Detection of Botnet Command and Control Traffic in Enterprise Networks

Pieter Burghouwt

DETECTION OF BOTNET COMMAND AND CONTROL TRAFFIC IN ENTERPRISE NETWORKS

Proefschrift

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft,
op gezag van de Rector Magnificus prof. ir. K. C. A. M. Luyben,
voorzitter van het College voor Promoties,
in het openbaar te verdedigen op 2 februari 2015 om 10:00 uur

door

Pieter BURGHOUWT

ingenieur Elektrotechniek
geboren te Voorburg, Nederland.

Dit proefschrift is goedgekeurd door de promotor:

Prof. dr. H. J. Sips

Samenstelling promotiecommissie:

Rector Magnificus,	voorzitter
Prof. dr. ir. H.J. Sips,	Delft University of Technology, promotor
Dr. M.E.M. Spruit,	The Hague University of Applied Sciences
Prof. dr. K.G. Langendoen,	Delft University of Technology
Prof. dr. ir. J. van den Berg,	Delft University of Technology
Prof. dr. ir. H.J. Bos,	VU University of Amsterdam
Prof. dr. S Etalle,	Technical University of Eindhoven
Dr. ir. J. Henseler,	Amsterdam University of Applied Sciences
Prof. dr. ir. D.H.J. Epema,	Delft University of Technology

Dr. M.E.W. Spruijt, heeft als begeleider in belangrijke mate aan de totstandkoming van het proefschrift bijgedragen.

Keywords: botnets, network intrusion detection, command and control traffic, cybersecurity

Printed by: Ipskamp Drukkers B.V

Cover design: Odine Burghouwt

ISBN 978-94-6186-414-7

An electronic version of this dissertation is available at
`http://repository.tudelft.nl/`.

This thesis is dedicated to my wife Carmen and my children
for their love and patience.

Pieter Burghouwt

ACKNOWLEDGMENTS

This thesis is the result of hard work over the last years. To combine PhD research with my work as a lecturer, a husband and a father was often challenging. Nevertheless, with great satisfaction I can look back at a phase of my life in which I was a student again. I enjoyed all aspects of it: from satisfying my curiosity in the research, to presenting my work at conferences in other countries.

This work would not have been possible without the support of many people. I express my grateful thanks to all of them.

First my thanks go to my supervisors Henk Sips and Marcel Spruit for their continuous and high quality support during my PhD studies. By their coaching and constructive feedback they enabled me to conduct the research and develop my research skills.

I thank the PhD committee for their review and their provision of high quality feedback that improved my thesis.

I also thank my employer The Hague University of Applied Sciences who offered me the unique opportunity to conduct my PhD research. My special thanks go to: Gert de Ruiter, Ineke van der Meulen, and other members of the board who enabled and patiently supported my study, to Jan Dirk Schagen who encouraged me to start the PhD research, and to all my colleagues who had to work harder because of my limited availability for teaching.

Finally but most importantly I thank my wife Carmen and my children for their patience, moral support, and their faith in me. My mental and sometimes physical absence must have been difficult for them but they never complained.

I could not have completed my research without the support of all these wonderful people!

Pieter Burghouwt
Delft, January 2015

CONTENTS

1

INTRODUCTION

1.1. RESEARCH BACKGROUND

We depend on the Internet.

In 2013, more than 72% of the population in the European Union used the Internet at least once a week [114]. Individuals use it for communication, for information, as a marketplace, and for leisure. Organizations use it to connect with their customers, partners, suppliers, branches and teleworkers. Governments use it to connect with civilians, enterprises, and other governments. The Internet brought forth completely new concepts, such as e-commerce, e-government and cloud computing.

Unfortunately, a new type of crime has also emerged along with the Internet: cybercrime. Its impact is significant. According to a Eurobarometer survey, held in 2013, about 50% of the Internet users in Europe is concerned about experiencing various types of cybercrime [115]. The global damage caused by cybercrime is not exactly known, but runs into billions of USD [4].

Botnets play an important role in cybercrime. A botnet consists of a large group of remotely controllable computers or bots. The bots are controlled by an individual or organization, referred to as the botmaster. Although there are some rare examples of benign botnets that perform legitimate tasks, most botmasters have malicious objectives and deploy bots exclusively for criminal operations. Without the knowledge or consent of the owner, computers are recruited as a bot by malware infection and subsequently deployed in diverse criminal activities, such as DDoS (Distributed Denial of Service) attacks, spam, click fraud, theft of sensitive information, and even cyber terrorism [27][126][136]. In this work the word botnet refers exclusively to malicious botnets.

1.1.1. SCHEMATIC OVERVIEW OF A BOTNET

Figure 1.1 gives a schematic overview of a botnet. Criminal attacks are launched from bots. Every Internet-connected computer, including: Personal Computers, mobile phones, network printers, embedded devices, and industrial process controllers, can be turned into a bot by malware infection.

1

The botmaster communicates with the bots in a special communication infrastructure, referred to as the C&C (Command and Control) Infrastructure. The botmaster is separated from the attacking bots by intermediate computers or stepping stones that complicate the trace back from discovered bots towards the botmaster by the C&C communication. The trace back complexity is further increased when the stepping stones are distributed over several countries with different legislation [69].

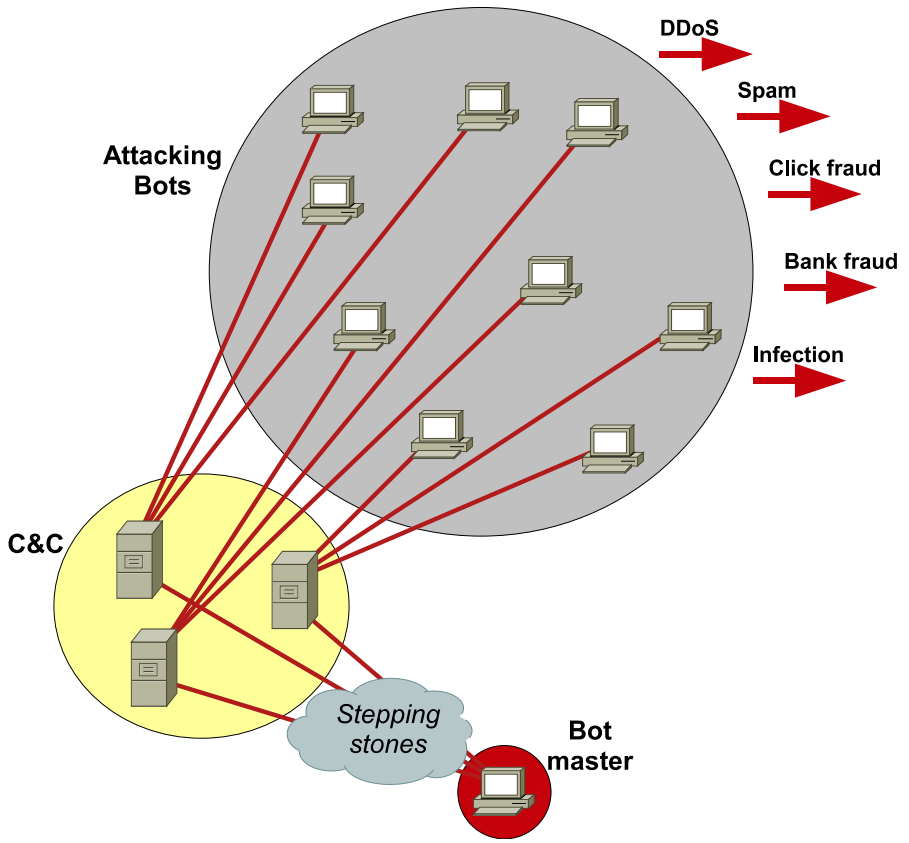

Figure 1.1: Schematic overview of a botnet.

1.1.2. DEFINING BOTNETS AND BOTS

It is difficult to accurately define a botnet [104]. Although it is evident that a botnet is a set of bots, connected to a botmaster, this definition is not satisfactory without the definition of a bot. Communication plays an important role, but the sole ability of malware to connect to other malicious instances is not a sufficient condition to classify an infected computer as a bot. Modern malware is practically always a combination of different components for: infection, attack, concealment, adaption, and communication. Not only popular media, but even scientific literature, often refers to the same malware instances with different terms, such as: virus, root kit, backdoor, RAT, or trojan. The

choice of the name often depends on a particular property or component that is highlighted by the referrer. Despite efforts to classify malware or attack classes in a taxonomy [69][63], there is no accepted uniform definition of a bot. In our work, instead of giving an exact definition of a bot, we present two basic conditions that must be met, to classify an infected computer as a bot:

1. *Remotely controllable:* A bot is remotely controllable. The communication channel for the remote control is designed as an infrastructure that can enduringly connect a large group of bots with each other and/or the botmaster. Depending on the obtained privileges, local files and hardware-related resources can become remotely accessible or controllable by the botmaster.

2. *Adaptable:* The bot has built-in functionality to change its settings, functionality, and program code. The botmaster can control the adaption by the C&C channel. This can include downloads of new settings and/or malware. Examples of adaptions are: addition of a specific attack capability, modification of the C&C communication, deployment of new countermeasures against specific types of detection, and complete removal of the malware and its traces.

Both conditions require specific pre-built provisions in the malware of any bot. If the involved malware lacks these provisions, the infected computer is not a bot. Hence a computer, infected by a simple backdoor or spyware agent that connects to a remote receiver is not a bot, since it cannot be adapted. A computer infected by a polymorphic virus is not a bot if there is no remote control after infection.

Both conditions also constitute the key facilities that turn a botnet into a universal tool for cybercrime.

1.1.3. THE EVOLUTION OF BOTNETS

The first botnets saw the daylight in 1989 with the introduction of IRC (Internet Relay Chat) and were intended for benign IRC control and usage [101]. In 1998 GTBot was one of the first popular malicious botnets, used for DDoS attacks. Its C&C infrastructure was based on IRC. It was followed by many variants that all used IRC or the HTTP protocol as C&C [8]. In addition to DDoS, other botnet applications followed, most notably, the distribution of spam [45]. Early botnets that were used for distribution of spam were Agobot and Bobax, around 2003 [105]. During the last decade, in addition of DDoS and spam, botnets were deployed for all types of attacks that require some type of collusion, such as: click fraud, phishing, and cycle stealing. In 2013 a new and emerging application, based on cycle stealing, is the mining of bitcoins by botnets[100].

All mentioned attacks have in common that the effectiveness is closely related with collusion between a large number of bots in a botnet. Collusion transforms a botnet into a computer grid, containing computers of different owners that work together to solve a common problem [24]. Unfortunately, in this case the common problem is a criminal task. Grid computing can be relatively cheap and highly scalable compared to centralized computing. If implemented correctly, an increase of the number of nodes will result in approximately an equal growth of computing power, bandwidth and/or storage. A huge grid with massive computing capabilities can rapidly be established by

the mentioned scalability of grid computing, combined with the exponential growth of malware infections by viruses or worms. This is supported by discoveries of very large botnets, with thousands to millions of computers, committed to Internet-related criminal activities. In 2010, the Conficker Working Group tracked more than 2 million different IP-addresses, belonging to bots in the Conficker botnet. In the same year, the National High Tech Crime Unit of the Netherlands' Police Agency (NHTCU) estimated the size of Bredolab, a botnet that was taken down, around three million [34]. Of course exact figures about botnet sizes are in some cases questionable [104] but convictions of botnet-related crime, like DDoS-related extortion and spam, show the significance of large botnets [34][55]. Several incidents also reveal that in conflicts, or situations of political tension, large botnets are used for terroristic attacks and warfare. A well-known example is the DDoS attack, launched from a massive number of bots, that targeted Estonian institutions for two weeks in 2007, during a period in which there was a strained relationship with Russia. Many similar incidents are reported from regions with war or political tension, like Korea, Georgia, West Bank, and Syria [126].

Botnets can be used for data theft. This can concern data that is directly stored on the infected computer, or captured from connected I/O devices, such as keyboard and screen. But also other data is vulnerable, if it is indirectly accessible from resources to which the bot has special access, such as other computers that are located in the same network. In addition to theft, data access also enables manipulation or destruction of information.

In case of botnets that target consumer computers, the stolen or manipulated data is often used for identity fraud and related bank fraud. In case of botnets that target enterprise networks, the objective is often espionage, data manipulation and sabotage. Not only criminal organizations are behind such attacks. There is growing evidence that governments use botnets systematically for cyber espionage [12][36][41][84].

In 2009 Ghostnet infected more than thousand computers worldwide of which 30% resided in government institutions, including governmental offices. Ghostnet bots were capable of searching and transferring files, and covertly operating devices such as webcams and microphones [36].

In 2010 Stuxnet sabotaged nuclear installations in Iran. Research revealed extremely sophisticated malware that could control industrial PLCs. Although often referred to as a worm, because of its advanced spreading mechanism, Stuxnet includes also bot functionality with C&C for information upload and updates. It is believed that foreign authorities were behind the attack, because of the advanced design of the malware [42].

In 2010 Google announced that it was infiltrated by malware that had stolen intellectual property and had targeted Gmail accounts of Chinese activists. The responsible malware was later named *Operation Aurora* by McAfee and research revealed sophisticated malware that used C&C. According to Google at least twenty other companies were also attacked [41][84].

Revelations about the NSA, by Snowden in 2013, include information about the infiltration of a large number of computer networks around the world by the U.S. government. In some cases malware was used with C&C functionality for control and transport of the harvested information [12].

1.1.4. BOTNET COUNTERMEASURES

Many countermeasures have been deployed against botnets. Apparently there has been no decisive countermeasure, because botnets are since many years the most important instrument of Internet related crime and massive numbers of computers are part of botnets. This is illustrated by many publications, such as:

- Research, conducted in 2009 among ISPs, in the Netherlands revealed that between 5% and 10% of all connected computers were part of a botnet [128].
- At the end 2013 of Microsoft announced that it had taken down the ZeroAccess botnet, in a cooperative effort with authorities and industrial partners. The bot consisted of several million bots. This was the eighth botnet takedown by Microsoft in the last three years [14].
- Symantec reported in 2012 that worldwide there is still a spam rate of 69% of all mail. Spam is almost exclusively sent by botnets. The spam rate is declining slightly since 2011, but targeted attacks and espionage increased by 42% [122].

In line with these publications, the continuous media attention also indicates that botnets are still a real and unsolved threat.

An important success factor of botnets is the high survivability. Since botnets are used for criminal activities, a botmaster will continuously anticipate the high risk of repressive actions. Like any critical infrastructure the botnet survivability is increased by redundancy and diversity. For botnet C&C communications, typical techniques that increase the survivability are IP fast-flux techniques [65] and peer-to-peer C&C between bots [66]. In contrast to legitimate infrastructures, invisibility is an important factor in botnet survivability, because repression can only take place after detection. A botnet has various types of activities that are eligible for detection, such as: the infection, the subsequent host activity, C&C communication, attacks, and money flow. Advanced stealth technology is at the disposal of botnets to stay below the radar of Anti-Virus solutions and Intrusion Detection Systems. This includes techniques, such as: code morphing, encryption, steganography, segmentation of botnets, tampering of detection systems, and the use of popular protocols and well known services [113]. Fortunately, the invisibility of a botnet has practical limitations. Important causes that limit the invisibility are:

- *attack traffic:* Traffic, such as DDos-traffic or spam give opportunity to discover bots;
- *other survivability measures:* Techniques as IP fast-flux and peer-to-peer can result in detectable anomalies;
- *malware installation:* during the initial installation of the malware a bot can be visible, because some bot-related stealth techniques, are not effective until after complete installation.
- *limited resources:* the effort, invested in invisibility by the increased complexity of program code, communication, and maintenance, must be covered by the botnet revenue.

Not all invisibility restrictions are always applicable. The level of invisibility depends on the type of botnet and its deployment.

1.2. THE ENTERPRISE NETWORK

Computer networks of organizations are often referred to as *enterprise networks*. An *enterprise network* comprises a generic model of a computer network that is exclusively used by a private or public organization. In addition to communication media and network components, it can include a large variety of end systems or computers, such as: desktop computers, servers, mainframes, industrial equipment, and connected (mobile) devices. There is one common administration. The size can range from a SOHO (Small Office Home Office) network with only one or a few computers in a LAN, to a large network with many LANs, optionally distributed over several locations and connected by WANs as private or virtual private networks. Large network technology corporations have defined enterprise networks in reference models, such the Enterprise Campus 3.0 Architecture of Cisco [25] and the FlexNetwork Architecture of HP [64].

We oppose *enterprise networks* to *public networks* that deliver connectivity to home networks and mobile consumer devices. The computers in a home network are typically: PCs, laptops, tablets, game consoles, and all types of embedded devices for personal use at home. It also includes smart phones for private use, connected by home WIFI.

In contrast to the public networks, enterprise networks have a common administration that includes control over the end systems and communication. Enterprise networks are isolated from public networks by well defined perimeters with controlled access, normally implemented by firewalls. Inside an enterprise network, the end systems are confined in compartments with different communication policies. For example a public accessible internet server is located in a Demilitarized Zone, or DMZ. Client computers are located in separate compartments with stateful traffic inspection, that only allows traffic if it is initiated by the clients.

There are small networks with properties of both enterprise and home networks, but in our problem statement we will use the simple classification of enterprise networks versus public networks with connected home networks and consumer devices. To make the distinction as clear as possible we clarify some special cases that can give rise to confusion:

- *Public LAN in a corporate network*: Some enterprise networks deliver public internet access to visitors or employees. If this part of the network is well confined, it can be seen as a separate public network that is not a logical part of the enterprise network.

- *Cloud computing*: Many components of the corporate network can be virtualized and remotely delivered by cloud services. This includes complete parts of the infrastructure, (Infrastructure As A Service). Although located on a different physical location, the virtual infrastructure, offered by the cloud can logically be seen as an integral part of the enterprise network of the client.

- *Mobile devices*: The trend of BYOD (Bring Your Own Device) introduces the se-

curity risk that mobile consumer devices become de facto part of the corporate network. This is prevented by allowing the complete consumer device only in a public compartment, as a visitor. In addition the device can be logically divided into in a confined secure workspace that is part of the enterprise network and an isolated consumer workspace that is connected with a public compartment of the network.

- *ISP*: Internet Service Providers have a public network for their clients and an enterprise network for management and control of the public network.

1.3. PROBLEM STATEMENT

We focus on network-based detection of bots in Internet-connected enterprise networks. With regard to the botnet threat and botnet detection, enterprise networks differ in several ways from public networks:

- *Attack type:* As described in Section 1.1.3, there are many incidents that indicate that botnets are systematically used for espionage (including information theft) and sabotage in organizations. The attacks are commercially or politically motivated. Enterprise computers, infected with malware that is designed to infiltrate an organization for stealing or modifying information, are usually part of a botnet, because:
  - A botnet provides the attacker scalable simultaneous control over many instances in multiple infiltrated organizations.
  - Bots are adaptable to specific encountered situations or sudden changes.
  - A carefully designed C&C infrastructure can contribute to the invisibility of the malicious communication.

  Since the effectiveness of espionage and sabotage depends highly on invisibility, these bots are normally not used for other more common bot activities, such as spam or DDoS, because this would increase the detection probability by noisy attack traffic.

- *Revenue per bot:* The average revenue of a specialized bot for espionage or sabotage in an enterprise network is obviously much higher than the average revenue of a generic DoS, or spam on a consumer network. The botmaster will put much effort in invisibility, to protect the organization-infiltrated profitable bot and assure its successful and prolonged operation. In addition to mentioned absence of attack traffic, the invisibility is further improved by maximum control over the infected computers to compromise any type of host-based detection, and discrete low volume C&C traffic.

- *Detection and repression environment:* The potential presence of sophisticated botnets for espionage and sabotage within the enterprise premises is regarded by many organizations as an Advanced and Persistent Threat (APT)[32][99]. As a potential victim, organizations will be highly motivated to respond to this risk with

comprehensive countermeasures, despite the associated considerable costs. This opposes the defense against bots on consumer computers. An average consumer has very limited skills and resources to detect a sophisticated botnet infection. [1]

The expected high level of invisibility of specialized bots in an enterprise network and the difference in detection environment give opportunity to specialized bot detection that anticipates specific properties of enterprise networks in addition to generic detection approaches.

To our knowledge, despite the existence of many proposed and implemented countermeasures, there is little research on botnet detection measures that anticipate the described specific environment of enterprise networks. This thesis addresses this knowledge gap. Our research focuses on *network-based detection* of *C&C traffic* in *enterprise networks*, because:

- Detection is a crucial component of all reactive countermeasures.
- All bots require C&C communication.
- C&C traffic detection in an enterprise network can directly identify infiltrated bots.
- Network-based detection has a low dependency on end systems and a low exposure to malware.

1.4. RESEARCH OBJECTIVES AND SUBSEQUENT RESEARCH ACTIVITIES

Our research aims for new network-based C&C detection approaches that anticipate specific properties of enterprise networks. We will not only consider properties or features that are used as direct input for classification of network traffic, but also other properties that are conditional for successful detection. The research is decomposed in three research objectives:

I Identify generic properties that are used in existing network-based C&C detection approaches and identify specific properties of enterprise networks that may lead to new detection approaches.

II Propose new network C&C detection approaches that anticipate on one or more of the identified enterprise network-specific properties.

[1] Some experts and authorities suggest that public ISPs, that connect consumer devices with the Internet, should play a more active role in countering botnets, despite the fact that they are not the primary victim. However the counter possibilities of public ISPs in the consumer network are limited because:

- An ISP's primary goal is the make money. An average private customer is not prepared to pay significantly more for advanced botnet countermeasures.
- Legislation on privacy and net-neutrality limit respectively the observation and policing of private traffic by a public ISP.
- A public ISP has no direct control over the connected end systems.

III Estimate the detection performance of these new approaches.

