The Vulnerability Ecosystem

Exploring vulnerability discovery and the resulting cyberattacks through agent-based modelling

Yorick Breukers
The Vulnerability Ecosystem

Exploring vulnerability discovery and the resulting cyberattacks through agent-based modelling

by

Yorick Paul Breukers

to obtain the degree of Master of Science
at Delft University of Technology,
to be defended publicly on Monday August 22, 2016 at 1:00 PM.

Student number: 4021053
E-mail: yorick.breukers@gmail.com
Project duration: March 1, 2016 – July 29, 2016
Thesis committee: Prof. dr. M. J. G. van Eeten, TU Delft, Chair
Dr. ir. W. Pieters, TU Delft, First supervisor
Dr. ir. I. Nikolic, TU Delft, Second supervisor
Dr. M. van Wieren, Deloitte Nederland, External supervisor

An electronic version of this thesis is available at http://repository.tudelft.nl/.
Before you lies my thesis for obtaining the Master’s degree in Systems Engineering, Policy Analysis, and Management. This thesis summarises the many different thoughts that passed my mind for the last five months, and hopefully, it is a coherent and convincing story. I found the subject highly interesting and can proudly say that I have enjoyed every day that I spent working on it. I am proud of the results and can honestly say that writing this thesis has been a rewarding experience.

This thesis dives into the topic of software vulnerabilities and the resulting attacks. The idea of this topic formed a while ago. However, I would like to thank Michel van Eeten for steering me in this direction. Although our time discussing my thesis was limited, the small time frame has proven invaluable, due to his ability to immediately point out any weak spots and his seemingly unlimited knowledge on this subject. In addition, I would like to thank Igor Nikolic for his very direct view on the application of the used method. His critical comments pushed me to extend the scope of the model and enabled the results that lay before you. Thirdly, I would like to thank Maarten van Wieren for his sober views and excellent comments on my thesis, and our philosophical discussions on topics completely unrelated to my thesis. In addition, I would like to thank the entire Deloitte Cyber Advisory team for their enthusiasm and offering a very stimulation environment to work on my thesis. Lastly, I would especially like to thank Wolter Pieters. His dedication and enthusiasm from the start have motivated and enabled me to finish this thesis within five months.

I would also like to apologise to the all the victims of my enthusiasm. Many of them where just in the wrong place at the wrong time and had to endure me talking about this topic without their consent. In the first place I would like to thank my friends and family for supporting me and providing the necessary distractions from time to time. Especially I would like to thank those who spend their free time reading through my thesis in order to reduce the seemingly endless supply of vulnerabilities in my text. A special word of thanks goes to my brother. I would like to thank him for his patience and willingness to read most of my thesis. Finally, I would like to thank Lisa for all her patience, endless support, and especially her joy.

Yorick Paul Breukers
Delft, January 2013
Numerous severe attacks on large organisations show that it remains a great challenge to minimise the risk and impact of a cyberattack. Software vulnerabilities are a major enabler for cyberattacks and are therefore responsible for a sizeable portion of that risk. The process of vulnerability discovery remains difficult to predict. This makes it difficult for software vendors to allocate sufficient resources to develop and deploy patches. In addition, organisations are challenged by the large number of patches that need to be deployed. The organisations are exposed to the risk of an unpatched vulnerability from the time that the vulnerability is discovered, until the moment that all systems within the organisation are patched. Better understanding the process of vulnerability discovery and the attacks caused by vulnerabilities, might help organisations to minimise their risk exposure and improve their odds in cyberspace. This challenge is the main goal of this thesis and is formalised by the following research question:

**Research question:** Which properties of the vulnerability ecosystem have a major influence on the vulnerability discovery rate of operating systems and what can vendors and organisations do to minimise the resulting attacks?

The complexity of this question lies in the system that surrounds the life cycle of a vulnerability. The lifecycle of a vulnerability is characterised by its discovery, exploitation, disclosure, and eventually, the patching of the vulnerability. Each of these phases is governed by different actors and mechanisms within the system. The discovery is the result of efforts of security researchers and hackers. Whether the vulnerability gets exploited or disclosed depends on their decision. The transfer of knowledge on the presence of a vulnerability or its exploit is facilitated by either the black markets for vulnerabilities, or bounty programmes through which the vulnerability is disclosed to the vendor. The exploitation and attack are actions performed by black hat hackers, while the development and release of a patch is primarily done by the software vendor, or open source community. Deploying the patch is the final action in the life cycle and the sole responsibility of organisations that use the software. The combined interaction of these actors and mechanisms is referred to as the vulnerability ecosystem, which is shown in the figure below.

The first part of the research question focusses on determining which properties of the vulnerability ecosystem have the most influence on the rate at which software vulnerabilities are discovered. These
are: the market share of the software, shared code between different software versions, learning effects of the discoverers, and the total number of vulnerabilities contained within the software. The second part of the question aims at analysing which properties of the vendors and organisations have the most influence on the resulting attacks. The properties that are tested for this part of the question, are: the vendor's ability to develop secure software, its patch development policy, and its patch release policy. The organisation's property that is analysed is the average time it takes to deploy a patch. An agent-based model is developed in order to simulate the vulnerability ecosystem and to find answers to both parts of the question. The model has been validated thoroughly; by comparing the behaviour of the model with empirical data, a comparison with existing research, and expert validation.

The results show that once software is released, it takes a while before it starts to pick up market share. The first discoveries of vulnerabilities coincide with the speed at which the software is adopted. This is caused by increasing interest in the software from discoverers. The transition of the discovery rate from a slow increase to the fast increase, is partially determined by the ease with which discoverers accumulate knowledge on the new software. This increase is constant until it becomes more difficult to find new vulnerabilities. However, this depends on how the software evolves, due to patches or by upgrades. The final decrease of the rate of discovery coincides with the users starting to adopt a newer version of the software and leave the current version behind. However, the validation of the model showed that this saturation of the rate of discovery is not realistic. Furthermore, additional experimentation revealed that this lack of saturation might be explained by a general increase of the interest in software vulnerabilities, higher amounts of shared code, and slow adoption of newer versions.

The results reveal that the vendor’s ability to develop secure software, and thus minimising the number of vulnerabilities in the software and the chance of introducing new vulnerabilities when releasing a patch, only has a limited influence on the number of attacks on the organisations. Far more important is how long it takes for the vendor to develop a patch and the timing of releasing a patch. In particular, continuously releasing patches can have a large influence on the number of attacks, when combined with longer patch development times. It is even more important that organisations deploy the patches as quickly as possible. The results show that an average difference of 60 days in the time it takes to deploy a patch, results in a 55% increase in the average attack frequency. This signifies the responsibility of the organisations themselves when it comes to minimising their risk exposure as a result of software vulnerabilities.

Based on the results, it can be concluded that both the defenders and vendors have a significant influence on the risk exposure of the defenders. The vendors can reduce the attack frequency on the defenders by balancing the right combination two key policies. First, how long it takes for the vendor to respond to the discovery of a vulnerability, and second, the timing of releasing the patch. However, this only works if the defenders also improve their capabilities and capacity for patch deployment, in order to significantly decrease the time it takes to test and deploy the patch throughout the systems within the organisation.

In addition to the direct answer to the research question, the model revealed that patching can lead to a surge in the number of attacks. Releasing a patch for a vulnerability that might only be known to the vendor or a small number of attackers, becomes public knowledge to all hackers, which opens the door the a huge increase in the number of attacks. This is exactly the behaviour that is observed in the model. The results show that not necessarily the number of vulnerabilities, but rather, the knowledge diffusion on their presence has a major impact on the number of attacks. Trying to patch all possible vulnerabilities, while taking into account the limited capacity of organisations to deploy all of them in a timely manner, might not be the best strategy. Instead, it might be worthwhile to allocate resources towards detecting and predicting which vulnerabilities have the highest likelihood of being exploited and primarily focus on those vulnerabilities.

The results have two interesting implications. First of all, the validation of the model and the additional experiments regarding the lack of saturation in rate of discovery, indicate that assuming saturation is not realistic and can potentially be easily explained by additional factors. Second, the result of this research, in combination with the lack of evidence that the number of vulnerabilities is decreasing, cast doubt on the effectiveness of public disclosure and bounty programmes. These mechanisms increase the number of patches, and as a result, put more strain on the organisation's capacity to deploy patches. At the same time, releasing a patch provides additional information to black hat hackers. It puts into question whether it makes sense for software vendors and security researchers to put tremendous effort in finding and fixing vulnerabilities, when it might not decrease the risk for the organisations.
## Contents

List of Figures                                      ix
List of Tables                                       xi

1 Introduction and approach                         1
   1.1 Cybersecurity: a general introduction         1
   1.2 Vulnerabilities                              2
   1.3 Problem and knowledge gap: a missing systems perspective 5
   1.4 Research approach                            6
   1.5 Thesis structure                             9

2 Vulnerabilities: the background                   11
   2.1 Perspectives on vulnerabilities              11
   2.2 The vulnerability life cycle                 15
   2.3 Related research and methods                 17
   2.4 Preliminary conclusion                       19

3 Influencing properties of the vulnerability ecosystem 21
   3.1 Influence on the vulnerability discovery rate 21
   3.2 Influence on the resulting attacks           26
   3.3 Preliminary conclusion                       28

4 Modelling complex adaptive systems                29
   4.1 The vulnerability ecosystem: A complex adaptive system 29
   4.2 Agent-based modelling and cybersecurity       30
   4.3 Simple versus complex: KISS or KIDS?          33
   4.4 Modelling process and challenges              34
   4.5 Preliminary conclusion                       35

5 Conceptualising the vulnerability ecosystem        37
   5.1 The agents                                   37
   5.2 Objects                                      44
   5.3 Environment                                  46
   5.4 Influencing properties and system overview    47
   5.5 Preliminary conclusion                       48

6 Model implementation and description               49
   6.1 Why NetLogo?                                 49
   6.2 Model narrative                              49
   6.3 Core mechanisms                              51
   6.4 Deviation from the conceptualisation          53
   6.5 Model dashboard                              53
   6.6 Model parameters                             54
   6.7 Verification                                 54
   6.8 Preliminary conclusion                       56

7 Exploration, experimentation and validation        57
   7.1 Confirmation versus exploration              57
   7.2 Exploring the model behaviour                58
   7.3 Setting up the main experiments               62
   7.4 Model validation                             65
   7.5 Preliminary conclusion                       72
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Section Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>Results</td>
<td></td>
</tr>
<tr>
<td></td>
<td>8.1 Influencing properties</td>
<td>73</td>
</tr>
<tr>
<td></td>
<td>8.2 Risk exposure</td>
<td>78</td>
</tr>
<tr>
<td>9</td>
<td>Conclusion and discussion</td>
<td>85</td>
</tr>
<tr>
<td></td>
<td>9.1 Answers to the research questions</td>
<td>85</td>
</tr>
<tr>
<td></td>
<td>9.2 Summary of the main findings</td>
<td>87</td>
</tr>
<tr>
<td></td>
<td>9.3 Reflection on the used method</td>
<td>89</td>
</tr>
<tr>
<td></td>
<td>9.4 Discussion on the implications</td>
<td>92</td>
</tr>
<tr>
<td></td>
<td>9.5 Recommendations</td>
<td>94</td>
</tr>
</tbody>
</table>

References 97

A Additional experiment: Saturation 103
B Additional results 107
C Model parameters 111
D Verification results 113
E Experimental setup 121
F Model exploration 125
List of Figures

1.1 Vulnerability life cycle (reprinted from Nappa, Johnson, Bilge, Caballero, and Dumitra (2015)) 3
1.2 Vulnerability discovery curve (adapted from Alhazmi and Malaiya (2005b)) 5
1.3 Research overview 7
1.4 Thesis structure 9

2.1 Life cycle of a vulnerability and its resulting intrusions (adapted from Arbaugh, Fithen, and McHugh (2000)) 16
2.2 Different states of a single host running vulnerable software (adapted from Arbaugh et al. (2000)) 16
2.3 Vulnerable systems and exploited systems over time (adapted from Rescorla (2005)) 17

4.1 Overview of the different parts of the research in relation to the agent-based model. 31
4.2 Lack of insight: changes in the vulnerability discovery rate over time 32

5.1 Conceptualisation of the vulnerability ecosystem 48

6.1 Flow chart of the model 50
6.2 Model dashboard 54

7.1 Vulnerability discovery rates of all software over 125 runs for non-experimental parameters 60
7.2 Vulnerability discovery rates of all software over 125 runs for experimental parameters 61
7.3 Comparison of the model behaviour and historic data on discovered vulnerabilities 67
7.4 Comparison of the agent-based model and the Alhazmi-Malaiya Logistic Model (AML) 68
7.5 Attack frequency after patch release (adapted from Arora, Nandkumar, and Telang (2006)) 69
7.6 Attack frequency and patching time of four semi-randomly selected vulnerabilities 70

8.1 Vulnerability discovery rates for market shares of 25%, 50%, and 75% 74
8.2 Vulnerability discovery rates for different knowledge build up times 75
8.3 Vulnerability discovery rates for different fractions of shared code. 76
8.4 Vulnerability discovery rates for different numbers of present vulnerabilities 77
8.5 Attack frequencies versus secure development capabilities 79
8.6 Attack frequencies versus patch development policies 80
8.7 Attack frequencies versus patch release policies 81
8.8 Overview of the attack frequencies for different patch development and release policies 82
8.9 Attack frequencies for proactive and reactive security strategies 83

9.1 The primary elements and relations of the vulnerability ecosystem 87
9.2 The vulnerability discovery rate and the influence of different properties during each phase 88

A.1 Vulnerabilities discovered versus different discoverer growth rates 104
A.2 Vulnerabilities discovered versus different amounts of shared code 105
A.3 Vulnerabilities discovered versus different amounts of shared code 105

B.1 Vulnerabilities discovered versus the number of present vulnerabilities 107
B.2 Vulnerabilities discovered versus the amount of shared code 108
B.3 Vulnerabilities discovered versus the knowledge build up time 108
B.4 Attack frequencies for different combinations of vendor properties 109

D.1 Total vulnerabilities per version after 4000 ticks 116
D.2 Attack frequency distribution for higher and lower market shares ............... 116
D.3 Attack frequency distribution for different vendor patch priority levels .......... 117
D.4 Attack frequency versus unpatched vulnerabilities .................................. 117
D.5 Missing data points in the parameter space. ............................................ 118
D.6 Vulnerabilities discovered over time for all versions of all vendors ............... 119
D.7 Vulnerabilities discovered over time for all second generation software ............ 120

F.1 Overview of the spread of different parameter settings per run ................. 125
F.2 Overview of the spread of different parameter settings per run .................... 126
# List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Overview of properties derived from hypotheses and assumptions in related research</td>
<td>23</td>
</tr>
<tr>
<td>3.2</td>
<td>Overview of properties derived from assumptions underlying VDMs</td>
<td>25</td>
</tr>
<tr>
<td>3.3</td>
<td>Selected properties that are suggested to influence the vulnerability discovery rate</td>
<td>25</td>
</tr>
<tr>
<td>3.4</td>
<td>Selected properties that influence the number of attacks</td>
<td>28</td>
</tr>
<tr>
<td>5.1</td>
<td>Overview of the elements of the vulnerability ecosystem</td>
<td>38</td>
</tr>
<tr>
<td>5.2</td>
<td>Overview of the properties and actions of vendors</td>
<td>39</td>
</tr>
<tr>
<td>5.3</td>
<td>Overview of the properties and actions of defenders</td>
<td>40</td>
</tr>
<tr>
<td>5.4</td>
<td>Overview of the properties and actions of discoverers</td>
<td>42</td>
</tr>
<tr>
<td>5.5</td>
<td>Overview of the properties and actions of attackers</td>
<td>43</td>
</tr>
<tr>
<td>5.6</td>
<td>Overview of the properties of software</td>
<td>45</td>
</tr>
<tr>
<td>5.7</td>
<td>Overview of the properties of vulnerabilities</td>
<td>46</td>
</tr>
<tr>
<td>5.8</td>
<td>Overview of the properties of exploits</td>
<td>46</td>
</tr>
<tr>
<td>6.1</td>
<td>Overview of the key parameters</td>
<td>55</td>
</tr>
<tr>
<td>7.1</td>
<td>General parameters for model exploration</td>
<td>59</td>
</tr>
<tr>
<td>7.2</td>
<td>Experiment specific parameters for model exploration</td>
<td>59</td>
</tr>
<tr>
<td>7.3</td>
<td>Implementation of the influencing properties in the model</td>
<td>62</td>
</tr>
<tr>
<td>7.4</td>
<td>Vendor properties and settings for risk exposure experiments</td>
<td>64</td>
</tr>
<tr>
<td>9.1</td>
<td>The likelihood of the defenders being attacked through vulnerabilities as a result of different vendor properties</td>
<td>89</td>
</tr>
<tr>
<td>A.1</td>
<td>Possible causes for the lack of saturation</td>
<td>103</td>
</tr>
<tr>
<td>C.1</td>
<td>Overview of the model parameters</td>
<td>111</td>
</tr>
<tr>
<td>E.1</td>
<td>Standard model parameter settings for experimentation</td>
<td>121</td>
</tr>
<tr>
<td>E.2</td>
<td>market share experiment settings</td>
<td>123</td>
</tr>
<tr>
<td>E.3</td>
<td>Learning effect experiment settings</td>
<td>123</td>
</tr>
<tr>
<td>E.4</td>
<td>Shared code experiment settings</td>
<td>123</td>
</tr>
<tr>
<td>E.5</td>
<td>Vulnerability depletion experiment settings</td>
<td>123</td>
</tr>
</tbody>
</table>
Introduction and approach

The Internet is a vital infrastructure for modern societies. It has become an inseparable part of our daily lives and every large organisation depends on its tremendous possibilities. Together with the digitalisation of most of the devices used today, this has resulted in the vast interconnectedness of our modern society. Unfortunately, this growth in interconnectedness through the internet, and the resulting interdependency of systems, is accompanied by exponentially increasing risks (Helbing, 2013). As organisations are moving most of their activities and transactions to cyberspace, so is the attention of people with malicious intentions. This has greatly increased the risks that organisations are facing through cyberspace. Dealing with the risks that companies face through cyberspace has proven to be a major challenge. The main goal of this thesis is to better understand these risks in order for organisation to improve their odds in cyberspace. This section starts with a general introduction on cybersecurity, followed by an introduction on the role of software vulnerabilities. This leads to the knowledge gap which forms the basis and justification for this thesis. This section concludes with the specification of the research questions, the goals, and the used method.

1.1. Cybersecurity: a general introduction

The news is riddled with numerous examples of cyberattacks. The National Cyber Security Center (2015) identifies threats ranging from professional criminals to scriptkiddies and hacktivists. This illustrates the broad range of possible attackers, each with different skill levels, resources, and objectives. Recent attacks show the sophistication of methods used by attackers to reach their desired objective. The Stuxnet attacks on Iran’s nuclear facilities in 2009 are a prime example and are believed to be carried out by U.S. and Israeli intelligence agencies (Menn, 2015). Other examples show that less sophisticated attacks can also have major impacts. The theft of 40 million credit card numbers at Target happened, despite advanced security systems, because security personnel ignored clear warning signals (Riley, Elgin, Dune, & Matlack, 2014). This shows that not only technical factors play a crucial role when it comes to cybersecurity, and thus requires organisations to address numerous factors in order to decrease their exposure to risks in cyberspace. This further complicates an already cumbersome task.

In order for an organisation to cope with these threats, they need an adequate cybersecurity strategy. Such a strategy determines, among other things, what technical measures are needed and how to respond when attacked. Determining the optimal strategy for an organisation, requires some under-
standing of the risks for that organisation. The risks for a bank for example, are different compared to those of a local retailer. Organisations need to understand which of their assets are at risk and how these assets can be compromised. In addition, organisations need to be aware what kind of attackers they are likely to attract. However, even if an organisation is fully aware of its threat landscape, its vulnerable assets, and has adopted security measures accordingly, this does not guarantee complete security.

Despite rapid development of cybersecurity solutions, complete security remains an utopia. Even organisations with mature cybersecurity practices remain vulnerable to both advanced and less advanced attackers. Highly sophisticated malware such as Stuxnet, Duqu, Flame, and Red October were able to deceive mature security solutions, and in some cases managed to avoid detection for years (Virvillis & Gritzalis, 2013). Other examples show that even unexperienced teenagers cause damage by using pre-made tools which enable them to easily conduct advanced cyberattacks (Lemos, 2000). These tools combine the knowledge and skill of advanced hackers and make them available to the masses. This greatly increases the risks for organisations in cyberspace, as even unskilled hackers can use a vast range of attacks by only pressing a button.

The vast majority of cyberattacks rely on vulnerabilities in software used by the targeted organisation. Software vulnerabilities played an important role in the aforementioned examples. Stuxnet, Duqu, and Flame each used unknown vulnerabilities, and Red October even used vulnerabilities for which a patch was already available (Virvillis & Gritzalis, 2013). Malware in general relies on zero-day and known vulnerabilities (Microsoft, 2011). The tools that are used by scriptkiddies rely on software vulnerabilities and automate the exploitation, making it easy to exploit vulnerabilities. Software vulnerabilities remain a serious issue for cybersecurity and thus a fundamental issue for the risk exposure of organisations. It is therefore important to improve understanding of the dynamics surrounding software vulnerabilities.

1.2. Vulnerabilities

Software has become a major enabler for modern businesses. Software drives communication, process, and transactions. It comprises of numerous lines of code; ranging from a few thousand for small pieces of software, to tens of millions lines of code for operating systems. It is impossible to ensure that all these lines of code are flawless. During the development it is inevitable that mistakes are made. Some of them do no harm; others however, might enable the software to be used for malicious purposes. These bugs are called vulnerabilities. However, vulnerabilities are not only the result of programming errors. They might also be the result of configuration or implementation mistakes. There are many definitions of a vulnerability. This project follows the definition by NIST (2002, p. 15) which states that a vulnerability is “a flaw or weakness in system security procedures, design, implementation, or internal controls, that could be exercised and result in a security breach or a violation of the system’s security policy.” This project focusses on those vulnerabilities that are the result of mistakes made by the software vendor.

According to some estimates, on average software contains 15 to 50 errors for every 1000 lines of code (McConnell, 2004). This implies a huge number of possible vulnerabilities. However, not all vulnerabilities can be used for exploitation, as not all have a code path that leads to an actual exploit (EEye, 2011). Still, a large number of vulnerabilities are reported each year. The National Vulnerability Database\(^1\) (NVD) offers an extensive overview of known vulnerabilities. It contains over 75,000 vulnerabilities; however, most of which are never exploited. According to Verizon (2015) between 50 and 100 unique exploited vulnerabilities are reported each year. Despite this seemingly low number, it remains

\(^1\)https://nvd.nist.gov
Vulnerabilities are a serious security threat, due to the fact that a new exploit for a yet unknown vulnerability, exposes each instance of that piece of software. This means that each organisation that uses that software is vulnerable to the exploit.

Vulnerabilities for which there is no patch available yet, are called zero-day vulnerabilities and are considered the holy grail for hackers. In combination with a working exploit, they can compromise every system that is affected by the vulnerability, and are therefore extremely valuable to hackers (Libicki, Ablon, & Webb, 2015). However, the large majority of the vulnerabilities in the NVD are not zero-day vulnerabilities. Most are either reported by security researchers or found by the software vendors themselves. The number of zero-day vulnerabilities that are used in the field is therefore limited. The majority of the exploits used in the field are based on vulnerabilities that are known for some time and for which a patch is already available (Verizon, 2015). This shows that both zero-day vulnerabilities and even well-known vulnerabilities have an impact on the risks for an organisation.

1.2.1. The vulnerability life cycle
For a more detailed look at vulnerabilities it is required to understand the life cycle of vulnerabilities. In general, four phases can be distinguished. These are: the point at which the vulnerability is discovered, the moment at which an exploit is developed for that vulnerability, the time at which the vulnerability is disclosed, and finally, the moment that a patch is released (Frei, May, Fiedler, & Plattner, 2006). It is important to understand that these actions do not necessarily need to take place in that exact order. Also, the moment when the patch is released is not necessarily the end of the life cycle of the vulnerability, as was illustrated by the Slammer virus case. However, this is the moment when the decay of the vulnerabilities starts and progresses as more systems are patched over time. Figure 1.1 (Nappa et al., 2015) provides an extensive overview of a linear life cycle of a vulnerability, including different stages in the patching cycle.

1.2.2. Uncertainty: how vulnerable are we?
The life cycle of a vulnerability is highly unpredictable. It is impossible to know which vulnerabilities are still waiting to be discovered, due to the huge number of bugs within software. This makes it hard to know which vulnerabilities are known and used by hackers. To make matters even worse; if a vulnerability is disclosed, it does not mean that the vendor or open source community has released a patch to fix it. This leaves the organisation exposed, even if the vulnerability is well known. Another challenge is getting a grip on which vulnerabilities are actually being exploited. The NVD continuously reports a vast number of vulnerabilities, each with a severity rating. This so-called Common Vulnerability Scoring System (CVSS) should enable organisations to determine which vulnerabilities are more severe, and
therefore require immediate attention. However, it is argued that these ratings do not reflect whether or not the vulnerability will actually be exploited by hackers (Allodi & Massacci, 2014). This makes it difficult for security operators to determine how vulnerable their systems actually are. This in turn makes it hard to allocate security resources accordingly.

1.2.3. Predicting vulnerability discovery
Dealing with the aforementioned uncertainty has proven to be a major challenge. An often proposed method is the Vulnerability Discovery Model (VDM). The goal of these methods is to determine the rate at which vulnerabilities are discovered relative to the age of the software. VDMs can be used to assess the risk associated with certain systems, due to the rate at which new vulnerabilities are likely to appear (Alhazmi & Malaiya, 2006b). This should enable organisations to determine with some degree of certainty how many vulnerabilities are likely to pose a threat in the near future. Some methods require access to the source code and are therefore only applicable to open source software (Roumani, Nwankpa, & Roumani, 2015). Most VDMs use past data on vulnerability discovery to predict future discovery rates (Alhazmi & Malaiya, 2005a; Alhazmi, Malaiya, & Ray, 2006; Woo, Joh, Alhazmi, & Malaiya, 2011). However, these models rely on several assumptions which limit their validity (Ozment, 2007). Other limitations are their inability to capture factors such as trend, level, and seasonality (Roumani et al., 2015). In general these models rely on data directly related to the technical characteristics of the software and fail to incorporate the broader context surrounding the software.

1.2.4. The vulnerability ecosystem
Technology has taken centre stage in most of the discussions on cybersecurity. However, Anderson (2001) argues that the economic perspective is at least as important as the technical perspective. More specifically, the incentives of actors and market failure can explain many of the insecurity witnessed in cyberspace (Anderson & Moore, 2006). The growing body of research on the economics of cybersecurity show its value and the importance for understanding and improving cybersecurity. This broader perspective should also be applied to vulnerabilities, as vulnerability discovery is not just the result of bad software development. Discovering vulnerabilities requires effort from both black hat hackers and white hat hackers. However, the factors, or more accurately the properties that determine the amount of effort spent on finding vulnerabilities has received little attention. This also includes the behaviour and interactions of software vendors and the actual users of the software. The goal of this thesis is to model the broader system that surrounds software in order to improve understanding on the properties of the system that influence vulnerability discovery and the resulting attacks.

1.2.5. Why some software is more attractive than others
The fact that a piece of software is badly developed, does not necessarily mean that it attracts a lot of hackers. Surprisingly little is known about the system properties that influence the vulnerability discovery rate. It is important to remember that there are different motivation for discovering vulnerabilities and that white hat hackers’ main goal is improving the security of software. One obvious explanation would be the market share or install base of the software. Software which is installed on a larger number of systems is a more attractive target, since this would enable the hacker to get access to more systems with the same method. The NVD for example contains much more vulnerabilities related to Windows compared to OSX, which has a significantly smaller market share. However, there is still no evidence that this is actually the primary explanation.

In general the vulnerability discovery rate of a piece of software follows a s-shaped curve (Alhazmi & Malaiya, 2005b). Figure 1.2 depicts the general development of the vulnerability discovery rate of
operating systems. During the lifespan of software the changes in the rate of discovery can be divided in three phases (Alhazmi & Malaiya, 2005b). The first phase is the learning phase in which hackers (both black hat and white hat) need to become familiar with the software. The second phase, termed the linear phase, during which the most vulnerabilities are discovered. And finally, the third phase, or saturation phase, where vulnerabilities are only discovered sporadically.

Figure 1.2: Vulnerability discovery curve (adapted from Alhazmi and Malaiya (2005b))

The main cause for this shape is attributed to the market share of the software (Alhazmi & Malaiya, 2005b), and does not consider other properties that might influence this shape. Besides the rational relation between the market share and the number of vulnerabilities, there is no hard evidence that this hypothesis actually holds. Another property that is suggested to influence the vulnerability discovery rate is a learning effect. Familiarity with the software might be an important determinant for how long it takes before the first vulnerabilities are discovered (Clark, Frei, Blaze, & Smith, 2010). Other suggestions are that a discovered vulnerability might lead to similar vulnerabilities, or that the methods and tools required for finding previous vulnerabilities might easily lead to subsequent vulnerabilities. Additionally, the fact that a vulnerability is found might be a sign of weakness to other hackers, resulting in more focus on that software (Clark, Frei, et al., 2010). Yet another explanation is the presence of shared code between different pieces of software (Nappa et al., 2015). If a vulnerability is found in a part of the software which is also used by other software, this vulnerability might also apply to that software.

1.3. Problem and knowledge gap: a missing systems perspective

Despite the numerous hypotheses on properties that influence vulnerability discovery rates, few research has aimed at analysing them and their relation to the observed vulnerability discovery rates. The question remains: How do the different relations, interactions, dependencies, technical and economic characteristics of the vulnerability ecosystem influence the vulnerability discovery rate? Research has been done on these relations on an individual level. Cavusoglu and Zhang (2006) assess the impact of patching strategies by modelling the interaction between a single vendor and a single organisation, or how a vulnerability propagates through different phases of the vulnerability life cycle (Al-Fedaghi, 2010). Another, yet valuable example is research by Arora, Krishnan, and Nandkumar (2004) on the relation between vulnerability disclosure and vendors’ patch release behaviour. Frei, Schatzmann, Plattner, and Trammell (2010) provide a model by combining some of the relations based on empirical research, yet they primarily focus on vulnerability discovery and disclosure. Despite extensive research
on these relations, a combined systems perspective, which might explain a part of the observed vulnerability discovery rate, is missing.

The problem surrounding the uncertainty of vulnerability discovery remains. The true exposure of organisations, due to the fact that they can never know how vulnerable they are, is a major concern. VDMs have been suggested to alleviate this uncertainty and enable vendors and organisations that use their software to focus security efforts. Despite promising results, their applicability remains limited due to the required access to source code or the reliance on past data on vulnerability discovery. The resulting problem is summarised by the following problem statement.

**Problem statement:** The uncertainty surrounding the properties of the vulnerability ecosystem that influence the vulnerability discovery rate, and how this in turn affects the risk exposure of organisations, makes it difficult for organisations to focus their security efforts and allocate resources accordingly.

VDMs offer some indication of how the vulnerability discovery rate will evolve over time for a particular piece of software (Roumani et al., 2015). However, they fail to grasp the underlying complexity of the vulnerability ecosystem. This means that changes in the external factors cannot be included in the forecast and might lead to invalid predictions. Combining the actors, relations, technical characteristics, and economic mechanisms of the vulnerability ecosystem might result in better insight in what causes hackers to focus on a particular system. Improved understanding of what properties influence vulnerability discovery and what should be done to minimise the resulting attacks, could help to decrease the risk exposure of large organisations.

### 1.4. Research approach

The aim of this thesis is to enhance understanding of the system properties that influence the vulnerability discovery rate and how these in turn influence the risks that organisations face in cyberspace. The objective, scope, contribution and research question are discussed below in more detail. The main method that is used for this research is agent-based modelling.