Adhering to these objectives and the addressed knowledge gap in Section 1.3, three consecutive research questions are formulated:

1. What are distinctive properties of existing network-based C&C detection approaches and which enterprise-specific properties are potentially suitable for new detection approaches?
   Research Activities:
   (a) Exploration of existing representative C&C detection approaches, including the relevant properties that are used or conditional for successful detection.
   (b) Identification of properties that are specific for enterprise networks and that can lead to new detection opportunities.
2. How can botnets be detected in an enterprise network, anticipating the identified new characteristic properties of question 1b?
   Research Activities:
   (a) Design of new detection approaches, based on the identified properties of Activity 1b.
   (b) Compare the new approaches with existing approaches as identified in Activity 1a.
3. What detection performance is expected from the newly identified detection approaches of question 2?
   Research Activities:
   (a) Identify the most important parameters that express the performance of C&C detection.
   (b) C&C detection modeling and error-related prediction of the approaches, as proposed in Activity 2a.
   (c) Experimental evaluation of the predicted performance, by testing the detection approaches of Activity 2a

1.4.1. CONTRIBUTIONS

The contributions of this thesis are summarized as follows:

1. We developed an ontology-based faceted classification to systematically identify and compare botnet countermeasures and botnet traffic detection approaches.
2. We identified inter-traffic causality, causality between user activity and traffic, the trustworthiness of traffic destinations, and DNS domain degree distribution, as new properties that can be used for botnet C&C anomaly detection.
3. We developed the CITRIC (CITRIC = Causal Inspection To Recognize Illegal Communication). CITRIC is a framework that analyzes and detects hidden botnet C&C

communication by building causal relationships between traffic flows, prior traffic flows, and user activity. As a part of the analysis CITRIC can produce Traffic Flow Causality Graphs, for manual evaluation of complex communication dialogues.

4. We proposed and evaluated TFC-detection. TFC is the acronym for Traffic Flow Causality. Botnet C&C traffic is detected by the observation of direct causes of traffic flows.

5. We proposed and evaluated UDI-detection. UDI is the acronym for Untrusted Destination by Identifier. It detects botnet C&C communication by determining the trustworthiness of contacted destinations by their identifier, such as IP-addresses and domain names.

6. We proposed and evaluated a DNS-based detection approach, based on anomalies in the degree distribution of domains, derived from DNS traffic in one or multiple enterprise networks.

1.5. THESIS OUTLINE

The remainder of this thesis is structured by the research questions and activities of Section 1.3, as shown in Table 1.1.

Table 1.1: Relation between the thesis content and the research questions and activities.

Chap.	Outline	Research Question & Activities
1	Introduction	definition of the research
2	Botnet detection in enterprise networks	1a, 1b, 3a
3	Detection by user activity	2a, 2b, 3b, 3c
4	Detection by causal relationships	2a, 2b, 3b, 3c
5	Detection by trusted destinations	2a, 2b, 3b, 3c
6	Detection by DNS distribution anomalies	2a, 2b, 3b, 3c
7	Concluding remarks	evaluation of the research

Chapter 2 covers existing network-based botnet detection approaches and identifies potential properties that can be used in new detection approaches in enterprise networks. New potential detection approaches, based on the identified properties, are derived.

Chapter 3, 4, 5, and 6 elaborate the derived anomaly detection approaches of Chapter 2 by presenting detector designs, modeling detection performance, evaluating through experiments, and discussing limitations. Chapter 3 explores detection by associating network traffic with user activity. Chapter 4 completes this concept by including traffic events in a complete causal analysis of network traffic flows. CITRIC, a framework, we developed for demonstration and evaluation of the detection approach, is presented. Chapter 5 elaborates UDI-detection. This approach identifies anomalous contacted destinations by estimation of the destination trustworthiness, derived from the destination identifier. A modified version of CITRIC is used for experimental evaluation.

Chapter 6 elaborates DNS-based detection of C&C detection in enterprise networks and presents detection by anomalies in the degree distribution of visited domains. The Chapters 3, 4, 5, and 6 take the format of separate research papers, since they are earlier published in proceedings of peer reviewed conferences. We have made adjustments and additions to the papers, to fit them as consistent chapters in this thesis.

Chapter 7 concludes the thesis by a retrospective evaluation of the complete research and the contributions of the thesis, including an overview of found limitations, and possible directions for future works.

2

COUNTERING BOTNETS BY THE DETECTION OF C&C TRAFFIC IN ENTERPRISE NETWORKS

This chapter starts with the introduction of the fundamentals of NIDSs (Network Intrusion Detection Systems), including the most important criteria to evaluate different detection approaches. However this knowledge is not enough for finding new detection approaches that anticipate specific properties of enterprise networks, because many choices are involved, not only concerning detection, but the complete countermeasure of which detection is a component.

Since there are no decisive countermeasures against botnets, multiple diverse countermeasures must be taken to establish a layered security defense. To assess the diversity by the most important similarities and differences between botnet countermeasures, a generic ontology-based faceted classification is presented. The classification system can be used as an instrument to select countermeasures components in a layered security defense, by their diversity and appropriateness for a specific environment.

Since our research focuses on the detection of botnet traffic, we extend the presented generic classification of botnet countermeasures with detection-specific facets that cover: measured features, knowledge, and state. We evaluate detection of sophisticated espionage bots in enterprise networks by these three facets. With the results we derive new detection approaches that anticipate on specific properties of enterprise networks and complement existing detection approaches.

2.1. NETWORK INTRUSION DETECTION

Detection of botnet traffic can be seen as a special type of intrusion detection that discerns normal traffic from malicious traffic. Detection, based on the observation of traffic features, is often referred to as network-based intrusion detection or NIDS. In this section we discuss common approaches of network-based intrusion detection. A generic detection model is presented in Figure 2.1. Traffic samples, such as IP packets, or flows, are fed to the detector. Relevant properties, referred to as features, are extracted from the traffic samples. The features are submitted as feature vectors to the classification process. The classification process uses knowledge to make the decision. The decision is binary, since one class is the intrusion traffic or malicious traffic, and the other class is the normal or legitimate traffic. In the case of botnets the malicious traffic can be divided in C&C traffic and attack traffic. Examples of attack traffic are: port-scans, spam, and DDoS attacks.

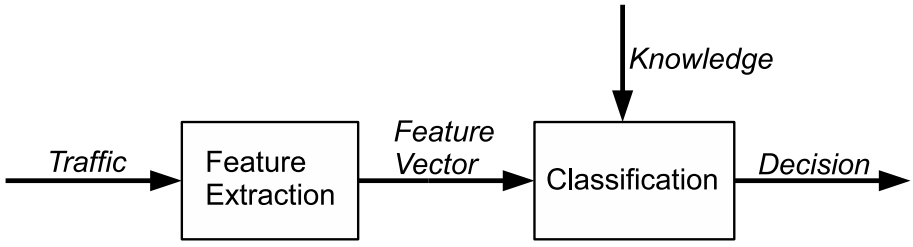

Figure 2.1: Generic model of a detector.

2.1.1. FEATURE EXTRACTION

In the case of network intrusion detection the data samples typically consist of IP packets. The accuracy of the classification depends strongly on the selection of appropriate features. Especially in the case of stealth Command and Control communication, it is a challenge to identify such features. The large number and the high diversity of features, proposed in literature, make it difficult to compare detection approaches. Examples of features are: packet rate, flow size, IP addresses, entropy, time, flow direction, flags, the presence or absence of payload strings patterns, port numbers, protocols, time stamp, time interval, etc.

In literature several ad hoc classifications of traffic features are presented, often introduced to explain the mechanism of a particular detection approach. Frequently the classification is only appropriate for this specific type of detection [49]. An example of a taxonomy that is not constrained to a particular detection approach, is the taxonomy of traffic features, proposed by Onut et al. [96]. This taxonomy starts by discerning basic features of a single packet, from derived features, extracted from multiple packets. It further differentiates the derived features by the way how packets are selected for the derived features. Selection criteria are: IP-address, traffic direction, time interval, and connection. With 26 classes of features, the resulting taxonomy is detailed. To provide more overview, we propose a simpler classification of traffic-related features that only distinguishes three classes of features:

- *Header(L2, L3, L4)*: The features are values of fields in the headers of the datalink (L2), network(L3) and transport(L4) layer. Examples are: source IP-address, Protocol (TCP/UDP), destination Port. The parsing of these features is normally done directly per packet.

- *Application data*: The features are parsed from the traffic payload. This includes formal fields of the application layer. Examples are string patterns, DNS replies and HTTP Cookies. The parsing of these features is normally done per flow or per connection.

- *Metadata*: The features are not directly parsed from headers or payloads, but indirectly determined. Examples are time-stamps, momentary packet size, byte count per flow, packet count per flow, entropy of a message, average packet size, etc. Metadata often implies aggregation of packets in flows.

2.1.2. KNOWLEDGE

The knowledge, needed for classification, can be obtained and represented in many ways. If all possible data instances are known, a set of reference vectors can be constructed, representing the knowledge. In such a case, classification consists of calculating the distances between the input vector and each of the reference vectors by correlation and subsequently determining the shortest distance, as shown in Figure 2.2. In practice, by the diversity of both normal and malicious traffic, it is very difficult to determine a set of reference vectors that completely covers all normal and malicious traffic. Since intrusion detection involves a binary classification, it is sufficient to have

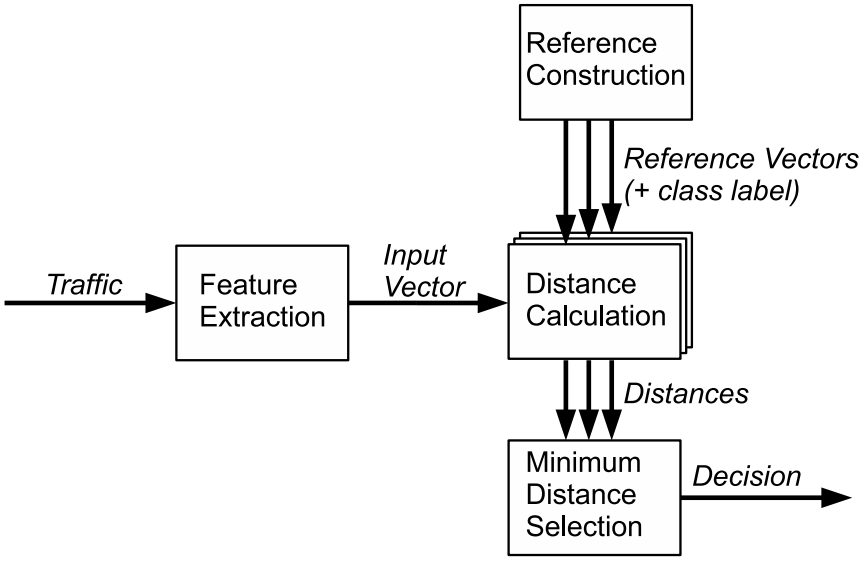

Figure 2.2: Detection with known reference vectors.

knowledge of only one of the two traffic classes. If the distance between the input vector

and reference vector is within a predefined range the known class is chosen. Otherwise the alternative or reject class is chosen. This results in two basic binary detection approaches with regard to knowledge: *Misuse detection* and *Anomaly detection*:

Misuse detection uses only knowledge of the attack, represented by specific signatures of known attacks. If traffic does not match the signatures, it is automatically classified in the reject class as normal. Examples of attack signatures are data patterns in payloads but also blacklisted ranges of IP-addresses, domain names, and ports. Signatures are often implemented as rules that allow for a more versatile, and complex matching and the inclusion of specific actions if a match occurs. This type of detection is often referred to as: misuse detection, signature detection, or rule-based detection. Figure 2.3 shows the basic setup. The use of intermediate results of prior traffic instances is modeled with

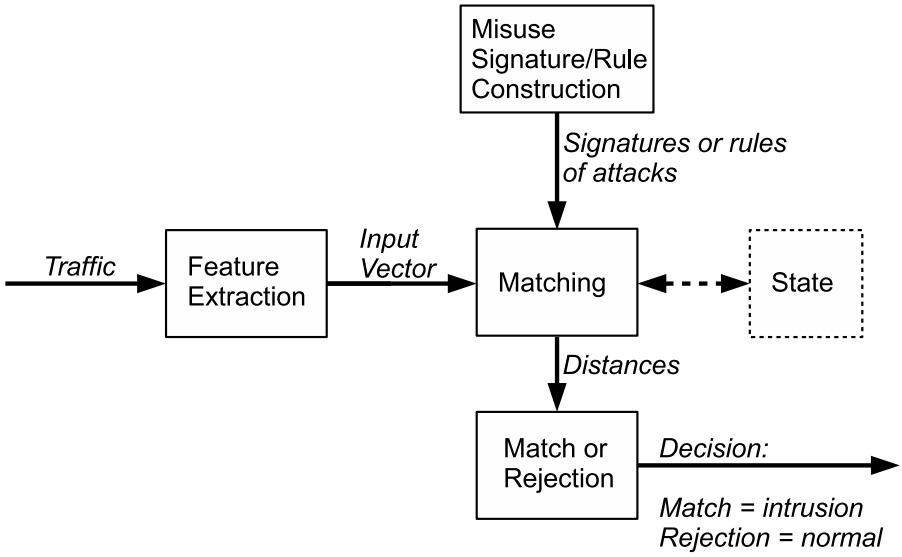

Figure 2.3: Misuse Detection.

optional state. An example is the SSL state of a connection, as implemented in Snort [109]. If a proper signature is present, misuse detection can accurately detect a known attack. This shows also the weak site of misuse detection: In case of new or unknown malicious traffic there are no signatures present, which results in classification as normal traffic. To minimize the risk of missing such zero-day malicious traffic, the database of signatures or rules needs to be continuously updated, anticipating the newest types of malicious traffic.

Anomaly detection discerns normal traffic from malicious traffic by knowledge of normal traffic. All traffic that does not match the normal profile, it is placed in the reject class and labeled as anomalous. Literature describes three basic techniques: *statistical, specification based* and *machine learning* [37][48][49][79].

- *Statistical*: A baseline of normal traffic is measured and statistical properties are stored as a normal profile. New traffic is evaluated on statistical properties and an anomaly score is calculated from its distance from the normal profile. The involved type of traffic features is metadata. The implementation is often a decision, or decision tree, based on threshold comparison. If the distance exceeds a certain threshold, the data sample is classified as anomalous. A method to determine the threshold by the first and second moment of the measured distances, is Chebyshev's inequality, as proposed by Denning and shown in Equation 2.1 [37].

$$P(|X - \bar{x}| \geq n.\sigma) \leq \frac{1}{n^2} \tag{2.1}$$

  $\bar{x}$ is the first moment, σ the second moment, and n a real number greater than 1. The formula gives an upper bound of the probability that the value of a stochastic variable deviates more than $n\sigma$ of $\bar{x}$.

- *Specification-based*: If the normal behavior is limited to a small number of possibilities, the normal profile can be described by rules and optional states. It works on the mechanism of least principle by classifying all traffic as anomalous if it does not comply with the predefined rules [49]. This is especially useful to detect deviations from known policies or protocols. The knowledge is typically manually constructed by an expert [48]. The implementation consists often of one or more binary decisions, represented by rules that are defined by an expert. Figure 2.4 shows a basic scheme of specification based detection.

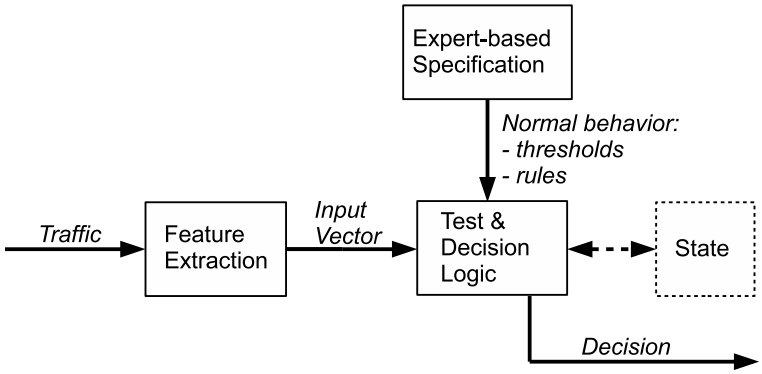

Figure 2.4: Anomaly Detection by predefined specifications.

- *Machine-learning*: The normal profile is automatically learned by a machine learning algorithm. In the case of supervised machine learning, the detector learns by a labeled training set of known normal and anomalous traffic. The training with labeled anomalous traffic makes the knowledge of supervised machine learning not only anomaly-based, but also partly misuse-based. Examples of machine learning algorithms are: Bayesian networks, neural networks, and artificial immune systems. Figure 2.5 shows a supervised machine learning detector. In the case of unsupervised machine learning, the detector will detect anomalies by identifying outliers in data by clustering algorithms such as K-means [70].

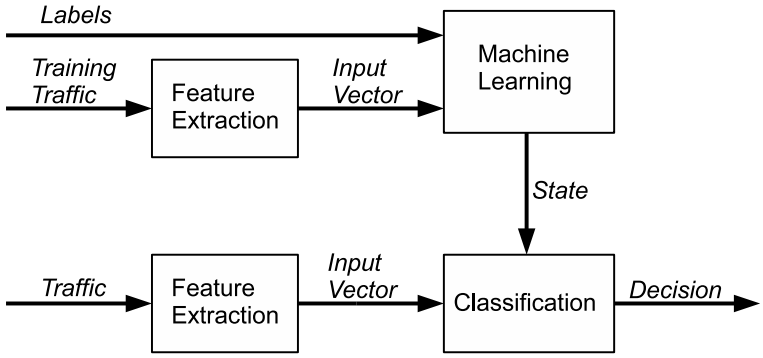

Figure 2.5: Anomaly Detection with supervised machine learning.

2.2. EVALUATION OF INTRUSION DETECTION METHODS

Intrusion Detection methods can be assessed on many evaluation criteria. The performance with respect to correct classification is one of the most important criteria. It will be elaborated in Section 2.2.1. Evasion, decision speed, exposure, and implementation complexity are relevant additional criteria which are often evaluated in literature concerning detection proposals. These will be elaborated in Section 2.2.2.

2.2.1. DETECTION PERFORMANCE

A network intrusion detector classifies traffic instances in two sets: malicious and normal traffic. In practice this classification will not always be correct. In some cases malicious traffic will be erroneously classified as normal traffic and vice versa. Since a binary intrusion detector classifies in either malicious or normal classes, and since the decisions can be either true or false, there are 4 potential outcomes. This is illustrated as a Venn diagram [129] in Figure 2.6.

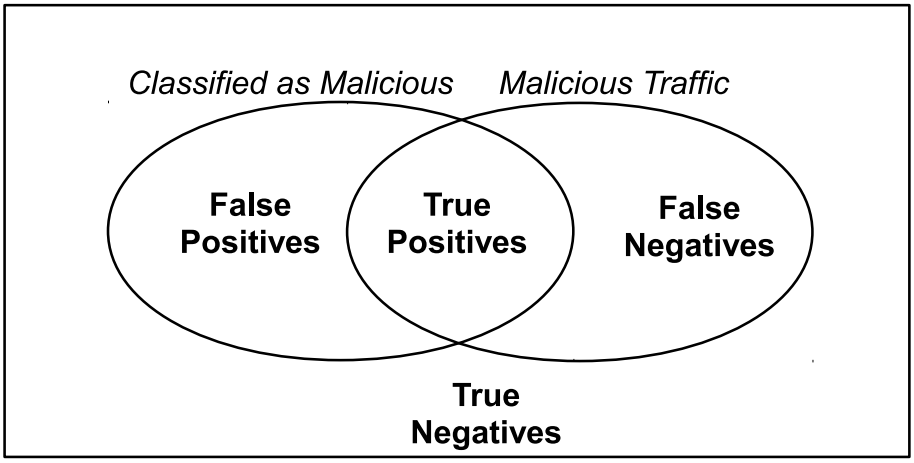

Figure 2.6: Venn diagram of binary intrusion detection with the actual and detected attacks.

The diagram represents the four potential outcomes as sets:

- *True Negatives*: The observed data instance does not belong to malicious traffic and the detector classifies the instance correctly as normal.
- *True Positives*: The observed data instance belongs to malicious traffic and the detector classifies the instance correctly as malicious.
- *False Negatives*: The observed data instance belongs to malicious traffic and the detector classifies the instance incorrectly as normal.
- *False Positives*: The observed data instance does not belong to malicious traffic and the detector classifies the instance incorrectly as malicious.

Another way to represent this is a confusion matrix as shown in Table 2.1 [78]. The error-

Table 2.1: Confusion matrix

		Detection Decision	
		Malicious (=POS)	Normal (=NEG)
Actual class	Malicious (=POS)	TP	FN
	Normal (=NEG)	FP	TN

related performance of the detection can be expressed as conditional probabilities that depend on the actual situation. The most popular probabilities are:

1. True Positive Rate (TPR)
   Also known in literature as Detection Rate (DR), Sensitivity, Recall, Power, and Hit Rate. As shown in Equation 2.2, the TPR equals the ratio of correctly detected malicious traffic events to all malicious events. Ideally the TPR=1.

$$TPR = \frac{\#TP}{\#TP + \#FN} = P(POS|Malicious) \tag{2.2}$$

2. Precision
   This is the ratio of correctly classified malicious traffic to all positives. Ideally the Precision=1 (Equation 2.3).

$$Precision = \frac{\#TP}{\#TP + \#FP} = P(Malicious|POS) \tag{2.3}$$

3. False Positive Rate (FPR)
   The FPR is also known as the False Alarm Rate (FAR) or Significance Level. It is the ratio of normal traffic events, erroneously classified as malicious, to all normal traffic. Ideally the FPR=0 (Equation 2.4).

$$FPR = \frac{\#FP}{\#TN + \#FP} = P(POS|Normal) \tag{2.4}$$

4. True Negative Rate (TNR)
   Also known in literature as Specificity (SPC). TNR equals 1-FPR, hence it provides the same information as FPR (Equation 2.5).

$$TNR = \frac{\#TN}{\#TN + \#FP} = P(NEG|Normal) \tag{2.5}$$

In many areas Recall (a synonym for TPR) and Precision are popular classification benchmarks. A related benchmark is F-Measure, which is the harmonic mean of Precision and TPR. A disadvantage of Precision (and F-Measure) is its dependency on the fraction of malicious traffic in the complete observation set. A high concentration of malicious traffic will result in a relatively high Precision, because the number of FP is relatively small in such a case. In the case of intrusion detection the fraction of malicious traffic in a complete observation set can vary significantly, depending on the network and observation time. Therefore in this thesis we will use another parameter or benchmark for the

analysis of proposed detectors that is also popular: FPR. We will use it together with DR (also a synonym for TPR that is commonly used in detector descriptions) The influence of detector settings on the DR and FPR can be very complex. ROC diagrams (Receiver Operating Characteristic diagrams) [43] are a simple means to evaluate the influence of a parameter on DR and FPR. As shown in Figure 2.7 an ROC diagram plots for a specific setting the DR and FPR. Perfect detection is positioned in the upper left corner.