#### 1.4.1. Objective

Current research focuses predominately on the trends rather than the cause. Hypotheses have been formulated, some in the form of assumptions made in VDMs, while others are only suggestions on how the observed vulnerability discovery rates came to be. This thesis aims to capture the complexity of the vulnerability ecosystem and by doing so, improve understanding on vulnerability discovery rates and what properties influence the attack frequency on large organisations as a result of discovered vulnerabilities.

The goal, together with the required inputs and outputs, are shown in figure 1.3. It shows the input required for the system description. This includes the goals and behavioural characteristics of actors, and high-level characteristics of the software. The relations between the elements of the system are described as the vulnerability ecosystem. The hypotheses are tested as an additional input. This can be in the form of different relations or different behavioural characteristics of the actors. The output, or units of analysis, are the attack frequency, vulnerability discovery rate, and vulnerability life cycle. Especially the last two outputs can be used for validation purposes. This allows the hypotheses to be tested by assessing their effect on the model output and compare this behaviour with empirical data.
1.4.2. Scope

This research topic can include a vast range of smaller topics. In order to ensure the focus and preserve depth, the main focus is on operating systems and will not cover detailed technical characteristics, such as the development process, code complexity, and specific software components. There are several reasons for focusing on the operating systems instead of other types of software. In the first place, they are widely used since every PC or laptop requires one. This in turn makes it an attractive target for both hackers and researchers. As a result there is more data available from both the scientific community, and reports on vulnerability-related security incidents. The scope is further narrowed down to only a few years of simulation. This is a sufficient timeframe to cover several software releases and prevent the need for excessive computational power. The final demarcation concerns the user-side of this project. This project focuses only on large organisations and will not consider consumers as end users. The main reason for this decision is the differentiation in the way patching is managed and the fact that large organisations have the means to adopt actual cybersecurity strategies.

1.4.3. Method: agent-based modelling

Combining the behavioural characteristics of the actors, the technical characteristics of the system and economic mechanisms require an extensive modelling approach. Due to this complexity and the various perspectives of the system, agent-based modelling is used as the main modelling method. It is a suitable method, due to its ability to capture emergent behaviour, enabling a natural description of the system, and its high flexibility (Bonabeau, 2002). In addition, agent-based modelling is recommended for situations where it is important that agents learn and adapt, where agents show strategic behaviour and anticipate, when agents assess and cooperate, and when the past offers no prediction for the future (Pidd, Seibers, Macal, Garnett, & Buxton, 2010). Agent-based modelling has been used in the field of cybersecurity. However, only for specific purposes such as: network simulation for data generation (Gorodetski & Kotenko, 2002) and DDoS attacks and defence (Kotenko, 2005; Kotenko, Konovalov, & Shorov, 2010). Apart from these examples, agent-based modelling has not been applied to the general field of cybersecurity. More specifically, it has not been used for research on vulnerabilities. This thesis follows the agent-based modelling process as proposed by Nikolic, van Dam, and Kasmire (2013). A more detailed discussion and justification is presented in chapter 4.
1.4.4. Contribution
This project is relevant from both a scientific and a societal perspective. From a scientific perspective this project is relevant for two reasons. First, this project contributes to ongoing research in the field of cybersecurity by providing insight in the system properties that influence the vulnerability discovery rate. Second, it examines how different properties influence the attacks induced by vulnerabilities. And third, it assesses the potential and value of agent-based modelling for this kind of research. From a societal perspective this project is relevant since it enables organisations to better understand the risk they face in cyberspace.

1.4.5. Question
This project revolves around the main research question, as presented below. The main research question is subdivided in several sub-questions in order to guide the research process. Each sub-question represents a part of the research process.

**Research question:** Which properties of the vulnerability ecosystem have a major influence on the vulnerability discovery rate of operating systems and what can vendors and organisations do to minimise the resulting attacks?

The main research question is divided in the sub-questions listed below. The first sub-questions have a more theoretical focus. This focus steadily shifts towards the modelling part of the project. The final sub-questions relate to answering the main research question by reflecting on the results from the model and providing recommendations.

1. What processes drive the general behaviour of the vulnerability ecosystem?
2. What properties of the vulnerability ecosystem are suggested to influence the vulnerability discovery rates and the resulting attacks according to the literature?
3. Which actors, technical characteristics, and mechanisms should be included in the conceptual model?
4. What can be learned from the agent-based model on the properties that are suggested to influence the vulnerability discovery rate?
5. What can be learned from the agent-based model on the influence of the properties of the defenders and vendors on the risk exposure of large organisations?

The first sub-question will be answered through a literature review of scientific literature. This results in a broad overview of the processes within the vulnerability ecosystem. This forms the starting point of the conceptualisation. The aim of sub-question 2 is to identify the properties that are suggested to influence the vulnerability discovery rate. Sub-question 3 covers the conceptualisation by combining the agents’ properties, behaviour, and interactions from sub-question 1. Sub-question 4 assesses the results from the experiments on what properties influence which parts of the vulnerability discovery curve. The final question, sub-question 5, explores what vendors and large organisations should do in order to minimise the risk exposure of large organisations in cyberspace as a result of software vulnerabilities.

1.4.6. Clarification of the term “properties”
Before diving into the details of the research approach, a clarification of the term “properties” is required. Properties in the context of this thesis refers to the characteristics of the vulnerability ecosys-
This system comprises the different actors, the software, the vulnerabilities in this software, and the relations and interactions between these elements. Therefore, the term properties refers to both behavioural characteristics of these agents and the fixed properties of the elements within the system or their relations.

1.5. Thesis structure

This thesis is divided into 9 chapters. Each chapter covers one or, when related, multiple parts of the research. The main focus of the different chapters and their relation to each other is shown in figure 1.4. Straight lines indicate the main sequence of the chapters. Dotted lines indicate a relation between specific parts of the chapters. The overview also shows which chapter covers which sub-question.

Figure 1.4: Thesis structure
Vulnerabilities remain a major problem for cybersecurity, as was discussed in the previous chapter. In order to get a better understanding of why this is such a problem, a more detailed understanding of the complexity behind the dynamics of the vulnerability ecosystem is required. The goal of this chapter is to provide background information on this complexity, by approaching the vulnerability ecosystem from multiple perspectives. In addition to these perspectives, an economic perspective is also discussed here. This chapter concludes with a more detailed introduction on the vulnerability life cycle and methods to predict vulnerability discovery rates.

2.1. Perspectives on vulnerabilities

This section discusses the different perspectives on vulnerabilities. The perspectives are based on the different actors that are part of the vulnerability ecosystem. The main reason for discussing the system from the perspective of the actors is the fact that their behaviour determines the life cycle of a vulnerability, starting with the creation and discovery of the vulnerability, and ending with the moment at which all vulnerable systems are patched. The main actors are the developers of the software; the software vendors, the black or white hat hackers who find the vulnerabilities; the discoverers, the black hat hackers who exploit the vulnerabilities; the attackers, and the organisations that are affected by the resulting cyberattacks; the defenders.

2.1.1. The discoverer’s perspective: a skilled search

Vulnerabilities are the results of bugs in software. As was mentioned in the previous chapter, software contains 15 to 50 errors for every 1000 lines of code on average (McConnell, 2004). However, not all these bugs can be exploited in order to gain access to a system. It is estimated that between 1% to 5% are actual vulnerabilities (Anderson, 2002; Longstaff, 2003; Alhazmi et al., 2006). There are several types of vulnerabilities. These include cross site request forgery, SQL injection, memory corruption, cross site scripting, and buffer overflows. Despite the large number of vulnerabilities within the software and many different types of vulnerabilities, discovering them remains a challenge. Discovering previously unknown vulnerabilities requires considerable technical skills and cannot be automated by applying algorithmic methods (Algarni & Malaiya, 2014). This might explain why the majority of those who discover a zero-day vulnerability are unlikely to find another one (Libicki et al., 2015). Finding vulnerabilities requires detailed knowledge about the software that is being examined, and the ability
to reverse engineer software when the discoverer has no access to the source code. Therefore, skill plays an important role when it comes to discovering vulnerabilities.

Another property that determines the behaviour of the discoverer is the motivation that drives the discoverers. Searching for vulnerabilities is not always a bad thing. Many discoverers put tremendous amounts of effort in discovering vulnerabilities, driven by ethical reasons. For some discoverers receiving credits for a discovered vulnerability is enough reward to actively search for vulnerabilities (Algarni & Malaiya, 2014). The important distinction is that the motivation of the discoverer influences what happens with a vulnerability when it is discovered. If the discoverer is motivated by improving software and reports the vulnerability to the software vendor, or when the discoverer offers the vulnerability to the vendor in return for financial reward, the vendor can use this knowledge to develop a patch. On the other hand, if the discoverer has malicious intentions, the vulnerability is likely to end up on a black market, where it is sold and used for exploitation. Despite efforts of software vendors to use bounty programmes to create financial incentives to report vulnerabilities to vendors, their prices do not match those on the black market (Libicki et al., 2015). Their effect on the behaviour of discoverers is thus limited. Therefore, the motivation of the discoverer has major impact on how a vulnerability affects the risk exposure of the defender.

2.1.2. The attacker’s perspective: getting to the gold

The required skills for vulnerability discovery make it a specialised trade, which is not practiced by the masses. Hence, it is useful to make a distinction between those who perform the skilled task of discovering vulnerabilities, and those who exploit vulnerabilities in order to perform the actual cyber-attack (Algarni & Malaiya, 2014). This distinction is facilitated by markets for vulnerabilities. These markets are discussed in detail in section 5.1.3. Using existing exploits requires much less skill and can therefore be done by a much larger group of people. Especially the existence of tools such as Kali Lunix make it relatively easy to perform cyberattacks without extensive technical skills. Kali Lunix is a tool for security testing and offers a broad range tools to conduct many different types of analysis and attacks. Although the main motivation behind the tool is to support ethical hackers in security testing, it is commonly known to be extensively used by black hat hackers as well. Tools like these enable even unskilled people to cause serious damage. This increases the likelihood of attacks, since it enables the masses to leverage the skills of more experienced hackers and discoverers.

The threat landscape in cyberspace comprises of a broad range of potential attackers. The National Cyber Security Center (2015) distinguishes between the following eight types of threats: professional criminals, nation states, terrorists, cyber vandals and scriptkiddies, hacktivists, internal actors, cyber researchers, and private organisations. Each types of threat has its own characteristics in the form of their goals, targets, means, and skills. This determines the attack strategy they apply and the extent to which they have access to vulnerabilities and exploits.

2.1.3. The vendor’s perspective: sell first, fix later

The main goal of a software vendor basically comes down to releasing and selling software. Making sure that the software goes to the market as soon as possible is important in order to secure its place in the market. The downside of this urge to release software as soon as possible, is that shorter development times have a negative impact on the quality of the software, and this software is more likely to contain vulnerabilities. Arora, Forman, Nandkumar, and Telang (2010) show that vendors with strong market positions have less incentive to release software on a short notice, and as a result their software contains less bugs. On the other hand, vendors in more competitive markets release software earlier with more bugs, and therefore require more budget for patching after the release of the software.
2.1. Perspectives on vulnerabilities

Software can be found in all parts of large organisations. They depend on it for most of their daily operations. The question rarely is: do we need software? Rather, the question is: what kind of software do we need? This makes the organisation dependent on software and especially, the software vendor. This effect is further strengthened by the lock-in effects which make an organisation strongly reliant on the vendor (Farrell & Klemperer, 2007). These and other characteristics of the software market offer little incentive for the vendor to improve security of the software (Anderson, 2001). This further complicates the reliance of the organisations on the vendors ability to provide secure software and decreases the incentive for vendor to provide it.

Software contains vulnerabilities, no matter how much effort vendors put in secure development and testing. The primary reason for those vulnerabilities is the rapid evolution and increasing complexity of software (Takanen, Demott, & Miller, 2008). The main goal of the software vendor is to provide software that satisfies their customers, and this includes providing secure software. However, finding vulnerabilities is just as hard for the vendor as it is for actors with malicious intentions. One widely used method is called fuzzing. Fuzzing is a “highly automated testing technique that covers numerous boundary cases using invalid data as application input to better ensure the absence of exploitable vulnerabilities” (Takanen et al., 2008, p. 1). Essentially brute forcing errors with the goal of uncovering potential vulnerabilities. However, fuzzing is by no means a panacea, as it is an expensive method, and only discovers relatively simple vulnerabilities (Libicki et al., 2015).

More complex vulnerabilities require dedicated researchers who are able to assess software from unique angles (Libicki et al., 2015). This is one of the reasons why the white market for vulnerabilities gets increasingly more attention from software vendors. In addition, there are multiple companies that offer vulnerability discovery services to software vendors. The white market however has to compete with the prices offered on the black market, which are substantially higher. This makes vulnerability discovery an almost prohibitively expensive task for vendors. To make matters worse, there is little evidence that increasing the rate at which vulnerabilities are found, will eventually deplete the number of vulnerabilities, and thus does not improve the security of the software (Rescorla, 2005). This decreases the attractiveness from a vendor’s perspective to allocate more resources to vulnerability discovery.

Another factor that adds to the complexity of the impact of vulnerabilities, is the discussion surrounding disclosure. When a vulnerability is found, it still takes time to develop a patch. In the meantime, the question remains whether the vendor, or the person who discovers it, should disclose the vulnerability or not. Publishing the vulnerability results in better informed customers and enables them to take mitigation actions. On the other hand, research has shown that if zero-days are disclosed, the number of malware variants exploiting them, increase dramatically, as well as the number of attacks (Bilge & Dumitras, 2012). In addition, disclosing vulnerabilities does have a significant impact on the release time of a patch by the vendor. Arora, Krishnan, Telang, and Yang (2010) show that when vulnerabilities are disclosed, on average, a vendor releases a patch 35 days earlier compared to when there is no disclosure. This shows that the way vulnerabilities are handled has a significant impact on the risk for the vendor’s customers.

2.1.4. The defender’s perspective: patch management

Dealing with vulnerabilities seems to be a straightforward task for organisations. When a vulnerability is discovered, they wait for the vendor to release a patch and implement it as soon as possible. However, this does not reflect reality. An interesting example is the Slammer virus, which struck networks worldwide and caused major outages. The vulnerability that was exploited by the Slammer virus was well known by then, and Microsoft had already released a security patch six months prior to the attack (McGhie, 2003). The entire incident could have been prevented if all these organisations had applied the provided patch. So why did this not happen? Cavusoglu and Zhang (2006) summarise some of
these reasons, including: the patch needs to be tested by organisations, sometimes it requires reconfiguration, distribution of patching is not always standardised, if a patch is needed for a critical system its implementation is complex and time consuming, and finally, patch management is expensive.

The way that patches are handled by the defender is determined by its security strategy. A cybersecurity strategy encompasses a great number of strategic decisions, managerial and operational decisions, and technical decisions (Da Veiga & Eloff, 2007). Broadly speaking, a distinction can be made between proactive versus reactive security investment strategies, and preventive versus detect and respond security strategies. Proactive refers to the practice of proactively analysing the threat landscape and investing accordingly. This method reduces security incidents, but it requires substantive upfront investment (Kwon & Johnson, 2011). Organisations that adopt a reactive strategy wait and observe security incidents and use that knowledge to allocate security investments. Therefore, this is more cost-efficient and can be preferred in some situations (Böhme & Moore, 2010). In addition, defenders can opt for a security strategy which focusses on prevention instead of detect and respond approach. This consideration refers to not only decreasing the likelihood of an attack, but also the impact of an attack.

The ability to patch also depends on the timing of the vendor to release a patch. There are two common release policies for vendors. Patch release is either event-driven or the more common periodical patch releases. For example, Patch Tuesday, which refers to the second Tuesday of each month when Microsoft releases all their patches. The main argument for periodical patch releases is to allow organisations to plan the patch implementation and decrease the burden on system administrators (Cavusoglu & Zhang, 2006). However, this does leave the system vulnerable in the meantime. It also raises the question on who is liable when an incident occurs and a patch was developed, but not yet released.

An additional property which influences the risk of an organisation is the amount of shared code between different software packages in relation to the number of packages used within an organisation. Nappa et al. (2015) show that shared libraries among different software packages significantly increase the risk, because a single vulnerability can effect multiple software packages and therefore leaves an organisation vulnerable, as long as one of the software packages remains unpatched. In addition, old versions of software that remain installed might also be vulnerable and are less likely to receive a patch. This further complicates the impact of vulnerabilities.

2.1.5. Economic perspective: markets for vulnerabilities

Financial gain plays an important role for criminals. This is no different for those that operate in cyberspace. There are of course other motivations for cyberattacks, such as hacktivism and terrorism. However, the large majority of the persons or organisations behind cyberattacks have financial gain as their main motivation (National Cyber Security Center, 2015). Hence, it is important to understand the economics behind vulnerabilities and exploits.

Vulnerabilities and especially their corresponding exploit can be very valuable on the black market, but what exactly determines the price of a vulnerability? The ease with which a vulnerability is discovered and exploited does not primarily determine the price, nor its severity (Libicki et al., 2015). A severe vulnerability, which might be difficult to exploit, can be less valuable compared to a less severe vulnerability which is easier to exploit (Microsoft, 2013). Other properties that determine the price are the duration that the vulnerability remains secret, the chance of the exploit being detected, and the ability to link several vulnerabilities, as in most cases multiple vulnerabilities and their exploits need to be linked in order to achieve the desired goal (Libicki et al., 2015).

Black markets have emerged in order to facilitate the trade between those that discover vulnerabilities and those who demand vulnerabilities. These black markets are organised and operated for
cybercrime (Ablon, Libicki, & Golay, 2014). Obscurity and privacy are an important aspect of these marketplaces. They operate like a regular market, where supply and demand determine the prices. Zero-day vulnerabilities for example, are extremely expensive, because they are hard to find, have a severe impact, and their corresponding exploit is hard to develop (Libicki et al., 2015). Another important property is the fact that a zero-day loses part of its value when it is used. This is due to the possibility of it being detected and thus increases the chance of the vulnerability being patched (Fidler, 2014). Only a tiny fraction of the vulnerabilities and exploits that are traded on the black market are zero-days, since they are not suitable for the mass market malware (Libicki et al., 2015). Most of the black market trade concerns vulnerabilities which are relatively new, but where a patch is already available. These are the so-called half-days, 1-days, or 2-days (Ablon et al., 2014).

In addition to the black market there are also grey and white markets for vulnerabilities. The grey markets are mostly funded by the government entities. The buyers on the grey market primarily operate for governments and intelligence agencies (Libicki et al., 2015). The goal of the buyer on the grey market is to support both defensive and offensive capabilities. This is in contrast with the black market where the main goal is to get access to systems and thus offensive capabilities. The motivation for the sellers to participate on the grey market is similar to the black market, which is financial gain (Libicki et al., 2015). Prices on both markets are roughly similar.

The white market on the other hand, is rather different. The white market is the result of efforts by software vendors to improve the security of software by stimulating vulnerability discovery and reporting it to the appropriate vendor. Supply and demand have less influence on the prices, since a vulnerability only has a single potential buyer. The financial motivation is of lesser importance, due to the fact that prices on the white market are significantly lower. Instead, white markets try to attract security experts for ethical reasons and by offering recognition (Libicki et al., 2015). The fact that there are three different kinds of market places, each with different characteristics and mechanisms, shows the complexity of the economics behind vulnerability discovery and trade.

### 2.2. The vulnerability life cycle

The previous section provided some insight on the vulnerability ecosystem from the perspective of the most important actors. However, it did not cover the vulnerability ecosystem from the perspective of the vulnerabilities. An interesting perspective on vulnerabilities becomes apparent by assessing their life cycles. Arbaugh et al. (2000) propose a vulnerability life cycle model based on three extensive case studies. Figure 2.1 shows how the number of intrusions caused by a vulnerability evolves over time. It starts with the creation of the vulnerability during the development of the software. After a while the vulnerability is discovered. This might lead to the first intrusions, however, only on a small scale. The bulk of the intrusions take place after the disclosure of the vulnerability. This increase starts to decline slowly after a patch is released and the life cycle ends when all the vulnerable systems are patched.

When looking at individual systems Arbaugh et al. (2000) distinguish three states in which a system can be. The different states and the way a system progresses through those states are shown in figure 2.2. The system is secure if there are no discovered vulnerabilities. If at any time, a vulnerability is discovered, the systems moves to the vulnerable state. When the vulnerability is exploited the system becomes compromised. The system returns to the secure state when it is patched.

Other models for the vulnerability life cycle have been proposed. In general these models are more extensive in the sense that they put more focus on the patching phase (Frei et al., 2010; Nappa et al., 2015), or emphasise that a vulnerability can pass through the phases in different sequences (Al-Fedaghi, 2010). However, the model proposed by Arbaugh et al. (2000) captures the core of the life cycle and provides a concise framework for analysing them. The following sections discuss the main
2. Vulnerabilities: the background

phases of the vulnerability life cycle.

**Discovery** A vulnerability is present in the software from the moment the software is developed. It is however not alive until the moment that the vulnerability is discovered. This gap between the moment that the software is introduced and the moment that the first vulnerabilities are discovered is called the honeymoon period. The length of this period is primarily determined by the familiarity of the discoverers with the software (Clark, Frei, et al., 2010). The relation between the discovery of subsequent vulnerabilities is however less clear. In general it is assumed that the discovery of vulnerability is an independent event. However, others have argued the this is not the case, because sometimes an entire class of new vulnerabilities is discovered at once (Ozment, 2007), or because the vulnerability is the result of a common mistake made by the developer and is likely to be found at multiple places (Anderson, 2002). Despite these arguments, vulnerability discovery remains hard to predict.

**Exploitation** A vulnerability on its own is not yet a problem. However, once an exploit is available it does become problematic. Timing is an interesting factor when assessing the rate of exploitation. Looking at figure 2.1 it can be seen that the attacks increase after the disclosure of the vulnerability. This is in line with findings from Arora et al. (2004), who show that undisclosed vulnerabilities are exploited...
far less often than disclosed or patched vulnerabilities. Even more interesting is that the majority of the attacks take place after a patch is released. This is contrary to what one would expect. One explanation is that the automation of the exploit enables a much broader range of hackers to use it and this causes a rapid increase in the number of attacks (Arbaugh et al., 2000).

**Disclosure** Whether or not to disclose a vulnerability is an ongoing debate. Numerous authors have discussed the merits and limitations of vulnerability disclosure (Arora et al., 2006; Böhme, 2006; Cavusoglu, Cavusoglu, & Raghunathan, 2007; McQueen, Wright, & Wellman, 2012; Ransbotham & Mitra, 2011). Those in favour argue that disclosing a vulnerability forces a vendor to develop and release a patch as soon as possible (Nizovtsev & Thursby, 2005). While others point out that full disclosure only works when the developer is able to release a patch on short notice, and under the conditions that the users patch promptly (Arora & Rahul, 2005). Disclosing a vulnerability publicly results in awareness among attackers. This explains why a majority of attacks occurs after the vulnerability is disclosed. This remains a heated debate with no definitive answer. Nevertheless, it is clear the disclosure has a major influence on the attack frequency.

**Patching** The final part of the life cycle is patching the vulnerability. Releasing a patch starts the decay of the vulnerability. However, this can take a long time. The patching process is shown in figure 2.3. The figure again depicts that the majority of the systems are compromised after the patch is released. The rate at which the systems are compromised decreases as more systems are patched. This decreases the number of available targets, and therefore the intrusion rate. As has been discussed in the first chapter, patching is not as straightforward as one would expect. There are many factors at play here. Especially for large organisations it requires planning and testing before it can be rolled out over all their system. This explains why it takes a long time before most of the systems are patched and the vulnerability reaches the end of its life cycle.

![Figure 2.3: Vulnerable systems and exploited systems over time (adapted from Rescorla (2005))](image)

### 2.3. Related research and methods

The previous sections discussed the multiple perspectives on the vulnerability ecosystem. However, it does not cover the methods that are proposed to deal with this complexity. Modelling the cybersecurity problems has been performed in many different ways for varying purposes. Two fields of research are particularly interesting; vulnerability discovery and prediction models, and vulnerability simulation...
models. Both fields are discussed in the following sections.

2.3.1. Vulnerability discovery models
Predicting the rate at which vulnerabilities are discovered is seen as an important step towards improving the odds of organisations in cyberspace. It enables vendors to predict how much resources they should allocate during which stage of the life cycle of their software. This enables them to release patches on a shorter notice and with improved quality. For the users, in this case large organisations, it allows them to anticipate when patches are likely to be released and therefore better prepare. Vulnerability discovery models (VDM) aim to achieve that. In addition, these models can be used to assess the risks when adopting new systems (Alhazmi et al., 2006). However, they are not widely adopted in the field and their benefits are not always clear. The goal of this section is to provide some background on the different types of VDMs, their methods, and their limitations.

The first model that assessed the number of vulnerabilities and their vulnerability discovery rate was proposed by Anderson (2002). This model was based on a thermodynamics analogy. The first models that are based on large scale vulnerability data were introduced by Alhazmi and Malaiya (2005a, 2006b); Alhazmi et al. (2006). The models were fitted using large data sets on vulnerabilities related to operating systems. These models reflect the s-shaped curve as shown in figure 1.2. A different approach is suggested by Roumani et al. (2015). Their model uses time series analysis to improve predictive capabilities.

Despite the progress being made in this field, there remain some limitations that diminish the validity and applicability of these models. One of the limitations of these models is the assumption that different versions of operating systems are independent of each other. However, this is not realistic since code reuse is very common among different versions of operating systems (Alhazmi et al., 2006). Another limitation is found in the simplifying assumptions that underly these models (Ozment, 2007). In addition, Roumani et al. (2015) point out that previous models do not incorporate the trend, level, and seasonality components of vulnerabilities. Another important limitation is the reliance on the past data for fitting most of the earlier models. This makes it difficult to apply these models to new software for which no, or limited, historic vulnerability data is available. One way of dealing with the lack of historical vulnerability data is analysing code characteristics (Rahimi & Zargham, 2013). The downside of this approach is that it requires access to the source code of the software.

These models provide valuable insights in how the rate of discovery evolves over time. In addition, they capture some of the behaviour of the vulnerability ecosystem. It should be noted that the goal of this thesis is not compete with, or replace VDMs. In contrast, this thesis provides additional insights in the properties of the vulnerability ecosystem in general, rather than focussing specifically on the software. Therefore, this thesis should be seen as complementary to the existing models.

2.3.2. Modelling the vulnerability ecosystem
Despite the extensive research conducted on the dynamics behind vulnerabilities, little research has focussed on the risk induced by vulnerabilities. Some have focussed on modelling the vulnerability discovery process, as is discussed in the previous section. However, few studies have aimed at covering a broad perspective on the vulnerability ecosystem. One example is the model proposed by Radianti and Gonzalez (2007). Their model uses system dynamics to analyse the growth of black markets in order to analyse the potential for white markets. The main limitation is their key focus on the market, without incorporating the individual response by vendors and defenders to newly emerged vulnerabilities. Although this paper shows the dynamics of the vulnerability markets to some extent, it does not cover the risk of the attacks as a result of the vulnerability markets. Some models have been presented
2.3. Related research and methods

that do focus on the relation between vulnerabilities and attacks. One of those models determines the progression of the number of attacks that occur through a vulnerability (Browne, Arbaugh, McHugh, & Fitthen, 2001). However, this model requires data on reported attacks for that specific vulnerability, and can therefore only be applied to that vulnerability. A more elaborate model presented in Joh and Malaiya (2011) focuses on the attacks as a result of discovered vulnerabilities and the patching by the users. An important limitation of this model is that it omits the vulnerability black market. Other work takes a very detailed look at the discovery process and trade of vulnerabilities (Algarni & Malaiya, 2014). However, it does not take into account the complexity of patch release and patch deployment. A third model has been presented that focuses on the decision made by the discoverers on what to do with a discovered vulnerability (Frei et al., 2010). It provides an extensive perspective on the potential options that the discoverer has. However, this model omits the important interactions between the vendors and organisations once a vulnerability becomes public, such as attacks, patch release, and patch deployment. Another model focuses on the time window between patch release and deployment (Cavusoglu & Zhang, 2006). However, given its game-theoretical nature, this model only focuses on a single vendor and organisation that uses its software. As a result, the model misses the complex interactions between discoverers, exploits and mechanisms such as black markets and bounty programmes.

A final, though highly relevant, paper by Arora et al. (2006), analyses the attack frequency as a result of the publication of a vulnerability through disclosure or patching. Based on empirical data, the paper argues that patching a vulnerability significantly increases the number of attacks. However, one of the limitations is that their analysis does not assess what kind of behaviour the software vendors and organisations exhibit, other than the publication and release of a patch by the vendor. Despite valuable contributions, none of these studies cover all the elements of the vulnerability ecosystem. By taking a broader perspective on the system surrounding vulnerabilities, this thesis analysis the influence of the behaviour of different actors on resulting cyberattacks.

2.4. Preliminary conclusion

This chapter provided background information on vulnerabilities in order create a more detailed understanding of the complexity behind the dynamics of the vulnerability ecosystem. It provided insight in the complexity by approaching the systems from the perspective of the main actors, which are: the developers of the software; the software vendors, the black or white hat hackers who find the vulnerabilities; the discoverers, the black hat hackers who exploit the vulnerabilities; the attackers, and the organisations that are affected by the resulting cyberattacks; the defenders. This chapter continued by discussing the four stages of the vulnerability life cycle in more detail. These are the discovery, exploitation, disclosure, and patching. The different sections identified the primary characteristics of the vulnerability ecosystem, and by doing so, it provided the answers to sub-question 1. These characteristics form the basis for the conceptualisation in chapter 5.
Influencing properties of the vulnerability ecosystem

The goal of this chapter is to identify the properties of the vulnerability ecosystem that influence the vulnerability discovery rate and the attack frequency as a result of discovered vulnerabilities. Both parts of the question require a different approach. The first part concerns the identification of the properties that are suggested to influence the rate of discovery. These properties are derived from hypotheses and assumptions from related literature. This is discussed in section 3.1. The second part of the question concerns the properties of vendors and organisations that influence the number of attacks enabled by vulnerabilities. This part requires a different approach and is discussed in section 3.2.

3.1. Influence on the vulnerability discovery rate

This section provides a detailed analysis of the properties that influence the vulnerability discovery rate of software and through this analysis, answers sub-question 2: what properties are suggested to influence the vulnerability discovery rates? The properties are derived from related research presented in the previous chapter. More specifically, two sources can be distinguished for these properties within the body of literature. The first source being hypotheses and suggestions on these properties, mentioned in the related work. The second source being the assumptions made within Vulnerability Discovery Models (VDMs). These assumptions are either the result of missing knowledge, or these assumptions are the result of the limitations of the chosen method. Both cases are worth investigating in more detail for this research.

There are numerous properties of the vulnerability ecosystem that influence the vulnerability discovery rate. Providing a truly complete overview of all these properties would be too cumbersome and too time consuming for the goal of the this thesis. Therefore, the decision has been made to cover only properties which are either non-technical or external to the software itself. Most of these properties have been covered in the previous chapter and for that reason, no extensive overview of related research is presented here. Instead, they are discussed briefly and only when it concerns direct mentioning of either assumptions or hypotheses on properties influencing the vulnerability discovery rate.

This chapter is structured as follows. Section 3.1.1 provides an overview of properties derived from hypotheses and suggestions from related research. This is followed by an overview of the properties
that are derived from assumptions made in VDMs in section 3.1.2. Based on this overview a selection is made in section 3.1.3. The remainder of thesis focusses on the properties selected in this chapter.

3.1.1. Hypotheses from related research

One the most influential works on vulnerability discovery rates and VDMs is from Alhazmi and Malaiya (2005a, 2005b, 2006a, 2006b) and Alhazmi et al. (2006). Their work focusses on several versions of Windows, Microsoft IIS, Apache, and Red Hat Linux. They are one of the first to present several VDMs based on historical vulnerability data. The resulting models follow the s-curve as shown in figure 1.2. They state several hypotheses for why their models, based on different software, follow the distinctive s-curve.