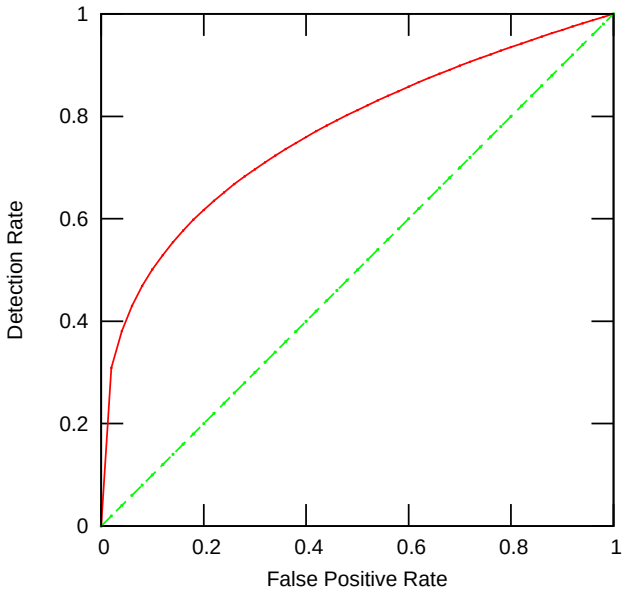

Figure 2.7: Example of an ROC diagram.

The straight line with FPR=DR marks completely random detection. In practice, detection will always have curves above this line. The FPR scale is often logarithmic, because practical values of FPR are normally very small. A small value of FPR is important, to prevent an excessive number of False Positives, as the vast majority of the analyzed traffic will be normal traffic.

2.2.2. OTHER EVALUATION CRITERIA

In addition to accuracy, other important evaluation criteria are evasion, decision speed, exposure, and implementation complexity.

Evasion

If the adversary is aware of an intrusion detection system, he can take countermeasures to evade or circumvent the detection system. The most important evasion techniques are:

1. Encryption of the C&C traffic
2. High and random delays between malicious traffic

3. Flow perturbation, including noise injection
4. Restriction of attack traffic
5. Source IP-address churn
6. Destination flux
7. Imitation of popular traffic

The first 5 techniques are derived from work of Stinson et al [119]. They systematically evaluated the evadability of bot and botnet detection methods and the amount of effort involved. Evasion possibilities of well-known detection approaches are qualitatively assessed by their implementation complexity and effects on utility. We added two additional techniques: destination flux and imitation of popular traffic. Destination flux is the frequent change of the destination of the C&C servers. This can be done in a simple way by DNS updates of the involved A record [65]. Additionally the authorative DNS server can be changed or even the hostname itself [137]. Evasion by imitation of popular traffic includes the use of popular services, such as services that are used by social media.

Decision Speed
The type of repressive action after detection depends on the moment of the detection decision. We distinguish three critical moments that depend on the detector state:

1. Immediate: the detector can decide immediately after capturing the first malicious packet. The malicious communication connection can be completely repressed.
2. After reception of normal traffic: the detector can detect malicious traffic after a state has been built by other normal traffic. The malicious communication connection can be completely repressed.
3. The detector can detect malicious traffic after reception of other malicious traffic. The malicious communication connection can therefore only be partly repressed.

Exposure
In addition to evasion, the adversary can attack the countermeasure itself. The extent to which a countermeasure is visible or reachable by the adversary is often referred to as the exposure surface. Passive listening network-based detectors have a very small exposure surface. However, if detection depends on software agents in potentially bot-infected end systems, the exposure surface will be significantly larger. Virtualization can reduce the exposure surface in end systems to some extent.

Cost
As with all countermeasures, the involved cost, related with its implementation and operation, must normally be lower than the prevented or mitigated damage [80]. In the design stage of a detector it is very difficult to assess quantitatively the cost of detection, but important contributing factors can be identified and qualitatively assessed. Important factors include:

- *Processing and memory resources*: It is evident that deep inspection of individual packets requires more processing and memory resources than the analysis of only aggregated flow attributes. This explains the popularity of flow-based detection systems. Intensive processing and memory requirements can result in high costs and poor scalability that affects the feasibility of the detection approach. In such case the suitability of an approach to be distributed over multiple systems, can improve the feasibility.

- *Knowledge maintenance*: This includes knowledge updates of malicious traffic, by means of new signatures or new training sets. It can also include new knowledge about normal traffic, for example by means of a new baseline.

- *Interception*: This involves the complexity of the data capture process. In general sensors in end systems, for example by software agents, are more complex to install and maintain, then a sensing mirror port on a central switch in the network. In some cases the traffic must be intercepted and decrypted in the network. This will result in processing costs, related with the processor-intensive decryption of many simultaneous flows, but in costs that are related with the installation and maintenance of the trust relationships between the intercepting detector and the observed end systems.

- *Costs of false positives*: A false positive results in subsequent actions that involve costs, related with unnecessary repression or further analysis.

2.3. CLASSIFICATION OF BOTNET COUNTERMEASURES

The discussed properties of network detection systems in Section 2.1, show a complex design space, which comprises many choices. In addition detection is a component of a complete countermeasure that includes other design choices, related to aspects, such as repression and implementation.

Instead of searching for one ultimate decisive countermeasure, a more realistic approach follows the principle of defense in depth and contributes to a layered security architecture, by combining multiple diverse countermeasures that can complement each other. The appropriateness of a countermeasure for a specific environment and the diversity between multiple applied countermeasures are important conditions for success. The complexity of existing botnet countermeasures and inconsistent vocabulary make it difficult to compare countermeasures, discover relevant similarities and differences, and assess its appropriateness for a specific environment. As a solution to this problem we introduce a classification of generic botnet countermeasures that identifies similarities and differences between countermeasures and supports the selection of countermeasures that are appropriate for a specific environment, such as a corporate network, or an ISP (Internet Service Provider). We will present ontology-based faceted classification of botnet countermeasures in Section 2.3.3 after discussing existing classifications of threats and countermeasures in Section 2.3.1 and classification approaches in Section 2.3.2. The Sections 2.3.4, 2.3.5, and 2.3.6 elucidate and motivate the proposed classification. Section 2.3.7 evaluates the model by classifying examples of existing very diverse countermeasures.

2.3.1. EXISTING CLASSIFICATIONS

Several classifications have been proposed to categorize botnets from different perspectives:

- One of the first botnet-related classifications is of Dagon et al. [31]. They propose effectiveness, efficiency, and robustness, to classify botnets by their topology.
- Anti-virus company Trend Micro proposes botnet classification by attacking behavior, C&C-models, rally mechanisms, communication protocols, observable botnet activities, and evasion techniques [86].
- Hachem et al. propose a botnet taxonomy by life cycle. They distinguish the subclasses: propagation and injection, C&C, and Application phases. [61].
- Rodriguez et al. present a survey of botnets by a life cycle-based taxonomy [108].

The relation between botnet properties and countermeasures is often complex and does not directly bring order in existing countermeasures. For example, the same countermeasure can be applied against different classes of botnets and conversely a particular botnet class can be successfully defeated by various classes of countermeasures.

A direct classification of countermeasures has been proposed for other logical cyber security threats. Most notably in the area of DoS-attacks many taxonomies of DoS attack types and DoS countermeasures have been proposed [6][22][89][116]. The proposed DoS taxonomies are closely related to taxonomies of the DoS threat itself. If we analyze

the motivation for a classification of countermeasures, we see the following recurring arguments:

- With a taxonomy of intrusion response systems, Stakhanova et al. aim at a better understanding of the problem, exposure of unexplored areas, and provision of a foundation for organizing research efforts in the field of intrusion response [117].

- Hamadeh constructs a taxonomy of Internet traceback techniques, to provide an overview in this research area [62].

- Weaver et al. propose a taxonomy of worms, based on target discovery, carrier, activation, payload, and attackers. The purpose of the taxonomy is a better understanding of the worm threat [134].

- A taxonomy of countermeasures against worms is described by Brumley et al. [15]. The concept of a design space, that shows degrees of freedom in the design of countermeasures, is used as a motivation of the taxonomy. Fundamental properties and working mechanism classify the countermeasures instead of implementation and specific details.

- Karresand [73] proposes a generic weapon-based taxonomy of malware, claiming the result of more consistent terms to describe the different types of malware.

In the domain of specific countermeasures against botnets we found three classifications in literature:

- Feily et al. present a taxonomy of four different classes of passive network-based botnet detection: misuse-based, anomaly-based, DNS-based, and mining-based [44]. For a clear view of similarities and differences between countermeasures, the classification is limited, because in practice a countermeasure can have properties of more than one class, such as an anomaly-based detector that mines in DNS payloads.

- Zeidanloo et al. classifies botnet detection in honeypots and Intrusion Detection Systems [139]. Intrusion detection is classified in the subclasses misuse detection and anomaly detection. Anomaly detection is classified in host-based and network based active and network-based passive subclasses. The suitability of this classification for determination of similarities and differences between countermeasures is limited in a similar way as with the taxonomy of Feily et al.

- A more detailed taxonomy of botnet detection and defense approaches is proposed by Khattak et al. [76]. The strictly unfaceted hierarchical structure of the proposed taxonomies results in graphs with many repeating classes. In some parts of the graphs, classes are not consistently repeated. This makes the classification difficult to use and extend for diversity analysis and synthesis.

2.3.2. TAXONOMY VERSUS ONTOLOGY-BASED FACETED CLASSIFICATION

The presented work in Section 2.3.1 shows that classification of cyber threats and countermeasures is a common approach for structuring knowledge. The classifications of the presented work use often the word *taxonomy* as a synonym for the resulting classification. We will use the word *taxonomy* restrictively, to refer to a specific classification strategy, and oppose it against another strategy to which we will refer as *ontology-based faceted classification.*

- We define a taxonomy as a hierarchical model of classes, ordered by inheritance and represented by a rooted DAG (Directed Acyclic Graph) with classes as vertices and inheritance relationships as edges. An object is classified by identifying exactly the class in the taxonomy from which it is instantiated. The rooted DAG structure allows for an easy stepwise selection of the appropriate class, starting from the root. Taxonomies have two limitations: They only model by a *"is-a"* relationship and it cannot efficiently model independent properties. For example if we want to differentiate by a taxonomy DAG between van cars and sedan cars, but also between diesel cars and petrol cars, the subclasses diesel car and petrol car are repeated as child classes in both the van and sedan parents. This is shown in the left diagram of Figure 2.8. One way to avoid this repetition is the modeling of diesel and petrol as atomic properties in the car-class, but this prevents further differentiation of cars by properties that depend on fuel types.

- A better classification strategy in this case is an *ontology-based faceted* classification. Gruber et al. defines an *ontology* as "an explicit specification of a conceptualization" [56]. Ontologies can have a hierarchical structure, similar to taxonomies, but class associations are not restricted to an *"is-a"* relationship [51]. In addition an ontology-based model can contain multiple hierarchical trees as *facets* that together classify the object [94]. With a proper choice of highly independent facets, each facet can directly identify a significant difference or similarity between countermeasures and a well chosen facet can provide information about the extent to which a countermeasure fits in a specific environment. In addition, an accompanying short notation can support a fast overview of the differences.

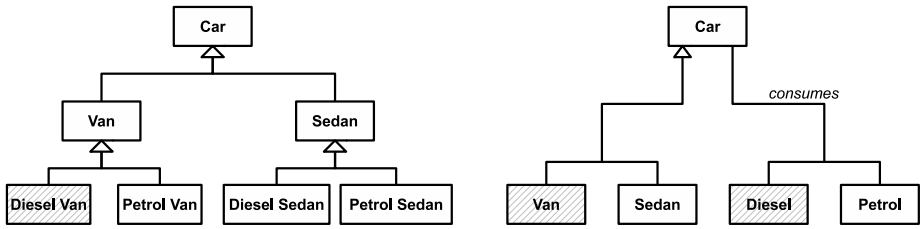

Figure 2.8: Comparison between taxonomy-based classification (left) and ontology-based faceted classification (right) of a diesel van.

In a taxonomy a countermeasure is identified by exactly one class. In a faceted classification, a countermeasure is identified by multiple classes of different facets that together

model the specific countermeasure. This allows for efficient and precise modeling of independent properties. In an ontology-based classification the edges of the DAG are not restricted to only *"is-a"* relationships, but can also represent specific semantic relationships. For our car example we could model the subclasses van and sedan by a *"is-a"* relationship. In addition to this facet, diesel and petrol are subclasses that belong to another facet, semantically expressed by the verb *"consumes"*. A specific car is identified by both facets, for example a Van/Diesel. This is shown in the right diagram of Figure 2.8. The result is a more compact classification without the repetition of classes.

2.3.3. A FACETED ONTOLOGY-BASED CLASSIFICATION OF BOTNET COUNTERMEASURES

Inspired by the discussed classifications in the existing work we present a new classification system for botnet countermeasure classification to identify similarities and differences between countermeasures and select countermeasures for a specific environment. Our classification meets the following requirements:

1. *diversity support*: The classification directly identifies differences and similarities between botnet countermeasures to evaluate diversity in combinations of countermeasures.

2. *environment specific*: the classification facilitates the selection of botnet countermeasures for a specific environment or situation, such as an enterprise network;

3. *minimal ambiguity*: All botnet countermeasures are classified by a simple process with the use of a consistent vocabulary that results in minimum ambiguity.

4. *extendable*: the classification is extendable with additional more refined classifications, to evaluate diversity between botnet countermeasures in more detail.

To meet the requirement of diversity support and minimal ambiguity we do not use a strict taxonomy but a faceted ontology-based classification as introduced earlier. The risk of ambiguity is decreased by choosing facets with easy-distinguishable classes and the representation of facets with all their classes in one graph. Environment-specific selection of countermeasures is supported by facets that identify the implementation and the object or activity that is countered. The ontology-based diagram model allows for further faceted classifications of a class. This makes the classification extendable.

Figure 2.9 presents the proposed ontology of botnet-related countermeasures as a UML class diagram. Although not primarily intended for representing ontologies, UML class diagrams have the advantage of being well-known in the Computer Science community. UML class diagrams have been proposed as a representation for ontologies by Cranefield et al. [28]. To limit the complexity of the diagram, we do not express cardinality or object constraints. The root of the ontology classifies botnet countermeasures by three important facets that identify: the *object* or *activity* that is countered, the *place* where the countermeasure is implemented, and the *moment* ot *time* of intervention. A countermeasure is classified by the combination of appropriate classes for each of the three facets. For example a Network Intrusion Detection System (NIDS) which can

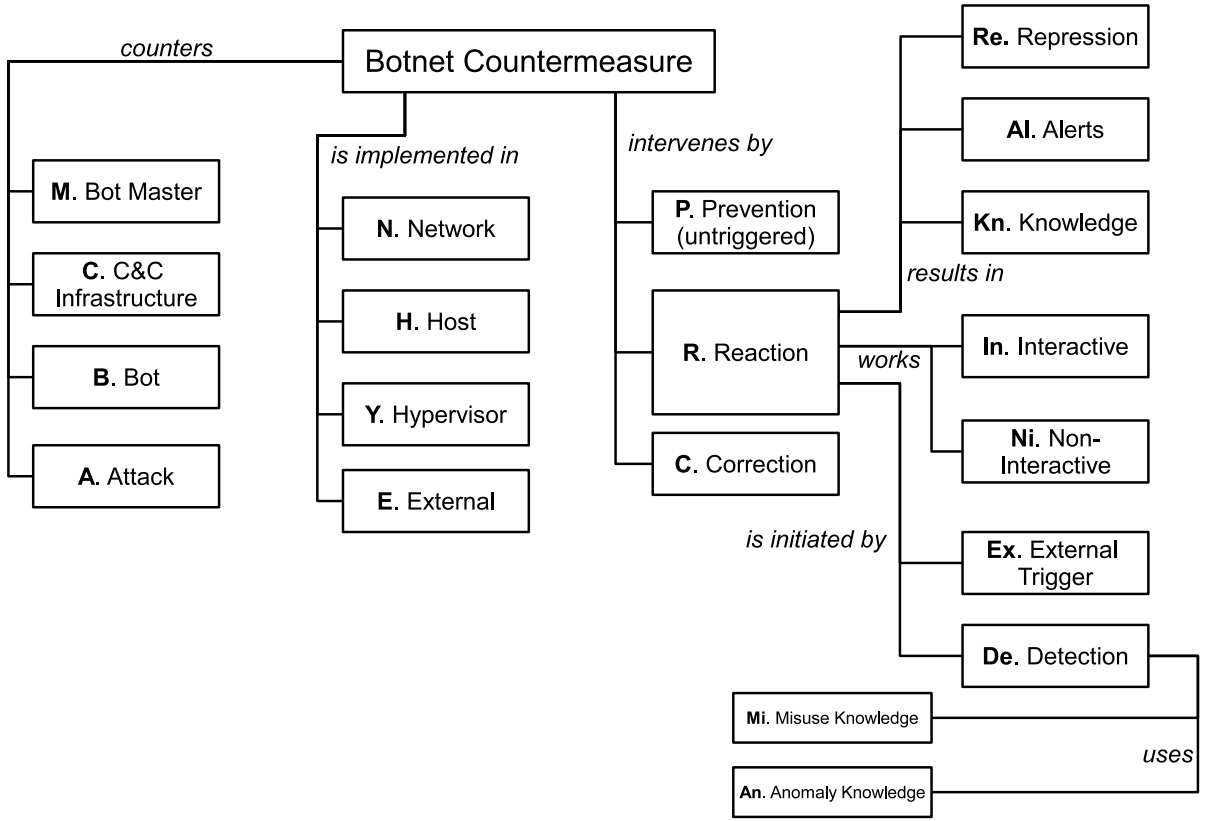

Figure 2.9: UML class diagram, representing an ontology with faceted classification of generic botnet countermeasures.

detect bot-activity in an enterprise LAN with potential bots, is classified as: Bot, Network, and Reaction. The Reaction class can be further refined by more detailed classes. Inspired by the Kendall notation in queuing theory [75], we can combine and express the matching classes in one systematic notation and use letters for the different classes. Classes of different facets are separated by slashes. In addition brackets, are used to express hierarchy. The opening bracket must follow immediately after a letter. If multiple classes in the same hierarchical level apply, the classes are enumerated by a $\wedge$ or $\vee$ representing respectively a logical AND, indicating that all classes must apply, or a logical OR, indicating that at least one of the classes must apply. Brackets are used to give precedence to a part of the logical enumeration expression. In this case the opening bracket can only follow immediately after a /, $\wedge$, or a $\vee$. If none of the classes of a particular facet apply, a dash "-" is used as a placeholder.

The root classifications are expressed by one letter. All other classifications are expressed by two letters. If the NIDS of our example does not infiltrate the botnet, detects by misuse knowledge, and does only alert a detected intrusion, the complete classification becomes: B/N/R(Al/Ni/De(Mi)).

2.3.4. MOTIVATION OF THE FACETS *"counters"* AND *"is-implementated-in"*

One of the most basic properties of a countermeasure is the object or activity that is countered. In case of a botnet the countered object is not trivial because botnets consist of a combination of components and activities that can be countered. We distinguish in this facet four different classes:

- *Bot Master*: The countered object is the criminal individual or organization behind the botnet. Examples are the bot herder, malware developer, mule, etc. Countermeasures focus on criminal investigation and prosecution.
- *C&C Infrastructure*: The countered object is the C&C infrastructure, consisting of components, services and communication that unite the bots to a botnet. Examples are: IRC-servers, HTTP-servers DNS-servers, stepping stones, C&C traffic in the core of Internet, etc. C&C countermeasures focus on detection, infiltration, and take down of the partial or complete botnet control structure. C&C countermeasures can be implemented in the network, but also in hosts that might participate in the C&C infrastructure.
- *Bot*: The countered object is the individual bot that receives instructions and carries out specific tasks. Bot countermeasures focus on the mitigation of specific bots by detection, shutdown, isolation, recruitment prevention, etc.
- *Attack*: Attack refers to the criminal activities of a botnet, such as spam, DDoS-attacks or espionage. Related countermeasures are often implemented in the vicinity of the victim, for example as perimeter-based defenses in a corporate network, antivirus solutions on a host, or observation and control of money or employees.

The classes can be combined in some cases. For example the results of C&C traffic detection can be used to identify bots, but also to identify C&C servers or domains. In

such a case the classification can be expressed as: B ∨ C. Other examples of countermeasures that can produce multiple classes per facet are:

- *Al* ∨ *Kn*: Detectors can produce at the same time alerts that can immediately be used for repression, such as the IP-address of a malicious bot, and knowledge that can be applied in other detectors, such as C&C signatures.
- *Mi* ∧ *An*: Infiltration-related detection relies on misuse-related knowledge for the infiltration itself and produces anomaly-related knowledge, related with the observed activities after infiltration.

Indirect effects are not considered for classification. For example the detection and shutdown of a bot implicitly represses its attacks, however, it is only classified as a bot countermeasure, because the cease of attacks from a repressed bot is an indirect effect. In addition, attacks can still continue from other bots.