One of the most mentioned properties that is suggested to influence the s-curve, is the market share. It is argued that the market share influences the effort that is put into discovering vulnerabilities, since a larger market share offers a bigger incentive to explore, as this is more profitable or satisfying for the discoverers (Alhazmi & Malaiya, 2006a). It is important to note that this is similar to the term installed base, as it was called in Alhazmi and Malaiya (2005b). Other research that mention the influence of the market share on the vulnerability discovery rate are Woo et al. (2011), S. Kim et al. (2010), and Roumani et al. (2015). This thesis refers to this property as market share instead of installed base. It can be argued that there is a clear difference between the term market share and installed base, as the first can also refer to the financial market share, whereas the latter only refers to the number of installations. However, with consistency in mind, this thesis refers to market share as the share of installed systems of the total population.

Age is another property which potentially influences the vulnerability discovery rate. The s-cure shows a period of little activity, followed by a period of rapid increase of the number of discovered vulnerabilities, and finally an ongoing period of little to no activity. According to Joh (2011) the age of the software has a significant impact on the vulnerability discovery rate. However, this suggestion has received relative little support from other authors. This might be due to other properties, such as increased market share or the introduction of a new version, that actually cause the changes over time, rather than just the age of the software. A new version causes users to switch and upgrade, which decrease the market share of the older version, and thus makes it less attractive for discoverers (Alhazmi & Malaiya, 2005b; Alhazmi et al., 2006).

Another property that hypothetically influences the vulnerability discovery rate, and especially the slow start of the curve is the familiarity with the software. Alhazmi and Malaiya (2005b) argue that there is a learning phase during which the discoverers need to get familiar with the software in order to gather the knowledge required to break the software successfully. This claim is supported by Khoo, Aloteibi, Anderson, and Meeks (2010) who spent about half their effort on getting familiar with the software and tools, before they started discovering vulnerabilities at a steady rate. This period is referred to as the honeymoon period by Clark, Frei, et al. (2010). It also has an influence on the impact of shared code between software. Therefore, this learning effect might have a major impact on the vulnerability discovery rate.

Shared code is argued to have a major impact on the vulnerability discovery rate. It influences the rate of discovery during the entire life time of the software. Clark, Frei, et al. (2010); Clark, Blaze, and Smith (2010) argue that due to software reuse in newer version, discoverers require less time to get familiar with the software since some parts are already familiar. This property might also have a significant influence on the vulnerability discovery rate during the end of the life time of the software. According to J. Kim, Malaiya, and Ray (2007); Nguyen and Massacci (2012) new vulnerabilities are discovered in older software because they share code with newer versions that are more popular. This is supported by findings from Nappa et al. (2015), where a large part of the vulnerabilities in their dataset
affected multiple applications.

Two additional properties might be relevant for the vulnerability discovery rate. These are the so-called ‘blood in the water’ theory and the potential economies of scale. The first is suggested by Clark, Frei, et al. (2010) and Clark, Blaze, and Smith (2010). They argue that the discovery of a new vulnerability might be a signal of weakness for that application, and might therefore attract additional discoverers, which results in an increased rate of discovery. The second suggested property, economies of scale, could be the case when vulnerability discovery tools and methods used by discoverers, can be successfully used for additional discoveries (Clark, Frei, et al., 2010). This might increase the vulnerability discovery rate over time.

Based on the hypotheses and suggestions mentioned in related literature, seven properties are identified that might influence the vulnerability discovery rate. Table 3.1 provides a structured overview of these properties and the related work in which they are mentioned.

Table 3.1: Overview of properties derived from hypotheses and assumptions in related research

<table>
<thead>
<tr>
<th>Influencing properties</th>
<th>Mentioned in</th>
</tr>
</thead>
<tbody>
<tr>
<td>Market share</td>
<td>Alhazmi and Malaiya (2005a, 2005b, 2006a, 2006b)</td>
</tr>
<tr>
<td></td>
<td>Alhazmi et al. (2006)</td>
</tr>
<tr>
<td></td>
<td>Woo et al. (2011)</td>
</tr>
<tr>
<td></td>
<td>S. Kim et al. (2010)</td>
</tr>
<tr>
<td></td>
<td>Roumani et al. (2015)</td>
</tr>
<tr>
<td>Software age</td>
<td>Joh (2011)</td>
</tr>
<tr>
<td>Learning effect</td>
<td>Alhazmi and Malaiya (2005b)</td>
</tr>
<tr>
<td></td>
<td>Khoo et al. (2010)</td>
</tr>
<tr>
<td></td>
<td>Clark, Frei, et al. (2010)</td>
</tr>
<tr>
<td>New version</td>
<td>Alhazmi and Malaiya (2005b)</td>
</tr>
<tr>
<td></td>
<td>Alhazmi et al. (2006)</td>
</tr>
<tr>
<td></td>
<td>Woo et al. (2011)</td>
</tr>
<tr>
<td>Shared code</td>
<td>J. Kim et al. (2007)</td>
</tr>
<tr>
<td></td>
<td>Clark, Frei, et al. (2010)</td>
</tr>
<tr>
<td></td>
<td>Clark, Blaze, and Smith (2010)</td>
</tr>
<tr>
<td></td>
<td>Nguyen and Massacci (2012)</td>
</tr>
<tr>
<td></td>
<td>Nappa et al. (2015)</td>
</tr>
<tr>
<td>Recently discovered vulnerabilities</td>
<td>Clark, Frei, et al. (2010)</td>
</tr>
<tr>
<td></td>
<td>Clark, Blaze, and Smith (2010)</td>
</tr>
<tr>
<td>Economies of scale</td>
<td>Clark, Frei, et al. (2010)</td>
</tr>
</tbody>
</table>

3.1.2. Assumptions from related research

This section provides an overview of the assumptions underlying the most common VDMs. As was mentioned earlier, these are relevant since they signal a knowledge gap, and therefore require assumptions on how these properties influence the vulnerability discovery rate. The second possibility is that these assumptions are the result of limitations of the chosen methods. In both cases these assumptions are a sign that further research is required in order to determine whether these assumptions are valid and what their impact is on the vulnerability discovery rates.

One assumption that is present in most of the VDMs that are published, concerns the independence of discovered vulnerabilities (Ozment, 2007). This assumption is the result of an ongoing discussion on whether or not the discovery of a vulnerability is independent of the discovery of another vulnerability. This assumption is not only the result of a lack of knowledge, it is also the result of limitations of the
used methods. The model presented by Anderson (2002) assumes that testing is random. However, in practice it is far more likely that a discoverer focuses on a particular type of vulnerability and method. The knowledge gained by finding the first vulnerability can therefore be used for finding another. This makes it unlikely that the discoveries are independent. Rescoca’s model (Rescoca, 2005) encounters the same issue, and thus simplifies the discovery process by assuming that it is stochastic in nature.

The second underlying assumption of VDMs is that software code remains static. However, as is argued by Ozment (2007) this assumption is false since code rarely remains static for long periods of time, due to patches or added features. This adds the possibility that new vulnerabilities are introduced, which is difficult to include in current VMDs. On the other hand it is doubtful whether this will result in better models, since it depends on how big the changes are. In case of minor code changes, in for example an operating system, the impact will likely be negligible.

The third assumption which is common in software is more crucial. It is caused by the uncertainty surrounding the number of vulnerabilities contained within the software. The crucial question is whether the number of vulnerabilities can be considered finite or not. Strictly speaking it is finite since the code has a finite size. However, due to size of most applications, the number of potential vulnerabilities can be considered infinite when compared to the number of vulnerabilities that are discovered. The assumption has a major impact on whether looking for vulnerabilities is actually a good idea (Rescoca, 2005). If it is finite, the security of the software increases since the number of remaining vulnerabilities decreases. However, if the number of remaining vulnerabilities remains extremely large compared to the rate at which they are found, the increase in security is minimal. The models presented by Alhazmi et al. (2006) for example assume that the number is finite. This discussion relates to the question whether or not it becomes more difficult to find new vulnerabilities as the easiest vulnerabilities are likely to be found first, and as time goes by only the vulnerabilities that are more difficult to find remain. Neuhaus (2012) argues that if this were the case, one would expect to see some form of power law in the rate at which vulnerabilities are discovered. However, they did not find any evidence that this is the case.

The final assumption which is common in VDMs is that different versions of software are independent of each other, and can therefore be modelled separately (Alhazmi & Malaiya, 2005a; J. Kim et al., 2007). This is clearly a concern within this field of research, as was pointed out in section 3.1.1. The assumption that different versions of software are independent is a recognised shortcoming of current VDMs. It potentially has a significant impact on the vulnerability discovery rates. Not only does it have an impact on learning effects, it also results in vulnerabilities being discovered in older software just because they are found in new software and both versions share similar code.

Based on these assumptions, four properties are identified that require further research in order to assess their impact on the vulnerability discovery rates. Table 3.2 provides a structured overview of these properties and where they are mentioned.

### 3.1.3. Selection of the influencing properties

The previous two sections identified a total of eleven properties which require further attention. These are: market share, software age, learning effect, new versions, shared code, recently discovered vulnerabilities, economies of scale, independence of vulnerability discovery, static code, vulnerability depletion, and independence of different software versions. This section discusses the selection of properties that form the main focus of this thesis. The first step is assessing the overlap between the different properties and the second step is the actual selection.

Most of these properties have relations to each other, or in some cases they even have a sizeable overlap. Market share for example is influenced by the introduction of a newer version. Therefore, there is an overlap between these properties. A second set of properties that show similarities are
3.1. Influence on the vulnerability discovery rate

Table 3.2: Overview of properties derived from assumptions underlying VDMs

<table>
<thead>
<tr>
<th>Influencing properties</th>
<th>Mentioned in</th>
</tr>
</thead>
<tbody>
<tr>
<td>Independence of vulnerability discoveries</td>
<td>Anderson (2002)</td>
</tr>
<tr>
<td></td>
<td>Rescorla (2005)</td>
</tr>
<tr>
<td></td>
<td>Ozment (2007)</td>
</tr>
<tr>
<td>Static code</td>
<td>Ozment (2007)</td>
</tr>
<tr>
<td>Vulnerability depletion</td>
<td>Rescorla (2005)</td>
</tr>
<tr>
<td></td>
<td>Alhazmi et al. (2006)</td>
</tr>
<tr>
<td>Independence of different software versions</td>
<td>Alhazmi and Malaiya (2005a)</td>
</tr>
<tr>
<td></td>
<td>J. Kim et al. (2007)</td>
</tr>
</tbody>
</table>

The independence of different versions and shared code. This is because shared code causes the dependence between different versions of software that share similar code. A third similarity, or overlap, is found between independence of vulnerability discovery and learning effects. Discoverers learn from discovering vulnerabilities, and this gained knowledge has a direct impact on finding additional vulnerabilities. The same argument holds for the relation between economies of scale and learning effects. These properties are combined in order to decrease the number of properties to be tested in later stages.

Two other properties are left out since they are either negligible or too technical in nature for the scope of this research. The age of software is one of them. The vulnerability discovery rate change over time, which hardly seems to be explained by just the age of the software. Rather, market share and learning effects seem far more likely. The other property which is omitted is static or changing code. This property relates to the technical characteristics of software and is therefore beyond the primary scope of this research. The following list of properties of the vulnerability ecosystem is the main focus regarding the vulnerability discovery rate. Each property is followed by a definition which is used for the remainder of this thesis.

Table 3.3: Selected properties that are suggested to influence the vulnerability discovery rate

<table>
<thead>
<tr>
<th>Selected properties</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Market share</strong></td>
<td>The number of systems on which the software is installed compared to the total number of systems.</td>
</tr>
<tr>
<td><strong>Learning effect</strong></td>
<td>The extent to which discoverers can use knowledge gained from prior exploration effort to find new vulnerabilities.</td>
</tr>
<tr>
<td><strong>Shared code</strong></td>
<td>The fraction of code shared between multiple versions of the software or shared between different applications.</td>
</tr>
<tr>
<td><strong>Recently discovered vulnerabilities</strong></td>
<td>The extent to which the discovery of a vulnerability attracts additional discoverers to focus on that application.</td>
</tr>
<tr>
<td><strong>Vulnerability depletion</strong></td>
<td>The extent to which there is an finite pool of vulnerabilities for a specific application.</td>
</tr>
</tbody>
</table>
3.2. Influence on the resulting attacks

The second part of the main research question focuses on what vendors and defenders can do to
minimise the number of attacks as a result of software vulnerabilities. Understand which properties of
the vulnerability ecosystem influence the number of attacks and what options the vendors and defend-
ers have is required, in order to answer this question. This section discusses the literature related to
this question. Since this section discusses some key aspects of both actors and approach the options
from their perspective, there is some overlap with the previous chapter. However, in order to present
a thorough and consistent selection of the options that are examined, this is unavoidable. This section
concludes with a selection of the options and translation to properties of vendors and defenders, which
will be analysed using the agent-based model further on in this thesis.

3.2.1. Options for the vendors

The first and most obvious option for the vendor is to reduce the number of vulnerabilities in the software
during development. There are certain methods that are used to actively reduce the number of bugs in
software, and thus reduce potential vulnerabilities. Fuzzing is one of the methods that is widely adopted
and has proven to be effective to reduce the presence of the more obvious vulnerabilities (Takanen et
al., 2008; Libicki et al., 2015). Other methods focus on improving the software development process
itself, such as: CLASP, SDL, and Touchpoints (de Win, Scandariato, Buyens, Grégoire, & Joosen,
2009). This shows that vendors have multiple options to try and reduce the number of bugs, and with
that, reduce the number of vulnerabilities in the software before the software is released.

Another option is to rely on external hackers to discover vulnerabilities and to report them to the
appropriate vendor. According to Algarni and Malaiya (2014) a majority of the discoverers are external
to the vendor. This means that a significant fraction of the information on the whereabouts of vulnerabil-
ities is beyond the reach of the vendor. In order to get a grasp on this information, bounty programmes
have been introduced. Bounty programmes attempt to persuade discoverers to responsibly disclos-
ing the discovered vulnerabilities to the corresponding software vendor. Despite that the efficiency of
such programmes has been put to question, they do provide vendors with knowledge on the presence
of vulnerabilities (Maillart, Zhao, Grossklags, & Chuang, 2016). Hence, persuading external discover-
ers to responsibly disclose vulnerabilities is a viable option to reduce vulnerabilities and thus the risk
of potential attacks through those vulnerabilities. Arguably, this option is also available to the defend-
ers. However, given the focus of the system and the required additional complexity involved with this
disclosure policy for specific defenders, this option has been omitted for the defenders.

Next to reducing the number of vulnerabilities and increasing the chances of finding them, the vendor
can release patches in order to reduce the exposure of the defenders to potential attacks. Different
vendors apply different approaches to patching. Schryen (2009) argues that there is no difference
between different types of vendors, but rather that patching is driven by the specific patch policy of
the vendor. Another important aspect of releasing patches is the response of the vendor to a publicly
disclosed vulnerability. Arora, Krishnan, et al. (2010) show that vulnerability disclosure speeds up the
patch release by 35 days on average. They also show that public disclosure influences the chance
that a patch is released. Releasing a patch does not necessarily reduce the risk for the defenders. A
valuable analyses by Arora et al. (2006) shows that attack frequencies increase in the short term as a
result of releasing a patch. This makes it difficult to determine whether releasing a patch is the right
thing to do in some situations. Adding to the uncertainty of the influence of releasing or not releasing is
patch is that we cannot measure how many attacks would have taken place if no patch was released in
the first place. This underlines the complexity and uncertainty of the system. Additionally, Cavusoglu
and Zhang (2006) show the importance of the timing of releasing a patch. Although their model takes
3.2. Influence on the resulting attacks

In a simplistic approach, it does indicate that not only the decision to develop a patch, but also the timing of releasing a patch might influence the potential risk for the defenders. Therefore, a distinction should be made between on the one hand the decision and time it takes to develop a patch, and on the other hand the timing of releasing the patch.

Numerous vulnerabilities are reported every day. Patching all those vulnerabilities has proven to be a challenge for large organisations, which require testing and deployment for the countless systems within the organisation (Beres et al., 2008). Prioritising which vulnerabilities induce the highest risk is also challenging, as it is difficult to determine which vulnerabilities are being exploited. CVSS scores are appointed to vulnerabilities that are reported in the National Vulnerability Database to indicate their severity level. However, a comparison between CVSS scores and exploits that are being traded on the black market, showed that CVSS scores are not a reliable indication of the severity of a vulnerability (Allodi & Massacci, 2012). Patching based on CVSS scores alone, only reduce the risk of being attacked by 3% – 4% (Allodi & Massacci, 2014). This makes it difficult to minimise the risk of being attacked by assessing one of the most used metrics for the severity of a vulnerability. An alternative method is proposed that applies machine learning to classify the severity of vulnerability (Bozorgi, Saul, Savage, & Voelker, 2010). They argue that their method performs better than current alternatives. However, this method is not widely adopted, and therefore not a suitable alternative yet. This shows that it remains difficult to determine which vulnerabilities are exploited, which in turn makes it difficult to effectively focus patching efforts.

### 3.2.2. Options for the defenders

Releasing a patch is one thing, the deployment of the patch on the defender’s side is a completely different story. There are numerous factors that underly the patch deployment process, such as: determining which patches to deploy, testing patches, and deployment over all the systems within the organisation (Cavusoglu, Cavusoglu, & Zhang, 2008). In addition, the alignment between patch releases from the vendor’s side and patch deployment on defender’s side is another factor that influences the risk and costs of vulnerabilities (Cavusoglu & Zhang, 2006). Beres et al. (2008) show with a simple model how different patching policies can influence the potential risk of an attack. Given the focus of this thesis, the patching behaviour of the defenders is an important part of the behaviour of the defenders and especially the influence they have on the resulting attacks should be further analysed. Provided that the way patches are managed by the defenders is determined by the cyber security strategy, the patch deployment by the defender will be further referred to as the defenders’ security strategy. Please refer to section 2.1.4 for a more detailed discussion.

In addition to options directly related to the reducing the exposure caused by vulnerabilities, the defender also has general security controls at its disposal. A defender could for instance decide to accept the vulnerabilities in the software and the resulting attempts to compromise its systems and focus on efforts to detect and respond to those attacks. Although this option might be effective, the scope of this research does not cover detailed security controls. Incorporating this option would also require the distinction between different types of vulnerabilities and attacks. Given the scope and time constraints, detailed security controls are not covered in this thesis.

### 3.2.3. Influencing properties of the vendor and defenders

This previous parts of this section identified what options the vendors and organisations have and how each of these options influence the number of attacks. Since this thesis focusses specifically on the vendors and defenders and not on the other actors, this section looks at the behaviour of vendors and defenders and how their behaviour influences the number of attacks. Behaviour is shaped by the prop-
Properties of the specific actors. In essence, the question is which properties of the vendors and defenders have the most influence on the number of attacks. In total six options are identified that enable vendors and organisations to influence the attack frequency as a result of discovered vulnerabilities. The options are: secure software development in order to reduce the number of vulnerabilities, incentivise responsible disclosure for external discoverers, patch development, patch release, focusing patching efforts, and patch deployment by the defenders. Two of these options will not be further covered in the remainder of this thesis: incentivising responsible disclosure, and focussing patching efforts. Incentivising responsible disclosure has been omitted, since it has been thoroughly covered in other research. Focussing patching efforts is also omitted, since this requires some way of categorising patches based on their properties. This would require the scope of this research to extent to detailed technical properties of both the software and vulnerabilities in order to provide a reliable distinction between different kinds of vulnerabilities. In addition, there is limited research available on methods that would enable to focus on more severe vulnerabilities, other than the currently used CVSS score systems. This is a topic on itself and is beyond the scope of this thesis.

The four remaining options are: secure software development, the decision and time it takes to develop a patch, the timing of the patch release, and patch deployment by the defenders. These options translate to the four properties that are shown in table 3.4. Hence, these properties are included in the conceptualisation of the vendors and defenders in chapter 5. In addition to the properties that are suggested to influence the vulnerability discovery rate shown in table 3.3, the four properties of the vendors and defenders are analysed in the remainder of this thesis.

Table 3.4: Selected properties that influence the number of attacks

<table>
<thead>
<tr>
<th>Property</th>
<th>Actor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Secure development capabilities</td>
<td>Vendor</td>
</tr>
<tr>
<td>Patch development policy</td>
<td>Vendor</td>
</tr>
<tr>
<td>Patch release policy</td>
<td>Vendor</td>
</tr>
<tr>
<td>Security strategy</td>
<td>Defender</td>
</tr>
</tbody>
</table>

3.3. Preliminary conclusion

The goal of this section was to provide an overview of the properties of the vulnerability ecosystem that are suggested to influence the vulnerability discovery rate and the resulting attacks on the defenders. The identified properties are the answer to sub-question 2, which is formulated as: What properties of the vulnerability ecosystem are suggested to influence the vulnerability discovery rates and the resulting attacks according to the literature? These properties are the main focus for the remainder of this thesis and their effect is analysed through the agent-based model. The properties are derived from related research. Regarding the properties that are suggested to influence the vulnerability discovery rate, the following five are identified as most relevant for this research: market share, learning effects, shared code, recently discovered vulnerabilities and vulnerability depletion. Regarding the properties of the vendors and defenders that influence that number of attacks as a result of vulnerabilities, the following four properties are identified: secure development capabilities, patch development policy, patch release policy, and security strategy.
Modelling complex adaptive systems

Agent-based modelling has proven to be a valuable approach for modelling complex adaptive systems. The goal of this chapter is to discuss why the vulnerability ecosystem is a complex adaptive system and why agent-based modelling is an appropriate method for the purpose of this research. In addition, this chapter discusses the initial hypotheses which are tested by means of the model. The chapter concludes with an overview of the modelling process and a discussion on the challenges for each of the phases of this project.

4.1. The vulnerability ecosystem: A complex adaptive system

Justifying the use of agent-based modelling requires a clear understanding of what kind of system is being analysed. Agent-based modelling is primarily useful to model Complex Adaptive Systems (Nikolic & Kasmire, 2013). A system is a broad term which encompasses many different meanings for many different perspectives. Ryan (2008) defines systems as: being an idealisations of a part of the real world; having multiple components which are interdependent; being organised; having emergent properties; having a boundary; being enduring; effect and being affected by the environment; exhibiting feedback; and having non-trivial behaviour. The discussion of the literature related to the vulnerability ecosystem in chapter 2 showed the complexity and interactions between different actors, their relation to the software, the emergence of discovered vulnerabilities and the resulting attacks, the obscurity of the markets, and uncertainty of the impact of patching vulnerabilities. The vulnerability ecosystem meets each aspect of this definition.

A complex adaptive systems is characterised by the adaptive nature of the system. Adaptivity “is to have the property of adaptation, or improvement over time in relation to environment” (Nikolic & Kasmire, 2013, p 26). Adaptivity can take on many forms. In case of the vulnerability ecosystem, the system adapts to appearance of new software, the discovery of new vulnerabilities, and the release of patches. Each of these events is in one way or the other a consequence of the previous event. Each agents either causes an event, or responds to it, which in turn changes the state of the system. Therefore, adaptivity plays a central part within the vulnerability ecosystem.

The final characteristic of a complex adaptive system is its complexity. Complexity is difficult to define and it completely depends on the perspective of the person who is observing the system. One might think that complexity depends on the extent of the observer’s knowledge of the system. However, the opposite might be the case. A better understanding of the system might reveal more about the
underlying interactions and dynamics, which might actually increase the complexity of the system from the observer’s perspective. Some parts of the vulnerability ecosystem are well-understood, whereas other parts of the system remain obscure and unpredictable. If the system is only perceived as vulnerabilities that need to be patched, most people involved in cybersecurity will agree that this description cannot be used for solving the problem, or not even for having a useful discussion. In order to better understand the vulnerability ecosystem, the system needs to be analysed from a perspective that takes into account a level of complexity that enables the improvement of our understanding. Therefore, the complexity of the vulnerability ecosystem is inherent to the approach and goal of this research.

Given the nature of the vulnerability ecosystem and the problem that is analysed, agent-based modelling is the most suitable method to model the system. Agent-based modelling offers great advantages over other modelling techniques. The work of Radianti and Gonzalez (2007) shows how simulation can be valuable to gain insight in vulnerability markets. However, their simulation applies system dynamics and fails to capture the complexity caused by the behaviour of different agents. Agent-based modelling would have been a more suitable modelling method, due to its ability to capture emergent behaviour, enabling a natural description of the system, and its high flexibility (Bonabeau, 2002). In addition, agent-based modelling is recommended for situations where it is important that agents learn and adapt, where agents show strategic behaviour and anticipate, when agents form alliances or cooperate, and when the past offers no prediction for the future (Pidd et al., 2010). These characteristics make it a suitable method to model complex adaptive systems such as the vulnerability ecosystem.

4.2. Agent-based modelling and cybersecurity

There are some examples where agent-based modelling has been used for cybersecurity related research. An interesting application of agent-based modelling is shown by Gorodetski and Kotenko (2002). They use it to simulate multi-agent systems which in turn are used to simulate network traffic. The resulting data is then used to test intrusion detection systems. Kotenko (2005) provides an approach for modelling and simulation, in which two communities of agents perform DDoS attacks and defend against the other’s attack. The author suggests an ontology and a software prototype. The paper by Kotenko et al. (2010) provides a working model of this case. The simulation is used to test different cybersecurity strategies and assess the botnet life cycle. The model is highly detailed and it models actual network characteristics on packet level. This shows that agent-based modelling can be used to model cybersecurity problems with varying levels of complexity.

4.2.1. Modelling vulnerability discovery and attacks

The main purpose of agent-based modelling for this project is to model and understand how the influencing properties from chapter 3 influence the vulnerability discovery rates, and which properties influence the resulting attack frequencies. This requires the model to represent the vulnerability ecosystem in such a way that it shows sufficient similarity with the real world system. This is needed in order to draw valid conclusions on the impact of the properties as identified in section 3.1.3. The main challenge is that there is no clear description of the vulnerability ecosystem. This challenge is addressed in the next chapter, where a conceptualisation of the vulnerability ecosystem is presented.

The complexity behind the vulnerability discovery rates is discussed in detail in chapter 2. The different actors have a major influence on the state of the system in the sense that each group of actors can influence the life cycle of the vulnerability. The model should represent the properties of the system, which include the high level properties of software, the actors and their behaviour, and mechanisms such as vulnerability markets and patch releases. In addition, the model should allow the initial hypotheses, presented in section 4.2.3, to be tested. These are based on the suggested impact of
the aforementioned properties on the vulnerability discovery rate and the extent to which properties of the vendors and defenders are expected to influence the attacks enabled by discovered vulnerabilities. The properties are included in the model in different forms. Some define differences in the relation between actors, or the characteristics of the software, whereas others change the behaviour of the actors. This can all be included in the same model, which allows for extensive simulation and analysis. An overview of the different inputs, the outputs, and the hypotheses is shown in figure 4.1.

![Diagram](https://example.com/diagram.png)

Figure 4.1: Overview of the different parts of the research in relation to the agent-based model.

### 4.2.2. Lack of insight

Many assumptions are made on what properties influence the vulnerability discovery rate of software. The previous chapter identified the following properties as the most relevant for this research: the market share, learning effects, shared code, recently discovered vulnerabilities, and vulnerability depletion. All these properties are mentioned numerous times. However, the true impact these properties on the vulnerability discovery rate is unclear. By modelling the vulnerability ecosystem it can be tested whether or not their influence is as significant as suggested by the literature. Testing different properties in the agent-based model, should result in different emerging patterns in the vulnerability discover rate over time. These hypothetical changes are illustrated in figure 4.2. The differences are not only interesting when they result in variations in the total number of discovered vulnerabilities, but also in in the rate of increase over time.

It is unclear what the best options are to reduce the risk as a result of discovered vulnerabilities. Research has been done on quantifying the windows of exposure. However, little is known about the direct impact on the defenders side. In addition, no research has tried to quantify what the relative influence is of different properties of the defenders and vendors on the attack frequency, and thus, the risk exposure of the defenders.

### 4.2.3. Initial hypotheses

Formulating expectations on the emergent behaviour can improve the value of the gained insight. According to Nikolic et al. (2013, p. 75): “well-formulated problems are more likely to yield useful insights.”
This includes the formulation of initial hypotheses on the expected emergent behaviour. The following hypotheses reflect the expectations of how the different properties from section 3.1.3 influence the vulnerability discovery rate and second, which properties of the vendors and defenders influence the risk exposure of the defenders.

**Vulnerability discovery rate**

- **Initial hypothesis 1:** It is expected that the market share has a major influence on the vulnerability discovery rate. This is supported by the many suggestions from related literature and the logic that a larger market share leads to more potential targets, which in turn yields higher gains for the attackers.

- **Initial hypothesis 2:** It is expected that learning effects have a small influence on the vulnerability discovery rates. However, they will become apparent in the sense that they result in a preference from the discoverers for certain software. This should become visible when increased learning effects result in a decreased tendency for discoverers to switch between software.

- **Initial hypothesis 3:** It is expected that shared code has a medium influence on the vulnerability discovery rate. This might be the result of lowering the barrier for discoverers to switch their focus between software that use a similar code base. This should result in increased switching between different applications or newer versions.

- **Initial hypothesis 4:** It is expected that recently discovered vulnerabilities have a minimal impact on the vulnerability discovery rates. This is partially supported by limited references in related work and the likelihood that other properties have a much stronger influence on the decisions made by the discoverers.

- **Initial hypothesis 5:** It is expected that vulnerability depletion has a large impact on the vulnerability discovery rate. The difference between a finite or infinite pool of vulnerabilities will basically come down to whether or not the discovery rate is influenced by a law of diminishing returns. This makes it harder to find subsequent vulnerabilities if more vulnerabilities are discovered.
Risk exposure

- **Initial hypothesis 1**: It is expected that the vendor’s capability to develop secure software has a large influence on the attack frequency, as this directly influences the number of vulnerabilities in the software.

- **Initial hypothesis 2**: It is expected that the patch development policy of the vendor will have a large impact on the number of attacks, as this directly influences how long an exploit can be used by the attackers.

- **Initial hypothesis 3**: It is expected that the vendor’s patch release policy only has a small influence on the number of attacks on the defenders that use the software, because the time difference between releasing patches for the different policies is relatively small.

- **Initial hypothesis 4**: It is expected that defenders with shorter patch deployment times are vulnerable for shorter amounts of times and are therefore less likely to be attacked as opposed to longer patch deployment times.

4.3. Simple versus complex: KISS or KIDS?

An important decision has to be made for every modelling exercise. It concerns the choice whether to build a simple model or a complex one. This is an ongoing debate in the field of agent-based modelling. There are two streams of views on the required complexity for agent-based modelling. The first one, KISS (“Keep It Simple, Stupid”) as proposed by Axelrod (2006), adheres to the belief that models should kept as simple as possible. Simple models are useful for communication, creating understanding and extension of the model (Hassan, Antunes, Pavon, & Gilbert, 2008). On the more complex side of the spectrum is the KIDS approach, or “Keep It Descriptive, Stupid.” This approach, as proposed by Edmonds and Moss (2004), is a response to the simplistic KISS approach. While recognising the value of simple models, the authors emphasise the lack of validity and usefulness as major drawbacks. Instead, they argue that modellers should build detailed models to start with, and only afterwards can modellers simplify the model, while preserving the original behaviour. A third approach is proposed to take the middle ground between the two ends of the spectrum (Hassan et al., 2008). They argue that data should become a more integrated part of the modelling process, rather than just a tool for validation.

The conceptualisation and formalisation of the vulnerability ecosystem requires some important decision regarding the complexity of the model. A more descriptive model would make it a more realistic representation of reality. However, this statement only holds under the assumption that the system is well understood and this is an unrealistic assumption. The current state of the research on the system surrounding vulnerability discovery and the underlying behaviour, are not sufficient enough to build descriptive models. The “in between-approach” of Hassan et al. (2008) would also be a troublesome match for this project. Data would become an integrated part of the modelling process, and this requires an abundance of data, which is a common problem within the field of cybersecurity research in general. The characteristics of this research make it a suitable candidate for the KISS approach. The lack of general understanding and missing theories on the dynamics of the vulnerability discovery rates and their impact on attack frequencies, require a more simple model. This minimises the number of assumptions, while at the same time improving the understanding of the system. In addition, a simple model is more easily extended and is thus more valuable for future research.
4.4. Modelling process and challenges

The modelling phase of this research follows the agent-based modelling process as proposed in Nikolic et al. (2013). This process consists of the ten steps as listed below. For each of these steps the goal and actions are discussed briefly. Some steps pose specific challenges for this project, which are discussed as well.

1. **Problem formulation and actor identification** The goal of this first step is to ask the following questions: What is the problem? Whose problem is being addressed? Which actors are involved? The answers to these questions are provided throughout chapters 1 – 3 of this report.

2. **System identification and decomposition** The second step focuses on gathering knowledge about the system. The output of this phase consists of three deliverables. The first is a list of agents with their properties, actions, and interactions. The second deliverable is the specification of these actions and how they affect the state of the agent. The final deliverable is a description of the environment. This step is particularly challenging, due to decisions that need to be made on what is included in the conceptual model and what not. This is covered by chapter 5 of this report.

3. **Concept formalisation** The aim of this step is to formalise the conceptualisation of the previous phase. The main purpose of this formalisation is to make the concepts explicit for the actual modelling in later phases. The concept formalisation contains many details on how certain aspects of the system should be modelled. The concept formalisation is not presented in this thesis. The initial formalisation required many additions, due to extensions in the model and is therefore obsolete.

4. **Model formalisation** The goal of the model formalisation is to translate the previous conceptualisation into a model narrative and pseudo code. The model narrative is a description of the events that take place within the model. In this report it is presented as a flow chart (figure 6.1) in chapter 6. The pseudo code is not included in this thesis.

5. **Software implementation** This step contains the actual modelling phase of this research. It is the translation of the model formalisation into actual code. In this case it is written in NetLogo (Wilensky, 1999). Given the size of the code, it is not included in this thesis. However, it can made available to anyone interested in continuing this work. Please contact the author or the first supervisor.

6. **Model verification** The focuses of this phase is to check whether the software implementation is correctly translated from the previous steps. This is done through a set of tests as presented in section 6.7 in chapter 6.

7. **Experimentation** The experimentation phase covers the simulation of the system with the purpose of generating the required data. This is preceded by the experimental design, which determines the input of the model and runtime settings. The experimentation focuses on the influencing properties from chapter 3. The experimental design is presented in chapter 7.

8. **Data analysis** Analysing the output is done in the data analysis phase. This analysis assesses for each of the influencing properties, how significant their impact on the vulnerability discovery rate is, under which conditions. In addition, this analysis assesses the impact of different properties on the attack frequencies or the defenders. The data analysis is performed in R and the most relevant results are shown in chapter 8.
9. **Model validation** This is a particular crucial phase and challenging step in this research. Its goal is to check whether the model is a valid representation of the real system. The validation consists of three validation methods: historic replay, literature validation, and expert validation. The extent to which the behaviour of the model is similar has an impact on the validity of the findings. The validation is discussed in chapter 7.

10. **Model use** This phase concerns the use of the model for answering the main question. This is covered in the final chapters of this thesis. The model can however, also be used for additional research. This is discussed in chapter 9.

### 4.5. Preliminary conclusion

This chapter presented a brief discussion on cybersecurity related research and agent-based modelling. It discussed the lack of insight that is addressed by the model and formulated the initial hypotheses that are tested. Based on the characteristics of this project, the KISS (Keep IT Simple Stupid) is preferred over a more descriptive approach, due to the lack of understanding of the system and limited availability of data. This lack of understanding on the dynamics behind the vulnerability ecosystem results in a strong dependancy on the conceptualisation phase of this project. Close attention should be payed on the conceptual decision made and how this influences the results of the agent-based model. Another challenge is the validation of the model since it requires comparison with empirical data, which is rather scarce. The following chapters focus on the modelling phase of this research.
Conceptualising the vulnerability ecosystem

Modelling the vulnerability ecosystem requires a clear and concise overview of the system’s characteristics. The goal of this chapter is to conceptualise the vulnerability ecosystem according to the guidelines provided by Nikolic et al. (2013). This conceptualisation is based on the extensive literature review of the complexity behind the vulnerability ecosystem as provided in chapter 2. It distils the core concepts, agents, behaviours, and interactions that are most relevant for understanding first, how the properties that are identified in chapter 3 influence the vulnerability discovery rate and second, how this in turn influences the attack frequency on large organisations. This improves systems understanding on how the vulnerability discovery rate affects the risk exposure of these organisations in cyberspace. An overview of the elements in the conceptual model is provided in table 5.1. Each of the elements is discussed in the following sections. The agents and their most important properties, actions, and interactions, are presented in section 5.1. The objects, which refers to the items in the model that are not capable of independent decision making, are discussed in section 5.2. The environment is discussed in section 5.3. Section 5.4 provides an overview of the conceptual model and a brief discussion on how each of the influencing properties is incorporated in the model.

5.1. The agents

Based on the identified actors from chapter 2, four agents can be distinguished that play a major role in the vulnerability ecosystem. These are the developers of the software; the software vendors, the black hat or white hat hackers who find the vulnerabilities; the discovers, the black hat hackers who exploit the vulnerabilities; the attackers, and the organisations that are affected by the resulting cyberattacks; the defenders. For each of these agents their properties, actions, and interactions are discussed below. In addition, the simplifying assumptions for each of the agents are discussed.

5.1.1. Vendors

Software plays a central role in the vulnerability systems. Each of the four identified agents interacts with the software in some form. The provider of the software has a major influence on the properties of the software and is discussed in detail in section 2.1.3. However, this research primarily focusses on the external properties of the software itself. For this reason, the vendor only influences the number
5. Conceptualising the vulnerability ecosystem

Table 5.1: Overview of the elements of the vulnerability ecosystem

<table>
<thead>
<tr>
<th>Overview of the elements</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Agents</strong></td>
<td></td>
</tr>
<tr>
<td>Vendors</td>
<td>Release software and patches for discovered vulnerabilities.</td>
</tr>
<tr>
<td>Defenders</td>
<td>Use software, switch to newer version, and apply patches.</td>
</tr>
<tr>
<td>Discoverers</td>
<td>Discover vulnerabilities and disclose them to the vendor or develop an exploit and sell it on the black market.</td>
</tr>
<tr>
<td>Attackers</td>
<td>Acquire vulnerabilities and attack defenders.</td>
</tr>
<tr>
<td><strong>Objects</strong></td>
<td></td>
</tr>
<tr>
<td>Software</td>
<td>Provided by a vendor and used by defenders.</td>
</tr>
<tr>
<td>Vulnerabilities</td>
<td>Present in software and discovered by discoverers.</td>
</tr>
<tr>
<td>Exploits</td>
<td>Exploit a vulnerability and are developed and sold by a discoverer.</td>
</tr>
<tr>
<td><strong>Links</strong></td>
<td></td>
</tr>
<tr>
<td>Software and patch provision</td>
<td>Relation between vendor and software.</td>
</tr>
<tr>
<td>Software usage</td>
<td>The relation between software and defender.</td>
</tr>
<tr>
<td>Affected software</td>
<td>The software that contains the vulnerability.</td>
</tr>
<tr>
<td>Affected defender</td>
<td>The defender that is vulnerable due to the vulnerability.</td>
</tr>
<tr>
<td><strong>Environment</strong></td>
<td></td>
</tr>
<tr>
<td>Market share</td>
<td>The market share of the software.</td>
</tr>
<tr>
<td>Exploit value</td>
<td>The value of the exploit on the black market</td>
</tr>
</tbody>
</table>

of vulnerabilities in the software, based on its capability to securely develop software. In addition, the vendors behaviour influences the defenders. Due to the reliance of the defenders on the vendor to release patches, the behaviour of the vendor has a direct impact on the risk exposure of its customers. Which in this case are the defending organisations. An overview of the vendors’ main properties and actions is provided in table 5.2.

**Properties** A vendor has five basic properties. The first property is that the vendor is the provider of certain pieces of software. This determines who is responsible for releasing a patch when a vulnerability is found. The second property is the patch release policy it applies, which primarily determines whether is patch is released or not (Schryen, 2009). In addition, the vendor has a certain capability to develop secure software (Libicki et al., 2015; Takanen et al., 2008). This determines how many vulnerabilities are in the software and it determines the chance that when a patch is released, it also introduces additional vulnerabilities. The final two properties determine if a patch will be developed, how long it takes, and when it will be released. There are three release policies (see section 2.1.4 for a more detailed discussion). The first is continuous, which means that a patch is released as soon as it has been developed. The second is periodical, which means that patches are released each month on a fixed day. The final policy is a combination of the two; patches are release monthly unless multiple attacks are reported, in that case a patch will be released as soon as it is ready.

**Actions** A vendor can perform three actions. The first is releasing new versions of the software and selling it to organisations (the defenders). It is assumed the eventually most defenders will adopt the new software in favour of the older version. The timing of a new software release is determined by the vendor’s properties. Since this research focuses on operating systems, this interval is approximately two to three years. The second action is whether or not to develop a patch. This decision is based on the vendor’s development policy, the severity level, and the number of reported attacks. Releasing the patch is the third action, which is determined by the vendor’s release policy.
Table 5.2: Overview of the properties and actions of vendors

<table>
<thead>
<tr>
<th>Vendor Properties</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Released software</td>
<td>The software that is released and can be used by the defenders. This software and consequent versions, rely on this vendor for patches. A vendor only releases one type of software.</td>
</tr>
<tr>
<td>New version release interval</td>
<td>Determines how long it takes for the vendor to release a new version.</td>
</tr>
<tr>
<td>Secure development capabilities</td>
<td>The ability of the vendor to minimise bugs in software. This influences the number of vulnerabilities in the software as well as the chance that a patch introduces a new vulnerability.</td>
</tr>
<tr>
<td>Patch development policy</td>
<td>The chance that a vendor develops a patch when a vulnerability is discovered and how long it takes.</td>
</tr>
<tr>
<td>Patch release policy</td>
<td>This determines the timing of the patch release. This can be continuously, monthly, or a combination of both.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Actions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Release new version</td>
<td>A new version of the software is released every 2 – 3 years. This is a fixed interval and is not influenced by other properties. Older versions of the software will remain to be supported by the vendor.</td>
</tr>
<tr>
<td>Develop patch</td>
<td>If a vulnerability is discovered and the vendor is aware of its presence, it can decide on whether or not to develop a patch based on the severity of the vulnerability.</td>
</tr>
<tr>
<td>Release patch</td>
<td>If a patch is developed the vendor will release the patch either through the standard procedure, which means at a fixed time interval, or through a priority procedure, which will release the patch when it is ready. This decision is made based on the severity of the vulnerability.</td>
</tr>
</tbody>
</table>

**Interactions** The strong position of software vendors and the lock-in effects, create a situation in which the influence of other parties on the vendor is minimal. However, a vendor is not completely immune to the actions of other agents. Empirical analysis has shown that the public disclosure of a severe vulnerability on average decreases the market value of a vendor by 0.6% (Telang & Wattal, 2005). They also show that when a company fails to release a patch at the time of disclosure, their market value drops, on average, with 0.8%. This shows that there is a small impact on the vendors state. However, due to its minor impact, this interaction is not included in the conceptual model. As security is becoming more of an issue, it is also becoming part of vendor selection (Godse & Mulik, 2009). Vendors that perform bad on releasing patches, might therefore be prone to losing customers. However, since this research focuses on operating systems, the lock-in effect is so strong that it is highly unlikely that defenders will switch to another vendor. Hence, both interactions are not included in this model. The only interaction that remains is between the vendor and the defender and concerns the release of software and patches.

**Simplifying assumptions** The vendor has a significant influence on the likelihood of finding vulnerabilities, through secure development and fuzzing (Libicki et al., 2015). Both methods however, are expensive and time consuming. For this reason and because of market pressure, most vendors prefer the ‘sell first, fix later approach.’ Therefore, the conceptual model presumes that the vendors only influences the number of vulnerabilities in the software, which is a relative simple representation of reality.
Another simplification is that the vendors have no in-house discoverers and completely rely on external discoverers.

5.1.2. Defenders

The defenders in this system refer to the organisations that use the software, as was discussed in detail in section 2.1.4. The attack frequency, which is also analysed next to the vulnerability discovery rate, is the result of the number of attacks on the defenders as a result of vulnerabilities. The defender is the most dependent actor within the system. It is influenced by most of the other agents, in the sense that their actions can have a large influence on the state of the defenders. An overview of the defender’s main properties and actions is provided in table 5.3.

Table 5.3: Overview of the properties and actions of defenders

<table>
<thead>
<tr>
<th>Defender Properties</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value at risk</td>
<td>Determines how attractive the defender is to attackers. Higher values at risk attract more attackers, as there is more to gain.</td>
</tr>
<tr>
<td>Number of systems</td>
<td>This refers to the number of systems used by the organisation, that run the software.</td>
</tr>
<tr>
<td>Software in use</td>
<td>The software that it is currently using.</td>
</tr>
<tr>
<td>Security strategy</td>
<td>Influences the time it takes for a patch to be applied. This is based on the severity level of the vulnerability and whether there are reported attacks for that vulnerability.</td>
</tr>
<tr>
<td>Maturity level</td>
<td>Determines the likelihood of a defender to successfully fend of an attack.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Actions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apply patch</td>
<td>The defender can apply a patch when it is released. The time it takes to apply a patch depends on the patch policy and the number of systems used by the defender.</td>
</tr>
<tr>
<td>Switch software</td>
<td>It can switch to a new version of the software when it becomes available.</td>
</tr>
</tbody>
</table>

Properties Based on the literature review in section 2.1.4, the defenders have the following properties that shape their behaviour. As an organisation they have certain assets that are their core value. At the same time, these are the assets that are at risk. This also determines the extent to which attackers are attracted to that organisation. The number of systems within the organisation influences the time it takes to deploy a patch (Beres et al., 2008; Cavusoglu & Zhang, 2006). Another important property is the type of software they use. This determines which vulnerabilities make the defender vulnerable. In addition, this also determines on which vendor the defender relies for releasing patches. In the end, this determines the defenders exposure. The final properties are the security strategy and the maturity level. Since this research focusses on vulnerabilities rather than all possible attack vectors, the security strategy only determines if patches are deployed and how long it takes before every system is patched (see section 2.1.4 for a more detailed discussion). The decision to deploy a patch is based on the severity level and reported attacks (Beres et al., 2008).

Actions The defenders have two action that are influenced by their properties. The first is the way patches are handled when they are released. This is determined by their security strategy. In case
of a reactive strategy, it will take more time before the patch is applied. With a proactive strategy, the patch is applied more timely, due to anticipation of security operators of the defender. The patching process follows the process described by Beres et al. (2008). This means that a patch has to go through several phases before it can be implemented. This causes a delay between patch release and the actual implementation. The second action that the defender can perform is to switch to a newer version. This decision is purely based on time, as it is assumed that most defenders will eventually adopt new operating systems.

Interactions There are two types of interactions. The first is applying the patches provided by the vendor. This decreases the number of vulnerable systems, and therefore makes that vulnerability less attractive for the attackers. The second is to report a vulnerability to the vendor in case an attack is detected. Since forensics remains a difficult and complex task, only a small portion of the attacks result in actual detection of the exploited vulnerability.

Simplifying assumptions Several simplifying assumptions underly the conceptualisation of the defender. The first simplification is the assumption that a defender has one combined asset. This means that there is no distinction between different assets within an organisation, which might attract different types of hackers. A second simplification is the security strategy. This research revolves around the way patches are handled and the chance of preventing a successful attack or mitigate the damage. It does not involve detailed characteristics of security controls and the security strategy is therefore only represented in form of a control strength. Another simplification is that an organisation only uses one operating system from a single vendor.

5.1.3. Discoverers
This group of agents is the main driver for the rate at which vulnerabilities are discovered. Section 2.1.1 discussed the diversity among discoverers in the form of their motivation and skills. This diversity would greatly increase the complexity of the model. The decision is therefore made to only distinguish between malicious and altruistic motivations. Most of the properties that influence the vulnerability discovery rate have a direct influence on the behaviour of the discoverers. An overview of the discoverers’ main properties and actions is provided in table 5.4.

Properties Vulnerability discovery is a highly skilled activity, which requires a high degree of technical skills and knowledge of the software (Algarni & Malaiya, 2014; Libicki et al., 2015). The skill level determines the ease with which they can discover new vulnerabilities and the chance to find vulnerabilities. The decision to disclose the vulnerability to the vendor, or to sell it on the black market, is to a large extent guided by the motivation of the discoverer (Frei et al., 2010). Based on the discussion on market for vulnerabilities in section , it is assumed that the price difference between the black and white market is too big for any direct form of competition, and therefore the only remaining drivers for responsibly disclosing vulnerabilities are based on ethical reasons. The third property is the learning effect of the discoverer. By putting effort in a certain piece of software, the discoverer develops knowledge and preference for that software. This increases the odds of finding new vulnerabilities (Khoo et al., 2010). Additionally, this creates a tendency not to switch its focus to other software. The software preference is based on the accumulated knowledge per software and the market share or the expected black market price in case of a black hat hacker. The final property is the tendency to sell an exploit multiple times, which is undesirable for the buyer, but difficult to prevent (Miller, 2007). This influences how often the exploit is sold and thus increase the chance of the exploit being detected by the defender. If a defender detects the exploitation, the related vulnerability is reported to the vendor.
<table>
<thead>
<tr>
<th>Discoverer Properties</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Skill level</td>
<td>Influences the chances of discovering new vulnerabilities and to accumulate knowledge. Higher skilled discoverers are more likely to discover new vulnerabilities, and accumulate knowledge at a higher pace.</td>
</tr>
<tr>
<td>Motivation</td>
<td>The motivation of the discoverer determines whether the vulnerability is disclosed responsibly, or sold on the black market. Discoverers can either be white hat or black hat.</td>
</tr>
<tr>
<td>Knowledge level</td>
<td>A discoverer has a certain knowledge level for each software that is released. By focussing on a specific software type and version, they accumulate knowledge. The amount shared code between software determines if this knowledge can also be applied to other software.</td>
</tr>
<tr>
<td>Software preference</td>
<td>The preference of the discoverer is based on accumulated knowledge of the software and the softwares market share or expected black market value of an exploit, depending on the discoverers motivation.</td>
</tr>
<tr>
<td>Software focus</td>
<td>The software on which the discoverer is currently focussing its discovery efforts.</td>
</tr>
<tr>
<td>Tendency to re-sale exploits</td>
<td>Determines how often an exploit is sold on the black market by the discoverer.</td>
</tr>
<tr>
<td>Actions</td>
<td>Description</td>
</tr>
<tr>
<td>Switch software focus</td>
<td>Switch to another piece of software if this is more appealing to the discoverer. This decision is made based on the preference as a result of the accumulated knowledge of the software and the software’s market share or expected black market value of an exploit, depending on the discoverers motivation.</td>
</tr>
<tr>
<td>Disclose or sell vulnerability</td>
<td>This decision is made based on the motivation of the discoverer. If the discoverer is a white hat hacker, the vulnerability is disclosed responsibly. If the discoverer is a black hat hacker, the vulnerability is offered on the black market.</td>
</tr>
</tbody>
</table>

**Actions**  
A discoverer has the following actions it can perform. The first action is selecting software on which to focus its discovery effort. This decision is driven by its previously accumulated knowledge of a particular system and its market share. When that system is no longer attractive, the discoverer will move on to the next one. The other action is deciding what to do with a newly discovered vulnerability. This decision is guided by its motivation (Frei et al., 2010).

**Interactions**  
Depending on the decision to disclose a vulnerability or sell it on the black market, the discoverer has different interactions. When the discoverer sells the vulnerability, it enables attackers to attack the defenders. When the discoverer chooses to disclose the vulnerability, it forces the vendor to decide on whether to develop a patch and when to release it. Both actions influence the defender, as in the first case the defender might be attacked, while in the other it has to decide on whether or not to apply the patch.
Simplifying assumptions  One important simplification is that the separation between the discoverers and vendors does not apply to open source development, since the discoverers are in most cases also developers. In that case the discoverers can also fix the vulnerability themselves. Another simplification is related to the distinction between internal and external discovers. Internal discoverers refer to those who work directly for the vendor, and those that are not directly related to the vendor. It is assumed that all discoverers are external to the vendors. The third, but important assumption, is that black hat discoverers also develop the exploit for a discovered vulnerability, which is turn offered on the black market. The final assumption is that all vulnerabilities, when disclosed, are disclosed responsibly. This is definitely not always the case, see section 2.1.1. However, in order to reduce the complexity and stay clear from the discussion on disclosure, other disclosure options are left out.

5.1.4. Attackers
The final group of agents is the attackers as was discussed in section 2.1.2. These are the malicious intended hackers that attack organisations for their own gain. Their behaviour is driven by their skill and perception of the potential target. In addition, they require access to exploits, which is determined by their means to buy them. An overview of the attackers’ main properties and actions is provided in table 5.5.

Table 5.5: Overview of the properties and actions of attackers

<table>
<thead>
<tr>
<th>Attacker Properties</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Skill level</td>
<td>Determines the kind of vulnerabilities they can use and their chances of a successful attack. Advanced attackers can use exploits for severe vulnerabilities and have a higher chance of success.</td>
</tr>
<tr>
<td>Means</td>
<td>The attacker’s financial resources that it can use to acquire exploits on the black market.</td>
</tr>
<tr>
<td>Attack strategy</td>
<td>Determines the attack strategy used by the attacker</td>
</tr>
<tr>
<td>Exploit preference</td>
<td>The attacker either has a set of targets and needs an exploit for their software, or acquires an available exploit and selects targets based on that exploit.</td>
</tr>
<tr>
<td>Acquired exploits</td>
<td>The exploits that the attacker can use to attack defenders. The attacker can only attack targets if they are vulnerable to one of the attackers exploit.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Actions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acquire exploit</td>
<td>If the attackers have the financial means, they will acquire additional exploits on the black market.</td>
</tr>
<tr>
<td>Select target</td>
<td>The attacker can select a target based on the defender’s value at risk. Advanced attackers can make better decisions since they are better capable of determining their opponents control strength.</td>
</tr>
<tr>
<td>Attack</td>
<td>Attacking the selected target. In case of low skilled attackers, they can try multiple targets at a time, whereas advanced hackers only focus on one target with an increased chance of success.</td>
</tr>
</tbody>
</table>

Properties  The attackers have four properties. The first is their skill level. This determines their ability to use exploits and it influence their chance of a successful attack. The second property of the attackers
is their means, or financial resources to acquire vulnerabilities on the black market. The third property of the attackers is their attack strategy. In order to narrow down the broad range of potential attackers, two archetype strategies are assumed. The first is smash and grab, in which the attacker tries to acquire as many assets as possible without being concerned about being detected. The second strategy is slow and stealthy, where the attacker moves slowly and therefore improves the chances of remaining undetected. The strategy is determined by the skill level of the attacker. Highly skilled attackers perform slow and stealth attacks, whereas less skilled attackers need to resort to smash and grab attacks, since their skill does not allow them to perform the more advanced slow stealth attacks. The final property is their exploit preference. Based on two mechanisms for the black market for vulnerabilities described by Sutton and Nagle (2006), two types of demand for exploits exist. The first is based on a specific group of targets or target software. The second is based on the availability on the black market. In this research it is referred to as either target-driven or availability-driven.

**Actions** An attacker can perform three actions. The first is selecting a target based on its perception of the most attractive target. Second, the attackers can acquire a vulnerability, or more precisely, an exploit. The third action is to use this exploit against the selected target(s). The process of attacking a large system can be extremely complicated and depends on the system architecture of the target organisation. A detailed representation of this process would be cumbersome to build, adds unnecessary sensitivity to the model, and requires numerous assumptions on how attacks propagate. The model used for this research therefore includes a relative simple representation, by determining whether the attack was successful or not based on the attack strength of the attacker and the control strength of the defender. It is assumed that attackers wait a while before they attempt to attack the same target again.

**Interactions** The first interaction is between the attacker and the discoverer of a vulnerability. The attacker requires a working exploit in order to perform an attack and therefore has a demand for vulnerabilities and their related exploit. This is facilitated by the black market on which they are traded for a price that is determined by the characteristics of a vulnerability. Developing an exploit requires only a fraction of the effort invested in the discovery. For this reason it is assumed that the discoverer does not just sell the vulnerability. Instead the discoverers directly sells the relating exploit. The second interaction is between the attacker and the defender in case an attack takes place.

**Simplifying assumptions** The conceptualisation of the attacker is relatively simple compared to reality. This is evident from the simple representation of the skills and strategy, whereas in reality there are many more types of attackers. The attack itself is also simplified as it does not include different stages or attack vectors. Instead the attack either succeeds or fails based on a chance relative to the attack strength of the attacker and control strength of the defender. Another assumption, which is mentioned above, is that the discoverer directly sells the exploit instead of only the vulnerability to the attacker. The final assumption is that when an attacker gets discovered, this increases the chance of the exploited vulnerability being detected and reported to the vendor.

**5.2. Objects**
The second category of elements within the vulnerability ecosystem are the objects. These are the elements of the system that cannot make independent decision. There are three main objects that play a central role within the systems. These are the software, the vulnerabilities within the software, and the exploits that are used to exploit the vulnerabilities. Each is highly related and in case of the
vulnerabilities they are an integral part of the software. For this reason, it could be argued that software and vulnerabilities are not separate objects. However, given that the vulnerabilities are the main unit of analysis in this research, this conceptual separation provides a clear distinction for the conceptualisation and flexibility for the analysis in later stages.

5.2.1. Software
Software cannot perform any actions and is therefore a passive element in the system. However, it does have certain properties that influence the behaviour of the agents. The first property is that it is provided by a vendor. The second property is its version. Please note that the version in this context refers to a new operating system from the same vendor (e.g. Windows 7 versus Windows 8). The release and implementation of a patch does not change the version number. The next property is that the software has a number of remaining vulnerabilities contained within the code. Do mind that these are considered as separate entities, as is discussed above. Related to this property is the number of discovered vulnerabilities. The final property is the amount code it shares with other versions of the software. This increases the odds that a vulnerability found for one version, also applies to another version. An overview is provided by table 5.6.

Table 5.6: Overview of the properties of software

<table>
<thead>
<tr>
<th>Software Properties</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vendor</td>
<td>The vendor that released the software and is responsible for releasing patches.</td>
</tr>
<tr>
<td>Version</td>
<td>Distinguishes between different operating systems from the same vendor.</td>
</tr>
<tr>
<td>Remaining vulnerabilities</td>
<td>These are the vulnerabilities that are yet to be discovered. The smaller the number of remaining vulnerabilities, the smaller the chances of finding new vulnerabilities.</td>
</tr>
<tr>
<td>Discovered vulnerabilities</td>
<td>The vulnerabilities that are discovered. They can either be secret, disclosed, available on the black market, sold, or be publicly known.</td>
</tr>
<tr>
<td>Shared code</td>
<td>The amount of code shared with other software. More shared code increases the likelihood that vulnerabilities also apply to other versions from the same vendor.</td>
</tr>
</tbody>
</table>

Simplifying assumptions It is assumed the software only shares code with other versions from the same vendor. Operating systems are known to share significant amounts code with subsequent versions. Despite that shared code libraries might also result in shared code with operating systems from other vendors, this is negligible compared to the shared code between software from the same vendor.

5.2.2. Vulnerability
A vulnerability is part of the software. However, it is represented as separate entity. It is discovered by a discoverer and has the following properties. It affects a piece of software and depending on the amount of shared code, it can also affect other software. It has a severity level, which influences the chances of a successful attack for the attacker. It also improves the value of the related exploit on the black market. A vulnerability also has a status. When the vulnerability is discovered, its initial status
is secret. Based on the motivation of the discoverer it is either disclosed to the vendor, or sold on the black market. If the a patch is released by the vendor or the vulnerability is detected by the defender after an attack, it becomes public. An overview of the properties is shown in table 5.7.

<table>
<thead>
<tr>
<th>Vulnerability Properties</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Affected software</td>
<td>The software to which the vulnerability applies. A vulnerability can apply to multiple types and versions of software. This depends on the amount of shared code between software.</td>
</tr>
<tr>
<td>Severity level</td>
<td>The severity of vulnerability. This can be low, medium, or high. The severity level influences the chances of a successful attack.</td>
</tr>
<tr>
<td>Status</td>
<td>A vulnerability can either be secret, disclosed, available on the black market, sold, or public. This depends of the who discoverers the vulnerability and how it is used.</td>
</tr>
<tr>
<td>Available patch</td>
<td>Whether or not there is a patch available for this vulnerability.</td>
</tr>
</tbody>
</table>

5.2.3. Exploit

An exploit is required in order for an attacker to perform an attack. It has two main properties. The first is of course the vulnerability for which it is developed. The second property is the skill level that is required to use it. It is assumed that unexperienced users cannot use complex exploits. Additionally, it is assumed that the discoverer also develops the exploit if the discoverer offers it on the black market. The two properties of an exploit are shown in table 5.8.

<table>
<thead>
<tr>
<th>Exploit Properties</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exploited vulnerability</td>
<td>The vulnerability for which this exploit is developed.</td>
</tr>
<tr>
<td>Required skill level</td>
<td>The skill level of the attacker that is required to use this exploit</td>
</tr>
</tbody>
</table>

5.3. Environment

The environment contains the elements that cannot be influenced by the other elements in the system. In this case there two external components (Nikolic et al., 2013). These are the market share of the software and the black market price of the exploits. They are the result of the aggregate behaviour of the agents in the model, rather than direct interaction.

5.3.1. Market share

The market share of software is an important element of the system. Although it directly relates to software, it is not a property of the software. The market share of the software is determined by the systems operated by the defenders that use the software as a fraction of the total number of systems of all the vendors.
5.3.2. Exploit value
The price of an exploit on the black market is a more complicated matter. Little is known on how prices on the black markets are established, as was discussed in section 5.1.3. Miller (2007) illustrates the challenges of selling vulnerabilities and exploits on the market and the influence of knowledge asymmetry on determining a price. It is suggested that the price of an exploit is determined by the severity of the vulnerability to which it applies, the amount of access it gives, and the supply and demand (Libicki et al., 2015). However, it remains unclear what properties exactly influence the price of vulnerabilities on the black market and how the prices mechanism work. Due to the lack of knowledge on black market pricing mechanisms, a pricing mechanism is assumed based on the severity level of the vulnerability, the market share of the software, and the supply and demand. Given the uncertainty, this simple but robust pricing mechanism minimises the risk of erroneous assumptions.

5.4. Influencing properties and system overview
The previous sections discussed the different elements of the vulnerability ecosystem. Together they form a high level conceptual model of the vulnerability ecosystem. Figure 5.1 provides an overview of the conceptual model. However, the conceptualisation of the properties that are suggested to influence the vulnerability discovery rate, as discussed in chapter 3, required further clarification. This is covered separately, since these properties are the main focus of this research, and they influence different parts of the system. Some influence the behaviour of the agents directly, while others result in different relations between agents. This section discusses how these properties are incorporated in the conceptualisation and how they relate the other elements in the system.

Market share The market share is included as the part of the environment as was discussed in the previous section. The market share has a powerful influence on the system. It determines to some extend the attractiveness of software for discoverers and it partially influences of exploits on the black market.

Learning effect This property is part of the property of the discoverers. If the strength of the learning effects increases, the discoverers accumulate knowledge quicker. This in turn improves the likelihood of them finding zero-day vulnerabilities for software they have more knowledge of. Therefore, this property only has a direct impact on how the discoverers behave.

Shared code The amount of shared code between different software has an impact on the number of systems and organisations that are effected by a vulnerability. If a vulnerability is found for a particular software, the amount of shared code increases the likelihood of other software also being vulnerable to this vulnerability. In addition, more shared code enables discoverers to also build knowledge of software that shares code with the software they are currently focussing on. It can therefore also influence the behaviour of the discoverers directly.

Recently discovered vulnerabilities This property determines how strong the signal of weakness is to other discoverers when a vulnerability is found. The discovery of new vulnerabilities might indicate that a piece of software yields more zero-day vulnerabilities and might therefore be more attractive. In this model the effect is represented as software becoming more attractive for other discoverers when the rate of discovery increases.
**Vulnerability depletion**  The number of remaining vulnerabilities is determined by this property. It influences the number of remaining vulnerabilities of the software and it can either be unlimited or a limited number, which decreases as more vulnerabilities are discovered.