The countered object or activity must not be confused with the place of implementation. For example bot detection can be implemented in a potential bot-infected host, but also as a separate device in the connected LAN. The place of implementation is important, since it affects the choice of detection features, repression possibilities, and the exposure of the countermeasure to the botnet. Therefore, an additional facet, by the place of implementation, is included in the ontology, which distinguishes between:

- *Network*: countermeasure implemented in an existing network component or as a separate appliance
- *Host*: countermeasure implemented as a software agent in a production computer
- *Hypervisor*: countermeasure implemented in the host that runs Virtual Machines
- *External implementation*: countermeasure not implemented in network, host, or hypervisor as described in the three above classes. Examples are countermeasures in separate devices, in an organization, or by human activity.

2.3.5. MOTIVATION OF THE FACET *"intervenes by"*

A botnet attack goes through various stages, such as developing the bot software, or actually attacking a victim. Every stage enables specific countermeasures. This suggests a time- or stage-related classification. Schiller and Binckley present a detailed lifecycle consisting of nine stages of an individual bot in a cycle: *infection, rally to C&C, retrieval of antivirus scanner measures, securing the bot, listen to commands, receiving commands, executing commands, reporting results and abandonment of the host, including erasure of evidence [113]*. Variations of this cycle can be found in various defense approaches. Gu et al. assumes a resembling lifecycle as a base for vertical correlation or dialog correlation [58]. Some details differ, like the presence of a stage that includes outbound scanning behavior. Leaving out optional stages, and involving the attacker, by adding stages for planning and evaluation, leads to a more generic lifecycle of the total botnet, as shown in Figure 2.10. Considering the requirements of our classification, the botnet life cycle seems a potential candidate for a time-related classification, because it

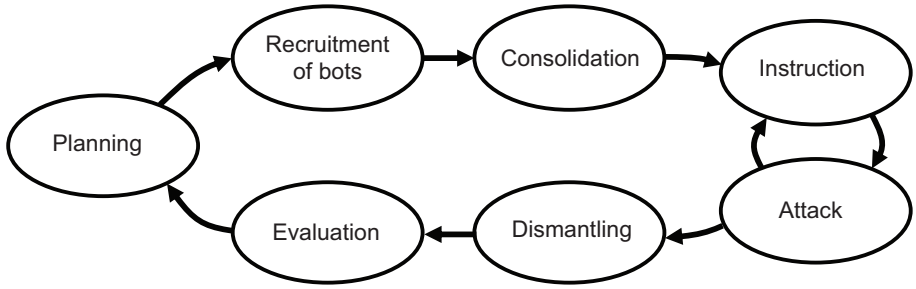

Figure 2.10: Generic lifecycle of a botnet.

is botnet-specific with clear boundaries and addressing diverse countermeasures. However, the life-cycle-based time classification depends on the countered object, as shown in Table 2.2. This dependency makes the life cycle an unsuitable facet in our taxonomy.

Table 2.2: Dependency of the botnet life cycle on the countered objects.

Time	Dependency on the countered object	Explanation
Planning	Bot Master	planning is only done by humans
Recruitment	Attack	recruitment is an attack type
Consolidation	-	possible in all places
Instruction	Bot, C&C	bots receive instructions from C&C
Attack	Attack	QED
Dismantling	-	possible in all places
Evaluation	Attacker	evaluation is done by humans

An alternative more object-independent and well-known time-related classification is the security event-cycle [98]. The stages of a security event divide defense measures in:

- Preventive measures
- Detective measures
- Repressive measures
- Corrective measures

Unfortunately, a part of this classification lacks consistency, because a botnet attack involves many parties with different perspectives. For example, from the perspective of a potential victim, the removal of bot software from a host is a preventive measure, but from the perspective of the owner of the infected host, a corrective measure. Inconsistency also arises by undefined system boundaries. For example an Intrusion Prevention System (IPS), blocking an IP-scan, can be considered a preventive countermeasure, because it prevents events to take place inside the area that is protected by the IPS. On the

other hand, it can be considered as a repressive measure inside a larger area, because it is blocking traffic, as a result of a detected event. To avoid these inconsistencies, we use in our ontology only three time-related classes, with clear boundaries:

- *Untriggered Preventive measures* = Preventive countermeasures that do not rely on any detection of botnet (related) activities. The word untriggered is used, to exclude preventive measures that need some kind of event as a trigger, because this involves implicit detection of botnet-related activity. An example of an untriggered preventive countermeasure is the timely updating of software by security updates to reduce the risk of infection. Firewalls and Intrusion Prevention Systems can give rise to confusion, because they can be seen as both preventive and reactive. The arrival of a packet, can be seen as a trigger, hence we classify them in our ontology as reactive.

- *Reactive measures* = countermeasures involving detection of botnet-related events and direct responses. This includes countermeasures that classify events as normal or anomalous.

- *Corrective measures* = measures, related to repair or the preparation of future repair of the botnet-caused damage

2.3.6. REFINEMENT OF REACTIVE MEASURES

The reactive class includes many different countermeasures, varying from a signature-based IDS to honeypots. In Figure 2.9 the ontology is refined by three facets:

- *Trigger*: A distinction is made between detective countermeasures and externally triggered countermeasures. A detective countermeasure measures features and classifies an event as botnet-related or not botnet-related. An externally triggered countermeasure receives a signal from another detector or an operator to take a repressive action or start collecting botnet specific knowledge. We will refine detection in Section 2.4.

- *Interaction with the botnet*: Interactive measures participate deliberately in the botnet as offender, C&C-entity, bot, or victim. This creates special opportunities for detection, repression, and research. Examples are: honey clients that are intended to become recruited in a botnet, injection of C&C-commands, infiltration of the criminal organization, a sandbox for evaluation of botnet-related malware. Non-interactive measures are measures that detect botnet activity, without direct participation in the botnet. Non-interactive countermeasures can have repressive responses that mitigate or stop of botnet activities without participation.

- *Result*: Repressive response involves some direct action with the aim of stopping or mitigating the botnet-attack or its effects. Examples include: the prosecution of offenders, blocking traffic, stopping malicious computer processes, exclusion of hosts by a blacklist. Alerts involve only the signaling of attack-related data that can directly be used for repression. Examples include IDS alerts. Knowledge involves

the collection of attack-related data that can be used in other detectors. The construction of blacklists of malicious IP-addresses and domains are an example of such knowledge.

Interaction with the botnet results often in a combination of detective and repressive effects. For example, the detection and deflection of an attack by a honeypot as a decoy, or the infiltration of a botnet to gather intelligence and disrupt by injection of certain C&C-commands.

2.3.7. EVALUATION OF THE ONTOLOGY-BASED CLASSIFICATION

We evaluated the proposed taxonomy by classifying well known diverse countermeasures, ranging from firewall, honeypot to even punishment of criminals. The result is shown in Table 2.3. The first column briefly mentions the countermeasure with optional

Table 2.3: Examples of countermeasures, classified according to the taxonomy.

Description of the countermeasure	Classification
Tracing the money flow [47]	M/E/R(Al∨Kn, Ni, De(Mi∨An))
Infiltration by undercover agents	M/E/R(Al∨Kn, In, De(Mi∧An))
Arrest of botmasters [77]	M/E/R(Re, Ni, Ex)
Deterrence by severe punishment [3][69]	M/E/P
Detection of IRC-signatures [52]	B/N/R(Al/Ni/De(Mi))
Correlation between C&C streams [59][121]	C∨B/N/R/(Al/Ni/De(An))
Detection of suspicious queries to DNS-blacklists [106]	C/N/R/(Kn/Ni/De(An))
Honeypot, used for C&C infiltration [125]	C/H/R/(Re∨Kn/In/De(Mi∨An))
C&C prevention by disabling certain server ports	C∨B/H/P
NIDS, detection by malicious signatures [109]	C∨B/N/R(Al∨Kn/Ni/De(Mi))
NIDS, detection by anomalies [58]	C∨B/N/R(Al∨Kn/Ni/De(An))
Software hardening and restriction of user privileges by kernel wrappers [46]	B/H/P
A stateful firewall, protecting the victim, by the default blocking all traffic that is initiated from the outside	A/N/R(Re/Ni/De(An))
A honeypot as a detector and deflector of attacks [103]	A/H/R(Re∧Al/In/De(An))
Resilience against DoS-attacks by deployment of sufficient servers [40][118]	A/H∧N/P
The use of Captchas, to prevent automated client actions [132]	A/H/P
Anti Virus Solution on a computer	B∨A/H/R(Re/Ni/De(An∨Mi))

references to literature. The second column shows the classification of the countermeasure in the proposed short notation.

We conclude that the proposed ontology-based classification meets the requirements

of Section 2.3.3 because:

- Table 2.3 shows that the ontology-based classification is capable of classifying very diverse measures in different classes that accurately reflect basic properties of the countermeasure. The classification per facet and the choice of the facets contribute to a simple classification process with minimal ambiguity or doubt.
- Countermeasures that counter a specific object or intervene during a specific phase of the threat, can be easily selected by the classification expressed by the accompanied short notation. The classification can also directly be used to select countermeasures by a specific implementation. This supports not only the selection of diverse countermeasure, but also countermeasures for a specific environment. For example, for the defense of an enterprise network, countermeasures that affect bot (B) and attack (A) can be selected from Table 2.3. A provider of a public network can select countermeasures that are not host-implemented, if the computers of the clients are not under its administrative control.
- The classification and the accompanied notation can be extended with refined facets and classes to cover further details. In Section 2.4 the detection class will be refined.

2.4. AN OVERVIEW OF EXISTING NETWORK-BASED C&C DETECTION

In this section, we discuss existing work and approaches on network-based botnet detection. We start by mapping the detection principles, as presented in Section 2.1, into a refinement of the detection-class of Figure 2.9. Figure 2.11 shows the result. The root class is the detection class of Figure 2.9. The diagram models only network-related detection, but can easily be extended to other detection types. To illustrate this, some classes are included that are not directly related with network-based detection, such as the class Host-based Features. In addition, classes use generic names, such as Activity instead of Traffic for the classes in the state facet.

For further clarification of the facets, we added key words in chevrons to the verb-based associations in the UML class diagram. The classification is faceted by the measured features, knowledge, and state. We will discuss each of the three classifiers.

Features

A distinction is made by the type of measured features. We focus on network-specific features and distinguish classes, according to Section 2.1. Additionally the specific header fields or application protocols can be added. Instead of including all possible header fields or protocols in the diagram, we added '...' as a placeholder for optional classes that specify the protocol. The three classes of the *Traffic Features* are highly related with evasion possibilities. Extraction of features in application data may be hampered by the botnet with standard encryption. Features, based on metadata, can easily be manipulated by random delays, flow perturbation, and the injection of noise in packets or flows. Finally header fields can often be adapted to imitate normal traffic. This includes the imitation of popular traffic and protocols by the use of the related well-known TCP or UDP ports. It also includes the periodic change of source and destination IP addresses to prevent correlation. *Host Features* are features, that are directly related with the host (typically the applications and operating system). *Other Features* are all features, that are not related with the host or network.

Knowledge

The reference knowledge that is used by the detection for its decision is modeled according to Section 2.1. *Misuse-related knowledge* is obtained from known malicious instances and stored as rules, blacklists, or signatures. The use of misuse-related knowledge introduces directly the problem of zero-day exploits. Special care must be given to the construction of the knowledge. Misuse-related knowledge must continuously be updated with the newest types of malicious activities, to minimize the risk of zero-day exploits. An advantage of detection by misuse knowledge is a low FPR. *Anomaly-related knowledge* models the normal activities. A deviation from normal is classified as anomalous. The knowledge of normal activity origins from different sources. In our model we distinguish: *statistics, specification*, and *machine learning*.

- Statistical knowledge requires a baseline of normal activity and optionally malicious activity. For the creation of a baseline of normal traffic, an uninfected network is required.

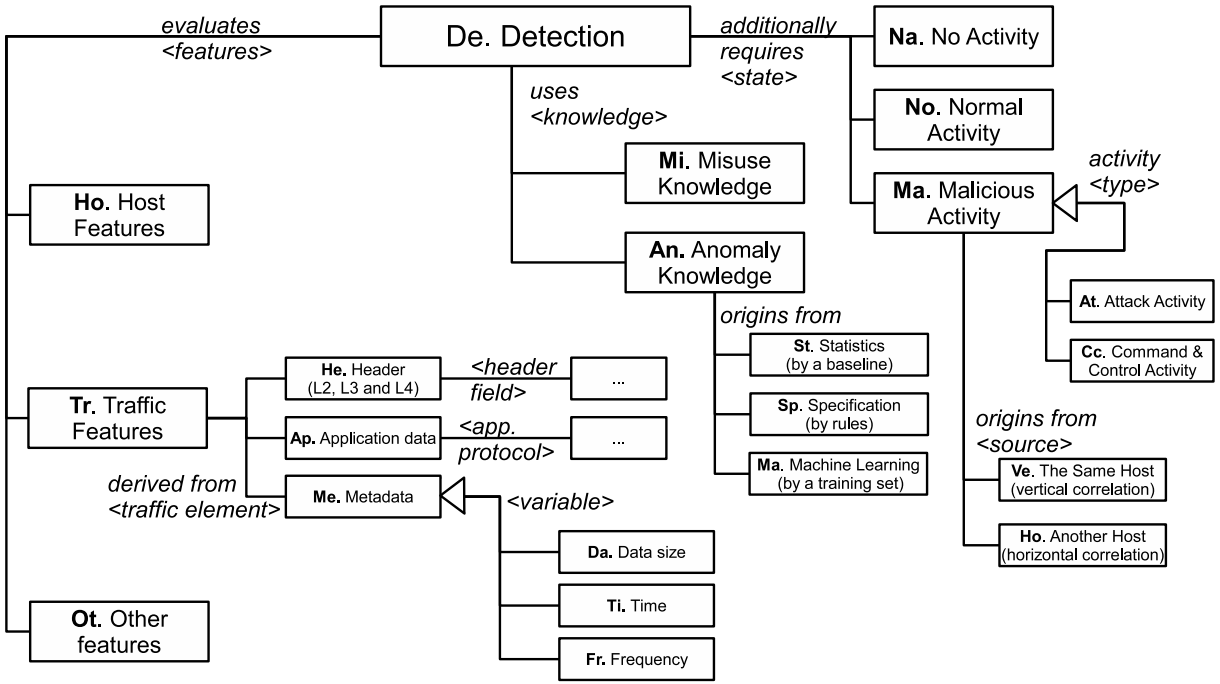

Figure 2.11: UML class diagram that represents an ontology with faceted classification of botnet detection. It is an extension of the detection class in Figure 2.1.

- Specification-based knowledge requires rules that specify normal behavior, drawn up by experts.
- Machine-learned knowledge for network detection is in most cases supervised and requires a representative training set of both normal and malicious traffic.

State
In addition to knowledge, the detection decision often depends on the momentary state of the detector. The momentary state is determined by previous normal or malicious activity. An important group of detectors that require a state, constructed from previous malicious activity, is the group of detectors that correlate different instances of botnet traffic. The consequence is that these detectors cannot immediately repress the first occurring malicious traffic event, since they need preceding malicious activity to classify the events as malicious. The classification further distinguishes preceding activity from the same host, often referred as vertical correlation, and preceding activity from other hosts, often referred as horizontal correlation. Additionally the state can be divided in C&C activities and attack activities.

2.4.1. CLASSIFICATION OF EXISTING MISUSED-BASED DETECTION

This section classifies different existing misuse-based detection approaches according to the ontologies of Figure 2.9 and 2.11. All discussed countermeasures are network-implemented non-interactive detectors that deliver potentially alerts and knowledge to counter bots or C&C traffic. By the root classification of Figure 2.9 this group of countermeasures is classified as: B∨C/N/R(Al∨Kn/Ni/De(Mi)). Therefore, we will only specify the detection-specific part of the notation. Misuse-based detection approaches:

- *Snort [109]: De(Tr(He∨Ap∨Me)/Mi/Na)* Snort is a misuse-based NIDS. We only classify the basic version of Snort, which works with signatures that span both the header and content of packets. Since it depends on misuse knowledge, it is ineffective against new malicious traffic types. In most cases no additional context is needed for the decision, which results in an immediate detection decision.
- *Rishi [52]: De(Tr(Ap(IRC))/Mi/Na)* Rishi is the name of a detection system that evaluates nicknames of IRC-based C&C traffic by known malicious string patterns. Its classification shows that is it is a comparable with Snort, but with a more specialized feature set
- *Botzilla [107]: De(Tr(Ap)/Mi/Na)* Botzilla actually combines two countermeasures. The first countermeasure is a sandbox that runs infected bot malware. Traffic signatures are derived from the content of the resulting C&C traffic. This can be seen as a separate countermeasure, classified by the ontology of Figure 2.9 as: C/H/R(Kn/In/Ex). The acquired knowledge is used to detect C&C traffic from bots in a separate detector, of which the detector part can be classified by the ontology of Figure 2.11 as De(Tr(Ap)/Mi/Na). Since the detection uses signatures of malicious traffic, the knowledge is classified as misuse.
- *Cocospot [39]: De(Tr(He∧Me)/Mi/Na)* As Botzilla, Cocospot combines two countermeasures. A sandbox is infected by bot malware and traffic signatures are de-

rived from the traffic size and timing the resulting C&C traffic. In a separate network-based detector traffic is correlated with the obtained patterns, derived from the sandbox. Basically the detector used misuse-related knowledge. However, instead of string matching, statistical footprints related with time and byte size of traffic fragments are used as features.

- *Provex: [110]: De(Tr(Ap∧Me)/Mi/Na)* Payloads of potential C&C traffic are decrypted by decryption protocols that are specifically used by botnets . The involved decryption protocols and keys are extracted from malware samples. In the detector the payload is statistically evaluated on the likeliness that such an encryption was applied. Since signatures of known C&C traffic are used in the process, it is misuse based.

2.4.2. EXAMPLES OF ANOMALY-BASED DETECTION

This section classifies different existing anomaly-based detection approaches according to the ontologies of Figure 2.2 and 2.11. For the same reason as with misuse detection, the generic classification of the countermeasure is again omitted in the notation. Anomaly-based detection approaches:

- *Botminer [59]: De(Tr(He∧Me)/An(St∧Ma)/Ma(Ho/At∧Cc))* Similarities in communication traffic and malicious traffic are cross correlated to identify C&C traffic and bots. The correlation requires attack traffic and C&C traffic from different hosts. Clustering is accomplished by unsupervised machine learning algorithms.

- *BotGAD [23]: De(Tr(He∧Ap(DNS))/An(St)/Ma(Ho∧Ve/Cc))* Group activities of bots in a network are observed. This results in detection of individual bots, but also of centralized C&C servers. A DNS-based solution that groups hosts with similar DNS-lookups is presented and evaluated as a case study. The decision is made comparing similarity related parameters with predefined thresholds, based on measured normal traffic. Before detection multiple C&C lookups must have taken place from different bots.

- *DNS anomaly detection [130]: De(Tr(Ap(DNS))/An(St)/Ma(Cc))* A combination of two DNS-based detection approaches that identify high query rates of a specific domain name or a high number of queries to non-existend domain. Both anomalies are often caused by bots that use a Domain Generation Algorithm (DGA) to create a resilient lookup mechanism for the C&C server. The approach needs many instances of C&C DNS traffic to exceed certain predefines threshold.

- *Bothunter [58]: De(Tr(He∧Ap∧Me)/An(St)/Ma(Ve/At∧Cc))* Bothunter uses vertical or dialog-based correlation to detect bot traffic. Since the detection of the dialog is based on specific knowledge of botnet stages, it is partly misuse-based detection. At the same time statistical anomaly detection is used to detect traffic of specific stages. The complete system is complex and requires observation of multiple malicious traffic events before a detection decision can be made.

- *Suspicious DNS-blacklists queries [106]: De(Tr(He∧Ap(DNS))/An(St)/Ma(Ho/Cc))* The detection identifies bots that query blacklists for counterintelligence. Fea-

tures are derived from the number of queries from a specific host and the number of times the DNS-name of the host itself is queried. Additionally the query rate over a certain time is evaluated to detect the presence or absence of diurnal effects.

- *IRC-based C&C flow correlation[121]: De(Tr(He∧Me)/An(St)/Ma(Ho/Cc))* The detection is based on derived features of IRC traffic flows. Statistical analysis of time and topology, by source, and destination addresses are used for detection.
- *Detection C&C by temporal persistence [50]: De(Tr(He∧Me)/An(St)/Ma(Ve∨Ho/Cc))* Detection is based on repeated visits to certain destinations, called persistent destinations, by bots. A base line is created of persistent destinations of normal traffic. New persistent destinations are classified as anomalous.
- *Detection of Peer to Peer C&C traffic [111]: De(Tr(He∧Me)/An(Ma)/Na)* The detection approach use 17 features to classify peer to peer C&C traffic. Since it uses supervised machine learning, datasets for training are required.
- *Not-a-Bot [60]: De(Ho∧Tr(He∧Me)/An(Sp)/Na)* Detection uses digital attestation to distinguish human-generated traffic from machine generated, such as bot- generated spam. Detection requires a host-implemented or hypervisor implemented attester that can monitor user activity of a potentially bot-infected host. The verifier is implemented in the network or related server.

2.5. NETWORK-BASED C&C DETECTION IN AN ENTERPRISE NETWORK

In Chapter 1 we described the differences between enterprise networks and public networks and discussed the threat of specialized bots, intended for espionage and sabotage in the enterprise network, that are difficult to detect. This section systematically identifies enterprise-specific traffic-based detection approaches for such bots. Detection focuses only on the C&C part of the bot traffic, since there is no guarantee that a bot produces attack traffic. In addition we assume that sophisticated botnets try to evade detection by the use of zero day traffic, limitation of the traffic volume, and imitation of popular traffic types. In order to find effective detection approaches of C&C traffic in enterprise networks, we will first identify important enterprise-specific characteristics that influence detection possibilities. With this knowledge each facet of the detection ontology is evaluated. This delivers a number of desired detection properties. Finally, we propose new detection approaches that anticipate the desired detection properties.