As can be seen from the relations of the different influencing properties to the other elements of the system, their role in the conceptual model differs. In some cases they are the property of an agent, whereas some influence the behaviour of agents indirectly.

### 5.5. Preliminary conclusion

This chapter discussed the conceptualisation of the vulnerability ecosystem. It showed the core concepts, agents, behaviours, and interactions that are most relevant for understanding first, how the properties influence the vulnerability discovery rate and second, how this in turn influences the attack frequency on large organisations. This provides the answer to sub-question 3; which agents, technical characteristics, and economic mechanism should be included in the conceptual model? The conceptualisation is presented in the different tables throughout this chapter. The conceptual model, combined with the concept formalisation form the basis for the software implementation, which is discussed in the next chapter.
Model implementation and description

This chapter aims to providing an overview of the software implementation of the model and a high level description of how the model works; the model narrative. In addition, this chapter covers the model verification and discusses several improvements. The software implementation in NetLogo is discussed in section 6.1. The model narrative is presented in section 6.2 and the core mechanisms of the model are discussed in section 6.3. There are some points where the formalisation deviates from the conceptualisation. This is discussed in section 6.4. Section 6.5 describes the user interface of the model in NetLogo, and section 6.7 concludes with the model verification.

6.1. Why NetLogo?
NetLogo is a broadly used platform for agent-based modelling. It enables easy and fast implementation of models, has a low barrier to entry, and has a large online community (Nikolic et al., 2013). In addition, the syntax is relatively simple and easy to interpret for non-experienced users. Another suitable option is Repast. It offers more flexibility compared to NetLogo, and might thus be more suitable for larger and more complex models. However, the structural uncertainty as a result of the limited research on the inner workings of the vulnerability ecosystem and lack of comparable agent-based models within this field, makes prototyping an important aspect of this approach. As a first attempt to capture the vulnerability ecosystem in an agent-based model, Netlogo provides enough flexibility to represent the system, while minimising the required experience, as opposed to a more complex and flexible language such as Repast. This minimises the development time and should promote future use of the model by less-experienced users.

6.2. Model narrative
The model consists of 16 core procedures. There are additional procedures for plotting and output generation. An overview of the core procedures is depicted in figure 6.1. This flow chart shows the sequence of the procedures or what triggers them. This flow chart is a simplification of the actual model logic and there are some minor deviations from the displayed model logic. However, this flow chart gives an overall impression of how the model works.

The first procedure is the initialisation or setup. It runs the model for the first time to create all the agents and objects with their required properties. This setup determines which agents gets what properties, based on the model parameters. It determines, among other things, the fraction of the
defenders that have a reactive security strategy and the fraction of the discovers that are white hat as opposed to black hat. It also determines the fraction of the attackers that have a target-driven or attack-driven exploit demand and it determines the fraction of advanced and basic attackers. Additionally, the setup sets the parameter that determines how far the agents can look back in time to make decisions.

The second procedure is the release of software by the vendors. Initially, all the vendors release the first version of their software. Defenders select a vendor randomly. The properties of the software is based on the properties of the vendor. This includes the number of vulnerabilities that are present within the software at the time of release. A new version of the software is released at a fixed interval.

The next procedure in the model is updating the market share, discovery rate, and expected prices. The market share is used for decision making by various agents. The discovery rate can optionally be used by the discoverers to determine whether or not they want to switch to other software. The expected market price is always used by the discoverers to determine the attractiveness of soft-
ware. The expected market price is based on the recent prices of a vulnerability relative to the demand and supply ratio for a piece of software. In case no vulnerabilities are offered, the price is based on the demand. The **discoverers select software** based on this information and their accumulated knowledge. White hat hackers are not driven by financial incentives and base their decision on the market share of software in combination with accumulated knowledge.

Discoverers are continuously searching for vulnerabilities in the software they select. They **discover vulnerabilities** based on probability. A successful discovery is determined by their knowledge level, their skill level, and optionally, the number of remaining vulnerabilities. The number of remaining vulnerabilities depends on whether or not it is assumed that the number of vulnerabilities is finite. The discoverers **update their knowledge** each iteration of the model for the software they currently focus on. Knowledge is also accumulated for software that shares code, relative to the amount of shared code. If a vulnerability is found, the discoverer decides whether to **disclose or sell** the vulnerability and exploit.

If the vulnerability is disclosed, the responsible vendor has to **assess the vulnerability** and decide whether or not to develop a patch. If the vendor decides to develop a patch, it takes a while before the **patch release** takes place. After a patch is released, the defenders have to **assess the patch** and determine whether or not to **deploy the patch**. Both the assessment and the deployment can take a long time, depending on the properties of the defender.

The following procedure concerns the decision of the defenders to **switch to a newer version** of the software. This decision is purely based on probability. Some defenders switch quickly, while others might stick to the current version for many years. When a defender switches to a newer version it becomes vulnerable to all the unpatched vulnerabilities at that moment.

If the discoverer decides not to disclose but to **sell the exploit** for the vulnerability on the black market, it requires some time to develop this exploit. After it is finished it is offered at the market. Interested attackers that have the means to buy it, can **acquire the exploit**. Depending on the attacker, it **selects a target** based on the exploits it has, or it acquires exploits based on the targets the attacker wants to attack. Both ways, the attacker requires an exploit before it can attack. If the attacker **attacks** it runs the risk of the exploit being detected. This however depends on the capabilities of the defender. If the exploit is detected, it is reported to the vendor, who has to assess whether or not a patch will be developed.

### 6.3. Core mechanisms

All decisions made by the agent require a decision mechanism. However, not all of these mechanisms are equally important and some require additional clarification. Five mechanisms are at the core of the behaviour of the model. These are the software selection by the discoverers, the process of vulnerability discovery, the mechanism behind black market prices of the exploits, the patch release by the vendors, and the patch deployment by the defenders. Each of these mechanisms is discussed below.

#### 6.3.1. Software selection by the discoverers

The selection of the software depends on how attractive the software is to a discoverer. This is different for each discoverer. The criteria differs among the discoverers depending on whether their motivation is “white hat” or “black hat”. Both base their preference on the previously accumulated knowledge. The black hat discoverers also base their preference on the potential value of exploits for each software. This potential value is based on recent black market prices and the expected demand. White hat hackers are not driven by prices on the black market and base their decision on the market share of each software. The discoverer’s knowledge level $K_d$ is determined by the the total time spend on that
software according to equation 6.1, where \( t \) equals the total time spend on a piece of software and parameter \( k \) is the knowledge build up time. This means that the added knowledge decreases for each subsequent time step. This provides a reasonable mechanism which rewards focussing on the same systems, while preventing unrealistically high levels of knowledge.

\[
K_d = 1 - e^{-\frac{t}{x}}
\]  

(6.1)

The discoverer’s preference \( P_i \) for each piece of software is determined by equation 6.2, where \( M_s \) is the market share of the software, \( V_e \) is the expected value of exploits for the software, \( K_d \) is the knowledge level for that software, and \( w_n \) the different weights assigned to each of them. This equation enables the incorporation of different parameters with varying weights, which delivers the flexibility that is needed, due to the uncertainty of how this mechanisms works in reality.

\[
P_i = w_1 \cdot M_s + w_2 \cdot V_e + w_3 \cdot K_d
\]  

(6.2)

6.3.2. Vulnerability discovery

The second mechanism is the discovery of a vulnerability by a discoverer. This discovery is modelled as a stochastic process, where the probability of finding a vulnerability is determined by a general vulnerability discovery chance, the knowledge level of the discoverer, and the number of remaining vulnerabilities in the software. The discovery chance \( D_v \) of a vulnerability is determined by equation 6.3, where \( D_g \) is the general discovery chance, \( K_d \) is the knowledge level of the discoverer for that software, \( R_s \) the number of remaining vulnerabilities in the software, and \( T_s \) the total number of vulnerabilities that are present in the software from the moment it is released. The advantage of this mechanism is that it takes into account the technical characteristics of the software as well as the discoverer’s knowledge of the software.

\[
D_v = D_g \cdot K_d \cdot \frac{R_s}{T_s}
\]  

(6.3)

6.3.3. Black market prices

The next mechanism discussed here is the market mechanism that establishes the price of the exploits that are offered on the black market. Suggestions have been made on how some factors influence the price of an exploit. However, no concrete pricing mechanism has been formulated. The mechanism presented here is a first attempt to formulate such a mechanism. The market price \( P_e \) for an exploit depends on a general base price for exploits, the market share of the software, the severity level of the vulnerability to which the exploit applies, and the supply and demand on the black market for that software, as discussed in section 5.3. This is shown in equation 6.4, where \( B_g \) is the general base price, \( M_s \) the market share of the software to which the vulnerability applies, \( S_v \) is the severity level price factor, \( D_e \) is the demand of exploits for the software on the black market, and \( S_e \) the supply. In order to stabilise the price volatility, both the demand and the supply are based on the average values over the last 30 days.

\[
P_e = B_g \cdot M_s \cdot S_v \cdot \frac{D_e}{S_e}
\]  

(6.4)

6.3.4. Patch development and release

Once a patch is discovered and the vendor becomes aware of its presence, the vendor responds by either releasing a patch or not. The argumentation behind this mechanism is presented in section 5.1.1.
This mechanism consists of two decision. First, the vendor determines if a patch will be developed. This decision depends on the patch development policy and the severity level of the vulnerability. A high severity level results in a high probability that a patch will be developed. A low severity level results in a smaller probability. In case of reported attacks, the vendor can decide to release the patch with priority, which significantly speeds up the development time. The second decision is how to release the patch. If a vendor has a continuous release policy, patches are release as soon as they are developed. In case of periodical release policy, patches are released every 30 days. In case of a semi-periodical release policy, patches are released every 30 days unless attacks are reported, in which case their are released once they are developed.

6.3.5. Patch deployment
If a patch is released, the defenders to which the patch applies, have to decide whether to deploy the patch over all the systems of the defender, and how much priority it has. A more detailed discussion on the reasons behind this mechanism is presented in section 5.1.2. Before the organisation can start with deployment, it has to test the patch. Depending on the defender’s security strategy this can take longer or shorter. After testing, the defender deploys the patch. The time it takes to patch all the systems depends on the number of systems used by the defender. If attacks are reported, the defender can decide to deploy the patch with priority. However, a defender can only deploy a limited number of patches with priority. This capacity to deploy patches with priority depends on the security strategy of the defender.

6.4. Deviation from the conceptualisation
The model as described above is the formalisation of the conceptual model as presented in chapter 5. However, there are some areas where the formalised model deviates from the conceptualisation. These deviations can be found in the representation of the objects. These objects are the software, vulnerabilities, and exploits. However, in the software implementation they are represented as agents that do not make independent decisions. In this case they are objects that are represented as passive agents in NetLogo. They only have properties and cannot perform any actions or influence any other agents. There are several reasons for this decision. The first reason is that this offers flexibility in how the other agents interact with the software. For example, other agents can create links with the software, vulnerabilities, and exploits. This makes the model less complex in the sense that the links determine how each agent relates to the software and vulnerabilities. These links represent who uses which software, which defender is vulnerable to which vulnerability, and which attackers can use an exploit for a specific vulnerability. In addition, this enables the generation of the price of an exploit for a specific vulnerability, based on the properties of that vulnerability. Therefore, the process of patching is also less complicated. Instead on monitoring exactly which patches have been applied by which defender, this can be replaced by the presence of a link between the vulnerability and the defender. Patching is then represented as just breaking the link. The final advantage of this representation is that it makes verification easier, since the modeller only has to visually inspect what links are generated, at which moments, and with what agents.

6.5. Model dashboard
The dashboard of the model is shown in figure 6.2. It shows the various slides and other parameter settings on left and bottom of the dashboard. The centre screen shows the agents and objects of the model. The output of the model is presented in the graphs on the right side of the dashboard. Not all graphs, buttons, and sliders are visible in this screenshot.
Several agents can be seen in the centre screen. The blue dots represent the software that is provided by the vendors. The software is positioned as a matrix. The columns represent different software from different vendors, while the rows are different versions from each software. The blue dot in the top left corner is software version 1 from vendor Alpha, whereas the software in the bottom right corner is the software version 4 from vendor Delta. The green dots represent the defenders who use the software. Since the screenshot was taken at the end of a simulation, nearly all defenders are situated around the latest version of the different software. The grey and white dots are respectively the black hat and white hat hackers. The defenders and discoverers centre around the software they are currently using or focussing on. The red dots represent the attackers who remain stationary during the simulation. The triangles are discovered vulnerabilities coloured according to their severity level and their current status.

6.6. Model parameters
The model includes numerous parameters. Some parameters only have a minor influence, while others are more significant. An overview and discussion on how the main parameters influence the behaviour of the model is provided in chapter 7. A complete list of the model parameters can be found in appendix C. The parameters that are used for the experimentation in the following chapters are listed in table 6.1.

6.7. Verification
This section discusses the verification of the model. Its main purpose is to verify that the model is a correct translation of the conceptual model. During the modelling phase continuous checks of each specific part of the model have been performed. This has resulted in numerous fixes during the development. However, this is far from sufficient and in order to minimise errors an extensive verification after the development remains necessary. In order to perform an effective verification, a well-structured set of methods is required. Nikolic et al. (2013, p. 100) describe four main parts of the verification process:
Table 6.1: Overview of the key parameters

<table>
<thead>
<tr>
<th>Model parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of defenders</td>
<td>The number of defenders.</td>
</tr>
<tr>
<td>Number of discoverers</td>
<td>The number of discoverers.</td>
</tr>
<tr>
<td>Number of attackers</td>
<td>The number of attackers.</td>
</tr>
<tr>
<td>Amount of shared code</td>
<td>The amount of code shared between subsequent software versions.</td>
</tr>
<tr>
<td>Shared code learning effects</td>
<td>The amount of knowledge that is accumulated based on shared code.</td>
</tr>
<tr>
<td>Knowledge build up time</td>
<td>The time it takes to approach full knowledge of a software version.</td>
</tr>
<tr>
<td>Total present vulnerabilities</td>
<td>The number of vulnerabilities that are present within the software at the time of release.</td>
</tr>
<tr>
<td>General vulnerability discovery chance</td>
<td>The general chance that a vulnerability is discovered by a discoverer.</td>
</tr>
<tr>
<td>Exploit base price</td>
<td>The standard starting price of an exploit.</td>
</tr>
<tr>
<td>Attack likelihood</td>
<td>The chance that an attacker attacks.</td>
</tr>
<tr>
<td>Exploit detection chance</td>
<td>The chance that an exploit is discovered during an attack.</td>
</tr>
</tbody>
</table>

1. Recording and tracking agent behaviour, in which relevant metrics are identified and recorded.

2. Single-agent testing, in which the behaviour of a single agent is verified.

3. Interaction testing in a minimal model, in which the interaction between agents is tested.

4. Multi-agent testing, in which the emergent behaviour of multiple agents is examined.

Each of these phases is described in the following sections. Ensuring that the code contains no errors is always challenging and hard to check. The goal of these tests is to take out all major errors and ensure that only an acceptable number of errors remains in order to minimise the negative impact on the model's behaviour and output. The detailed results of these tests are shown in appendix D. No structured input and output check is performed for the final verification of the model. The reason for omitting this test is that the interaction between agents and objects primarily takes place through links. This provides the advantage of visually inspecting whether the interaction take place as is intended. Other actions take place in the form of list mutations. These have been thoroughly and continuously monitored during the development. For this reason, no structured test is required to check input and output within the model.

6.7.1. Recording and tracking agent behaviour

During this test the following metrics were monitored: the number of attacks that were performed on the defenders over time, the number of vulnerabilities in the four different states (secret, disclosed, available on market, and public), the prices of the vulnerabilities on the market, the average prices of the vulnerabilities per software, and the amount of acquired exploits by each of the attackers. Any inconsistency in the input and output indicates a potential error. This test however, yielded no significant errors. The small number of discovered mistakes might be due to some overlap with the metrics that were monitored during the development process when testing newly added code.
6.7.2. Single-agent testing
The aim of this set of tests is to verify the behaviour of a single agent. This is done by closely monitoring
the behaviour of a single agent. Nikolic et al. (2013) define two fundamental tests for single-agent test-
ing. The first is theoretical prediction and sanity checks, which involves the formulation of hypotheses
that describe predictions on how agents should theoretically behave under certain conditions. The sec-
ond test is called breaking the agent, which refers to the practice of finding the edge of the parameter
range in which the agent should behave normally. This is done by providing the agent with extreme
inputs and monitoring the resulting behaviour. For this verification, only the theoretical prediction and
sanity checks are performed. The reason for omitting the breaking the agent test is that this was con-
sistently done during the development, and therefore does not require additional testing in this phase.
For the theoretical prediction and sanity checks, numerous theoretical prediction are formulated and
checked. An overview of these hypotheses and their outcome are presented in appendix D.1. During
this set of test, three errors were found. All three have been fixed, revalidated, and confirmed.

6.7.3. Interaction testing in a minimal model
The third series of test aims at running the model with the minimum number of agents required to run
the model. For this model, the four types of software are required as the underlying model logic requires
four vendors, each with their own software in order for some iterations to work properly. For the other
agents; the attackers, defenders, and discoverers, only one of each is required to correctly run the
model. The following hypotheses are tested with four vendors, one discoverer, one defender, and one
attacker. The hypotheses and the results can be found in appendix D.2. Setting up the model for this
already revealed two significant errors. Two additional errors were found after testing the hypotheses.
All the errors are fixed, revalidated and confirmed.

6.7.4. Multi-agent Testing
Multi-agent testing aims at testing similar hypotheses as the previous tests. However, these hypotheses
should be simpler and more accurate. In addition to these tests, two other tests are performed. These
are the variability test, which takes a closer look at the variability of the output of the model over multiple
runs. The third multi-agent test is the timeline sanity test. The goal is to test whether the output can
be explained by the model logic. If not, this might indicate that there is something wrong in the model.
The results for all three tests are shown in appendix D.3. The theoretical prediction tests uncovered
no additional errors. However, the variability check and time line sanity revealed two additional errors
that slightly skewed the behaviour. Both were fixed and additional tests revealed no further problems.
Based on the results from the different verification tests, it can be concluded that the number of errors
is reduced to an acceptable level.

6.8. Preliminary conclusion
The model contains 16 key procedures that form the logic of the model. In addition to these
procedures, five core mechanisms have been discussed in more detail in this chapter. These
are the software selection by the discoverers, vulnerability discovery, the black market pricing
mechanism, the mechanism behind patch development and release, and the mechanism behind
the patch deployment. This chapter concluded with the verification of the model. Each of the
tests has revealed different errors. The verification of the output data of the experiment also
revealed an error in LHS of one of the experiments. Based on the verification it can be conclude
that the number of errors is reduced to an acceptable level.
Exploration, experimentation and validation

The goal of this chapter is to analyse the behaviour of the model and determine the experimental setup. This chapter covers the exploration of the model behaviour in general, the experimental setup for the main experiments, and the validation of the model. The distinction between exploration and experiment is discussed in section 7.1. The first step is to explore the behaviour of the model. This is covered in section 7.2. Based on the findings of the exploration, this chapter continues with the setup for the main experiments in section 7.3. The validation of the model is covered by section 7.4.

7.1. Confirmation versus exploration

The serves two main purposes. The first is to analyse which properties of the vulnerability ecosystem influence the rate at which vulnerabilities are discovered. The second purpose is to better understand which properties of the vendors and defenders have most influence on the number of attacks as a result of discovered vulnerabilities. Understanding what is exactly tested is essential before continuing with the experiments. Nikolic et al. (2013) describe two types of hypotheses related to generative nature of agent-based models:

1. Under the specified conditions, a macroscopic regularity of interest emerges from the designed agent-based model.
2. A range of clearly identifiable emergent behaviours and regularities can be established from this agent based model of a system.

This research typically falls into the first category. Certain behaviour is observed in the real world, but it is unknown what causes this behaviour. In this case, the rate at which vulnerabilities are discovered is known and the goal is to find answers to what the underlying properties of the system are that lead to that observation. The second type of hypothesis focusses less on recreating real world phenomena, rather it focusses on exploring what possible worlds lead to what kind of behaviour. The first type aims more towards falsifying or confirming hypotheses, while the second aims more towards exploration. Given the uncertainty surrounding the mechanisms and properties of the vulnerability ecosystem, this project also borrows elements from the second type of hypothesis in addition to the first. This overlap between the types of hypotheses has some implications for how to approach the experimentation. Instead of
moving directly towards testing the initial hypotheses from chapter 4, exploration of the model behaviour is needed in order to make sure that the right things are tested in the correct way.

### 7.2. Exploring the model behaviour

The obscurity and uncertainty surrounding the inner workings of the vulnerability ecosystem make it difficult to define some of the system’s mechanisms with certainty. Therefore, it is undesirable to start experimenting without understanding how the model behaves under different conditions. The goal of this section is to run the model under various conditions in order to better understand how the model responds to the different parameters. This results in a better understanding of the sensitivity of the model.

The exploration is done in two steps. First, exploring the model behaviour when varying the parameters that are not the direct focus of this research; the so-called general parameters. These are in essence the parameters to which the model is likely to be sensitive, but are not part of the actual experiments. Second, exploring how the parameters that are part of the actual experimentation influence the behaviour of the model in addition to the experiments in the following stage. These are the so-called experiment specific parameters. The reason why the exploration is divided in two steps is because of the large difference in the expected influence of both set of parameters. The first set of parameters are likely to have a very large influence on the rate of discovery. While the second set of parameters, those that are part the main experiments, are expected to have a relative smaller influence. Exploring the model behaviour with all the parameters would obscure the influence of parameters that are likely to have a relative small influence, while that “small” influence might prove relevant for setting up the main experiments. In addition, there is a high uncertainty on what would be valid values for the first set of parameters.

#### 7.2.1. Exploration setup

Testing the model for a variety of parameter settings can soon become an uncomprehending activity, due to the large amount of possible combinations of parameter settings. Nikolic et al. (2013) provides an interesting example of how testing different parameter settings for 14 variables can soon become an experiment that would take 10 times longer than the estimated age of the Earth. For this reason a more subtle and efficient approach is required. One such approach is Latin Hypercube Sampling (LHS). LHS “is a statistical technique that guarantees uniform sampling with the desired granularity of the scenario space given a Y dimensional parameter space and with a limit of X experiments” (Nikolic et al., 2013, p. 109). This should result in an efficient set of parameter settings that together cover a uniform range of all possible combinations. The LHSs for this project were created in R (R Core Team, 2015), using the Latin Hypercube Samples package by Carnell (2016).

The model includes numerous parameters that each influence the model in their own way. However, not every parameter is equally important. A complete list of the model parameters can be found in appendix C. Testing varying settings for all these parameters would be problematic, even when using LHS. Hence, a selection of the model parameters has been made that is used for exploring the general behaviour of the model. These general parameters are assessed first. Five parameters are tested for exploring the model behaviour. Table 7.1 shows which parameters are explored and the range of values that is used for the LHS. The fifth parameter, the total present vulnerabilities, is also included in the general parameters even though it is also part of the experiment specific parameters. However, since the total present vulnerabilities can be a strong limiting condition for the amount of vulnerabilities that are discovered, its general impact in combination with the other four parameters should be explored. This first set of parameters are selected, since each of them has a direct influence on the rate at which
7.2. Exploring the model behaviour

vulnerabilities are discovered, or the decisions made by the agents regarding. Therefore, they are expected to have a much larger influence on the behaviour of the model, compared to other parameters. The values range is based on a heuristic selection of possible realistic values.

Table 7.1: General parameters for model exploration

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value range</th>
</tr>
</thead>
<tbody>
<tr>
<td>General vulnerability discovery chance</td>
<td>[0.001 - 0.05]</td>
</tr>
<tr>
<td>Rationality noise</td>
<td>[0.1 - 0.5]</td>
</tr>
<tr>
<td>Exploit base price</td>
<td>[5000 - 20000]</td>
</tr>
<tr>
<td>Exploit detection chance</td>
<td>[0.001 - 0.05]</td>
</tr>
<tr>
<td>Total present vulnerabilities</td>
<td>[100 - 600]</td>
</tr>
</tbody>
</table>

After exploring the parameters that are not part of the main experiment, the model behaviour for the parameters that are central for the research question of this thesis, have to be explored. This step is necessary in order to determine how to experiment in the following stage. There are four parameters that represent the properties that are suggested to influence the vulnerability discovery rate, as discussed in chapter 3. A similar setup as the previous step is applied to this one. Table 7.2 provides an overview of the parameters and the value range that is tested using LHS. One exception is the market share of the vendors. For the implementation, each vendor gets an assigned market share which is set by a vendor specific parameter. Because of the dependency between those parameters, it is not possible to include them as separate parameters in the LHS. Therefore, fixed parameter settings were chosen. Two of the vendors have a market share of 15% and the two remaining vendors have a market share of 35%. The results of both simulations are presented and discussed in the next section.

Table 7.2: Experiment specific parameters for model exploration

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Market share</td>
<td>[15% - 35%]</td>
</tr>
<tr>
<td>Total present vulnerabilities</td>
<td>[100 - 600]</td>
</tr>
<tr>
<td>Knowledge build up time</td>
<td>[10 - 300]</td>
</tr>
<tr>
<td>Shared code</td>
<td>[0.1 - 0.6]</td>
</tr>
</tbody>
</table>

7.2.2. Exploration results

The previous section discussed the setup of the exploration of the model’s more general behaviour. For the first part of the exploration, five parameters are selected and tested. The LHS for this part of the exploration consists of 25 uniformly distributed sets of parameter settings. The output data from the model is analysed using R. Given the nature of agent-based modelling, several repetitions for each set of parameters is required to give a reliable dataset for analysis. The behaviour space tool in Netlogo and the latin hypercube samples were used for running the different simulations. The resulting datasets range between 100 to 300 MB each. Processing and analysis this amount of data requires powerful data analysis tools. In addition to performance and flexibility, open source software is preferred in order to stimulate further use of this model and enable the reuse of the code for data analysis. Therefore, a most suitable tool for this research is R, provided by the R Core Team (2015). It offers a huge range of open source package for all kinds of analysis combined with a large community for support, and equally important; it is also very powerful and efficient. The packages used include ggplot2 (Wickham, 2009), reshape2 (Wickham, 2007), and (d)plyr (Wickham, 2011; Wickham & François, 2015). All the data processing and analysis are performed in R. The code is not included in this thesis due to its size. Interested readers can contact the author or the first supervisor.
The results of this part of the exploration are shown in figure 7.1. The box plot shows the number of discovered vulnerabilities over time for the four software generations. Each generation consists of similar versions of software from the four vendors. The first generation consists of the software that is released at $t = 0$. The newest generation software, generation 4, consists of the software that is released at $t = 2700$. Please note that the severe outliers are not shown in this graph in order to make the newer generations better visible. In some extreme combinations of the vulnerability discovery chance and the total present vulnerabilities, the number of discovered vulnerabilities for the 1st generation software, could range up to 1500. There are several observations that can be made based on the figure 7.1. The first observation is the large difference between the first generation software and the following generations. The growth in the number of discovered vulnerabilities is noticeably steeper for the first generation. Another clear difference are the outliers. There are some cases in which the number of discovered vulnerabilities are much higher than others. This can be explained by a combination of the parameter settings as well as the properties for that specific vendor. A third observation is that there are numerous cases where very few vulnerabilities are found. This is evident from the spread of the first and third quantile, whisker lines that go all the way to 0. This can be explained by very low values of vulnerability discovery chance, which indicates that the model is highly sensitive to this parameter.

A more detailed analysis of the the influence of specific parameters is shown in appendix F. Graph F.1 in the appendix shows for each of the parameters how they influence the behaviour of the model. None of the parameters seem to have an unexpected large influence on the model behaviour, except for the vulnerability discovery chance. This was to be expected. However, graph F.1 shows that there is a strong influence on the rate of discovery. The sensitivity of the model to this parameter is therefore considered for the main experiments. The analysis also shows that the impact of the total number of vulnerabilities in the software only has a limited impact on the number of discovered vulnerabilities. This is surprising, as it was expected that this would have a much larger impact, which was the reason to also include this parameter in this part of the exploration.

The results of the second part of the exploration are shown in graph 7.2. The graph shows how
the model behaves under various settings of the total present vulnerabilities, knowledge build up time, shared code, and market share of the vendors. At first sight, both graphs show similar behaviour. However, there are some very clear differences between them. One of the difference is the spread of the number of discovered vulnerabilities between the different runs. The spread is smaller compared to the high spread in graph 7.1. This shows that the model is less sensitive to the second set of parameters. This confirms the initial expectations. The second part of the exploration also shows that for these parameters, the model results in far less runs with a number of discovered vulnerabilities close to zero. As with the first part of the exploration, the model behaviour shows a difference in the vulnerability discovery rate between the first generation software and the subsequent generations. The primary explanation for this difference is the initialisation of the model. During the initialisation, all the defenders and discoverers can only pick software from the first generation, while with the subsequent generations, the defenders and discoverers are spread out over multiple generations.

![Model behaviour for varying experimental parameter settings](image)

Figure 7.2: Vulnerability discovery rates of all software over 125 runs for experimental parameters

7.2.3. Implications for the main experiments

Based on the exploration there are two key implications for the experimental setup of the main experiments. The first is that the vulnerability discovery chance has a very large impact on the model behaviour. The implication for the main experiments is that in order to reduce noise in the data, the parameters that are not directly part of the main experiments should be set at fixed values. Therefore, controlled experiments are needed to ensure that the outcome of the experiments primarily shows the impact of the model property that is examined, and not the noise caused by other elements of the model. For that reason, only the parameter that is being investigated is varied in each experiment.

The second observation is the noticeable difference between the number of discovered vulnerabilities for the first generation software and the subsequent generations. The fact that fewer software is available in the first period of the model generates skewed results. The same can be said for the final generation, since there is no newer version available and eventually most defenders and discoverers will switch to the last available generation. The first and fourth generation software are therefore less...
valid, since they respectively miss a previous or subsequent generation of software that also attract discoverers and defenders. This is a clear case in which the model requires a warm up and cool down period. Therefore, the main experiments should only focus on software from the second and third generation. In addition, due to the difference in the simulated lifespan of both generations, the second generation is preferred over the third generation, since it allows for an analysis of a longer simulated lifespan, given that the second generation software is introduced earlier.

7.3. Setting up the main experiments

The model exploration provided insight in how the model behaves in general. However, the main purpose of the model was to investigate how the five identified properties from section 3.1 influence the vulnerability discovery rate, and second, which properties of the defenders and vendors influence the risk of large organisations in cyberspace, which were identified in section 3.2. Both require different experiments. Experimenting with the influence of the five identified properties requires a specific setup for each of the properties. Experimenting with the properties of the defenders and vendors, requires a combined experiment, since the interactions of these properties is of particular interest. This experiment has a less predetermined focus, and is therefore of a more exploratory nature. The setup for each group of experiments is discussed in the sections below.

7.3.1. Testing the effects of the influencing properties

Chapter 3 identified five properties of the vulnerability ecosystem that are suggested to influence the vulnerability discovery rate of software. The goal of this section is to provide a detailed description of the setup of the experiments that is used to investigate each of the properties individually. As was mentioned earlier, each of the properties is implemented by one or multiple parameters in the model.