2.5.1. ENTERPRISE-SPECIFIC CHARACTERISTICS

As discussed in Chapter 1, there are significant differences between enterprise networks and public networks. We identified four enterprise-specific characteristics that have influence on the detection possibilities:

1. An enterprise network allows for comprehensive control and restriction of traffic because:

   (a) In a corporate network, the number of different applications and the associated network traffic types are much more restricted than in public networks. This makes it easier to define unnecessary traffic.

   (b) In contrast to public networks, enterprise networks have more freedom to implement traffic restrictions, because they are less limited by net neutrality regulations and the wish of consumers not to restrict their diverse traffic.

2. An enterprise network allows for detailed traffic observation. This includes the content of traffic by deep packet inspection (DPI). In public networks such observation is limited by privacy regulations and in addition, technical complexity limits DPI in the connected home networks.

3. An enterprise network allows for comprehensive control of end systems (hosts) because:

   (a) The intended function of end systems in an enterprise network is better defined than in public networks. This includes the installed applications, client-server role, connected hardware and the interaction with a user.

   (b) The end systems of an enterprise network are under one administration. In a public network the control is scattered over the different customers of the public network.

4. The administration of a corporate network, aware of security risks, can and will put more effort in countermeasures then a consumer or a public provider that connects consumers. If necessary, countermeasures can include significant changes in hardware, software and procedures.

2.5.2. SELECTION OF CLASSES IN THE DETECTION ONTOLOGY

The facetted classification of Figure 2.11 allows for a systematic selection of classes that support the detection of sophisticated botnet C&C traffic and optimally make advantage the identified enterprise-specific characteristics. We identify here the most suitable classes per facet.

- *Detection Features*: The opportunity of DPI in an enterprise network allows for inspection of *Application Data*. Since corporate application traffic often consists of DNS and HTTP(S), many bots will use these protocols to mimic normal traffic. In these cases successful detection can depend on subtle elements of the content. Even the inspection of encrypted TLS/SSL-traffic by an intercepting proxy is possible, since a trust relationship between the end systems and the proxy can be established by an enterprise PKI (Public Key Infrastructure). The physical presence, complete control, and well-defined function of many end systems in an enterprise network allow for the observation of *Other Features*, such as actions of users or connected hardware.

- *Knowledge*: Many existing detection approaches depend on misuse related knowledge. Although effective against known C&C traffic, additional anomaly knowledge is required to detect zero day traffic. In practice knowledge acquired by machine-learning also depends on misuse related knowledge, because malicious samples are part of the training set and the ability to recognize complete new types of C&C traffic is severely limited by the finite complexity of practical machine learning. This makes knowledge that origins from *Statistics* or *Specification* the most interesting classes for detecting zero-day C&C traffic. Most existing anomaly-based detection approaches are limited to statistics, because it is difficult to define normal traffic by a specification. However in a corporate network with its limited traffic diversity, detection of anomalies by specification of normal traffic is a feasible option.

- *State*: Many existing C&C detection approaches are based on correlation that requires some kind of prior *Malicious Activity*, before C&C traffic is detected. In case of sophisticated bots, intended for espionage, there is no attack traffic. Long delays between malicious activities make the correlation-based detection less feasible because of the required resources for constructing, storing, and using the state. The amount of required resources can be extensive: the state includes both normal and malicious activities from the past, since a separation of activities is not possible before detection. The problem can be mitigated by the use of a state that remains compact and scales well with an increasing observation interval or network size. Another disadvantage of detection that relies on previous malicious activity, is the implication that in the past at least one malicious activity must have

taken place without detection at that moment. In contrast, detection approaches that do not require a state (*No State*) or a state that only depends on *Normal Activity* allow for detection at the first occurring C&C traffic event. This is especially interesting in a corporate network, where immediate repression of bots and traffic within the premises is feasible.

To summarize the identification: the following classes as suitable for detection of sophisticated botnet traffic in enterprise networks:

1. Features, derived from application data, especially DNS and HTTP(S)
2. Features from other sources than the computer network
3. Knowledge related with statistics
4. Knowledge related with specification
5. No state, or a state of normal activities
6. A state of prior C&C traffic, if the state remains compact and scalable

This does not mean that detection, based on these classes, replaces other detection approaches. It must be seen as an addition. For example, misuse-related detection remains a very important way to detect malicious traffic with a low FPR. The identified classes can be used to find new effective countermeasures that operate alongside other detection approaches in a layered security architecture.

2.5.3. NEW DETECTION APPROACHES

We propose three new approaches for C&C detection in an enterprise network, based on the identified classes. Each approach is described in detail in the chapters 3 to 6. Table 2.4 gives an overview of the detection approaches with the specific detection classes.

- *Chapter 3 and 4: Direct causality*
  $B/N \wedge (H \vee E)/R(Al/Ni/De(An))$
  with $De=De(Ho \wedge Tr(He \wedge Ap(DNS \wedge HTTP) \wedge Me)/An(St \wedge Sp)/No)$
  Egress traffic is evaluated on its direct cause. Direct causes are events that immediately trigger a new traffic flow. Chapter 3 starts with an exploration of user events. By measuring the time between user activity and traffic, automatic traffic is distinguished from human-generated traffic. Human activity can be measured by a software agent on the observed computers or by special external hardware that directly observes activity of the keyboard and mouse of a specific computer. The identification of user events as direct causes for new traffic is especially interesting for the detection of C&C traffic that uses services of popular social media. In Chapter 4 the approach is further completed by also including traffic events as direct causes for new flows. We define and construct TFC Graphs (TFC=Traffic Flow Causality) that provide an overview of traffic flows and their direct causes. In this way the root cause of each traffic flow is identified and associated with normal traffic or C&C traffic. We will refer to the complete approach as TFC detection. The collection of all normal traffic graphs represents a state that is required to detect C&C traffic.

- *Chapter 5: Trustworthiness of visited destinations*
  B/N/R(Al/Ni/De(An))
  with DE=De(Tr(He∧Ap(DNS∧HTTP)∧Me)/An(St∧Sp)/No)
  Traffic is evaluated on the trustworthiness of the contacted destination identifier, such as an IP-address, hostname, or URI. If the destination identifier origins directly from a human, prior traffic from a trusted destination, or a defined set of legitimate applications, the destination is trusted and its associated traffic is classified as normal. The required knowledge origins primarily from specification that defines how legitimate destination identifiers are obtained in a enterprise network. Statistical knowledge is necessary for evaluation of hostnames. We refer to this detection approach as UDI-detection (UDI=Untrusted Destination Identifier). Chapter 5 also compares UDI detection with TFC-detection and highlights the important similarities and differences.

- *Chapter 6: Domain degree distribution*
  B/N/R(Al/Ni/De(An))
  with DE=De(Tr(Ap(DNS))/An(St∧Sp)/Ma(Cc/Ho))
  The degree distribution of resolved DNS-domains is evaluated. Domains with an unexplained popularity are classified as anomalous. In contrast to the other approaches, this approach requires a state that includes C&C traffic, before it can successfully detect new C&C traffic. Only if a minimum amount of bots has contacted a C&C domain, this domain and its traffic can be classified as anomalous. Since this state only consists of domain names and the number of computers that queried a domain, this approach allows for scalable distribution over multiple enterprise networks.

Table 2.4: Proposed detection approaches and their use of classes that are suitable for botnet detection.

Detection Approach	DNS/ HTTP features	Host features	Statistics	Specification	No Malicious State	Compact Malicious State
Direct causality	x	x	x	x	x	
Trustworthiness of visited destinations	x		x	x	x	
Domain degree distribution	x		x			x

In the the chapters 3 to 6 we will also identify and discuss the limitations of each of the approaches. Detection evasion is and important part of this. In Section 7.1, as a part of the concluding remarks, we will continue this discussion by combining and

extending the presented approaches. This leads to proposed future steps as a follow-up of this study.

3

TOWARDS DETECTION OF BOTNET COMMUNICATION THROUGH SOCIAL MEDIA BY MONITORING USER ACTIVITY

A new generation of botnets abuses popular social media like Twitter, Facebook, and Youtube as Command and Control channel. This challenges the detection of Command and Control traffic, because traditional IDS approaches, based on statistical flow anomalies, protocol anomalies, payload signatures, and server blacklists, do not work in this case. In this chapter we introduce a new detection approach that measures the causal relationship between network traffic and human activity, like mouse clicks or keyboard strokes. Communication with social media that is not assignably caused by human activity, is classified as anomalous. We explore both theoretically and experimentally this detection approach by a case study, with Twitter.com as a Command and Control channel, demonstrate successful real-time detection of botnet Command and Control traffic, and discuss limitations.

This chapter is a minor revision of the paper with the same title published in the Proceedings of the ICISS2011 Conference [19].

3.1. INTRODUCTION

Social media, like Twitter, Facebook, and Youtube create a new communication opportunity for malicious botnets. A Command and Control (C&C) channel is crucial for a botnet. Bots need regular instructions from the botmaster, for example to initiate or synchronize an attack, upload harvested data, or update the malware [113]. A large number of countermeasures has been developed and deployed against C&C in general. Botnets react to these measures by using new communication mechanisms that are harder to detect and repress [119]. With the arrival of social media based C&C, there are now 4 major C&C-mechanisms:

1. IRC is a simple and proven C&C technique. It is attractive for the botmaster, because of its simple management, with a real time or "tight" control over the botnet through a permanently established connection. However especially for this type of C&C, many countermeasures have been developed. An example is the work of Cook et al. [27].

2. A more sophisticated C&C-mechanism uses peer-to-peer communication, based on protocols like Kademlia [33]. The intrinsic decentralized structure is difficult to counter and the use of peer-to-peer protocols is widespread, due to popular file sharing and communication protocols, like Bittorrent and Skype. However the absence of centralized control makes management complex, and due to firewalls and NATs, many peers cannot receive incoming connections, resulting in network of which the availability and stability highly depends on a limited number of nodes [90].

3. Another C&C-mechanism uses HTTP to exchange information. The popularity of HTTP makes anomaly detection difficult. In addition, many botnets decentralize the HTTP-service by fast fluxing A-records, IP-addresses of DNS-servers, or even domain names [93]. This makes it difficult to bring the service down by IP or domain blacklisting [102].

4. A rather new C&C-mechanism uses social media for C&C. A botmaster posts instructions as messages on a popular social medium, like Twitter or Facebook. Bots fetch the instructions by regularly polling certain pages. Examples of such botnets are: Trojan.Whitewell, that uses Facebook [81], TwitterNET, that uses Twitter [92], and Trojan 2.0, that uses Google groups [54].

If a botnet imitates the communication patterns of normal users that visit a popular social medium, detection will become very difficult with conventional network IDS-techniques, because there are no anomalous addresses, domain names, protocols, or ports involved and a large fraction of the daily legitimate traffic of normal computers consists of visits to social media. A further increase of the C&C invisibility is possible by steganographic techniques, to hide the commands in apparently normal messages, account flux, by making the bots visit different accounts or even different media, and SSL/TLS encryption, to impede content inspection in the network. Many social media offer already HTTPS access as an alternative to the still popular unencrypted HTTP access.

Difficult detection is not the only reason that social media based C&Cs have the potential to become the most important botnet control mechanism on the Internet. Other beneficial properties from the perspective of the botmaster are: simple implementation, simple management of applications that read from and write to social media, scalability, and high availability of the social media services.

In this chapter we address the detection of social media based C&C-traffic, by introducing a detection mechanism that measures causality between user activity and network traffic. The presence or absence of certain key strokes and mouse clicks is used to determine if network traffic is user- or botnet- originated.

Section 3.2 gives an overview of related existing work in the areas of botnet C&C-detection and the observation of user activity in the detection process.

Section 3.3 introduces a detection mechanism that uses mouse clicks and key strokes as a proof of user commitment to visit a social medium. If egress traffic starts within a small time window directly after a user event, it is classified as user intended. Traffic outside this time window consists of both legitimate traffic that is indirectly induced by user events, and botnet C&C traffic that is not induced by any user event.

The detection mechanism is elaborated with Twitter.com as representative example of a social medium.

Section 3.4 discusses detection improvements, related with legal automatic traffic and potential evasion techniques.

3.2. RELATED WORK

Most existing detection approaches in enterprise networks, as discussed in Chapter 2, have limited results against botnet traffic that carefully imitates user originated visits to popular social websites, due to the close resemblance to legitimate traffic.

Misuse-based detection, such as Snort [109] is very limited, because it is difficult to define and find misuse-related signatures of social media C&C. The protocols and destinations of the social media C&C traffic are also used for legitimate traffic and in addition the payload can imitate normal application traffic by steganography.

The resemblance to normal traffic also complicates anomaly-based detection. A mixture of C&C traffic with a large volume of resembling legitimate traffic in the same network complicates the search of similarities between different instances of C&C traffic and the search for correlation with attack traffic. If well implemented, it cripples a large variety of detection approaches, such as Botminer [59] and the approach of Giroire et al [50]. Detection approaches that anticipate DNS anomalies, such as fast changes in DNS-records [65], are not effective, because C&C over social media uses DNS in a normal way.

Honeypots [103] that are recruited in a botnet can detect C&C over social media, because all activity is anomalous by definition. However, a honeypot itself is not a production system, hence after detection, a signature of the malicious traffic must be identified, derived and distributed for misuse detection. As already mentioned, identification of traffic signatures of C&C over popular social media can be very difficult.

Another aspect, which is related with the detection mechanism, to be presented in this paper, is the measurement of user commitment. A CAPTCHA, as introduced by Ahn et al. [132], is a popular test to remotely distinguish between computer programs and

users. CAPTCHA's are often used in Web 2.0 applications, to prevent automated account creation. Obviously an intensive use of CAPTCHA's does not promote the accessibility of a website. As most users do not create a new account every five minutes, the nuisance is limited in that case. However, the use of CAPTCHA's for every social medium visit, results in an unacceptable burden. Of course cryptographic signatures can be used to relate later traffic with an initial CAPTCHA verification, but it can result in a complex trust mechanism. Vo et al. [131] propose a system that authenticates physical computers with social media accounts after an initial CAPTCHA test. However once authenticated, the system does not really prevent botnets to use the account or visit public pages of users.

Gummadi et al. propose NAB, Not-a-Bot [60]. The system uses TPM-backed attestations of user activity, that are sent with web- and email requests to an external verifier, where the decision is made if the traffic is legitimate or not. Gummadi et al. focus primarily on the implementation of the attestation mechanism and give no details of the time-related detection mechanism. Verification of the detector performance is done indirectly by combining offline keyboard- and mouse logs with network traces. It shows that NAB can reduce the rate of intense bot-generated traffic, like DoS and spam. NAB, can be seen as complementary to our work, by delivering a possible practical implementation of a trusted agent that attests user events to the actual detector.

Kartaltepe et al. [74] present a systematic overview of the evolution in social media botnets and discuss the possibilities and limitations of detection approaches. The work defines a process on a client computer, that does not interact with human input, as *self concealing*. This can be a detectable property in a host-based IDS, to identify suspicious processes. It is different from our approach, that identifies normal traffic to social media by the direct presence of prior user activity.

3.3. DETECTION PRINCIPLE

Network traffic does not occur spontaneously. Something triggers a traffic flow. In the case of a visit to a social medium, the trigger is usually a keyboard or mouse event, caused by a human user. However if a bot visits a social medium to fetch new instructions, or upload harvested information, the traffic is not triggered by user events, but by internal state changes of the malware. This allows for detection of botnet traffic to social media by the absence of user events that could potentially have triggered the traffic.

Figure 3.1 shows a schematic overview of our proposed detection system. Network traffic (3) is captured from an inserted network bridge. If a traffic flow is initiated to a social medium, without preceding keyboard events (1) or mouse events (2), the flow is classified as potential bot-originated by the causality detector.

There are several ways to implement the taps to intercept the keyboard and mouse events. For example by hooks in the operating system of the client computer that catches the events. Another possibility is the insertion of a hardware device that monitors the signals on the bus from the user devices to the client computer. In case of a virtual client computer, the tap can even be implemented in the hypervisor. The possibility of implementing taps, causality detection, and bridge completely outside the client computer results in a detector that is less visible and more resistant against direct attacks.

The causality detection works with a small time window that starts directly after a

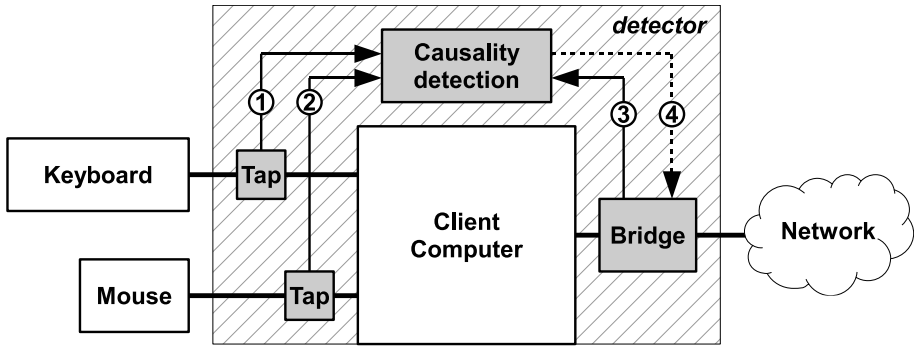

Figure 3.1: Schematic overview of a detector which correlates activity of keyboard (1) and mouse (2), with captured network traffic (3).

user event and discriminates between flows that are caused or not caused by the user event. It does not depend on known signatures; hence communication of zero day-bots can also be detected. Moreover, in contrast to most anomaly detection mechanisms, the classification is real time, resulting in the potential block of a malicious flow at the moment of the first egress packet.

In the remainder of this section we elaborate on the detection algorithm, estimate the performance, and show by experiment that it really detects botnet traffic. For the ease of the explanation, we focus on Twitter.com as a representative example of a popular social medium, hence all results can be extended to other social media. We assume that all legal traffic to Twitter.com is directly caused by user events and all bot-originated traffic is not synchronized with user activity. In Section 3.4 we discuss important exceptions to these assumptions, like legitimate automatic traffic and detector evasion by synchronization with user activity.

3.3.1. DETECTION OF BOTNET TRAFFIC TO TWITTER.COM

The detection algorithm proposed in this section is linked to browser access to Twitter.com with Internet Explorer and Firefox. In Section 3.4 we discuss alternative client scenarios to communicate with Twitter. In the browser scenario we identify three specific user events that can exclusively trigger Twitter traffic:

- Left mouse click, typically on Twitter link;
- Enter key, typically during login or completion of message or “Tweet”;
- F5-key to reload a page

Normal Twitter traffic always starts with the request of an object from Twitter.com. Examples of requested Twitter.com-objects are: a timeline with tweets, a search instruction, or a login. Directly after the loading of the first html-formatted object, additional objects, like scripts, media, advertisements, and tracking objects are loaded from other domains, like Twimg.com and Google.com. Bots that use Twitter as a C&C-channel must

start with a request of a Twitter.com-object, because other sequences are unusual and raise suspicion. The Twitter.com-object can directly contain C&C instructions or results, or link to other objects with C&C-related content. Our detection algorithm tests for the presence of relevant user events within a certain time frame, just before an egress flow to Twitter.com is initiated. Figure 3.2 shows the relevant timing.

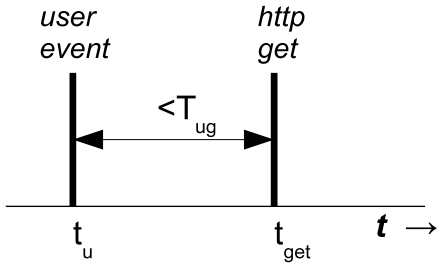

Figure 3.2: Time diagram of a user event that directly results in a visit to Twitter.com. Parameters are defined in Table 3.1.

A Twitter.com flow is classified as non-bot if the flow is user induced, as illustrated by the implication:

$$t_{get} - t_u < T_{ug} \rightarrow user\,induced \tag{3.1}$$

A complicating factor is DNS. If the client cache does not contain a valid DNS record of Twitter.com, a DNS lookup will take place between the user event and the actual visit to Twitter.com. Figure 3.3 shows the relevant timing for the second more complex scenario. The Twitter.com flow with is now classified as non-bot if the flow is user induced, as illustrated by the implication:

$$(t_{dnsq} - t_u < T_{ud}) \wedge (t_{dnsa} - t_{dnsq} < T_{dd}) \wedge (t_{get} - t_{dnsa} < T_{dg}) \rightarrow user\,induced \tag{3.2}$$

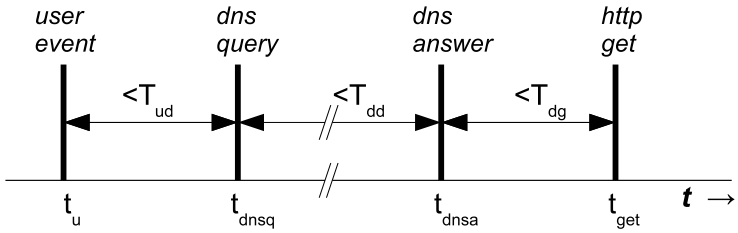

Figure 3.3: Time diagram of a user events that results in a DNS-lookup, followed by a visit to Twitter.com. Parameters are defined in Table 3.1.

Detection performance depends largely on a proper value of the defined windows T_{ug}, T_{ud}, and T_{dg}. Large windows increase the probability that egress botnet traffic is

Table 3.1: Description of the parameters, used in the algorithm.

Parameter	Description
t_u	user event(release in left mouse click, press of F5 or Enter key)
t_{get}	HTTP GET-request to Twitter.com
t_{dnsq}	DNS query to resolve Twitter.com
t_{dnsa}	DNS answer to resolve Twitter.com
T_{ug}	maximum permitted time between t_u and t_{get}
T_{ud}	maximum permitted time between t_u and t_{dnsq}
T_{dd}	maximum permitted time between $t_{dnsq}$and t_{dnsa}
T_{dg}	maximum permitted time between t_{dnsa} and t_{get}

coincidentally initiated inside a window, with a false negative as a result. Conversely small windows increase the probability that user-induced traffic starts just after a window, with a false positive as a result. The size of T_{dd} depends on the maximum expected response time of the DNS-server. The value of T_{dd} is not critical in the detection process, because it is not a response time of the potentially infected client computer. Moreover causality between DNS request and DNS response can also be determined by the UDP client port or DNS payload instead of timing.