Table 7.3: Implementation of the influencing properties in the model

<table>
<thead>
<tr>
<th>Property</th>
<th>Implemented as</th>
</tr>
</thead>
<tbody>
<tr>
<td>Market share</td>
<td>Market share vendor alpha</td>
</tr>
<tr>
<td></td>
<td>Market share vendor beta</td>
</tr>
<tr>
<td></td>
<td>Market share vendor gamma</td>
</tr>
<tr>
<td></td>
<td>Market share vendor delta</td>
</tr>
<tr>
<td>Learning effect</td>
<td>Knowledge build up time</td>
</tr>
<tr>
<td>Shared code</td>
<td>Amount of shared code</td>
</tr>
<tr>
<td>Recently discovered vulnerabilities</td>
<td>Recent rate of discovery</td>
</tr>
<tr>
<td>Vulnerability depletion</td>
<td>Total present vulnerabilities</td>
</tr>
</tbody>
</table>

Before continuing, standard parameter values need to be set in order to make the experiments comparable. Additionally, setting fixed parameter values decreases the noise in the model and enables us to look at the specific impact of the parameter that is being investigated. This decreases the need for countless simulations that would be needed to minimise the noise caused by varied parameter settings. The following parameter settings are chosen for the main experiments. These are either based on existing data or literature, or in case no data is available estimated based on what is reasonable given the results from the model exploration. The chosen parameter values are elaborated on in appendix E. These parameter settings form the controlled setting in which the specific parameter values are changed for each experiment. The decision to do these experiments in a controlled setting might have consequences in the sense that the relation between the different properties are not fully assessed.
Another consequence is that these experiments do not fully benefit from the rich complexity that the agent-based model has to offer. However, there are two arguments why this setup is appropriate. The goal of the thesis is to analyse the influence of specific properties and this influence becomes most obvious in a controlled setting. The second argument relates to the first, but is of a more practical nature. It would be possible to do an extensive LHS and analyse each of the properties based on that single experiment. However, this experiment would require a huge number of replications in order to reduce the noise to a level that reveals the specific impact of each property. For these reasons fixed parameter settings are chosen for the following experiments as is shown in appendix E, table E.1.

**Experiment 1: Market share** The first experiment aims at investigating the exact impact of a software’s market share on the rate at which vulnerabilities are discovered for the software. As one can see in table E.1, the market share of the software is not the actual market share of the specific piece of software. Rather, it is the market share of the vendor. Strictly speaking, these are different metrics. However, when considering the model formalisation they are nearly the same. The market share of the vendor determines how many defenders use its software. Given that defenders do not switch between vendors, they only use one software version from a single vendor, and the defenders switch to newer software at constant rate; these values are nearly identical. For this experiment three different market share settings are used that together cover a broad range, as is shown in appendix E, table E.2.

**Experiment 2: Learning effect** The learning effect is implemented as the parameter that determines the speed with which knowledge is accumulated as a result of the time spent focussing on a piece of software. This relation is implemented by means of equation 6.1. This parameter is the knowledge build up time and for this experiment its value ranges between 20 to 320. With a knowledge build up time of 20 it takes around 60 days for the knowledge level to approach 1. While a knowledge build up time of 300 it takes around 1000 days for the knowledge level to approach 1. The different values for this experiment are shown in appendix E, table E.3.

**Experiment 3: Shared code** The amount of shared code is the fraction of code that is shared with the previous version of the software. A higher amount of shared code increases the chance that a vulnerability also applies to another version from the same vendor. For this experiment the amount of shared code varies from 0.1 to 0.6, meaning the in the first case the software only shares 10% of its code with the previous version, while in the second case it shares 60% of its code with the previous version. Although higher fractions of shared code are definitely possible, it is unlikely that each part of the shared code is used in the same way, or enables the same access. Higher values are therefore omitted. The various parameter settings are shown in appendix E, table E.4.

**Experiment 4: Recently discovered vulnerabilities** This experiment is less straightforward and reveals some problems with the conceptualisation. Recently discovered vulnerabilities refers to the phenomenon that occurs when more vulnerabilities are found, this attracts more discoverers. This property of the system is included in the decision mechanism that the discoverers use to select software on which to focus their discovery efforts. For this experiment the standard decision mechanism as discussed in section 6.3 needs to be altered. An additional procedure is incorporated in the model that determines the recent relative rate of discovery. However, this creates some problems in the sense that it significantly changes the general model behaviour. A more detailed discussion on this specific experiment and the problems is presented in the next chapter in section 8.1.4.
Experiment 5: Vulnerability depletion  The goal of the final experiment for testing the effects of the influencing properties on the vulnerability discover rate, is to investigate how the model behaves under the assumption of a finite pool of vulnerabilities. The parameter of this experiment is the total number of vulnerabilities present in the software at the moment of release. For this experiment the total present vulnerabilities varies from 100 to 600, as is shown in appendix E, table E.5. This range covers a reasonable number of possible present vulnerabilities. Higher values are unnecessary given the number of discovered vulnerabilities of existing operating systems.

7.3.2. Testing the risk impact for the defenders
The second set of experiments focusses on the risk exposure of large organisations in cyber space. The previous experiments and model exploration focus on the behaviour of the model as a result of the more general properties of the vulnerability ecosystem. The aim of this second set of experiments is to uncover what properties of vendors and defenders have the most influence on the risk exposure of large organisations in cyberspace. Risk exposure refers to the number of attacks on the defenders. It tells us something about how many attacks can the defender expect, given the properties of the defender and the properties of the vendor that provides the software that is used by the defender. More interestingly, which properties result in the most attacks on the defender? It should be noted that the number of attacks is analysed and not the number of successful attacks. The main reason for this decision is the fact that no detailed attack and defence mechanisms are included in the model. In addition, this research focusses on the attacks enable by vulnerabilities, and since the model only includes attacks that are enable by vulnerabilities, the number of successful attacks is less relevant given the scope of this research.

The parameters for this experiment are the properties of the vendors and defenders. The vendor’s properties influence the system in three ways. In the first place, the vendor’s ability develop secure software directly influences the chance that discoverers find vulnerabilities, since there are more of them. It also influences the chance that a patch introduces a new vulnerability. Second, the chance that a vendor releases a patch depends on how it assess the potential threat and severity of a given vulnerability. Some vulnerabilities are not patched at all, while others are only patched after attacks are reported. Third, the final property of the vendor that influences the system is the timing of the patch release. The different properties and their settings for this experiment are shown in table 7.4.

<table>
<thead>
<tr>
<th>Property</th>
<th>Possible settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Secure development capabilities</td>
<td>high</td>
</tr>
<tr>
<td>Patch development policy</td>
<td>high</td>
</tr>
<tr>
<td>Patch release policy</td>
<td>continuous</td>
</tr>
</tbody>
</table>

The secure development capabilities of the defender correspond to the chance that more vulnerabilities are introduced in the software when released. “High” means no additional vulnerabilities, “medium” means 10% additional vulnerabilities, and “low” introduces an 20% additional vulnerabilities. The patch development policy corresponds to the chance and speed of patch development. “High” means that on average that there is a 90% chance that a patch is released with a development time of 15 days. “Medium” means that on average that there is a 85% chance that a patch is released with a development time of 40 days. “Low” means that on average that there is a 80% chance that a patch is released with a development time of 65 days. The different patch release policies are described in section 5.1.1.

The defender has two properties that are of interest for this experiment. These are the defenders security strategy and the maturity level. Note that the latter only influences the chance that an attack
fails or succeeds, and in case of advanced attackers also the likelihood that they target the defender or not. However, as this only applies to a small fraction of the attackers and the focus on the number of attacks rather than the number of successful attacks, this property is not analysed in this experiment. The remaining property is the defender’s security strategy. In this model it determines how the defender handles patches once they are released by the vendor. A more detailed description of this mechanism can be found in section 6.3.5. The security strategy can either be proactive or reactive. The proactive strategy results in a patch deployment time between 15 – 135 days, depending on the severity, time it takes to test the patch, and the number of systems of the defender. The reactive strategy results in a patch deployment time between 20 – 180 days.

This experiment uses LHS for setting the different property settings for vendors and defenders as described above. The rest of the parameters remain fixed. However, there are some slight differences between these parameter values and the fixed parameter settings of the previous experiment. More attackers are present in this experiment, and the vulnerability discovery chance is reduced slightly in order to decrease the number of available exploits for the attackers to use. The was necessary in order to prevent an unrealistic abundance of exploits on the black market. The results of both the experiments are discussed in the next chapter.

7.4. Model validation
Validity is an essential part of every research. One of the main criticisms concerning the use of agent-based models, is how to deal with validation. Conventional validation techniques do not always apply to agent-based models (Louie & Carley, 2008). The first part of this section discusses what type of validation methods should be applied to this research. The second part of this section provides an overview to the applied validation methods and their results.

7.4.1. Type of research and suitable validation methods
Louie and Carley (2008) provide an extensive discussion on how to address the validation of an agent-based model depending on the purpose of the model. They distinguish the following three categories of questions that are generally addressed by simulation studies: positive, normative, and plausible. The positive questions concern the questions that observe the world, provide a description and then try to find an explanation for the observed behaviour. The normative questions aim at finding what is preferred and what should be done. The plausible questions are of a more exploratory nature in the sense that they try to find answers to what might happen under certain conditions. As was discussed in section 7.1, this research tries to address two questions. The first being what properties of the model might explain the observed vulnerability discovery rates, and the second being how different properties of vendors and defenders influence the risk exposure of large organisations in cyber space. The first question falls into the positive category, while the second question belongs to the normative category. This has consequences for especially the first question, as it requires a comparison with empirical data in order to achieve a credible level of veridicality (Louie & Carley, 2008). However, the authors do not describe the specifics of how such a method should be applied or what is considered sufficiently valid.

A more concrete overview of possible validation tests is provided by Nikolic et al. (2013). They describe four types of tests that can be used for validating agent-based models. These are: historic replay, face validation through expert consultation, literature validation, and model replication. A historic replay involves recreating a well known situation and test whether the model shows similar behaviour. For face validation, subject matter experts and problem owners discuss the behaviour of the model in order to determine whether it is realistic and meets the purpose for which the model is designed. Literature validation uses academic literature to compare the findings from the model with the findings
from related literature. The final validation method is model replication. For this method a second model is created using either a different system decomposition or another modelling technique and compare the behaviour of both models. This final method however, requires the development of a second model, which is too time consuming for this thesis. Based on the recommendations by Louie and Carley (2008) and the methods discussed by Nikolic et al. (2013), the following types of validation method are used for the validation of this model: historic replay focussed on the vulnerability discovery rate of existing operating systems; literature validation of the vulnerability discovery rates and the life cycle of vulnerabilities and the resulting attacks; and face validation of the conceptualisation, behaviour, and results.

7.4.2. Historic replay

A historic replay requires historic data to which the model can be compared. However, data for most outputs of the model are scarce. There is data available on when and which vulnerabilities are found; The National Vulnerability Database contains an extensive set of reported vulnerabilities. Therefore, this part of the validations focusses on one of the two main outputs of the model; the vulnerability discovery rate. However, extracting this data and processing it is a time consuming process. Due to the time constraints of this project, a more efficient source of data is the vulnerability statistics from CVE Details (2016). This website provides an overview of multiple statistics on the discovered vulnerabilities, including the number of reported vulnerabilities per year for some operating systems. The downside of this source is that it does not provide detailed statistics on separate operating systems other than Windows. For this reason, only Windows operating systems are included. The model exploration showed that, due to the required warm up and cool down period, only 6.5 years of the lifespan can be simulated reliably. Hence, only Windows versions with a lifespan longer than 6 years should be used for validation. Based on these constraints, the comparison is made with the number of reported vulnerabilities over time for Windows XP, Windows Vista, and Windows 7. The simulation data used for this validation is the same as the data used for market share experiment. The shown behaviour is the average number of discovered vulnerabilities of all the simulated runs. The results of this comparison are shown in graph 7.3. Before continuing with discussing the results from this comparison, it has to be noted that there is a difference between the number of reported vulnerabilities and the number of vulnerabilities that are actually discovered, as not all vulnerabilities are reported. However, for this comparison it is assumed that there is no significant difference.

Graph 7.3 shows the number of discovered vulnerabilities each year. The lines are fitted using polynomial regression. Linear regression has also been tested. However, the linear model showed a higher adjusted squared error. An interesting observation can be made based by only looking at the three Windows versions. There seems to be a strong increase in the rate at which vulnerabilities are discovered for subsequent Windows versions. This might be the result of increased discovery efforts from both security researchers as well as black hat hackers, increases in the number of lines of code, or decreasing quality of development. No matter the reason, these differences obfuscate the comparison. For this reason and due to the sensitivity of the model to various parameters, the shape of the curve is analysed rather than the absolute values.

Comparing the behaviour of the model with historic data shows some interesting differences. First of all, the rate of discovery for the windows versions in the first two years is relatively slow compared to the model. However, this rate increase for all three of them afterwards. The second observation is that the time of transition between the learning phase and the linear phase moves closer towards the transition shown by the model with newer Windows versions. The third an final observation is that the saturation phase is less clear for the Windows versions compared to the model. This can point towards two things: either the saturation phase does not take place, or it takes much longer than 6
years to become obvious. Another problem is that the overall number of discovered vulnerabilities in general is increasing. That could explain why no saturation is visible for the Windows versions. Additional experiments are performed to determine whether different parameterisations of the model result in similar behaviour. Theses experiments are discussed in appendix A.

**Saturation of the discovery rate** Based on this comparison it is concluded that for the first years the model shows sufficient similar behaviour to the observed rates in the real world. For later years however, this similarity decreases as the rate of discovered vulnerabilities for the operating systems keeps increasing, while the model’s rate of discovered vulnerabilities seems to decrease. This difference does not only have implications for this model, as this undermines the general assumption that there even is a saturation phase. The data from all three Windows versions do not show a decrease in the first 6.5 years. Although not shown in this graph, the rate of discovery of Windows XP appears to be decreasing, but only after 12 years. This has implications for the validity of the model. Although saturation seems to be a logic consequence of decreasing interest in software over time, the empirical data does not confirm this. The additional experiments in appendix A show that a combination of an increasing interest in vulnerabilities in general, higher amounts of shared code, and slower adoption of newer versions provide a plausible explanation of the lack of saturation. This leads to the conclusion that the general structure of the model remains valid, rather it points to a less valid parameterisation. This is not surprising, given the difficulty of determining some of these parameters. In addition, the model assumes an equal time between the release of different versions. However, in reality the time between different releases can vary significantly. For example: Windows Vista was released 5 years after Windows XP, while Windows 7 was released 2,5 years after Windows Vista. This limitation should be kept in mind when translating the results from the model to a broader context. This means that any behaviour related to discovery saturation, should be reviewed critically.
7.4.3. Literature validation: Fitting the AML model

This second validation method aims at comparing the output of the model with related literature. There are two papers that provide valuable means for comparison, each of them with a different focus and unit of analysis. The first paper focusses on vulnerability discovery rates by Alhazmi et al. (2006) and the second paper focusses on the number of attacks as a result of patches becoming public by Arora et al. (2006), which is addressed in the next section.

The s-shaped curve of the rate of vulnerability discovery was first introduced and described by Alhazmi and Malaiya (2005b). The authors developed an time based model logistic that can describe this shape. They show that the model could fit the rate of discovery for several operating systems. The Alhazmi-Malaiya Logistic Model (AML) is described by equation 7.1. Where $y$ is the cumulative number of discovered vulnerabilities, $A$ is the proportionality constant, $B$ the total number of discovered vulnerabilities, and $C$ is a constant that is introduced by solving the differential equation from which this equation is derived. See Alhazmi and Malaiya (2005b) for further details.

$$y = \frac{B}{Be^{-ABt} + 1} \quad (7.1)$$

The authors apply this model to several operating systems, of which Windows XP is the most recent. In order to make a fair comparison only $B$ is changed to the number of discovered vulnerabilities in the agent-based model at $t = 6$, which is 269. The other parameters remain equal to the parameters of the model fitted to the empirical data of Windows XP, as they are based on empirical analysis (Alhazmi et al., 2006). The results are shown in graph 7.4. The graph shows that there is a noticeable difference in the rate of discovery. The deviation starts right from the start. The AML model increases slower compared to the ABM model. However, this difference remains stable for the following years. This difference might be explained by the other parameters in the model. However, it cannot be checked whether this is true, since parameters $A$ and $C$ cannot be based on the ABM model. What is interesting though is that apart from the deviation in the earlier years, both models show a nearly identical shape.

![Graph comparing ABM model and AML model](image-url)
Agent-based models versus logistic models The data shows strong similarities between both models. Given the amount of effort required for building an agent-based model one might argue that the logistic model is a better option. However, the logistic model cannot be used for the purpose of this research. It only shows the rate at which vulnerabilities are discovered based on the technical characteristics. Instead, this research tries to analyse properties of the system surrounding the software. This requires a modelling technique that can capture a much higher degree of detail. In addition, the AML model cannot be used to analyse the resulting attack frequencies. This validation test shows that the agent-based model performs similar to the AML model and can also be used to address a much broader range of questions. This leads to the conclusion that the agent-based model is a powerful and versatile alternative to the logistic model.

7.4.4. Literature validation: Comparing empirical attack frequencies

The second comparison with related literature concerns the number of attacks after a vulnerability becomes public. Arora et al. (2006) extensively analysed attack frequencies before and after the disclosure and patch release for a vulnerability. This section assesses whether the attack frequencies from the model increase in a similar way when a vulnerability becomes public. For this comparison, additional alteration have been made to the model to record the required data. Arora et al. (2006) analyse four related situations. From these four situations, the one as described by figure 7.5 is the most relevant for this research and for comparison with the model. The figure shows how the attack frequency evolves over time. Before the patch is release, the vulnerability is only known to a few, and as a result few attacks occur. However, if a patch gets released, the vulnerability becomes widely known. This causes a spike in the number of attacks, which slowly decreases after more defenders apply the patch. The model behaviour is shown in figure 7.6.

![Figure 7.5: Attack frequency after patch release (adapted from Arora et al. (2006))](image)

The four graphs show the analysis of four vulnerabilities from a total set of 100 vulnerabilities from 10 runs. From those 100 vulnerabilities the vulnerabilities that were not used by any attackers were filtered out. The four vulnerabilities in the graph are randomly selected from the remaining group. It shows for each of the vulnerabilities the number of times the vulnerability was exploited and when the patch was released, which is indicated by the blue dotted line. As can be seen from the four graphs, in three of the four cases, at first the attack frequency shortly increases after a patch is released and than slowly decreases. This is similar to the behaviour observed by Arora et al. (2006), as shown in figure 7.5. This behaviour is less clear in the top right graph. The graphs also show that after 150 – 200 days the attacks drop to a minimum as a result of the adoption of the patch. There are also differences. The attack increase in the attack frequency is relatively small compared to the analysis by Arora et al. (2006). This might be the result of fewer attackers in the model compared to the real world.
However, this should not be a problem, as this model is not designed for analysing absolute numbers. The behaviour of the model shows sufficient similarities with the work of Arora et al. (2006) to further support its validity.

**Similar behaviour despite a lack of data** This final comparison showed surprising similarities in the increase in the attack frequencies between the agent-based model and the empirical based attack frequencies provided by Arora et al. (2006). Although the agent-based model is primarily based on available literature and empirical data, some important parts of the model are still based on assumptions and heuristic estimates. Despite the limited validity of those parts of the model, the validation with Arora et al. (2006) shows that the general behaviour of the model remains valid. Not only does this support the validity of the model, it also shows the potential of agent-based modelling in the absence of empirical data, as long as there is some knowledge on the behaviour of the individual agents.

**7.4.5. Face validation**

The final step is to validate the model through face validation with experts. The goal of this method is to determine the validity of the model, by enabling experts in the field of cybersecurity to assess the validity of the behaviour of the agents, the general patterns of behaviour, and whether the model is applicable for the desired goal (Nikolic et al., 2013). Given the specificity of the model, and the rather unspecific definition of both “cybersecurity” and “expert,” it is important to determine what kind of experts are required. As the model’s primary focus is on the vulnerability discovery, the resulting attacks, and patch management, the ideal expert would have knowledge of both secure software development and cybersecurity policies of large organisations. Additionally, an expert on attacker behaviour would be desirable. However, those experts are hard to come by. For this validation two experts were found that meet the criteria of both knowledge on secure software development and cybersecurity policies of large organisations at CISO level, with additional experience in patch management.

Structured interviews were used to assess if the experts believed the model to be valid. In order to minimise potential bias, the experts were only told what the model should represent, by giving them a high description of the vulnerability ecosystem. Following the description of what the model should
represent, the experts were asked the following questions:

- What is your opinion on the extent to which the conceptualisation covers your image of the system surrounding vulnerability discovery, the resulting attacks, and patch management?

- Given the models mechanisms and procedures, do you believe that this covers the most important characteristics, and if not, what mechanisms are missing?

- Based on the behaviour of the model, does this resemble you perspective on the expected behaviour, and if not, what could cause this deviation?

- What would be your conclusions, based on the results of the experiments, on the vulnerability discovery rate and the risk exposure of large organisations?

In general, both experts believed that the conceptualisation covered most aspects of the vulnerability ecosystem. There were some points that might be valuable to include in a next version. The first being the lack of public disclosure. However, as one expert noted, this is similar to offering a vulnerability on the black market for free. He argued that publicly disclosing a patch is malicious, and therefore part of the motivation of the hacker. In that sense, it is included in the model in some form. Another observation was that the visualisation of the conceptual model of the vulnerability ecosystem, does not show what happens with vulnerabilities that become publicly known. Although this is not shown in the visualisation, this is present in the model. One expert noted that the bounty programmes only facilitates the transfer of knowledge on vulnerabilities between discoverers and the vendor. In reality, large organisations also offer bounties in the form of their responsible disclosure policy. They in turn inform the vendor of the vulnerability. The result is similar to what the conceptual model achieves through the direct interaction between the vendor and discoverer. A final observation is the limited range of the type of the attackers. Despite these comments both expert were confident in the validity of the model.

The experts were provided with a high level overview of the model logic and procedures, as presented in figure 6.1. Both experts believed that the mechanisms that guide the software selection by the discoverers is realistic, given the different motivations of the discoverers. One expert noted that the time it takes to develop an exploit for a released can be less than 24 hours. The model assumes that it takes several days. However, the expert argued that in most cases it takes less than 24 hours before the relating exploit to become public. One expert noted that a valuable addition to the model would be to take into account different types of vulnerabilities with different levels of access, based on the part of the software that the vulnerability affects. Both experts confirmed the validity of including two types of exploit preference mechanisms, as some attackers just target whatever they can target with their available exploits, while others select specific targets and acquire the necessary exploits. An important observation by one of the experts regards what happens with a vulnerability once its exploitation is detected by a defender after an attack. The model assumes that it becomes public, while in reality it can also be reported to vendor without it becoming public knowledge. This has a negative impact on the validity of the model. The final comment is that the model assumes that the software remains supported throughout its lifespan. This is not always the case. However, given that the model only simulates 6.5 years of the lifespan of the software, this should be no problem. Both experts believed that the model is valid enough for its current purpose.

The final two questions concern the behaviour of the model and the conclusions that were reached based on the results. Regarding the behaviour, both experts were shown the rate of discovery compared to other operating systems as shown in figure 7.3, and asked to identify which curve represented the models behaviour. Both could easily identify the model, due to decreased rate during the end of
the simulation. However, both experts agreed that a general increase in the effort of looking for vulnerabilities, would obfuscate such a slow down in the rate of discovery. Regarding the conclusions, both experts came to similar conclusions based on the results from the experiments. One expert noted that not necessarily the benefit of bounty programmes should be questioned, rather he argued, vendors should put more effort in prioritising which vulnerabilities to patch and which to keep secret. In general, both expert believed that the model, its behaviour, and the conclusion based on the results, are valid.

7.5. Preliminary conclusion
This chapter covered three important topics: the exploration, the experimental setup, and the validation. The exploration of the model revealed that the model is sensitive to some parameters, which might lead to obfuscation of the results. Therefore, these parameters should be fixed during the experimentation. In addition, the exploration showed that the second generation software is the most reliable and should be used for the main experiments. The validation confirmed that validity of the model in general. However, it revealed an interesting discrepancy regarding the assumed saturation of the discovery rate. The model showed saturation, while the empirical data does not. Additional experiments revealed that not necessarily the structure of the model, but the parameterisation explains the saturation observed in the model behaviour. Nonetheless, this means that any results based on behaviour related to discovery saturation, should be reviewed critically.
The main goal of this chapter is to discuss the results from the experiments. The previous chapter discussed the behaviour of the model in general and provided a detailed description of the setup of the different experiments. These experiments are divided in two groups with different purposes. The aim of the first set of experiments is to examine the influence of the five properties on the vulnerability discovery rate. The goal of the second set of experiments is to analyse how different properties of the defenders and vendors influence the risk exposure of large organisations in cyberspace. The results of the both sets of experiment are provided and discussed respectively in sections 8.1 and 8.2.

8.1. Influencing properties

The first part of the main research question concerns the influence of the market share, learning effect, shared code, recently discovered vulnerabilities, and vulnerability depletion, on the rate at which vulnerabilities are discovered. Section 7.3.1 described the five experiments that examine each of these properties. The results from the experiments are presented in the following sections.

8.1.1. The influence of market share

The initial hypothesis on the influence of the software’s market share on the vulnerability discovery rates was that it would have a large influence on the rate at which vulnerabilities are found. This was based on the idea that a larger market share offers more potential targets and thus increases the value of a vulnerability. The results from this experiment are shown in graph 8.1. The graph shows the vulnerability discovery over time for software with a market share of 25%, 50%, and 75%. The solid lines represent the average of all the runs for the three percentages of market share. The dark shade of blue represents the runs with a market share of 25%. The lighter shaded blue line represents the runs with a market share of 50%, and the lightest shade of blue represent the runs with a market share of 75%. The area surrounding each of the lines represents the total spread of all the runs for that specific value of market share. It should be noted that the graph starts at \( t = 900 \) which is the time at which this version of the software was introduced. Please refer to section 7.2.3 for further explanation on the chosen time interval.

As can be seen from the graph, the market share has a very large effect on the rate at which vulnerabilities are discovered. The graph shows 6.5 years of the lifespan of the software, after which the average difference is nearly 150 vulnerabilities. However, these values should be interpreted as relative
values, due to the sensitivity of the model to the chosen parameters. Based on these results it can be concluded that a market share of 75% is likely to yield 65% more vulnerabilities compared to a market share of 25%. The graph also shows that higher market shares lead to disproportional higher numbers of discovered vulnerabilities, as the difference between the number of discovered vulnerabilities is larger between market shares of 50% and 75%, compared to the difference in market shares between 50% and 25%. A final observation is the difference between when the first discoveries appear for different values of market share. There is a considerable difference between the first discoveries of vulnerabilities in the software with market shares of 75% and 50% compared to the software with a market share of 25%. This can be explained by the observation that when vendors release a new version of their existing software, it is more likely that they reach a large group of users in a shorter time. This is because, on average, the absolute number of users that upgrade is larger for vendors with a larger market share. This at the same time increases the attractiveness of the new version of software to the discoverers. Therefore, the higher the market share of a vendor, the more defenders in an absolute sense and the lower the barrier for the discoverers to also switch to the newer version.

8.1.2. The influence of learning effects

The second experiment aims at examining the influence of learning effects for the discovers. The initial hypothesis was that learning effects would only have a small influence on the vulnerability discovery rate. The results from this experiment are shown in graph 8.2. The graph shows the number of discovered vulnerabilities over time for different knowledge level build up times. This determines how fast the discoverers accumulate knowledge for a piece of software, and this in turn influences their chances of successfully discovering vulnerabilities. The parameter values are 20, 40, 80, 140, 220, and 320. Where the lower values are represented as darker shades of blue and the higher values as lighter shades of blue. As can be seen from the graph, the difference between the number of discovered
vulnerabilities is smaller compared to those of the market share experiment in graph 8.1. Given the large differences in the parameter values that are tested in this experiment, one might expect larger differences in the resulting number of discovered vulnerabilities. This can be explained by the law of diminishing returns according to which the knowledge accumulation is modelled.

![Graph 8.2: Vulnerability discovery rates for different knowledge build up times.](image)

What is clearly visible from graph 8.2, is that there seem to be three phases that are linear by nature. The first is the steep increase after the introduction of the software. Followed by a less steep linear phase, which coincides with the introduction the next software version. And finally, an even slower rising linear phase, which follows right after the introduction of the subsequent version. These differences in the rate of discovery can be explained by the transition of the discoverers and defenders to the next generation. On average, the defenders move to a newer version in about one year after its release. This is relatively short compared to reality, where large organisations might stick to an operating system for numerous years. These clear differences are therefore more an artefact of the model, than an observation that is relatable to the real world. However, these differences were less clear in the market share experiment, which focussed on the movement of defenders to newer versions. This can be explained by the higher sensitivity of the model to the differences in market share, compared to varying knowledge build up times.

More interesting are the differences around the time the first vulnerabilities are discovered. One would expect the first vulnerabilities to be discovered much earlier in runs with shorter knowledge build up times. However, as can be seen in graph 8.2, the first vulnerabilities are discovered around the same time. The actual difference is in the speed at which the rate moves from limited discoveries right after the release to a steep increase of the number of discovered vulnerabilities. This shows that the learning effect has the most influence on the transition between the learning phase to the linear phase.
8.1.3. The influence of shared code

The previous experiment showed three distinguishable linear phases, which are likely to be explained by the next experiment. This experiment focuses on the influence of shared code on the rate of discovery. The initial hypothesis was that shared code would lower the barrier for the discoverers to move to a newer version and that it would have a medium impact on the actual rate of discovery. The results from the experiments are shown in graph 8.3.

Figure 8.3: Vulnerability discovery rates for different fractions of shared code.

The graph shows the results for six different fractions of shared code, ranging from 0.1 to 0.6. Each run is grouped according to the different fractions of shared code. Large fractions are represented by lighter shades of blue and higher fractions can be recognised by the darker shades of blue. What becomes immediately clear is that there is no difference between varying amounts of shared code and the rate of discovery in the first two years of the lifespan of the software. The first deviations become apparent only after the introduction of the following version. This makes sense, as the defenders and discoverers start moving away from the software and fewer agents stay behind. Therefore, the continuing increase in the number of discovered vulnerabilities becomes the result of vulnerabilities being found in the newer versions which might also apply to the previous version. This relation becomes stronger as the amount of shared code increases, which is visible in the graph. The influence of the amount of shared code increases with the ageing of the software. Hence, it is concluded that the amount of shared code only influences the number of discovered vulnerabilities once a newer version is adopted by the fast majority of the defenders.

8.1.4. The influence of recently discovered vulnerabilities

As was mentioned in section 7.3.1, assessing the influence of recently discovered vulnerabilities on the vulnerability discovery rate proved to be challenging. In theory, as described by Clark, Blaze, and Smith (2010), the discovery of new vulnerabilities might be a signal of weakness to other discoverers.
and therefore attracts more of them. It turned out that this model is unable to test whether this influence can explain observed rates of discovery. There are several reasons why the model is not suitable for this experiment. The relation seems straightforward; new vulnerabilities are discovered, more discoverers start focussing on the software, which results in even more discoveries. The model showed chaotic and unrealistic behaviour when this property was included in the software selection decision mechanism. The problem is that discovers base their decision on where the most vulnerabilities are discovered and at the same time the expected price on the black market. The behaviour of the model is highly sensitive to the balance between the discovery rate and the potential market value of exploits, which in numerous cases would result in a situation were all the discoverers stay with the first generation of software, while all the defenders moved on. This makes the experiment unrealistic and inconclusive.

8.1.5. The influence of vulnerability depletion

The final experiment in this set aims at the vulnerability depletion. Vulnerability depletion is an interesting property, especially when considering how it is currently handled in this field of research. The models that assess vulnerability discovery rates or other related research, mostly assume that the number of vulnerabilities in the software can be considered finite compared to the rate at which they are discovered. The goal of this experiment was to test whether this assumption has a large influence or not. The initial hypothesis was that assuming finite or infinite vulnerabilities has a large influence on the rate of discovery. The results from this experiment are shown in graph 8.4. It shows the number of discovered vulnerabilities over time grouped according to the number vulnerabilities in the software at the time of release, varying between 100 – 600. The darker lines indicate lower numbers of present vulnerabilities, while lighter lines indicate higher numbers of present vulnerabilities. Depending on the properties of the software vendor, these values can differ, as does the chance that a patch introduces a new vulnerability. Therefore, the number of present vulnerabilities decreases when a vulnerability is discovered, and might increase due to the release of a new patch.

![Graph 8.4: Vulnerability discovery rates for different numbers of present vulnerabilities.](image-url)
The results show that the number of present vulnerabilities has a large influence on the rate of discovery. It was expected that as the number of remaining vulnerabilities decreases, as does the chance of finding a new one. The range between the number of discovered vulnerabilities, due to different number of present vulnerabilities, is considerable. The results also show that the effect becomes smaller as the number of present vulnerabilities increases. The difference in the number of discovered vulnerabilities after 6.5 years, between 100 and 200 present vulnerabilities, is larger than the difference between 200 and 600 present vulnerabilities. This shows that there is a certain threshold beyond which the total capacity of discoverers to find new vulnerabilities is too small to be seriously influenced by the number of present vulnerabilities. This difference starts to become visible about one year after the release of the software. The runs with 100 present vulnerabilities at the time of release, start to deviate from the others around this time. The results show that for this situation the threshold is somewhere in the lower range values tested in this experiment.

Based on these results it is concluded that assuming finite or infinite vulnerabilities has a significant influence on the resulting outcome. This not only confirms the initial hypothesis, it also confirms that rate of discovery, which relates to the total capacity of the discoverers, combined with the number of present vulnerabilities, has a strong influence on the rate at which vulnerabilities are discovered. This in turn has implications for research that is based on this assumption.

8.2. Risk exposure

The second question that is addressed by the agent-based model, is which properties of the vendors have the most influence on the risk exposure of large organisations in cyberspace, and what should be done to minimise this risk. Section 7.3.2 discussed the setup of this experiment in detail. The main goal is to uncover what strategies of the defenders and vendors should be preferred and what kind behaviour should be abstained from. Chapters 2 and 5 discussed the relation between the defenders and vendors in cyberspace. The vendor has three main properties that influence the risk exposure of the defenders. In the first place, vendor’s ability to develop secure software, which influences the number of vulnerabilities in the software. Second, the chance that a patch is developed once a vulnerability is discovered and how long the development takes. And third, the release policy of the vendor, which can either be continuous, semi-periodical, or periodical. Please refer to section 5.1.1 for further details.

The defender remains highly dependent on the vendor for releasing patches. Apart from deploying patches, the defender has limited options to minimise its risk exposure caused by vulnerabilities. As was discussed in section 2.1.4, patching is not as straightforward as one might think. It requires planning, testing, and needs to be deployed over all systems within the organisation. This property is included in the model as the security strategy of the defender. This can either be proactive or reactive. In the first case the defender has more capacity for testing and deployment, which results in shorter deployment times. In the second case, there is less capacity to anticipate patches, which can result in longer patch deployment times. The experiment was setup as described in the previous chapter, which includes a Latin Hypercube containing different settings for the properties of both the vendor and defender. The main output and unit of analysis in this experiment is the attack frequency as a result of different properties of both the defenders and vendors. The following four sections present and discuss the results from this experiment for each of the properties of the vendors and defenders.

8.2.1. Secure software development capabilities

The vendor’s ability to develop secure software, is the first property that is examined here. The initial hypothesis was that this would have a large influence on the attack frequency, as this directly influences the number of vulnerabilities in the software. The results from this analysis are presented in figure 8.5.
Each point represents the combined number of attacks on the defenders that use software from a vendor with that property for a single run.

The box plot in figure 8.5a shows that the medians of the total number of attacks on vendors with different properties are close to each other. This casts doubt on the initial hypothesis that the vendor’s ability to develop secure software would have a large impact on the number of attacks on the defenders that use its software. Analysis of Variance (ANOVA) is often used to analyse the difference in distributions. However, this test only works under the assumption that the population is normally distributed. The probability distributions in figure 8.5b indicate that the attack frequencies might not be distributed normally. The Shapiro-Wilks Normality Test confirms that the attack frequencies are not normally distributed with $p < 2.2e^{-16}$. Therefore, the Kruskal-Wallis Test is applied, since this test does not assume a normal distribution and can test more than two populations. The Kruskal-Wallis test shows (with $p = 6.2e^{-10}$) that there is a significant difference between the different secure development capabilities. However, as can be seen in figure 8.5a, the relative differences between the median of each group are very small and not as large as one might expect. Another observation from the data is that there appears to be a bifurcation for all three groups. Figure 8.5b shows that there seems to be a gap in the attack frequencies for all three groups around 8000 – 9000. This bifurcation can be explained by the patch release policy, as will be discussed in section 8.2.3.

In conclusion, the secure development capabilities of the vendor only have a minimal impact on the number of attacks on its users. This can be explained by the large influence of other properties that cause the higher attack frequencies. If all values above 8500 from figure 8.5 are taken out, the difference will increase. Another explanation is the access to exploits. A large number of the attacks are the result of attackers that use publicly available exploits. The number of exploits that are publicly available is for most part determined by the number of discovered vulnerabilities and more importantly, how many of them become publicly known. The results show that the vendor’s ability to develop secure software has a smaller influence than expected on the attacker’s access to exploits and thus, the resulting attacks.
8.2.2. Patch development policy

The property that is examined in this experiment, is the vendor’s patch development policy. This policy determines the chance that a patch will be developed and how long the development takes. The initial hypothesis was that the patch development policy would have a large impact on the number of attacks, as this directly influences how long a exploit can be used by the attackers. The results are shown in the box plots in figure 8.6

![Attack frequency versus patch development policy](image)

![Attack frequency probability per patch development policy](image)

Figure 8.6: Attack frequencies versus patch development policies.

The results from this experiment indicate that the patch development policy only has a small influence on the number of attacks on the users of the software, as is shown by the medians in figure 8.6a. This means that the chance that a patch will be developed by the vendor and the speed of the development, does not have a large influence on the risk of large organisations in cyberspace. However, the results do show an interesting difference between the “high” patch development policy and the “low” and “medium” patch development policies. The differences, as shown in figure 8.6b, leads to the conclusion that vendors that do respond quickly to an emerged vulnerability, can reduce the chance
of relative high attack frequencies.

8.2.3. Patch release policy

The final property of the vendor that is analysed in this experiment is the patch release policy. This determines when a patch is released once it is finished. The initial hypothesis was that this would only have a small influence on the number of attacks on the defenders that use the software. It was expected that continuous patch releases would slightly reduce the time that a vulnerability can be exploited and thus slightly decrease the number of attacks on the defenders, however not significantly. The results from this experiment are shown in figure 8.7.

![Image](attachment:image.png)

(a) Attack frequencies (box plot)

Figure 8.7: Attack frequencies versus patch release policies.

The experiment delivers surprising results. Continuously releasing patches has a severe negative impact on the number of attacks on the defenders, while the differences between periodical and semi-periodical release policy is negligible. The Kruskal-Wallis test shows that the difference in attack frequencies for the different vendor patch release policies, is significant with $p < 2.2e^{-16}$. Not only is the difference significant, the absolute difference is also large. These results are contrary to what one might expect, as the continuous release policy would enable defenders to deploy patches faster. However, there is an important underlying dynamic that coincides with the release of a patch. Releasing a patch signals the presence of a vulnerability to the entire internet. Finding vulnerabilities is difficult. Developing an exploit is relatively easy. Releasing a patch for a vulnerability that might only be known to the vendor or a small number of attackers, becomes public knowledge to all attackers, which opens the door for a huge increase in the number of attacks. This is exactly the behaviour that is observed in these results. The continuous release of patches does not only provide the defenders with the ability to patch a vulnerability, it also enables a serious increase in the number of attacks if the defenders fail to deploy the patch on a short notice. This is shown in figures 8.7a and 8.7b. The continuous stream of patches, in combination with the inability for defenders to adopt patches in timely manner, enables a surge in the number of attacks.

The bifurcation between the attack frequencies below 8500 and those above, can be explained by a combination of two properties of the defenders. Figure 8.5 shows that attack frequencies above 8500 occur for each level of secure development capabilities. The first differences in the higher attack
frequencies become visible in figure 8.6. This shows that these higher attack frequencies only occur for “low” and “medium” patch development policies, while a “high” patch development policy shows no attack frequencies above 8500. The most clear difference between the higher and lower attack frequencies, becomes visible in figure 8.7, since it shows that the higher attack frequencies above 8500 only occur with a continuous patch release policy. An overview of the most relevant properties of the vendor and the resulting attack frequencies for different patch release and development policies, is shown in 8.8. The higher attack frequencies only appear for the continuous release policy. This leads to the conclusion that the combination between a “low” or “medium” patch development policy and a continuous patch release policy, are a worst case combination, which results in an increased number of attacks on the defenders.

![Number of attacks versus different patch release policies](image)

**Figure 8.8:** Overview of the attack frequencies for different patch development and release policies

### 8.2.4. Cyber security strategy

The final property that is analysed the the cyber security strategy of the defenders. As was discussed in section 7.3.2, this experiment only looks at how defenders handle patches, as opposed to an entire set of security measures. The initial hypothesis was that defenders with shorter patch deployment times (proactive security strategy) are vulnerable for shorter amounts of times and are therefore less likely to be attacked, as opposed to longer patch deployment times (reactive security strategy). The results from this experiment are shown in figure 8.9.

The results show significant differences in the distribution of the attack frequencies between the different security strategies. This was to be expected, since defenders with proactive security strategies have an average patch deployment time of 40 days, whereas the average deployment time of defenders with a reactive security strategy is 100 days. This difference between the parameter values is large. However, this difference is realistic as deployment times for large organisations can differ significantly and range up to several months. The results show that the average difference in patch deployment of 60 days results in a 55% increase of the median attack frequency. This confirms the initial hypothesis...
and shows that the defenders themselves have significant influence on the chance of being attacked. An interesting observation is the multimodality of the distribution of the attack frequencies for proactive security strategies, as shown in figure 8.9b. This might be caused by the properties of the vendors that supply the software that is used by the defenders. Apparently, the defenders’ ability to apply patches on a short notice, reduces the risk for those defenders of being exposed to the large increase in the number of attacks that follow the release of a patch. Based on these results it can be argued that the defenders themselves have a significant influence on their risk exposure.
Conclusion and discussion

The results presented in the previous chapter are only valuable when taking into account a broader perspective. This comes down to answering the following question: Given the scope, limitations, and validity of the model; how do the results from the agent-based model relate to the real world challenges for cybersecurity? This chapter provides an answer to this question. The main findings are summarised in section 9.2. A reflection on the validity of the results and the value of agent-based modelling is presented in section 9.3. The implications of the finding are discussed in a broader context in section 9.4. Section 9.5 concludes with recommendations for vendors and large organisation, and for future research.

9.1. Answers to the research questions

**Sub-question 1: What processes drive the general behaviour of the vulnerability ecosystem?**
Based on a review of related literature, it is concluded that the primary processes can be arranged according to the four main stages of the life cycle of a vulnerability. These are the discovery, exploitation, disclosure, and patching of the vulnerability. Each of these phases is governed by different actors and mechanisms within the system. The discovery is the result of effort by security researchers and hackers. Whether the vulnerability gets exploited or disclosed depends on their decision. Vulnerability markets are the key mechanisms that facilitate the transfer of exploits and knowledge on vulnerabilities. Exploitation is performed by black hat hackers, while the development and release of a patch is primarily performed by the software vendor, or open source community. Deploying the patch is the final action in the life cycle and is the sole responsibility of the defenders.

**Sub-question 2: What properties of the vulnerability ecosystem are suggested to influence the vulnerability discovery rate and the resulting attacks according to literature?**
An extensive literature review revealed that the following properties are suggested to influence the vulnerability discovery rate: the market share of software, shared code between different versions, learning effects for the discoverers, recently discovered vulnerabilities within the software, and vulnerability depletion as a result of finite vulnerabilities contained within the software. The following four properties of vendors and defenders have an effect on the number of attacks: secure development capabilities, patch development policy, patch release policy, and the defender’s security strategy.
Sub-question 3: *Which actors, technical characteristics, and mechanisms should be included in the conceptualisation of the vulnerability ecosystem?* To the author’s knowledge, there is currently no comprehensive conceptualisation of the vulnerability ecosystem. The goal of this question was to provide a conceptual model for this research and future use. Based on a literature review, the following actors have been identified: vendors releasing software, defenders using software, discoverers searching for vulnerabilities and attackers exploiting vulnerabilities. The main object within the vulnerability ecosystem is the software containing vulnerabilities. The main mechanisms are the black market for vulnerabilities and bounty programmes. The primary properties of the vendor is its ability to develop secure software, its patch development policy, and its patch release policy. The defender’s primary properties are its value at risk, its number of systems, and its security strategy. The discoverer’s primary properties are his or her skill level, knowledge of the software, and motivation. The primary properties of the attacker are his or her skill level, means to acquire exploits, and attack strategy. The software is mainly characterised by the version, the number of remain vulnerabilities, and the amount of shared code. Vulnerabilities are characterised by the software to which it applies, the severity, and its status. This status can be secret, offered on the black market, disclosed, or patched. These are the main actors, technical characteristics, and mechanism that comprise the conceptualisation of the vulnerability ecosystem.

Sub-question 4: *What can be learned from the agent-based model on the properties that are suggested to influence the vulnerability discovery rate?* The goal of this question was to analyse the influence of the properties that are identified in sub-question 2 on the behaviour of the system. The results confirm that the market share of software has a large influence on the rate of discovery throughout the life cycle of the software. Learning effects for the discoverers have the largest influence on the transition between the low rate of discovery in the early years (the learning phase) and the subsequent high rate of discovery (the linear phase). However, the overall influence of learning effects is limited. The results also showed that the amount of shared code primarily has an influence on the rate of discovery when most of the defenders start to adopt newer versions of the software. Due to limitations in the model, it was not possible to realistically examine the influence of recently discovered vulnerabilities. The final property is the number of present vulnerabilities in the software when assuming finite vulnerabilities. The results show that it has a large influence on the rate of discovery starting during the linear phase until the end of the software’s lifespan. In general, these findings confirm the initial hypotheses from related literature. In conclusion, the market share has a major influence on the rate of discovery, while shared code primarily influences the rate of discovery during the end of the lifespan of the software.

Sub-question 5: *What can be learned from the agent-based model on the influence of the properties of the defenders and vendors on the risk exposure of large organisations?* This question focusses on the influence of the vendor’s secure development capabilities, its patch development policy, its patch release policy, and the defender’s security strategy. The defender’s security strategy determines how long it takes to deploy a patch once it is released by the defender. The results from this experiment are surprising, since they indicate that not necessarily the number of vulnerabilities, but rather, the knowledge diffusion of their presence, influences the attack frequency. It shows that the vendor’s secure development capabilities only have a limited influence on the number of attacks on the defenders. Rather, the time it takes for the vendor to develop a patch and the timing of releasing the patch, have a large influence on the number of attacks on the defenders. In particular, a continuous patch release policy can cause a major increase in the number of attacks when combined with a relative long patch development time. The results also show that the defender’s ability to deploy patches in a
timely manner has a major influence on the number of attacks.

**Main question:** Which properties of the vulnerability ecosystem have a major influence on the vulnerability discovery rate of operating systems and what can vendors and organisations do to minimise the resulting attacks? The answer to the main research question comprises the answers to sub-questions 4 and 5. The primary property that influences that vulnerability discovery rate remains the market share of the software. Also, shared code has a major influence on the rate of discovery for an operating system, once its market share starts to decrease as a result of the introduction of a newer version. Minimising the risk of attacks from the vendor’s side depends on the right combination between the time it takes to develop a patch and the patch release policy. From the defender’s side, fast deployment of patches is essential if an organisation wants to minimise the risk of being attacked.

### 9.2. Summary of the main findings

The aforementioned answers enhance existing knowledge by providing a detailed analysis on how different properties influence the behaviour of the system. The main findings are based on these answers and revolve around three key topics. First, the system surrounding software vulnerabilities. Second, the properties of the system that cause the vulnerability discovery rate to evolve over time. And third, the options that software vendors and large organisations have to minimise the attacks as a result of software vulnerabilities. The main findings are summarised in the sections below.

**The vulnerability ecosystem** Some work has been done on describing the system surrounding vulnerabilities. However, most work only focusses on specific parts of the system. On the basis of an extensive review of related literature, the vulnerability ecosystem is conceptualised as the primary elements and their relations, as shown in figure 9.1.

![Figure 9.1: The primary elements and relations of the vulnerability ecosystem](image-url)

The conceptualisation shows how the four key actors interact with each other and what their perspective is on the software and its vulnerabilities. In addition, it shows how bounty programmes and market places are positioned in the vulnerability ecosystem. A more detailed version of this model can be
found in chapter 5, figure 5.1. This trimmed down version is useful for communication purposes and for providing an overview of the roles of the different elements in the system.

**Vulnerability discovery rate**  This research finds its basis partially in the s-shaped curve describing how the rate of discovery evolves over time, as shown in chapter 1, figure 1.2. This shape is first proposed by Alhazmi and Malaiya (2005b) and is the main result of two transition phases, namely the transition from the learning phase to the linear phase, and the linear phase to the saturation phase. The results from the experiments showed that the s-shaped curve requires further refinement in the sense that the transition from the linear phase to the saturation phase is not as clear as described by Alhazmi and Malaiya (2005b). This relates to the fact that saturation is not clearly witnessed in the results from the model, nor for real operating systems as was discussed in section 7.4.2. The additional experimentation presented in appendix A, indicated that high amounts of shared code and increasing interest in vulnerability discovery can possibly explain the lack of saturation. Figure 9.2 shows how the rate of discovery evolves over time according to the model. In addition, it shows how the phases and transitions between them are influenced by the properties of the vulnerability ecosystem. The grey curve and the dotted grey transition lines indicate the original shape, as proposed by Alhazmi and Malaiya (2005b).

![Figure 9.2: The vulnerability discovery rate and the influence of different properties during each phase](image)

Once software is released, it takes a while before it starts to pick up market share. The first discoveries coincide with the speed at which the software is adopted. This is caused by the increasing interest from discoverers. The speed at which the rate of discovery goes from a slow increase to the fast increase of the linear phase, is partially determined by the speed at which discoverers accumulate knowledge on the new software. The linear phase continues until it becomes more difficult to find new vulnerabilities. However, this depends on how the software evolves due to patches and upgrades, which is an interesting point of discussion. This discussion is picked up in section 9.4. The transition to the final phase coincides with adoption of the next version by the users. This process can take
numerous years, which causes the transition to take place over a long time. The increase beyond this point is primarily determined by the vulnerabilities that are discovered in newer versions and, due to shared code, also apply to older versions.

**Risk exposure due to vulnerabilities** This leads to the last main finding of this thesis; the options that vendors and large organisations have for minimising the attacks as a result of software vulnerabilities. The model was used to analyse which properties of both the defenders and vendors have the most influence on the number of attacks on the defenders. The results were quite surprising. The vendor’s ability to develop secure software, and thus to minimise the number of vulnerabilities in the software and the chance of introducing new vulnerabilities when releasing a patch, only has a limited influence on the number of attacks on the defenders. Patch development time and patch release time are far more important determinants according to the model results. In particular, continuously releasing patches combined with longer patch development times, can have an adverse influence on the number of attacks. An overview of the likelihood of being attacked through a vulnerability as a result of different vendor properties is shown in table 9.1.

<table>
<thead>
<tr>
<th>Patch development policy</th>
<th>Patch release policy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short development time</td>
<td>Medium risk</td>
</tr>
<tr>
<td>Medium development time</td>
<td>Higher risk</td>
</tr>
<tr>
<td>Long development time</td>
<td>Higher risk</td>
</tr>
</tbody>
</table>

Apart from the vendor’s responsibility to determine whether a patch should be released as well as reducing the development time, it is even more important that organisations implement the patches as quickly as possible. The results show that an average difference in the time that it takes to deploy a patch of 60 days results in a 55% increase of the median attack frequency. This signifies the responsibility of the organisations themselves when it comes to minimising their risk exposure as a result of software vulnerabilities. This is interesting, given that the defenders are completely reliant on the vendors to provide them with the means to minimise the impact of vulnerabilities.

Based on the results, both the defenders and vendors have a significant influence on the risk exposure of the defenders. The vendors can decrease the attack frequency of the defenders by balancing the right combination of two key policies. First, how long it takes for the vendor to respond to the discovery of a vulnerability, and second, when it releases the patch. However, this only works if the defenders also improve their capabilities and capacity for patch deployment, in order to significantly decrease the time it takes to test and deploy the patch throughout the organisation.

### 9.3. Reflection on the used method

This section reflects on the conceptualisation and formalisation of the agent-based model, and the extent of the validity of the results. In light of this reflection, this section also discusses the value of agent-based modelling and a systems perspective on the vulnerability ecosystem.

**Validity and scope** Modelling requires an abstraction of the real world. In this case it is an abstraction of the conceptualisation, which in turn is an abstraction based on the observer’s view on the vulnerability ecosystem. This raises two questions: First, to what extent is the model a valid translation of reality? Second, does the scope include all the necessary parts to be representative of the actual system? Both questions and their implications for the results are addressed here.
The validity of the model has been thoroughly assessed through the methods commonly used for validating agent-based models. In general, these methods confirmed the validity of the model to the extent that is required for this research. However, there are several areas that show the limitations of the model. The most important point of discussion concerns the saturation phase. The comparison with empirical data on the number of discovered vulnerabilities for three Windows systems shows that the model deviates from the observed rates. In particular, the model shows a saturation of the rates after several years, whereas the empirical data does not. In contrast, the data that was used for fitting the AML model by Alhazmi et al. (2006), was from older operating systems and their data did show saturation. Additional experimentation showed that increasing the discoverers over time and assuming higher amounts of shared code can partially explain the lack of saturation. Nonetheless, the fact that the model shows saturation, whereas this tendency is not observed in real data, limits the validity of the results regarding the final phase on the lifespan of the software. This indicates that the assumptions and parameterisation of the model might not be a correct translation of reality, even though they are partially based on existing research. This highlights a limitation of how agent-based modelling is applied in this thesis, as it assumes that previous work is correct, whereas in reality it might be outdated, or even incorrect.

Scoping decisions have been made which potentially influence the behaviour of the model and limit the generalisability of the results. The primary scoping decisions concern the focus on large organisations and operating systems. Large organisations are characterised by their high value at risk and slow response. Advanced hackers are much more likely to use valuable exploits in order to attack large organisations than wasting time on individual consumers. In addition, the challenges surrounding patch deployment for large organisations, do not apply to smaller organisations and individual users. This makes it difficult to translate the results from the model to smaller organisations and individual users. The second scoping decision concerned the focus on operating systems. There are many different types of software with varying levels of complexity and importance to the organisation. Focussing on operating systems has important implications for the generalisability of the results. Operating systems are required on all systems, whereas other types of software are “optional”. In addition, vulnerabilities in operating systems offer different levels of access compared to vulnerabilities in other software. Another difference is that a system in most cases only runs one operating system, whereas it can run a multitude of other software at the same time. This means that some of the assumptions in the model are questionable when applied to software other than operating systems. This includes the assumption that an organisation only uses an operating system from a single vendor, and the assumption that discoverers can pretty accurately determine the market share of the software. Another difference is the ease of patching. Less critical software can be patched automatically, while operating systems are more likely to perform a critical function, and therefore require testing before a patch can be deployed. Despite these differences, the dynamics of the model are likely to be similar to other widely adopted software, while being less applicable to more specific or less critical software.

This leads to the discussion of how the results from the model should be interpreted. The first limitation is that the scale of the model is not representative for the number of actors in the real world. Additionally, some of the parameters are heuristically chosen and difficult to validate. Hence, the results should not be interpreted as absolute values. Rather, the results show what underlying properties of the system influence the observed differences. For instance, the results reveal that a continuous release policy greatly increases the number of attacks. These results should not be interpreted as evidence that continuously releasing patches is dangerous. Rather, this should be interpreted as an indication that releasing a constant stream of patches can lead to effects that are opposite to the initial intentions. Inherently to the nature of modelling, the purpose of this model is not to provide accurate predictions or determine possible outcomes with absolute certainty. The model and the results should be viewed as a
tool to structurally combine our current understanding of the system, and based on that understanding, determine the underlying cause of the observed behaviour. The results from the agent-based model of the vulnerability ecosystem should be viewed from this perspective.

**Limitations and possible improvements** Although the scope of the model is wide compared to existing work and covers most parts of the vulnerability ecosystem, the level of detail could be improved. One of the main limitations of this model is the relative simple representation of the markets for vulnerabilities. However, this remains a challenge, since limited information is available about the inner workings of especially the vulnerability black market. Algarni and Malaiya (2014) provide a possible source for improving the black market in the model or incorporating additional markets. However, the problem remains that there is limited knowledge on the pricing mechanisms, especially for the more obscure black and grey markets for vulnerabilities.

A second limitation is the limited diversity of both the discoverers and the attackers. Given the large range of potential attackers and the varying levels of skills, means, and professionalism, the two types of attackers in the model are an abstract representation of reality. In addition, the model does not include highly organised attackers such as Advanced Persistent Threats (APT). These can either be advanced criminal organisations, or military and intelligence agencies supported by nation states. Advanced persistent threats include both the discoverers and attackers, in additional to nearly unlimited means. This would change the threat landscape and the prices on the black market. Widening the scope of the model by incorporating a broader range of potential threat actors, might provide additional insights on the interactions with the attackers. This includes the influence of APTs on the market prices and availability of exploits, in addition to differences in the frequency and severity of attacks.

The relatively simple representation of the actual attack and the response is a third limitation of the model. In the current model, the success of an attack is determined by a chance relative to the attack strength of the attacker and control strength of the defender. For this thesis the current representation suffices, because it shows sufficient similarity to find valid answers to the research question. However, if one would want analyse the interactions between attackers and defenders in detail, a more realistic mechanism is required. Such a mechanism should distinguish between the three primary phases of an attack: infiltration, reconnaissance, and exfiltration, in order to capture the influence of more detailed defence capabilities of the defenders. Each part of the attack should consider different defensive mechanisms in relation to each phase of the attack. Different durations of an attack should also be included. In reality an attack can take place over several months or even years, a possibility currently not included in the model.

A final limitation of the model is the representation of software itself. Software in general, and especially operating systems, are very complex objects. Given the intangible nature of software, it is difficult to determine what its key characteristics are, except from a huge collection of lines of instructions, each of which determines how the software behaves on a micro-level. A possible solution would be to represent software as a large list of lists. Each list could represent a part of the source code of software, and can either contain a vulnerability or not. A discoverer is limited by its ability to only look at one list at a time. Defenders run an instance of the list of lists as their version and by applying a patch, remove a vulnerability from a specific list. This is a complex and maybe cumbersome solution. However, it might prove valuable for further use of the model, as it enables the incorporation of more technical characteristics of the software and as a result differentiate between different types of software.

**Value of agent-based modelling and a systems perspective** Applying agent-based modelling in this field of research is a novel approach. Therefore, a discussion of its value in relation to empirical methods is appropriate. Cybersecurity has proven to be a complex and challenging subject, due to
the obscure nature of most cyber-related activities. Empirical research is invaluable for improving our understanding of the vulnerability ecosystem. However, if data is scarce or even unavailable, different approaches are needed. Agent-based modelling is a valuable method to explore possible scenarios and test our assumptions in the absences of specific empirical data. This requires a sound empirical foundation for constructing the model. Agent-based modelling should not be seen as a panacea for every question. Rather, it should be seen as a tool to slightly extend our reach when empirical methods cannot be used.

Valuable work from different perspectives has focussed on the vulnerabilities as an important part of cybersecurity-related challenges. The main knowledge gap that forms the foundation of this thesis is a missing systems perspective. Taking a broader perspective on the vulnerability ecosystem has led to the conceptualisation of the vulnerability ecosystem and the agent-based model. The final point of discussion concerns the benefits of using a systems perspective on vulnerability discovery and the resulting attacks. Chapters 2 and 3 discussed how many different parts of the system influence its behaviour. Despite this work, two questions remain unanswered.

Previous work has shown how vulnerability rates are likely to evolve over time by using technical characteristics of the software to model and predict this process (Alhazmi et al., 2006; Woo et al., 2011). However, the influence of properties external to the software and their interaction remains unclear. Analysing their influence, provided with limited data, requires an approach that can structurally simulate the relation between different parts of the system. The same argument applies to analysing which properties influence the attacks as a result of discovered vulnerabilities. Ozment (2007) and Bilge and Dumitras (2012) show that releasing a patch significantly increases the number of attacks. However, the data does not enable us to analyse how different behavioural properties of the vendors and defenders directly influence the number of attacks, since it is difficult to measure on a sufficient scale who is attacked, through which vulnerabilities, and what their behavioural properties are. Again, a broader perspective in combination with agent-based modelling enables us to structurally compare plausible interactions and their influence on the number of attacks. Taking a narrow perspective, and thus omitting possible interactions, would have resulted in less valid results. Hence, the systems perspective is inherent to the inability to measure certain interaction and the need to incorporate their combined influence.

9.4. Discussion on the implications
The findings have some interesting implications. In this section four topics are briefly discussed concerning the saturation of the rate of discovery, the impact of assuming finite or infinite vulnerabilities, the options to minimise the risk for the defenders, and finally, the cost of security.

Saturation of vulnerability discovery Before discussing whether or not saturation of the rate of discovery is realistic, a short discussion is needed on the value of understanding vulnerability discovery rates. In general, it is argued that being able to predict the rate of discovery is valuable, because it enables vendors to get better insights in when their software is likely to reveal increased numbers of vulnerabilities (Alhazmi & Malaiya, 2005a; Ozment, 2007; Nguyen & Massacci, 2012). This in turn enables the vendors to anticipate increases in the rate of discovery and ensure that sufficient capacity is in place to assess vulnerabilities, develop patches, and prepare the patches for release to the users. However, the merits of predicting vulnerability discovery only hold if there are large variations in the rate of discovery during the lifespan of the software. Hypothetically, the eventual decline in the rate of discovery seems reasonable, because searching for vulnerabilities in software that is nearing the
9.4. Discussion on the implications end of its lifespan is irrational. This argument summarises why some authors assume the saturation of the rate of discovery. However, reality does not confirm this hypothesis. The validation showed that saturation in the behaviour of the model does not reflect the continuous increase of the number of discovered vulnerabilities for Windows operating systems. There are several possible explanations why saturation is not observed: First, an increasing interest in software vulnerabilities in general; second, the ongoing use of outdated operating systems; and third, large amounts of shared code. The additional experiments suggested that the combined influence of these factors can prevent saturation. Based on the empirical data it can be concluded that saturation is not realistic. Furthermore, the additional experiments indicate that the lack of saturation might be easily explained. This shows that assuming large variations of the rate of discovery during the lifespan of the software is not realistic. This limits the potential merits of being able to predict the rate of discovery in general.

Vulnerabilities: finite or infinite? The question whether the number of vulnerabilities present in software can be considered finite or infinite, is an ongoing debate. It has dire consequences for some of the proposed solutions that aim to reduce the risk exposure due to vulnerabilities. As is pointed out by Rescorla (2005), bounty programmes only work if they result in a noticeable decrease in the number of remaining vulnerabilities. If the number of vulnerabilities in the software is so high, that even patching dozens of vulnerabilities does not significantly decrease the number of remaining vulnerabilities, these methods will not work. Then, they will only provide additional information to the attackers and discoverers on the presence and location of vulnerabilities, and make defenders that have not implemented the patch (yet) even more vulnerable. This problem is exactly what is observed in the results from the model. It shows that in some situations, releasing more patches results in even more attacks.

The results of this research do not, and cannot, provide a conclusive answer to the question whether vulnerability depletion is actually happening. What the results do provide, is that assuming either of them has consequences for effectiveness of public disclosure and bounty programmes. Zhao, Grossklags, and Liu (2015) show that bounty programmes reveal an initial peak in the number of discovered vulnerabilities in new software, after which, vulnerabilities are discovered at a stable rate. More recent work by Maillart et al. (2016) shows that discovery rates for bounty programmes do slowly decrease over time. However, this seems to be the result of discoverers moving to newer software, which offers quicker rewards due to the presence of more reachable vulnerabilities. Their findings, together with newly introduced vulnerabilities as a result of constantly changing code (Ozment, 2007; Neuhaus, 2012), show that depleting the number of vulnerabilities in software is difficult. The results of this thesis, in combination with the lack of evidence that the number of vulnerabilities is decreasing, cast doubt on the effectiveness of public disclosure and bounty programmes. It calls into question whether it makes sense for software vendors and security researchers to put large amounts of effort in finding and fixing vulnerabilities, when it might not decrease the risk for the defenders. Instead, it might be worthwhile to allocate resources towards detecting which vulnerabilities are actively being exploited, and primarily focus on those vulnerabilities that are being targeted by the attackers.