3.3.2. EMPIRICAL ESTIMATION OF OPTIMAL TIME WINDOWS

The optimal value of T_{ug}, T_{ud}, and T_{dg} is the worst case response time of the involved client computer. This value is determined by many factors, like the amount of processing involved in the response, hardware capabilities, real time properties of the operating system, and the actual load of the system. We estimated the values by measuring Twitter-related response times on a HP DC7900, a main stream Personal Computer, in a test set up, as in Figure 3.1, but with a logging facility instead of a causality detector. The keyboard/mouse taps were implemented by a separate microcontroller board, inserted in the PS2-cables of both the mouse and the keyboard. Captured events were signaled to the logging device over a high speed serial connection. All other functions were implemented in a fast computer with a Linux operating system.

We repeatedly visited the same timeline of a Twitter account with a Firefox browser under Windows XP. Response times were measured for 3 different user events, both under idle and high load conditions. The latter condition was established by simultaneous downloads of 5Mb/s+ traffic from 10+ sources. The three user events were: F5 reload, mouse click (M1), and mouse click with empty DNS-cache (M2). Every measurement was repeated 100 times. The average and extreme response times for each scenario are presented in Table 3.2.

Only 2 of the 600 measured response times were above 100ms, with a maximum response time of 163ms.

3.3.3. THEORETICAL PERFORMANCE OF THE DETECTOR

To determine the Detection Rate (DR) and False Positive Rate (FPR) as defined in Section 2.2.1, we start by an analysis of visits to Twitter.com with a valid DNS cache. This simple

Table 3.2: Summary of measured response times of Twitter.com visits with a HP DC7900 Windows XP PC. F5 is the response to a F5-reload, M1 is the response to a Mouse click with preloaded DNS-cache and M2 with empty DNS-cache.

		t(ms) Idle			t(ms) High concurrent load		
Case	Interval	t_{min}	t_{av}	t_{max}	t_{min}	t_{av}	t_{max}
F5	t_{get}-$t_{u,key}$	0.1	3	163	0.2	15	115
M1	t_{get}-$t_{u,mouse}$	16	17.6	32.1	16	27	45
M2	t_{dnsq}-$t_{u,mouse}$	4.7	5.8	7.1	5.3	14	21
	t_{get}-t_{dnsa}	0.3	0.6	2.7	0.3	3.2	7.3

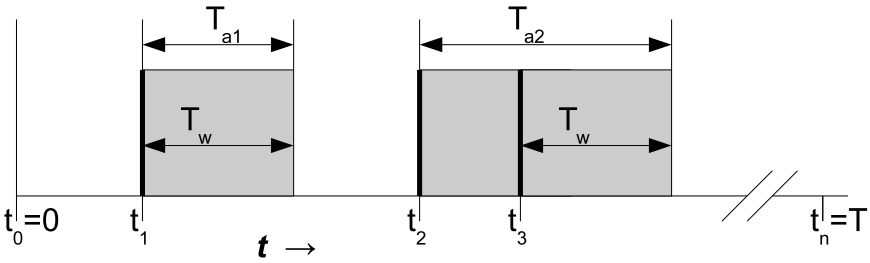

Figure 3.4: Example of 3 user events (t_1, t_2, and t_3) with their causal windows with length T_w. Traffic that is initiated outside these windows is classified as anomalous.

scenario uses only one time window $T_w = T_{ug}$. At the end of the next subsection we will extend the analysis for the more complex scenario with DNS-traffic.

Only a Twitter.com flow that starts within the causal window T_w is classified as user-caused and hence normal. All other Twitter.com flows are classified as anomalous. Figure 3.4 illustrates a generic example of three causal windows of which two are partly overlapping. Overlap can take place if the window size is in the order of magnitude of the minimum time between user events.

Detection Rate Assume multiple events during some observation interval T and define t_i as the moment the i^{th} user event is observed. The boundaries t_0 and t_n are not user events, but the respective start and stop time of the observation interval. Let $h(t)$ be a function that has the value of 0 if t is inside a causal window, and the value of 1 if t is outside any of the causal windows:

$$h(t) = \begin{cases} 0 & if \exists i, 0 \leq t - t_i \leq T_w \\ 1 & if \nexists i, 0 \leq t - t_i \leq T_w \end{cases} \tag{3.3}$$

The Detection Rate (DR) is the probability that a bot-originated flow starts outside any of the causal windows and hence is classified as anomalous.

$$DR = P\big(h(t) = 1 | t = random\,start\,of\,C\&C\,flow\big)$$
$$= \frac{1}{T}\int_0^T h(t)dt = \frac{1}{T}\sum_{i=1}^{n} H_i \tag{3.4}$$

with

$$H_i = \begin{cases} 0 & if\ t_i - t_{i-1} \le T_w \\ t_i - t_{i-1} - T_w & if\ t_i - t_{i-1} > T_w \end{cases} \tag{3.5}$$

We define T_{av} as the average time between 2 successive user events during the observation interval. If $T_{\text{av}} >> T_{\text{w}}$ the effect of overlapping windows is negligible and Eq. 3.4 reduces to:

$$DR \approx 1 - \frac{T_w}{T_{av}} \tag{3.6}$$

Eq. 3.6 shows that the DR is optimal with a small causal window T_{w} and a large average time between user events.

To estimate the DR in a worst case scenario, we used the reported results of odometric measurements of the Workpace RSI-software among 50,000 users during a 4 week period [123]. The work shows a daily peak of 3400 mouse clicks and 11,600 key strokes during 2.8 hours. If we assume that 5% or less of the pressed keys is a carriage return or F5, the average time between user events can be calculated:
$T_{av} = (2.8 * 3600) / (3400 + 11600 * 0.05) \approx 2.5s$. Based on 163ms as the maximum measured response time in Table 3.2, we choose T_{w}= 200ms, resulting in a detection rate: $DR = 1 - (0.2/2.5) = 0.91$.

In practice the the average DR is expected to be higher, because:

1. the average time between user events over a day is significantly higher than 2.5 seconds, because most computers are only intensely used during a small part of the day[123];

2. the effective frequency of left mouse clicks is lower, because the presented frequency also includes right mouse clicks and double clicks[123].

In the more more complex scenario with a DNS-lookup there are two critical windows: T_{ud} and T_{dg}. By replacing both windows by one window $T_{\text{w}} = T_{\text{ud}} + T_{\text{dg}}$, the DR is calculated by the same math. Based on 21+7.3ms as the maximum measured response time in Table 3.2, we choose T_{w} = 21+7.3 = 30ms, resulting in a DR of 0.99.

False Positive Rate False Positives are caused by a response time that is larger than the defined causal window T_{w}. Eq. 3.7 expresses the resulting False Positive Rate (FPR). It depends on the distribution of the response time.

Table 3.3: Experimental results of the detection algorithm during an 8 hours observation of an infected computer, with $T_{ug} = T_{ud} = T_{dg} = 200$ms and $T_{dd} = 10$s.

Parameter	Value
total left mouse click rel. events	1620
total Enter key press events	414
total F5 key press events	28
total flows	7170 (1467 bot + 5703 user)
Detection Rate	0.987 (1448 of 1467 illegal flows)
False Positive Rate excluding reloads	0 (0 of 43 legitimate Twitter flows)
False Positive Rate including reloads	0.4 (17 of 43 legitimate Twitter flows)

$$FPR = P\left(t_{get} - t_u > T_w \,|\, user\ triggered\ flow\right) = 1 - \int_0^{T_w} p(t)dt \tag{3.7}$$

The function p(t) is the probability distribution of the response time . The observations in Section 3.2 did not lead to a well-defined model for this distribution, because many factors influence the response time of the computer. However it is considered safe to assume that $FPR < 0.01$ for the choice $T_w = 200ms$ in the case of the observed computer, because in our tests none of the 600 measured response times exceeded 163ms.

3.3.4. EXPERIMENTAL EVALUATION OF THE DETECTION ALGORITHM

We tested the algorithm in a real situation. The detector of Section 3.1 was implemented with time windows $T_{ug} = T_{ud} = T_{dg} = 200ms$, as explained in Section 3.3, and $T_{dd} = 10s$. In addition to logging, the traffic to Twitter.com, outside the specified time window was stopped by the realtime insertion of firewall rules. During 8 hours a client computer was used for text processing, email reading/writing, drawing, browsing, and of course using Twitter. At the same time the computer was infected by a bot. The used bot was a recent variant of the Twitternet bot as discovered by Nazario [92]. This variant is also known by Kasperski lab as Backdoor.Win32.Twitbot.c. It polls every 20 seconds a Twitter account and uses SSLv1. Of course in this case we could directly detect this flow by the rarely used SSLv1-protocol and the high query-rate. However, to test the performance of the our detection mechanism, we only used the algorithm of Section 3.2. Table 3.3 presents the results of the experiment. The results show a succesful detection of botnet traffic and support the theoretically derived performance of the detection algorithm.

We provide in Table 3.3 two values for the FPR. The 17 false positives were all caused by automatic reloads of already open Twitter pages. This results in a very high FPR of 0.4. However, in the case of an enterprise computer that only provides basic usage of Twitter, the effect of blocking all automatic Twitter traffic does not create a serious problem, because the user can still update the pages by a manual reload. If we consider all automatic traffic as undesired, the effective FPR in this experiment equals to 0. The issue of automatic traffic is further discussed in the next section.

3.4. SPECIAL CASES OF TWITTER TRAFFIC

In the previous sections we have assumed that legal traffic to Twitter.com could only be triggered by human activity and that botnet traffic was never triggered by human activity. Unfortunately, in reality the situation is more complex. We elaborate in this section on two important cases:

- automatic legal traffic by applications that poll Twitter periodically;
- user synchronized botnet traffic to evade detection.

3.4.1. AUTOMATIC LEGAL TRAFFIC

Automatic notifications, which were blocked in the experiment, are an example of legal automatic traffic to Twitter.com. In the case of Twitter we identify three important groups of legal automatic traffic:

- If a timeline or a search page of Twitter.com is loaded, the browser polls approximately every 30 seconds Twitter.com for new messages. New messages result default in a notification, but the page with the new content is not automatically loaded.
- A timeline or hash tag can be polled as RSS-feed. An agent loads a certain Twitter page in RSS-format periodically. Update frequencies are low (typically once a day for IE and once an hour for Firefox).
- A timeline or hash tag can also be polled by special agents like Tweetdeck. These applications periodically load timelines or search results of Twitter.com in .json, .xml, or .rss format.

In the absence of direct user activity, our detector will classify these traffic types as botnet-originated. We explore here some solutions to deal with this false positive problem but restrict ourselves to solutions that can be implemented on the network in the neighborhood of the client.

1. The most rigorous solution is instantly blocking this type of traffic, as we did in the experiment. In an enterprise network with computers that run a limited number of well defined applications this can be a viable option for the Twitter case, because employers can still use Twitter, but only manually.
2. A more elegant solution is complete inspection of the received traffic content. In this way legitimate triggers for new traffic (in addition to the user events) can be identified. This solution will be worked out in Chapter 4.
3. Additionally a white list can be applied, allowing automatic traffic to certain locations of Twitter.com. This can only work if the majority of this list is automatically maintained. For example the detector can allow Twitter locations that have successfully been visited by earlier user events. Also CAPTCHA's can be used, as a condition to manually add locations to the whitelist.

3.4.2. EVASION BY USER SYNCHRONIZED BOTNET TRAFFIC

The presented detector can only detect botnet traffic if the traffic is not synchronized with user activity. However it is possible to design a bot that holds the C&C traffic until a suitable user event takes place. In that case the detector will erroneously classify the traffic as human originated with a false negative as a result. We will refer this type of evasion as piggybacking.

At first glance, piggybacking on user events appears to be a technique that results in an increased C&C visibility that leads to new detection possibilities instead of evasion. In the area of keylogger detection, synchronisation between malware and keystrokes is a property that is used in detection approaches [97]. However we assume here that the bot will sparsely produce C&C-traffic, synchronized with user events. We discuss here some solution directions for piggybacking, related to Twitter.

1. Detection evasion by piggybacking requires the observation of user events. A host-IDS can detect background processes that poll or hook user events. In the case of enterprise computers, with well-defined applications this does not have to lead to many False Positives. The malware can evade this type of detection by disrupting the detection process or by modification of the programs that run legitimately in the foreground, however, this raises extra evasion difficulties for the bot.

2. A more complete observation of user activity, allows the detector to estimate the probability of traffic generation after a user event. This is possible by defining states that belong to certain user behavior. For example a high number of keystrokes in a short time can reveal that the user is text-processing. If Twitter traffic is started after the Enter-key and at the same time new key strokes are observed, the traffic can be classified as anomalous. Another detectable anomaly is the typing of a non-Twitter URL, directly followed by traffic to Twitter. Of course a sophisticated bot can wait until the right conditions are met, but in those circumstances there is a high probability of two simultaneous flows to Twitter.com, the bot flow and a user flow, which is again a detectable anomaly. The left mouse click is an attractive event for a bot to synchronize, because it is the most important trigger of legitimate Twitter traffic. Again observation of earlier and later events can lead to detection of an anomalous situation. For example in case of a double click, the bot has to start its communication before it can observe the possible second mouse click, because if it waits to long, the traffic will be outside of the causal window. The result is a detectable anomaly. Also the presence of other user activity immediately after the mouse click can indicate anomalous traffic.

3. The observation of received traffic content can be involved in the detection decision. An example of a detectable anomaly by this approach is the situation that a webpage is loaded that does not contain a link to Twitter and directly after a completion a mouse click is followed by a connection to Twitter.

The solution directions show that evasion by piggybacking also causes new detection possibilities. We will come back on piggybacking and evasion in Chapter 4.

3.5. CONCLUSIONS

In this chapter we explored a detection approach that can detect botnet C&C-traffic to Twitter.com, by observing only causal relationships between observed network traffic and observed user events that potentially can trigger traffic to Twitter.com. The three observed user events that can trigger a visit to Twitter.com are: the Enter key press, the Left Mouse Button release, and the F5 key press. Theory, combined with empirical data, predict an acceptable detection rate and false positive rate. An experiment with real user traffic and botnet traffic supports the capability of this approach to detect and optionally block immediately C&C-traffic traffic to Twitter in the case of enterprise computers with restricted applications. The detection approach is limited to traffic to Twitter.com, related with certain use cases. With its focus on user activity and Twitter-specific scenarios, this chapter must be seen as a first exploration of detection by causal relationships. The next chapter elaborates traffic flow causality in addition to causality between user activity and traffic. This results a in a more generic detection approach that is capable of detecting a large variety of botnets by building Traffic Causality Graphs.

4

DETECTION OF COVERT BOTNET COMMAND AND CONTROL CHANNELS BY CAUSAL ANALYSIS OF TRAFFIC FLOWS

The C&C communication of a botnet is evolving into sophisticated covert communication. Sophisticated techniques, such as encryption, steganography, and recently the use of social network websites as a proxy, often impede timely detection of botnet traffic by the existing techniques, as discussed in Chapter 2. The concept of causality between user activity and traffic, explored in Chapter 3, is extended in this chapter to a more generic causality model between traffic flows and between traffic flows and user activity.

Identifying the direct causes of traffic flows, allows for real-time specification-based anomaly detection of C&C traffic of many C&C traffic types. In addition it allows for offline forensic analysis of traffic. The proposed causal analysis of traffic is experimentally with traffic that included various types of Command and Control traffic.

This chapter is a minor revision of the paper with the same title published in the Proceedings of the CSS2013 Conference [20].

4.1. INTRODUCTION

The success of existing C&C detection techniques is often due to the presence of noisy attack traffic, as DDoS attacks, spam, and network scans. However, as discussed in Chapter 1, botnets can use sophisticated covert C&C communication that produces minimal noise. This is particularly true for espionage bots that infiltrate an enterprise network: there is no or very limited attack traffic and the C&C communication can be hidden by the imitation of popular traffic types, including the use of social networks as a proxy as described in Section 3. Secrecy can even further be enhanced by steganography and encryption. If these techniques are applied in the right manner and not related with other observable attack behavior, detection of C&C communication can become extremely difficult [91].

In this chapter, we introduce a new approach to detect covert communication by identifying direct causal relationships between network flows and prior events. We will refer to this type of detection as *Traffic Flow Causality detection* or *TFC detection*. TFC detection can be deployed as a network IDS. It addresses the common situation of typical client computers, as PCs or mobile devices, located in a LAN of a corporate network, and protected from the Internet by a stateful firewall as shown in Figure 4.1. TFC detection inspects passively the network traffic per computer. TFC detection includes the observation of user activity, as introduced in Chapter 3, obtained from the mouse and keyboard by additional hardware or a software agent, and transported to the TFC detector by a separate channel. An infected computer will regularly produce traffic by

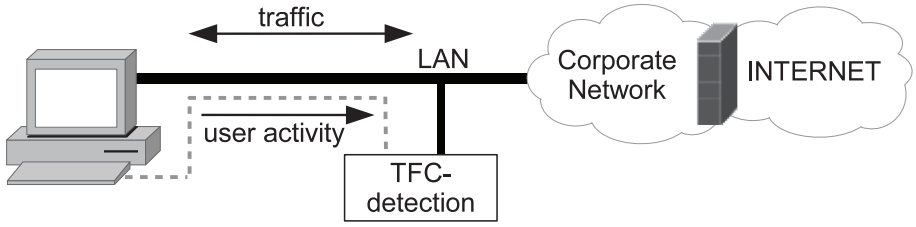

Figure 4.1: Deployment of TFC-detection in a LAN.

"phoning home" to a malicious entity in addition to the legitimate traffic. We assume this situation throughout the paper.

The captured traffic can be organized as bidirectional flows: an aggregation of all IP-packets between two computers, on both sides identified with a unique IP-address and a Layer 4 port. A stateful firewall forces all bidirectional flows to initiate from the client computer. Traffic flows are often caused by other traffic or user activity. An example is the visit to a website: A mouse click can trigger a DNS flow, followed by a HTTP flow to the resolved IP-address. The downloaded HTML object contains URLs of other HTTP objects that can result in new HTTP flows with or without intermediate DNS flows. The flows can be organized by their causal relationships, in a tree-shaped graph, as shown in Figure 4.2. We will refer to this type of graph as a *Traffic Flow Causality graph* or *TFC graph*. In addition we will refer to the first flow of a tree as the *root flow* and the tree itself as a *TFC tree*. If a new flow starts, it will be either a child, connected to an existing tree, or the root flow of a new tree. The result is a forest of TFC trees, growing on the event of

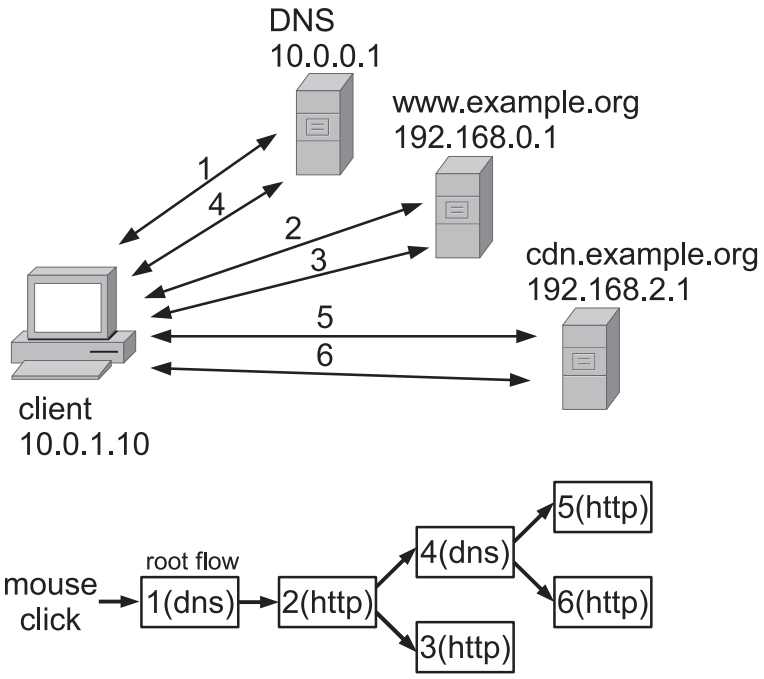

Figure 4.2: Scenario of a visit to the website Example.com by a mouse click with the TFC graph of the resulting traffic flows (vertices) and their direct causes (edges).

each new flow.

A TFC graph can be constructed by observing traffic and user activity, as mouse clicks and key strokes. An important step in the construction process is the selection of the most suitable event as the direct cause of a new flow, since there can be multiple events that qualify as potential cause of a new flow. We have developed for this selection the *Optimal Cause Selection algorithm* or *OCS-algorithm*. It evaluates the time between an event and a new traffic flow, and for some events, the presence of a reference to the destination of a new flow in the payload of prior traffic. If a root flow, initiated from a client computer, is not caused by user activity, it must be caused by an automatic process. This can be legitimate traffic, such as an automatic update of a normal application, but it can also be the C&C traffic of a bot instance. A root flow and its offspring are classified as anomalous, if the root flow is not caused by user activity. In addition, a whitelist can prevent anomaly classification of traffic that is caused by known legitimate automatic processes. This method allows for the detection of all types of "phone home" traffic. The detector does not need a state of prior malicious traffic, which makes it suitable immediate detection of one single C&C traffic flow, as described in Section 2.5.2.

Section 4.2 gives an overview of related work. Section 4.3 elaborates the construction of TFC graphs and the associated anomaly detection by the OCS algorithm. Section 4.4 presents CITRIC, a self-developed framework that implements TFC detection and

visualization of TFC graphs. Section 4.5 describes experiments, conducted with CITRIC and traces of normal and botnet C&C traffic. Section 4.6 elaborates evasion and improvements. Section 4.7 and 4.8 summarize conclusions and future work.