Improving the odds of the defenders The results from the risk exposure experiment show that the response by both the vendor and defender has a significant influence on the likelihood of being attacked. Not necessarily the number of vulnerabilities, but rather, the diffusion of the knowledge on vulnerabilities strongly affects the number of attacks. The model shows that once vulnerabilities become public, the attack frequency increases significantly. This is in line with findings from Arora et al. (2006) and Bilge and Dumitras (2012). This research extends on their work by assessing which combination of properties of the vendor has the most influence on the attack frequency. Especially the combination of a continuous release policy with a long average patch development time can result in
a surge in the attack frequency. This effect is reinforced by the defender’s (in)ability to apply patches. The results show that there is much to be gained if organisations improve their capacity to enable faster patch deployment. This is an additional argument why public disclosure and bounty programmes might not be as effective as is currently assumed, since they increase the number of patches, and as a result, put more strain on the organisation’s capacity to deploy patches. This points towards the need for more focus on vulnerabilities that are being exploited, instead of trying to patch them all. Only the combined effort of both the defenders and vendors to significantly reduce the time that the defenders are exposed to a known vulnerability, will result in an actual improvement of the odds of large organisations in cyberspace.

The cost of security  This leads to the final point of discussion: what are the costs of improved security and more important, who should pay for them? This is always a hard question. One could argue that the vendor is responsible for making a product that is secure in the first place. However, developing completely secure software is prohibitively expensive as is pointed out by Libicki et al. (2015). Even if vendors would substantially enlarge their effort to minimise the number of vulnerabilities in their software, this will also significantly increase the cost of the software, which eventually will have to be paid for by the defenders. Cavusoglu et al. (2008) propose to share the costs of patch development, patch deployment, and costs as a result of cyberattacks between the defenders and vendors. They argue that this would achieve a social optimal outcome. However, as they point out, this is not a realistic solution, due to the legal complexity required to institutionalise such an arrangement. In addition, the results from this thesis show that the defenders themselves have sufficient options to decrease their risk by improving their patch deployment capacity.

The problem remains that there is little incentive for the vendors to release patches in a timely manner, as lock-in effects prevent true competition and thus, fail to provide such an incentive. A possible solution would be for policy makers to force vendors to speed up patch development by setting deadlines. However, this will introduce a new problem. Either, such legislation would force vendors to develop patches for every known vulnerability, which would increase the number of patches and as a result, put more strain in the defenders patch deployment capacity. Or, the legislation would have to be very complex and include all kinds of exceptions, in order to enable the vendors to decide which vulnerabilities should be patched or not. Both ways, such legislation would fail to achieve the intended social optimal outcome. This shows why it is difficult to allocate the costs and incentives in a social optimal way.

9.5. Recommendations
This final section summarises the main recommendations. Given the focus of this thesis, there are three types of recommendations for three perspectives: the software vendors, large organisations, and future research. Each is addressed in the sections below.

9.5.1. Software vendors
Software vendors should focus on balancing their patch release strategy. Continuously releasing patches, increases the diffusion of the knowledge on vulnerabilities to hackers. This could pose a risk to large organisations that use the software, if they are unable to deploy the patch in a timely manner. Given the organisation’s general inability to quickly deploy released patches, semi-periodically releasing patches is the safest option. This enables users to plan upcoming patch releases, yet enables the immediate release of patches for vulnerabilities that pose an extremely high risk. Second, it is recommended that software vendors put more effort in analysing which vulnerabilities are being
9.5. Recommendations

exploited, as opposed to trying the fix all known vulnerabilities. This decreases the available knowledge for potential attackers, and minimises the strain on the organisation’s capacity to deploy patches.

9.5.2. Large organisations

Every delay in the deployment of a patch increases the risk of being attacked. The results show that an average difference in the time that it takes to deploy a patch of 60 days, results in a 55% increase of median attack frequency. Therefore, large organisations should improve their capacity in order to significantly speed up the deployment of patches. In addition, large organisations should align their patch deployment policy with the vendor’s patch release policy. This enables the organisation to prepare upcoming patch releases. As a result, this should minimise the patch deployment time and narrow down the window of exposure to the vulnerability. Organisations should identify the major bottlenecks regarding patch deployment within their organisation, and assess the costs of removing those bottlenecks. The results show that even a small reduction in the average patch deployment time should result in noticeable reduction in the risk exposure.

9.5.3. Future research

The model used in this thesis could be used as a starting point for further research. The design of the model enables easy integration of additional modules. Some of the recommended improvements are discussed in section 9.3. Extending the model’s attack and defence procedures could be used to simulate more realistic attacks. As a result, the potential losses can be simulated, as well as different defence strategies. This would enable organisations to make better-informed decisions on security investments. One of the limitations of this model is the assumption on the mechanism that determines the price of an exploit. Improved understanding on the mechanisms that drive the black markets could provide the knowledge needed to improve the model. This could improve the validity of this model and future models. In addition, it allows policy makers to explore which policies could disrupt or decrease the attractiveness of the black market.

Based on the results and discussion, it is recommended to focus future research efforts on bounty programmes in order to better assess what their influence is on the risk exposure of large organisations and other users. In particular, it would be valuable to know what fraction of the disclosed bugs should be patched, relative to the organisation’s capacity to deploy them. Another valuable topic is the impact of focussing patching efforts on those vulnerabilities that are being exploited. This requires research on whether or not this method reduces the risk for large organisations, and second, this requires monitoring and prediction methods to analyse and predict which vulnerabilities are being exploited or traded on the black market.
References


Conclusion and discussion


security applications conference (pp. 251–260). ACM. doi: 10.1145/1920261.1920299


doi: 10.1109/TSE.2007.70712
doi: 10.1109/ARES.2013.32
Additional experiment: Saturation

The validation in chapter 7, section 7.4.2 revealed that the assumed saturation of the rate of discovery is not observed in the empirical data for Windows XP, Windows Vista, and Windows 7. The analysis looked at the cumulative number of discovered vulnerabilities for the first 6 years after the release of the software. The lack of an observed saturation shows that assuming that the rate of discovery will eventually slow down is not realistic. However, the idea that saturation should be observed arrises from a logical set of assumptions. Discoverers are interested in software that is attractive to them. No matter the motivation, vulnerabilities for software that has a large install base seems more valuable, both from a financial perspective and a repetitional perspective. If a newer version of the software is available on the market, most users will eventually switch to that newer version. The number of users for the older version will steadily decline until it is no longer worth to look for vulnerabilities. This seems a logical set of assumptions. However, the empirical data does not support this line of thought. Than what might explain this lack of saturation?

The goal of this additional set of experiments is to show what properties can potentially cause the lack of saturation. Given that this question is beyond the scope of the primary research questions, no extensive literature review is conducted. Rather, these experiment focus on the three properties that are plausible candidates to explain the observed behaviour. Table A.1 shows three possible causes for the lack of saturation and how they can be simulated in the model.

Table A.1: Possible causes for the lack of saturation

<table>
<thead>
<tr>
<th>Possible cause</th>
<th>Model property</th>
</tr>
</thead>
<tbody>
<tr>
<td>Increasing interest in vulnerabilities in general</td>
<td>Discoverer growth rate</td>
</tr>
<tr>
<td>Vulnerabilities shared between versions</td>
<td>Amount of shared code</td>
</tr>
<tr>
<td>Slow adoption of newer versions</td>
<td>Switching probability</td>
</tr>
</tbody>
</table>

The interest in software vulnerabilities has increased in the last two decades. Looking at the overall number of vulnerabilities reported in the NVD reveals that the number of discovered vulnerabilities recorded in the database has increased for each consecutive year. Software has become more complex and interconnected over the past year. However, it seems unlikely that this can explain the vast increase seen in the last few years. More likely is a general interest in cybersecurity and software vulnerabilities in general. This can be tested by the agent-based model by increasing the number of...
discoverers over time. For this experiment, the following four scenarios are simulated: No increase (growth_rate = 0), the number of discoverers increases by 50% over time (growth_rate = 0.007), the number of discoverers increases by 100% over time (growth_rate = 0.014), and the number of discoverers increases by 200% over time (growth_rate = 0.028). The results from this experiment are shown in figure A.1. The results show that an increasing population can cause the saturation effect to be less obvious. Especially the increase of 200% over time shows that only a very limited amount of saturation is shown.

![Figure A.1: Vulnerabilities discovered versus different discoverer growth rates](image)

The second possible explanation for the lack of observed saturation are the high amounts of shared code. This might cause that vulnerabilities that are discovered in the newest version of the software also apply to the older versions. High amounts of shared code increases that probability and obfuscates the saturation. For this experiments three scenarios are simulated: 20% of shared code, 40% of shared code, and 60% of shared code. Since the main experiments already analysed the influence of shared code on the rate of discovery and given the influence of an increasing population of discoverers, this experiment assumes a small increase of the number of discoverers over time of 50%. The results are shown in figure A.2. As can be seen from the graph, the amount of share code might for a large part explain the lack of observed saturation.

The final possible explanation that can be tested using the agent-based model is the slow adoption of newer versions. The parameterisation assumes that defenders switch to a newer operating system on average 500 days after the release of a newer version. For the next experiment two additional parameter values are tested: 1000 and 1500 days. Although these values represent a very long time, the fact that Windows XP is still used shows that this is not unrealistic. These values are tested together with an increase in the number of discoverers of 50% over time and 40% shared code in order to get a combined view of the different possible causes. The results are shown in figure A.3. As can be seen from graph, different switch time values do have a large influence on the shape of the curve. This
indicates that the time it takes for the majority of the defenders to adopt a newer version, only has a small influence compared to the previous two properties.
The additional experiments show that each of these properties can possibly explain why saturation is not observed in reality. An increasing number of discoverers over time as well as the amount of shared code seem to have the largest influence. The speed at which defenders adopt a newer version has a smaller influence. These experiments are an addition to the main experiments and have not been constructed as thoroughly. Furthermore, the values that are selected for these experiments are heuristically selected. However, these values are not unreasonable. The results should be interpreted as an preliminary investigation into a plausible explanation and are by no means conclusive. In conclusion, the lack of saturation can possibly be explained by a general increase of the interest in vulnerabilities, higher amounts of shared code, and slow adoption of newer versions.
Additional results

The primary results from the experimentation are discussed in chapter 8. This section provides a more detailed overview of the results. The results discussed here are based on the experiment for determining which properties influence the vulnerability discovery rate. The results from the experiments regarding the risk exposure are presented in sufficient detail in chapter 8. As is discussed in section 7.3.1, the market share experiment has a slightly different setup. Since varying that parameter is different from the parameters for the other experiment, they are not included in this section. The fitted lines are based on the Loess method. The shaded area indicates the 95% confidence interval.

Figure B.1: Vulnerabilities discovered versus the number of present vulnerabilities

Figure B.1: Vulnerabilities discovered versus the number of present vulnerabilities
Figure B.2: Vulnerabilities discovered versus the amount of shared code

Figure B.3: Vulnerabilities discovered versus the knowledge build up time

Figure B.4 provides a detailed overview of how each combination of properties of the vendor influ-
ences the number of attacks on the defenders. It shows that the highest attack frequencies only appear for continuous patch release policy in combination with a low and medium development policy. A more detailed discussion on this figure is presented in section 8.2.3.

Figure B.4: Attack frequencies for different combinations of vendor properties
The list below contains an overview of the parameters within the final implementation of the model.

<table>
<thead>
<tr>
<th>Model parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Number of defenders</strong></td>
<td>The number of defenders.</td>
</tr>
<tr>
<td><strong>Number of discoverers</strong></td>
<td>The number of discoverers.</td>
</tr>
<tr>
<td><strong>Number of attackers</strong></td>
<td>The number of attackers.</td>
</tr>
<tr>
<td><strong>Amount of shared code</strong></td>
<td>The amount of code shared between subsequent software versions.</td>
</tr>
<tr>
<td><strong>Shared code learning effects</strong></td>
<td>The amount of knowledge that is accumulated based on shared code.</td>
</tr>
<tr>
<td><strong>Knowledge build up time</strong></td>
<td>The time it takes to approach full knowledge of a software version.</td>
</tr>
<tr>
<td><strong>Total present vulnerabilities</strong></td>
<td>The number of vulnerabilities that are present within the software at the time of release.</td>
</tr>
<tr>
<td><strong>General vulnerability discovery chance</strong></td>
<td>The general chance that a vulnerability is discovered by a discoverer.</td>
</tr>
<tr>
<td><strong>Rationality noise</strong></td>
<td>A random noise that is added to the preference values of the agents to obscure decision making.</td>
</tr>
<tr>
<td><strong>Weights of the decision mechanism</strong></td>
<td>Different weights assigned to the factors in the decision mechanism for software selection by the discoverers.</td>
</tr>
<tr>
<td><strong>Fraction reactive / proactive</strong></td>
<td>The fraction of defenders that have a reactive strategy versus a proactive strategy.</td>
</tr>
<tr>
<td><strong>Fraction white hat / black hat</strong></td>
<td>The fraction of white hat discoverers as opposed to black hat discoverers.</td>
</tr>
<tr>
<td><strong>Fraction advance / basic</strong></td>
<td>The fraction of advanced attackers versus basic attackers.</td>
</tr>
<tr>
<td><strong>Fraction target driven / exploit driven</strong></td>
<td>The fraction of attackers that have demand for exploits based on preferred targets as opposed to attackers that buy exploits based on availability.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Defender maturity levels</td>
<td>The spread of different maturity levels among the defenders.</td>
</tr>
<tr>
<td>Discover skill levels</td>
<td>The spread of different skill levels among the discovers.</td>
</tr>
<tr>
<td>Look back time</td>
<td>The number of days that the defenders look back in time when making decisions.</td>
</tr>
<tr>
<td>Software selection chance</td>
<td>The chance that a discoverer reassess its software preference.</td>
</tr>
<tr>
<td>Increase of means</td>
<td>The average amount with which the attackers accumulate means.</td>
</tr>
<tr>
<td>Exploit base price</td>
<td>The standard starting price of an exploit.</td>
</tr>
<tr>
<td>Attack likelihood</td>
<td>The chance that an attackers attacks.</td>
</tr>
<tr>
<td>Exploit detection chance</td>
<td>The chance that an exploit is discovered during an attack.</td>
</tr>
</tbody>
</table>
Verification results

This appendix discusses the verification of the model in more detail. Its main purpose is to verify that the model is a correct translation of the conceptual model. The following four sets of test were performed. The results of the single-agent tests, interaction tests, and multi-agent tests are presented here.

1. Recording and tracking agent behaviour, in which relevant metrics are identified and recorded.
2. Single-agent testing, in which the behaviour of a single agent is verified.
3. Interaction testing in a minimal model, in which the interaction between agents is tested.
4. Multi-agent testing, in which the emergent behaviour of multiple agents is examined.

D.1. Single-agent testing
For the single-agent test, the following hypotheses have been tested. These hypotheses cover most of the behaviours and interactions in the model.

Theoretical prediction and sanity checks

- If all defenders select the same software to use, this software should report a market share of 1. All other software should report a market share of 0. Error found. Only software that had any remaining users were asked to update their market share. Fixed, revalidated, confirmed.

- If a software has no vulnerabilities offered on the black market, this should increase the potential price on the market, and therefore attract most of the defenders. Confirmed.

- If the discoverer has full rationality and the preference of one of the software becomes higher than the of the current software, the discoverer should change to the one with the highest preference. Confirmed.

- If the discoverer has a knowledge level of 10 (which is way above what is actually possible) it should discover vulnerabilities at a very high rate. Confirmed.

- If the discoverer’s skill level is high, most of its discovered vulnerabilities should have high and medium severity levels. If the discoverer’s skill level is medium its discovered vulnerabilities should have high and medium and low severity levels. Confirmed.
• If the market share of the software on which the discoverer is currently focusing reaches zero, there should be no interest from the attackers, and therefore the average market price for those vulnerabilities should be 0. **Confirmed.**

• If the motivation of a discoverer is white-hat, all the discovered vulnerabilities from this discoverer are disclosed to the relevant vendor and the status of the vulnerability is set to disclosed. **Confirmed.**

• If a vulnerability (exploit) is sold by a discoverer who is likely to sell the vulnerabilities multiple times, this should result in increased number of reported attacks and therefore a shorter time until a patch is released. **Confirmed.**

• If a vulnerability does not have any remaining defenders that are affected by it, it should have a market price of zero and not by acquired by any attacker, or used in any attack. **Error found.** Rare situations could potentially lead to attackers buying a vulnerability that could not be used. **Fixed, revalidated, confirmed.**

• If an attacker does not have sufficient means, it can only acquire exploits that are available because they are publicly known. **Error found.** A limiting condition prevented the attacker from acquiring public exploits in case of few defenders. **Fixed, revalidated, confirmed.**

• If an advanced attacker selects a target it basis its decision on the value of that target divided by its control strength. Advanced attackers should therefore prefer targets with low maturity levels, as this leads to a low control strength. **Confirmed.**

• If an attacker with a target driven exploit preference has attacked all possible targets based on the exploits it has, the attacker has to wait until new exploits become available. **Confirmed.**

**D.2. Interaction testing in a minimal model**

This series of test aims at running the model with the minimum amount of agents required to actually run the model.

**Initial errors:**

• An error was found in the target selection procedure for attackers with a target driven exploit preference. **Error found.** Three types of software with a market share higher than 0.05 were requested, but only one could meet that requirement. A check is added to prevent this error. **Fixed, revalidated.**

• An division error occurred when calculating the market share. **Error found.** In rare situation an defender could be initialised with zero system. **Fixed, revalidated.**

**Theoretical prediction and sanity checks**  The following hypotheses are tested with four vendors, one discoverer, one defender, and one attacker.

• Vulnerabilities are only discovered for the software on which the single discoverer is currently focusing, since there is no effort invested in other software. **Confirmed.**

• Since there is only one defender, the market of the software that it is using should be 1 and therefore attract the discoverer no matter its motivation. **Error found.** Knowledge levels were relative to the total accumulated knowledge, this should not be the case. **Fixed, revalidated, confirmed.**
All attacks should focus of the single defender and therefore only the patches released by the vendor that is providing that software should be released with priority. Confirmed.

Defenders that move to a new system should become vulnerable to all the unpatched vulnerabilities of the newer version. Error found. When switching to a newer version, no links were made with the existing vulnerabilities for which no patches were available. Fixed, revalidated, confirmed.

If no switching between vendors takes place, all vulnerabilities that are found should be in the software from a single vendor, as the software from this vendor is the most attractive due to its 100% market share. Confirmed.

If switching between vendors does take place, vulnerabilities should be found in software for multiple vendors. Confirmed.

D.3. Multi-agent Testing
Multi-agent testing aims at testing similar hypotheses as the previous tests. The following hypotheses have been tested.

Theoretical prediction and sanity checks

- Given that half of the attackers are target driven when acquiring an exploit on the black market, software that has highest combined value at risk of its users should have the most discoverers on average. Confirmed.

- Shared code between different versions of the software should result in a continuous increase over time. Even software that is no longer receiving any focus from any discoverers should still see a slow but decreasing increase of newly discovered vulnerabilities. Confirmed.

- Discoverers accumulate knowledge and therefore have a preference for staying with the software from the vendor on which they have focussed before. Confirmed.

- Not all defender should switch to newer versions of the software. In some cases a defender might not even switch at all. Confirmed.

- The majority of the attacks should target defenders that use software for which the most vulnerabilities are discovered. As the price of these vulnerabilities will be the lowest. Confirmed.

- The total rate at which vulnerabilities are found should correspond with the rate of attack, since discovered vulnerabilities are should lead to an increase in exploits on the black market and this in turn lowers the prices of the vulnerabilities. This should enable more attackers to buy exploits and perform attacks. Confirmed.

Variability testing For this test a 100 runs of the model were used while monitoring the number of discovered vulnerabilities for a specific software version, the number of attacks on defenders who used software from one specific vendor, and the histograms of the resulting output is shown in figures D.1. The histograms with the attack frequency density show no unexpected or unexplainable variability, as there seems to be no significant anomalies in the distributions.
Verification results

The second variability test again looks at the attack frequency per vendor, however, for this test there are two distinctive groups of vendors. Two vendors with a high market share and two vendors with a low market share. Figure D.2 shows the distribution of attack frequencies for both groups. It shows no evidence of anomalies that require further investigation.

The following test assess the distribution of the density of the number of unpatched vulnerability for different patch priority levels of the software vendor. The results are shown in figure D.3. The results show that there is a difference between the different populations regarding the number of unpatched vulnerabilities. The difference is most visible in the tail, which especially for the low patch priority level, seems to be multimodal with a second modal around 550. The other populations also seem multimodal. However, their second modal is close to the larger modal and less obvious. This behaviour can not
directly be explained by the logic of the model. Further investigation revealed that there some minor problem with the patch release procedure as a result of a wrong initialisation of the patch development duration. This problem has been fixed.

Despite the fixed error, the behaviour as shown in figure D.3 remained nearly unchanged. There seems to be a bifurcation in the attack frequency. Further analysis is performed to take more detailed look at the attack frequency itself, rather than it being the result of different patch priority levels. Figure D.4 shows that a bifurcation appears around 6000 attacks.

![Figure D.3: Attack frequency distribution for different vendor patch priority levels](image)

![Figure D.4: Attack frequency versus unpatched vulnerabilities](image)
This clearly shows something is causing higher attacks to occur in some situations, which cannot be explained by the number of unpatched vulnerabilities. Further investigation revealed that the seemingly undesired behaviour is actually the result of the behaviour of the agents as intended. The attack frequency is caused by the different patch release policy. This is further discussed in the experimentation results.

**Data consistency test** In addition to the aforementioned tests, a data consistency test was performed. The analysis of the data from the experiments revealed that some data point that were expected in the parameter space were not present in the results. This is shown in figure D.5. The highlighted area should have shown attack frequencies as well. However, as can be seen they are not there. Further investigation revealed an error in the LHS that was used for the experiment. The missing values are the result of missing combinations in the LHS and as a result, no experiments were simulated using those combinations of experiments. New experiments were used to generate the results as presented in chapter 8.

![Figure D.5: Missing data points in the parameter space.](image-url)

**Time line sanity** The final verification test is assessing the behaviour of a longer time span. Since the result is limited to a simulation time of 4199 ticks, since a longer time span causes problems a fifth generation of software is introduced. This causes a problem with some of the reporters due to the structure of some of the lists in the model. In order to save simulation time, this test was performed 3600 ticks, as this covers all generations of the software. The results are shown in graph D.6.
The test reveals no unexplained behaviour. However, it is difficult to distinguish individual runs, due to the density in the some parts of the graph. An additional test was performed that only analyses the second generation software. The experiment focus primarily on this generation, which makes it a logic candidate for this test. The results are shown in graph D.7. Each run is given an different colour in order to make it easier to distinguish the behaviour of individual runs. The behaviour shows some interesting behaviour. For a majority of the runs we can see that there is some sort of step wise increase over time. There some clear periods of rapid increase, followed by a period of few discoveries. This behaviour is unexpected. Further investigation revealed a error in the procedure that determines when discoverers switch to other software. The parameter value that determines how often the discoverers switch was unreasonably high, which resulted in cyclic behaviour of rapid switching with no real cause. Slowing down the number of times the discoverers switch software resolved this issue.
The variability test and time line sanity test revealed no further errors. It is therefore concluded that the number of errors is the software implementation is reduced to an acceptable level.
Experimental setup

This appendix contains the details on the experimental setup. Each of the tables below describes the parameter values for each experiment. They are discussed in chapter 7. The chosen values are discussed in this appendix.

Table E.1: Standard model parameter settings for experimentation

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of defenders</td>
<td>200</td>
</tr>
<tr>
<td>Number of discovers</td>
<td>50</td>
</tr>
<tr>
<td>Number of attackers</td>
<td>50</td>
</tr>
<tr>
<td>Market share for each vendor</td>
<td>25%</td>
</tr>
<tr>
<td>Amount of shared code</td>
<td>0.25</td>
</tr>
<tr>
<td>Knowledge build up time</td>
<td>100</td>
</tr>
<tr>
<td>Total present vulnerabilities</td>
<td>250</td>
</tr>
<tr>
<td>General vulnerability discovery chance</td>
<td>0.03</td>
</tr>
<tr>
<td>Exploit base price</td>
<td>10,000</td>
</tr>
<tr>
<td>Attack likelihood</td>
<td>0.1</td>
</tr>
<tr>
<td>Exploit detection chance</td>
<td>0.01</td>
</tr>
</tbody>
</table>

The number of defenders, number of discoverers, and number of attackers are difficult to determine. The population of the actual number of agents is much larger than the model can handle, due to the required computing power needed to run this model on a true scale. In addition, it is difficult to determine the true size of the population of defender, attackers, and discoverers. The second problem is the ratio between the attacker, defenders, and discoverers. It is difficult to determine a number of attackers per defender. However, we can assume that the population of potential targets is larger that the number of attackers, since we are only looking at defenders that actually have a viable chance of penetrating a defender. On the other hand, we know that there is a reasonable chance that attacks from different attackers can coincide with each other (Herley, 2013). Therefore, the ratio of the number attackers versus the defenders should not be unreasonably small. The current 1/4 ratio between the attackers and defenders is a reasonable assumption based on the aforementioned criteria.

The market share is determined by the number of vendors in the model. In reality, there are only a few vendors who develop operating systems. The market share of these operating systems differs greatly. Windows for example has a market share of up to 90%, while some Linux distributions and
OSX share the remainder of the market. However, since there is a dedicated experiment to determine the influence of the market share, the market is set equally to 25% for each vendor and its software.

The amount of shared code is set to a fraction of 0.25 in order to resemble a reasonable overlap between software. The actual percentage of code shared between operating systems varies significantly, since it depends on different factors. How are shared code libraries used? How many changes are made for the newer version? Is the similar code used differently in other versions? This greatly differs between different vendors and even different versions. Windows is known for having a large amount of legacy code. However, there is a large chance that code is used differently, and vulnerabilities in these sections of code, provide different levels of access, due to chances in the interaction with other code. The amount of shared code of 25% provides a reasonable estimate that is on the lower side of the potential range. However, based on possible changes surrounding the code, it is unlikely that a similar vulnerability provides the same access in different versions. Taking into account how the software is conceptualised, assuming much higher amounts of shared code would result in an overestimation of the influence of the shared code between software.

The knowledge build up time is equally difficult to determine. It depends on how quickly the discoverers learn, how easy it is to understand the software, and even more difficult; when do you fully understand software? Research has shown that a few weeks are required to get familiar with the software, after which vulnerabilities are discovered at a steady rate (Khoo et al., 2010). Operating systems on the other hand are huge collections of code, which makes it unrealistic that a discoverer can become familiar with the code in a few weeks. A knowledge build up time of 100 would mean that reaching a thorough understanding of the code takes around one year, while the significant levels of knowledge are accumulated after around 100 days. This is a realistic assumption since it gives an advantage for experience discoverers, while spending a few weeks on an operating system should result in the first discoveries. This makes switching worth while and at the same rewards discoverers that focus on a single operating system.

The number of total present vulnerabilities is impossible to determine. There are some estimates on the number of vulnerabilities per 1000 lines of code (Libicki et al., 2015). However, given that this model does represent a valid population the number of discoverers, assuming such values would result in a very high number of vulnerabilities compared to the number of discoverers. The model exploration shows that the model is sensitive to the number of present vulnerabilities, although not extremely. In order to ensure that it becomes harder to discover vulnerabilities if the code remains static, a value is chosen that can actually be depleted by the population of discoverers.

The final four parameters are even more difficult to determine. These are the general vulnerability detection chance, exploit base price, attack likelihood, and the exploit detection chance. These parameters have a high influence on the behaviour of the system as is shown by the model exploration. The chosen values do not resemble valid estimation. Rather, they are heuristic estimates that appear to result in stable behaviour. However, it has to be noted that this is not tested thoroughly. Monte Carlo simulation or other methods could have been used to confirm the assumption that these values are within a stable range of values. However, due to time constraints no such test is performed. On the other hand, the model exploration shows that results are distributed normally, which makes it unlikely that clear shifts in the state of the system can be induced by parameter values that lie within the chosen range. Therefore, we can conclude that the chosen parameter values are useful and adequate for current purpose of the model. However, for future use of the model, it is recommended to reassess the chosen values in order to ensure that they are still adequate.

The following tables show the values that are chosen for the experiments. The value selection for these parameters is discussed in chapter 7.
Table E.2: market share experiment settings

<table>
<thead>
<tr>
<th>Simulation</th>
<th>Vendor A</th>
<th>Vendor B</th>
<th>Vendor C</th>
<th>Vendor D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>25%</td>
<td>25%</td>
<td>25%</td>
<td>25%</td>
</tr>
<tr>
<td>2</td>
<td>50%</td>
<td>16.7%</td>
<td>16.7%</td>
<td>16.7%</td>
</tr>
<tr>
<td>3</td>
<td>75%</td>
<td>8.3%</td>
<td>8.3%</td>
<td>8.3%</td>
</tr>
</tbody>
</table>

Table E.3: Learning effect experiment settings

<table>
<thead>
<tr>
<th>Simulation</th>
<th>Amount of shared code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.1</td>
</tr>
<tr>
<td>2</td>
<td>0.2</td>
</tr>
<tr>
<td>3</td>
<td>0.3</td>
</tr>
<tr>
<td>4</td>
<td>0.4</td>
</tr>
<tr>
<td>5</td>
<td>0.5</td>
</tr>
<tr>
<td>6</td>
<td>0.6</td>
</tr>
</tbody>
</table>

Table E.4: Shared code experiment settings

<table>
<thead>
<tr>
<th>Simulation</th>
<th>Total present vulnerabilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>2</td>
<td>200</td>
</tr>
<tr>
<td>3</td>
<td>300</td>
</tr>
<tr>
<td>4</td>
<td>400</td>
</tr>
<tr>
<td>5</td>
<td>500</td>
</tr>
<tr>
<td>6</td>
<td>600</td>
</tr>
</tbody>
</table>
Model exploration

Graph F.1 shows how the vulnerability discovery chance, the exploit detection chance, the rationality noise, and the total present vulnerabilities influence the behaviour of the model. Each of the four graphs is shaded according to the value of the respective parameter.

![Graphs showing model behaviour](image)

Figure F.1: Overview of the spread of different parameter settings per run
Lighter shades of blue refer to higher parameter values and darker shades of blue refer to lower values. The most interesting result is the graph in the top left corner. It implies a strong correlation between vulnerability discovery chance and the vulnerability discovery rate. This is very obvious, but should be kept in mind throughout this report. The graphs also show that the influence of the other parameters is much less severe. Another noticeable outcome is the graph in the bottom right corner. It show the impact of the number of total vulnerabilities. As can be seen by the spread on the colour for each run, there is no clear relation between the parameter value and the model behaviour. So the total number of vulnerabilities only has a small influence on the behaviour of the model compared to the impact of the vulnerability discovery chance.

A similar analysis is performed for the parameters that are included in the main experiment. The resulting behaviour is shown in graph F.2. Each of the four graphs is shaded according to the value of the respective parameter. Lighter blue indicates higher parameter values and darker blue indicates lower parameter values.

The first graph (top left) shows the impact of the market share on the model behaviour. As was noted in the setup, no varying parameter settings were included in the LHS. Instead, fixed market share values where assigned to the vendors. The number of discovered vulnerabilities in software from vendors with a market share of 35% are coloured light blue. The number of discovered vulnerabilities in software from vendors with a market share of 15% are coloured darker blue. The sharp contrast is caused by the order of printing the lines. However, a clear difference can be seen. The other factor seems
to have a significant impact on the model behaviour is the learning effect, as is visible in the bottom left graph. Shorter knowledge build up times appear the result in much higher number of discovered vulnerabilities. This can be explained by its relation to the vulnerability discovery chance.