4.2. RELATED WORK

Our work involves anomaly detection by identifying direct causes of traffic flows, obtained from passively captured network traffic and user activity. There are many proposed and implemented anomaly detectors that analyze captured network traffic. We compare here our work with other work that involves associations between traffic flows.

Cui et al. present Binder [29], a detection approach that also measures causal relations between user events and traffic events. Zhang et al. propose a similar approach, called CR-Miner[140]. CR-Miner constructs Traffic Dependency Graphs that resembles our TFC graphs. However, both Binder and CR-Miner have fundamental differences with our approach:

- Both depend on host-internal information, as process IDs. Our TFC-detection captures passively traffic outside the observed host. Even the capture of user events can be implemented in hardware, outside the software environment of the host.
- CR-Minor constructs dependency trees by the Referer field in the HTTP header. To prevent tampering, CR-Minor signs from within the browser each HTTP header. This requires a browser that is not compromised by malware. However on an infected computer, the risk of browser compromise is significant. Another problem is that legitimate HTTP requests do not always specify the Referer field. In contrast our approach identifies direct causes by time measurements and matches between the destination of a new flow and a reference in the payload of an earlier flow. Forgery of the destination of a new flow by malware is difficult, since it will result in failed communication with the remote malicious instance.
- Our TFC graphs include all flows per host, including DNS, which allows a more complete analysis of covert communication.

Burghouwt et al. demonstrate the detection of a Twitter-based C&C channel by causally relating Twitter.com traffic with user events [19]. Our TFC detection extends this work, by identifying causal relationships between flows. In addition TFC detection does not depend on Twitter specific properties.

Karagiannis et al. present Blinc to classify traffic flows by their originating applications [72]. They use graphlets to represent flows that share L4 ports or addresses. Unlike our approach, no causal relationships are evaluated.

Iliofotou et al. introduce Traffic Dispersion Graphs that present connection patterns between different hosts [68]. Our TFC graphs are different, because they represent flows and causal relationships instead of hosts and connections.

Asai et al. map causal relations between flows in Traffic Causality Graphs to profile application traffic [5]. The resulting graphs are constructed without the use of destination references in the payload, resulting in a high uncertainty of the identified causal relationships. In addition user events are not taken into account.

4.3. CAUSAL ANALYSIS OF FLOWS AND ANOMALY DETECTION

A bidirectional flow or dialogue is the aggregation of IP-packets, exchanged between a local and remote computer, and identified by the 5-tuple {prot, IP_{local}, IP_{remote}, $port_{local}$, $port_{remote}$ }, referring to the IP addresses and port numbers of the local and remote computer. RFC5103 [127] refers to these bidirectional flows as biflows. The direction of a bidirectional flow is defined as the direction of the first packet. Since in the remainder of this paper all traffic flows are bidirectional, the adjective "bidirectional" will be omitted. In this section we present the algorithm that identifies causal relationships between flows, and detects anomalous flows by the cause of the root flow.

4.3.1. THE DIRECT CAUSE OF A FLOW

We define the direct cause of a traffic flow as the event that ultimately triggers the flow. In addition, there can be multiple indirect causes or additional preconditions that must be satisfied. If a web page shows a hyperlink and a user clicks on the link, the induced flow is directly caused by the mouse click and indirectly caused by the already open web page with the hyperlink.

Traffic events are the most common direct cause of a new traffic flows. An example is the reception of an IP-address in a DNS reply that is used immediately as destination of a new HTTP flow. Flows that are caused by traffic events contribute to the branches of trees, but can never be a root flow.

User events are certain user actions by mouse, keyboard, touch screen or another input device. Popular actions that can cause new traffic flows are the release of a mouse button and the press of the Enter key. Flows, directly caused by a user event, are always root flows.

Process events are state transitions in software processes that trigger automatically new flows, such as a software update or a check for new email. Most legitimate flows, caused by automatic processes, are well known per host and their destinations can be defined in a whitelist. Flows, directly caused by a process event, are always root flows.

Server events are new flows, initiated from external computers to the observed computer. This is only possible if incoming connections are allowed. Since this is normally blocked for clients behind a stateful firewall, we will not elaborate further on this event type.

A traffic flow is classified as anomalous if the direct cause is a process event, and the remote address or hostname is not whitelisted. Exact determination of the direct cause of a flow requires detailed analysis of the program execution of all processes inside the observed host. This is complex, platform dependent, and entails a high risk of detection and compromise by the malware. The solution for this problem is the selection of direct causes from traffic events and low level user events, both captured outside the monitored host. If no direct cause can be selected from the observed traffic and user events, and if the remote host is not whitelisted, then the flow is classified as malicious. However, the host-external approach is less accurate, because in some cases the direct cause of a new traffic flow must be selected from multiple available candidate causes. If the wrong cause is selected, a root flow can erroneously be absorbed as a branch in a tree or associated with a user event. The absorption of a malicious root flow in a tree of legitimate flows or its association with user event, results in a False Negative. On the

other hand, if a direct cause is not found, a flow can erroneously be dispersed as a root flow of a new tree. This can result in a False Positive.

4.3.2. OPTIMAL SELECTION OF THE DIRECT CAUSE

To select from the traffic and user events the most likely direct cause of a new flow, with minimal absorption and dispersion risk, we developed the Optimal Cause Selection Algorithm or OCS Algorithm.

Algorithm 4.1 OCS Algorithm for TFC graph building and anomaly detection.

for $each\ new\ flow$ **do**
  if $(findDNSEvent(X.IP_{remote}, T_{dns}))$ **then**
    $cause = DNSEvent$; $addToFoundTree(X)$;
  else if $findURLEvent(X.name_{remote}, T_{url})$ **then**
    $cause = URLEvent$; $addToFoundTree(X)$;
  else if $findUserEvent(X, T_{user})$ **then**
    $cause = USEREVENT$; $createTree(X)$;
  else if $findHTTPSEvent(X, T_{https})$ **then**
    $cause = HTTPSEVENT$; $addToFoundTree(X)$;
  else if $findHTTPEvent(X, T_{http})$ **then**
    $cause = HTTPEVENT$; $addToFoundTree(X)$;
  else if $isWhiteListed(X)$ **then**
    $cause = WHITELIST$; $createTree(X)$;
  else
    $cause = UNKNOWN$; $createTree(X)$; $signalAnomaly(X)$;
  end if
end for

For every new flow the OCS algorithm tries to find a direct cause by searching in succession through different types of events that have occurred in a defined time window before the start of the new flow. The algorithm differentiates traffic events in four different types. Combined with user events, OCS distinguishes five different types of events:

1. *DNS Event*: The detector caches recently captured DNS-lookups. If a new flow X is non-DNS, the function $findDNSEvent(X.IP_{dest}, T_{dns})$ searches in its cache for a DNS translation that matches the remote address $X.IP_{remote}$. If the most recent matching DNS-record is received within the time window T_{dns} before the start of the new flow X, the event is chosen as direct cause of X. Flow X will become a child of the DNS-flow that carried the record and automatically inherits the normal or anomalous classification of the related tree.

2. *URL Event*: The detector caches the hostnames of URLs, parsed from recently captured HTTP-payloads. If a new flow X is non-DNS, the remote IP address is first translated to a hostname, $X.name_{remote}$, by the DNS-cache. If the new flow is DNS, the hostname in the DNS-request is chosen as $X.name_{remote}$. Then the function $findURLEvent(X.name_{remote}, T_{url})$ searches for a cached URL with a matching hostname. If the most recent matching URL is received within the time

window T_{url} before the new flow X, the event is chosen as direct cause of X. Flow X will become a child of the HTTP-flow that carried the URL and automatically inherits the normal or anomalous classification of its tree.

3. *User Event*: Mouse clicks and specific key presses, as the Enter key are seen as events that can trigger new flows. An agent captures and sends user events from the observed computer to the detection system. Software implementation of the agent increases significantly the exposure to the malware. Malware with root privilege can suppress or mimic user events. The exposure can be reduced by the implementation of the agent in a hypervisor [60], or in hardware that reads the electrical connection of input devices [19]. The function $findUserEvent(X, T_{user})$ selects the most recent event as direct cause of flow X if it is within T_{url}. The flow is classified as a user caused root flow.

4. *HTTPS Event*: An HTTPS event is defined as a sudden decline of the received IP packet size in a flow. This occurs typically when the last part of a requested object is received. It does not necessarily indicate the termination of the flow, because in the case of a persistent connection, as in HTTP1.1, other object requests and replies can follow within the same flow. The completion of an object downloaded, indicated by a sudden decrease of the packet size, can initiate a new flow X that depends on the received object.
   The function $findHTTPSEvent(X, T_{https})$ searches for the most recent HTTPS event that occurred within the time window T_{https} before the start of the new flow. If a HTTPS event is found, Flow X will become a child of the HTTPS-flow and will automatically inherit the normal or anomalous classification of the tree.

5. *HTTP Event*: Similar as HTTPS, but with packet size decrease in HTTP traffic.

If no suitable event types are found and the remote destination is not whitelisted, the flow is classified as an anomalous root flow. In practice not all URLs will be identified in the payload. Typical causes of missed URLs are: TLS-encryption of the payload, as HTTPS, client caching of a prior received HTTP messages, and complex scripts that compose URLs from received code and data. This can result in tree dissection and false positives. OCS solves the problem of missed references by searching for the more generic HTTP and HTTPS events, after an unsuccessful search for a URL event. Since this search is only based on time and not on string matching, the related expiration windows T_{https} and T_{http} must be kept as small as possible, to reduce the risk of false negatives by absorption. For simplicity OCS evaluates only traffic reference events in DNS and HTTP traffic. However, this type of cause evaluation can be extended to other less popular traffic types that carry references in the payload, such as SIP traffic that carries IP-addresses for media streams.

4.3.3. DETECTION PERFORMANCE

In some cases TFC-detection can miss anomalous flows, by selecting the wrong event as direct cause. This is especially the case for the HTTP, HTTPS, and user events that are merely selected by their presence within a time window. The probability that an HTTP, HTTPS, or user event is erroneously selected as the direct cause of a malicious

flow, starting at a random moment, can be approximated by $T_{window}.f_{event}$ if $T_{window} << 1/f_{event}$ with f defined as the average frequency of HTTP, HTTPS, or user events. A True Positive is only possible if none of the event types is selected as a direct cause. The probability of erroneously selecting DNS or URL events is relatively small, because in addition to the presence of the event in a time window before the start of the flow, there must be a match of an IP address or a hostname. Assuming mutual independence between all events, the DR, as defined in Section 2.2.1, is estimated by Equation 4.1. We also assume that there is no intelligent evasion, by bots that piggyback on certain events by waiting for a suitable moment and optionally using popular servers in the C&C communication. We will discuss this type of evasion in Section 4.6.1.

$$DR \approx (1 - T_{user}.f_{userEvent}).(1 - T_{http}.f_{httpEvent}).(1 - T_{https}.f_{httpsEvent}) \tag{4.1}$$

Equation 4.1 shows that the DR benefits from small time windows. The frequency of user events can be minimized by optimal selection of only those user events that are really potential triggers of a new flow. Other solutions to improve the DR by removing the HTTP and HTTP factors are discussed in Section 6.

4.4. CITRIC: PRACTICAL IMPLEMENTATION OF TFC GRAPH CONSTRUCTION AND DETECTION

TFC graph construction and detection is evaluated by our self-developed framework, called CITRIC (Causal Inspection To Recognize Illegal Communication). The main components of CITRIC are:

- *Traffic sensor*: Device that captures passively real time internet traffic in PCAP format.
- *User event sensor*: Either a hardware device that is inserted between keyboard, mouse, and the computer, or a software agent to capture and signal user events.
- *Flow aggregator*: A software object that constructs bidirectional flows of captured IP-packets. When a packet is captured and the flow already exists, flow parameters are updated. In case of a new flow, the aggregator calls the appropriate analyzers, to place the flow in a tree.
- *DNS analyzer with cache*: A software object that analyses DNS traffic and caches name-to-IP translations.
- *HTTP analyzer*: A software object that searches for potential URL events, received in HTTP traffic. In addition it searches for generic HTTP and HTTPS events by monitoring the payload size between two consecutive ingress packets of the same flow. All potential events are temporarily stored in a cache.
- *Cause Analyzer*: The central software object that constructs TFC graphs by the OCS algorithm with adjustable time windows.
- *Anomaly detection alert*: A software object that logs important events and alerts detected anomalies

All components, with the exception of the user event sensor, are implemented in C++ on a Linux computer that bridges all LAN traffic. In addition to real-time capture, CITRIC can analyze files with prerecorded traffic in PCAP format, including signaled user events.

4.4.1. IMPLEMENTATION ISSUES AND SOLUTIONS

During development many implementation-specific issues had to be solved. The most important issues are summarized below.

1. Some operating systems use for DNS flows the same client port or a limited set of client ports. This creates a risk that different DNS queries are aggregated in the same flow. To prevent this, the DNS transaction number is added to the flow tuple, to keep the tuple unique.

2. Virtual hosting and Content Delivery Networks use IP addresses with multiple hostnames, resulting in ambiguity of the translation of an IP address to a hostname by cached DNS records. This is solved by first expanding the search for potential causal relationships to all possible names that map to the same IP of a new flow, and then choosing the most recent URL that refers to any of the hostnames.

3. In some cases scripts construct host names by combining strings. When there is no exact match between a hostname and the cached URLs, CITRIC will test a match of at least the 4 last characters of the second level domain name. In case of a well-known public suffix, such as .co in .co.uk, this partial match is expanded to the third level domain name.

4. To accomplish a fast search of matching destinations and references, hash tables are used. For the described experiments the hash is 8 bit, resulting in a 256 times smaller average seek time than in the case of a linear search. Similar hash techniques are implemented for fast DNS search by IP address and flow search by tuple.

5. Many popular websites only deliver gzipped HTTP replies. CITRIC can unzip and merge chunked HTPP replies.

6. If a DNS query fails or the answer takes too long, some DNS clients will swiftly repeat the question, resulting in multiple DNS flows that query the same name. This can lead to multiple root flows, instead of one. This is solved by combining identical queries that start within a small time window.

CITRIC is explained in more detail in Appendix A. The source is made publicly available [17].

4.5. EXPERIMENTAL EVALUATION

We have evaluated TFC detection experimentally by running CITRIC with four different traces of traffic, captured in a controlled environment.

The first trace is used for the evaluation of False Positives and contains the complete captured traffic from visits to the 30 most popular websites of the Internet. Visits

Table 4.1: Overview of the three different traces, infected with botnet traffic.

Bot	C&C type	Injected in Top30 trace
Kelihos	HTTP [35]	10 x DNS and HTTP flow
Tbot	Peer to Peer [26]	10 x TCP flow, port 9001
Twebot	Twitter as proxy [92]	10 x DNS, HTTP, and HTTPS flow

to popular websites result in many additional traffic flows, caused by advertisements, mesh-ups, scripts, etc. The number and variety of direct causes tests the TFC detection under difficult circumstances. Missed causes will result in false positives. The popular websites were derived from the rankings of Alexa [2] and Google [53]. All doubles were removed and of the remaining sites only the 30 most popular were used. Each visit started by typing the name in the browser address bar, followed by at least one typical activity, as a login on Facebook, a search in Google, etc. Traffic was captured and stored in PCAP-format by Gulp [112], a capture tool with a low probability of packet loss. The visiting computer was a laptop with a fresh installation of Windows 7 and a Firefox browser with plugins for Flash and Java. At the start of the capture both the web cache and DNS cache were emptied. A software agent, installed on the visiting host, captured mouse clicks and key strokes and transmitted every event as a special UDP-packet. Hence the events were automatically part of the captured traffic. The resulting trace, to which we will refer as Top30, contains 113505 packets, representing 4179 flows. The most popular protocols are DNS, HTTP and HTTPS. Since we assume that the captured traffic is not contaminated with malicious traffic, every detector alert is a false positive.

The other traces, intended for the evaluation of the C&C detection, are composed of the Top30 trace with injected C&C traffic. For each trace we infected a clean Windows 7 instance with real malware and manually isolated exactly one representative tree of the captured C&C traffic. The malicious tree was injected in the Top30 trace at ten different equidistant times. We developed for the injection special software that could modify packet timestamps, IP-source addresses, and colliding ephemeral L4 port numbers of the malicious traffic. The result was a consistent trace with exactly ten separated C&C trees, during the thirty legitimate website visits. We composed in this way three infected traces in PCAP format, each with a different type of C&C traffic as shown in Table 4.1.

4.5.1. EMPIRICAL DETERMINATION OF THE OPTIMAL WINDOWS SIZES

The accuracy of a TFC graph depends on the size of the five time windows as defined in Section 4.3.2. To minimize the FPR, all windows must be chosen as large as possible. However, Equation 4.1 shows that an increase of T_{user}, T_{https}, and T_{http} results in a decrease of the DR, because these windows, in contrast to T_{url} and T_{dns}, are not accompanied with a string match condition. This does not imply that T_{dns} and T_{url} can be chosen arbitrarily large, because this will decrease the DR, if the covert channel uses a popular server as intermediary, as explained in Section 4.3.3. For the determination of the optimal value of T_{dns} the unnormalized cumulative distribution of the DNS delay, $CDF_{dns}(T)$ has been obtained by measuring in the clean Top30 trace the time between each DNS reply and the first usage of the IP address in the A-record. We use an unnormalized distribution function to express the results directly as flows. Delays,

during which one or more user events take place, are excluded, because in those cases it is not clear what triggers the new flow. As shown in the graph of Figure 4.3, the resulting $CDF_{dns}(T)$ bends sharply to an almost horizontal line at T=1ms, indicating that most DNS related delays are smaller than 1ms. About 90% of 726 measured delays is less than

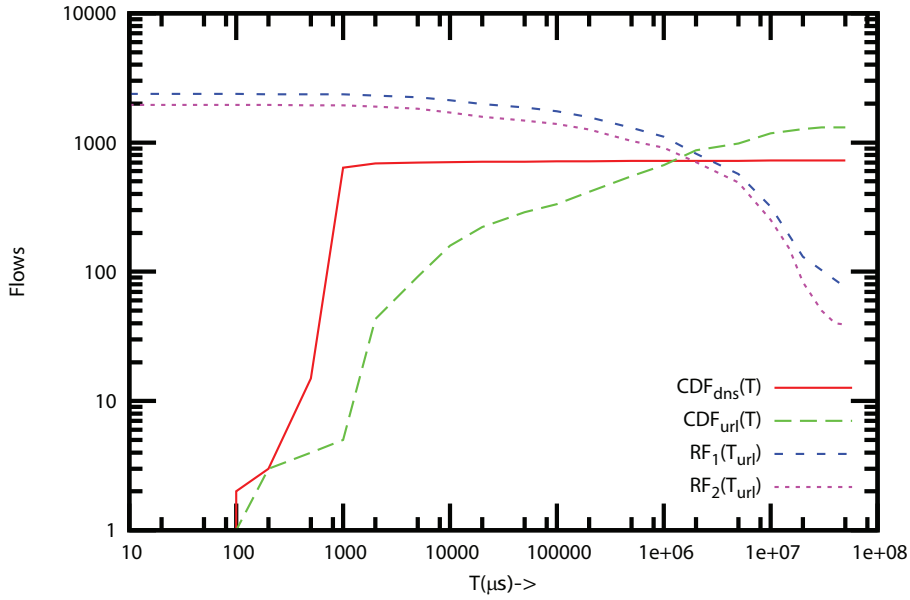

Figure 4.3: Cumulative flow distributions and root flows as a function of the applied time window. $CDF_{dns}(T)$ and $CDF_{url}(T)$ are the unnormalized cumulative distribution functions of flows that are respectively caused by a prior DNS or a prior HTTP flow. $RF(T_{url})$ is the number of identified root flows with varying window T_{url}. $RF_1(T_{url})$ does not apply an HTTPS window, $RF_2(T_{url})$ applies an HTTPS window of 500ms. Other settings can be found in the text.

2ms. We choose T_{dns}=500ms as an optimal time window, because the 6 flows with delays above 500ms are all related with DNS prefetches from distant websites that loaded very slowly.

To obtain an optimum value for T_{url}, $CDF_{url}(T)$ is measured in a similar way as described for $CDF_{dns}(T)$. In this case the distribution does not show a sharp bend. This implies that we need a large window size T_{url} to identify all flows that are caused by a received URL. However, a large size of T_{url} increases the risk that user caused root flows are absorbed in other trees, since the OCS algorithm, evaluates the URL events before user events. Although this does not affect directly the FPR, the constructed TFC graph will become less accurate. To analyze this effect, the algorithm was run for different values of T_{url}, with T_{dns} set on 500ms, and all other time windows set to zero. Figure 4.3 shows the number of root flows by $RF_1(T_{url})$. Manual inspection revealed that the undesired absorption of user caused root flows occurs at window sizes above 10s. To prevent this, T_{url} is set to 10s. However, with this window size, URL-based relations with a delay of more than 10s are missed. We solved this problem by adding a second findURLEvent() test with a window size of 30s immediately after the findUserEvent() test.

Manual inspection also revealed that the observed long delays are caused by missed references in HTTPS traffic and to a more limited extent by missed references in HTTP traffic. To compensate for the missed references in HTTPS traffic T_{https} is set to 500ms. With this value there is no absorption of user caused root flows by prior HTTPS events. To visualize the effect of the applied HTTPS window, the algorithm was run again for different values for T_{url}, with T_{dns}=500ms, T_{https}=500ms, and with the other windows equal to zero. $RF_2(T_{url})$ shows the number of root flows. Compared to $RF_1(T_{url})$ the number of flows is lower. The steepest point of the $RF_2(T_{url})$ curve is around t=20s. Finally we ran the OCS-algorithm for different values of T_{user}. The False Positive Rate (FPR) can directly be derived from the number of detected anomalous flows, since the Top30 trace did not contain malicious flows. The expected Detection Rate (DR) for the Top30 trace was indirectly calculated by Equation 4.1. Figure 4.4 shows the influence of T_{user} on the FPR and DR in an ROC graph. In general, an ROC graph shows the effect of a parameter on the DR and FPR of a detector [43]. The ROC curves of three different setups are displayed:

- Setup 1: Detector with T_{dns}=500ms, T_{url}=10s/30s, T_{https}=500ms, T_{http}=*0s*. Optimum at T_{user}=100ms with FPR=0.005 and DR=0.95.
- Setup 2: Detector with T_{dns}=500ms, T_{url}=10s/30s, T_{https}=500ms, T_{http}=*50ms*. Optimum at T_{user}=100ms with FPR=0.0017 and DR=0.85.
- Setup 3: All windows zero except T_{user}. Unsuitable for detection

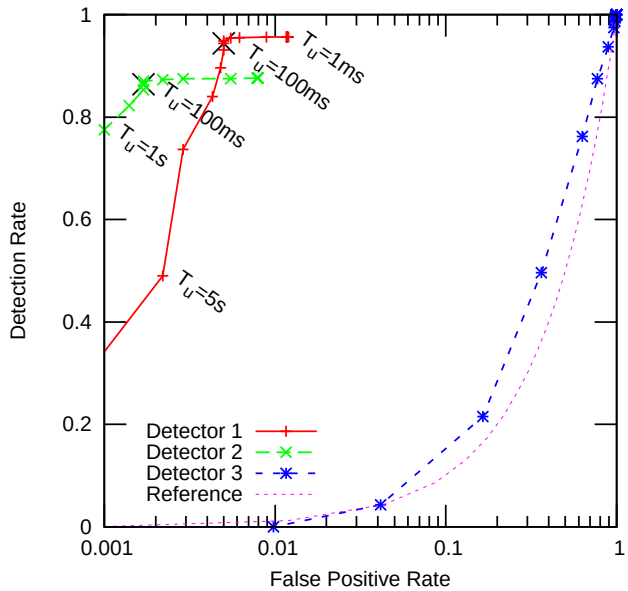

Figure 4.4: ROC graph of TFC detection with different settings and a varying T_{user}.

Table 4.2: Experimental performance of TFC detection with 4 different traces.

Trace	FP	TP	FPR	DR
cleanTop30	7	-	0.0017	-
Top30+Kelihos	7	8	0.0017	0.8
Top30+Tbot	7	8	0.0017	0.8
Top30+Twebot	7	7	0.0017	0.7

The ROC graph shows that Setup 3 is totally unusable. The reason that we tested Setup 3 is to show that a simplified algorithm, that only evaluates user events as direct causes, does not work. The ROC graph of Setups 1 or 2, shows that TFC detection can optimally detect covert channels for T_{user}=100ms. In all cases only the communication of the user event agent was whitelisted.

4.5.2. TFC DETECTION OF REAL C&C TRAFFIC

The Detection Rate (DR) of Section 4.5.1 was indirectly estimated by the non-overlapping accumulated time of the open windows in the clean Top30 trace and was not really measured with malware. Therefore we also tested TFC detection with the three malware infected traces. The applied window sizes are equal to Setup 2 of Section 4.5.1 with T_{user}=100ms. Table 4.2 shows the results. The observed DR is close to the estimated value of Section 4.5.1. False Negatives are the result of (1) the absorption of malicious root flows in legitimate traffic, caused by the non-zero windows T_{http} and T_{https}, and (2) the wrong classification of root flows as user-caused, by the non-zero window T_{user}. One malicious tree of the TweBot C&C traffic was absorbed in a legitimate tree with by a URL to Twitter.com. The experiments show that TFC detection performs, as predicted in Section 4.3 and it detects real C&C traffic, including the covert traffic of social media based C&C.

4.5.3. VISUALIZATION OF THE TFC GRAPHS

TFC graphs reveal causal relations between flows. This can be used in forensic situations to isolate a tree of related flows. We have built a tool as an extension of CITRIC to visualize each tree separately by post-analysis of the CITRIC logs. Flows are represented by the vertices, annotated with the most important properties as destination and protocol. Direct causes are represented by the edges, annotated with the event type and the delay between parent and child. The cause of the root flow is indicated as an extra annotated vertex with specification. Optionally colors can indicate the protocol. Figure 4.4 shows an automatically generated image of a visit to Google.com. The tree starts with a user event. After some DNS and HTTP redirection flows, the main page and additional objects are loaded. A large number of DNS stubs in one of the branches of the tree is caused by browser DNS prefetching.

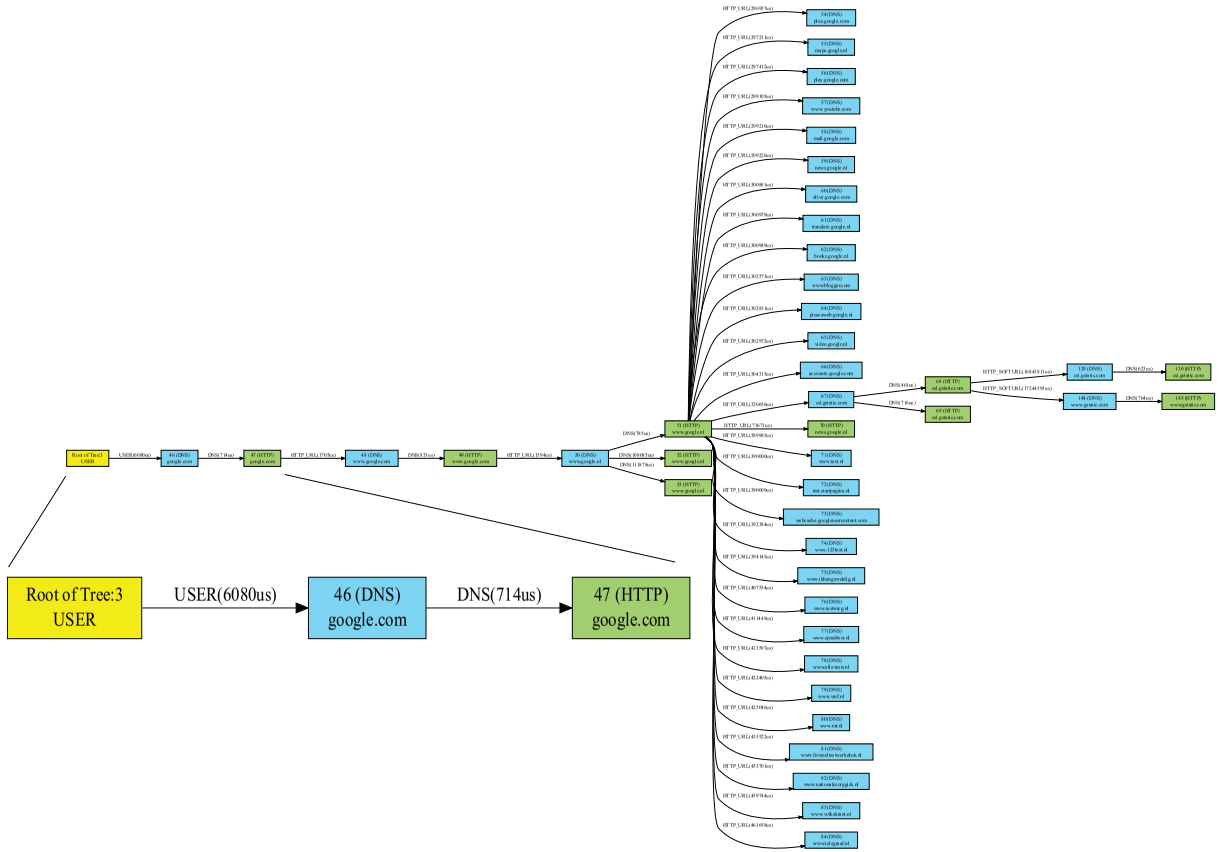

Figure 4.5: Part of a TFC Graph of a visit to Google.com, automatically visualized by CITRIC. For readability of the generated captions in this figure, the initial part of the tree is manually magnified.

4.6. EVASION AND RELATED IMPROVEMENTS OF TFC DETECTION

Discovery of TFC detection by malware is difficult, because both traffic and user activity are captured passively and can completely be implemented outside the software environment of the observed computer. If the malware is aware of TFC detection, it can adapt its communication to evade detection by piggybacking on legitimate events. The malware monitors traffic or user events and waits for a suitable moment to initiate communication. In general successful piggybacking requires specific privileges, to monitor in the background network traffic and user activity. Monitoring these type of events is an anomalous activity that can be detected by a host IDS, especially in the case of computers in enterprise networks with well defined legitimate applications.

4.6.1. SOLUTIONS AGAINST PIGGYBACKING

Piggybacking on HTTP and HTTPS events can be prevented by removing these events from the OCS algorithm. HTTPS events were introduced, to compensate the inability of the OCS-algorithm to inspect the payload of HTTPS traffic for the more selective URL events. This problem can be more fundamentally solved by a TLS/SSL interception proxy that is trusted by the observed client computers. If a client computer uses the certificate of the proxy, all communication can be decrypted in the LAN or proxy, and the more selective findURLEvent() function can find potential URL events, making the findHTTPSEvent() superfluous.

HTTP events were introduced to compensate the inability of finding all potential destination references in the HTTP payload. Sometimes URLs are on-the-fly composed by client scripts. Improvement of the search for destination references in HTTP traffic can be achieved by feeding the received payloads to an environment that emulates the browser engine of the client, to compose references in a similar way as the browser. This would make the findHTTPEvent() superfluous.

With the removal of the HTTPS and HTTP events, it still remains possible to piggyback on user events. A possible solution is a more complete observation of the user input, to estimate the likelihood that a particular root flow is generated by a particular user event. An example is the search of potential host and object names in the typed input from the user, and matching these with destinations of new flows. This is not possible with mouse clicks on hyperlinks, but inspection of recently received hyperlinks and whitelisting can limit the number of possible new destinations after a click. A totally different solution is the replacement of the direct capture of user events by alternative heuristics that select the direct cause of a root flow by the likelihood that a destination reference is clicked or typed by a user. This is a component of the UDI-detection approach that will be presented in Chapter 5.

Piggybacking on URL and DNS events is much more difficult than the other events types, because the malicious and legitimate communication must now simultaneously visit the same destination. This can be the case when popular websites, as social media, are used as proxy. This type of piggybacking can be countered by not only matching the hostname, but also the path and object of the URL reference. An example is the matching of Twitter.com/tlab32768 instead of Twitter.com. It is highly unlikely that both

malware and legitimate processes visit the same resource on the same host within a short time interval. This type of solution will be further discussed in Chapter 5.

4.7. CONCLUSIONS

By identifying the direct cause of traffic flows, it is possible to organize the traffic in tree-shaped graphs and detect C&C communication by anomalous causes. The OCS algorithm selects the optimal direct cause for each new traffic flow from passively captured traffic events and user events. Experiments with representative popular HTTP traffic and different types of malicious C&C traffic support the effectiveness of TFC detection. While TFC detection allows for real-time detection of all types of covert traffic, it is particularly suitable for detection of covert botnet C&C to popular websites.

In addition to real-time detection, the visualization of the constructed TFC graph can be used in forensic analysis.

TFC detection is a typical enterprise-specific countermeasure that anticipates the enterprise-specific characteristics of Section 2.5.2. Control over the network allows for the capture of URLs and IP-addresses, as *Application Data* in the payload of HTTP and DNS. Control over the hosts and the connected hardware allows for the measurement of user activity. Causality depends only on knowledge that origins from *specification* and *statistics*. Finally it does *not* require a state that depends on prior *malicious activity*, hence immediate detection of one single C&C instance is possible. This allows for complete repression of all C&C traffic.

Although user events need to be captured in addition to network traffic, the risk of compromise by the malware of infected hosts can be kept low by an implementation outside the software environment of the observed computers. A complete host-external hardware implementation minimizes the risk, but introduces deployment complexity. In some enterprise environments, such as a VDI environment, the external observation of user events is relatively easy, because all user activity transported over the network[88].

While FTC detection can successfully and feasibly detect malware, evasion of the detection is possible by starting C&C traffic in a time window directly after after a specific event. Different types of detection enhancements are possible to impede such an evasion.

5

DETECTION OF BOTNET COMMAND AND CONTROL TRAFFIC BY THE IDENTIFICATION OF UNTRUSTED DESTINATIONS

Although the causal detection of Chapter 4 is capable of real-time detection of low volume C&C traffic, even if popular traffic is imitated, deployment is difficult in some enterprise networks, because of the required observation of user events.

In this chapter we present an alternative approach that is also capable of detecting Command and Control traffic in an enterprise network. Instead of causality, the trustworthiness of the traffic destinations is used for classification. If the destination identifier of a traffic flow origins directly from: human input, prior traffic from a trusted destination, or a defined set of legitimate applications, the destination is trusted and its associated traffic is classified as normal.

In addition to the real-time detection of zero day malicious traffic, this approach does not depend on time measurements and special agents for the observation of user events. This results in a less complex implementation, a low exposure to malware by the completely passive host-external traffic monitoring and the prevention of evasion by piggybacking. Experimental evaluation demonstrates successful detection of diverse types of Command and Control Traffic.

This chapter is an extended version of the paper with the same title, published in [21]

5.1. INTRODUCTION

As discussed in Chapter 1 and 2, it is crucial for an organization to identify infected computers in its premises. Infected computers can attack other computers, steal sensitive information and disturb critical production processes. Infected computers are often bot instances that participate in a botnet. In addition to normal traffic, they produce malicious traffic, consisting of occasional connections or *phone home calls* to a C&C (Command and Control) entity on the Internet and optionally they generate attack traffic, such as DDoS and spam.

In this paper we present a new approach to detect botnet activity in an enterprise network. Similar to the approaches of Chapter 3 and 4, this is an anomaly-based approach that does not depend on misuse-related knowledge, such as signatures [109] or blacklists of malicious hosts [16][38]. Also similar to the approaches of Chapter 3 and 4, no prior botnet activity is required in contrast to many other anomaly-based approaches that evaluate correlation between different malicious traffic flows [57].

Instead of evaluating causal relationships between traffic and user events, this approach is based on trust of traffic destinations. Trust is a complex concept and can be defined in many different ways. We use a context-specific definition of trust, derived from a more generic definition from Olmedilla et al. [95]. In our context, which is an enterprise network with inside potentially bot-recruited computers, we define trust as *the measurable belief of the organization that a specific entity does not collude in a botnet.* We assume that the organization trusts its employees and a defined set of legitimate software applications if deployed on an uninfected computer. On the other hand, the enterprise computers with the installed OS and software instances, are not trusted, since they can be compromised and recruited in a botnet. Traffic destinations are initially not trusted, because they can be part of a C&C infrastructure that is contacted by an inside bot. However, a destination becomes trusted by transitivity, if its *identifier* origins from another trusted entity. The *identifier* of a destination can be an IP-address, name, URI, or any other data that is used to direct the traffic to a remote computer or resource.

Evaluation of the origin of *destination identifiers* enables the detection of C&C traffic. Traffic is classified as normal, if the destination identifier origins directly from: human input, a legitimate application, or the received content from a trusted destination. All other destination identifiers are not trusted and the associated traffic is classified as anomalous.

We will refer to this anomaly detection approach as *Untrusted Destination by Identifier Detection* or *UDI Detection.* Section 2 describes the details of UDI detection. Section 3 presents a practical implementation for experimental evaluation. Section 4 evaluates UDI detection by experiments with real traffic. Section 5 elaborates evasion possibilities. UDI detection is compared with other work in Section 6. Finally Section 7 concludes and proposes future work.

5.2. UDI DETECTION APPROACH

For UDI detection we assume the typical scenario of client computers in a segment of an enterprise network, protected by a stateful firewall, that blocks all traffic that is ini-

tiated from outside. This enforces inside bots as the initiator of all C&C communication(*phone home*). All traffic from and towards the inside computers is passively captured and evaluated by the UDI detector as shown in Figure 1. To limit the number of detection decisions, the UDI detector organizes all traffic in flows by protocol, source and destination IP address, and UDP/TCP port numbers. In Chapter 4 we worked with bidirectional flows, to optimally explain the concept of causality. In this chapter we use the more common definition of a unidirectional *flow* that refers to an ingress or egress flow. This definition allows for a better explanation of the detection mechanism.

The stateful firewall assures that each ingress flow is associated with exactly one existing egress flow with swapped IP addresses and ports. The detector evaluates the egress flows on trust of their destinations. An egress flow is only classified as normal if its destination is trusted. Ingress flows inherit the trust and anomaly state of their associated egress flows.

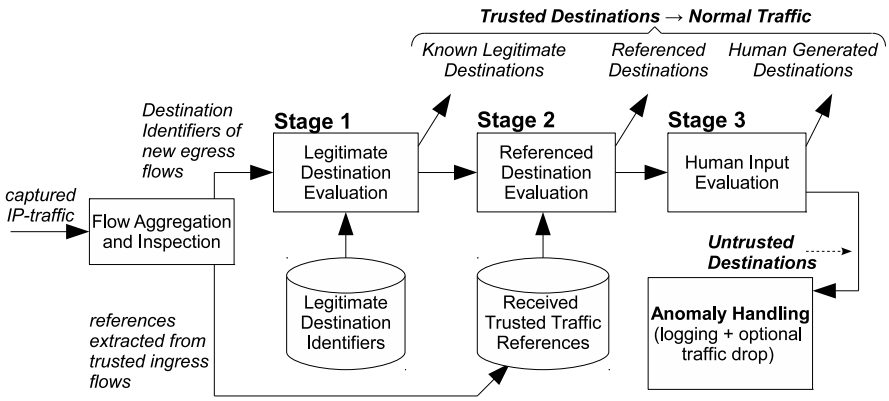

Figure 5.1: Schematic overview of UDI detection.

For each new egress flow, trust is determined by the the origin of its *destination identifier* in the three consecutive decision stages of Figure 1.

The first stage tests if the destination identifier is present in a predefined set of legitimate destinations, used by trusted applications. This typically includes destinations of servers for software updates, browser home pages, and local management traffic. Flows to these destinations are classified as normal and not further evaluated.

The second stage tests if the destination identifier matches a reference that was received in the payload of prior ingress flows from a trusted destination. Examples of such references are URLs in HTTP content and IP-addresses in DNS replies. If the destination identifier matches a reference, the destination is trusted, and the associated flow is classified as normal and not further evaluated. If there is no match, the destination identifier is forwarded to the third stage.

The third stage evaluates the remaining destination identifiers on the likelihood of being directly inputted by a human. We assume that humans normally enter destinations that can be distinguished from machine-originated input by differences in complexity and surprisal. For example, humans will normally not type very long names or

IP-addresses. These and many other features can be used in heuristics to differentiate between human and machine origin. If the destination identifier is estimated as human input, the destination is trusted and its flow is classified as normal and not further evaluated. The remaining destinations identifiers represent untrusted destinations that belong to flows that are likely automatically generated by illegal processes.

The combination of the three stages results in a system that can immediately detect botnet phone home traffic, even if it has a low volume and uses popular traffic types, to stay below the radar of existing Intrusion Detection Systems. The passive monitoring and real-time classification of UDI detection, allow for implementation in an edge-router, or a network Intrusion Prevention System, to prevent any contact between an inside bot and outside C&C entities.

The necessary deep packet inspection of all received traffic payloads and the management of a set of known trusted legitimate destinations, are especially feasible in enterprise networks. Deployment in the networks of public ISPs with connected consumer devices and home networks is more difficult, due to the high diversity of consumer end systems, the lack of control and transparency in consumer networks, and privacy regulations.

5.2.1. LOGICAL DESTINATION IDENTIFIERS

Before further elaborating UDI detection, we present a more precise definition of the destination identifier of a flow, and will refer to this as the *logical destination identifier* or *ldi*. We assume a local computer that initiates an egress flow X to a remote destination that is identified by ldi_X, as defined by Equation 1.

$$ldi_X = \textit{host-id}_X + \textit{resource-id}_X \tag{5.1}$$

- The *host-id* identifies the contacted remote host of flow X. It is determined by the destination IP address of the flow as shown in Equation 2. A computer normally acquires a destination IP address by the translation of a hostname, performed by a translation service, in most cases DNS. If the translation is observed before flow X, the host-id will be the hostname. In all other cases it is directly the IP-address.

$$\textit{host-id}_X = \begin{cases} hostname(IP_{dest,X}) & if\ hostname(IP_{dest,X}) \neq 0 \\ IP_{dest,X} & if\ hostname(IP_{dest,X}) = 0 \end{cases} \tag{5.2}$$

  In this formula $IP_{dest,X}$ is the destination address in the IP-header of egress packets of flow X. The function *hostname()*delivers the IP address, if a valid translation exists from hostname to IP address. Otherwise *hostname()* is 0.

- The *resource-id* in Equation 1 identifies a specific resource of the remote host. It is extracted from the payload of the egress flow. If not present in the payload, the *resource-id* is defined as zero. An example of a resource-id is the *path/querystring*, used in a HTTP GET request. In this particular example the complete *ldi* is very similar to a URI. A completely different example is an ICMP flow of a ping. In this case the *resource-id* in the *ldi* is zero.

The basic assumption of UDI detection is the low probability that a trusted destination belongs to the C&C infrastructure of a local bot-infected computer, or provides *ldi's* of the C&C infrastructure. However this assumption does not hold for trusted destinations that deliver translation services, such as corporate DNS-servers. Malware from infected computers can contact the DNS-server for resolving a hostname of a C&C server. In such a case the trust stateof the DNS-server should not transfer to translated destinations. Therefore the *ldi* of an egress DNS flow is defined by the query sent to the resolver for translation (Equation 3).

$$ldi_X = query_X \qquad if\ X = DNS\ flow \tag{5.3}$$

This means that a DNS flow, towards a DNS-server with a query that refers to an untrusted destination, is classified as anomalous. If an anomalous DNS flow is not immediately blocked, the received IP-addresses in the DNS answer are not placed in the list of trusted received traffic references, despite the fact that they are delivered by a trusted corporate DNS server. DNS is by far the most important protocol that generates resolver flows, but there are other protocols that can be regarded as resolvers or translators, such as the Bittorrent Tracker Protocol, that resolves hashes of file names to IP-addresses of peers.

5.2.2. FORWARD REFERENCE EXTRACTION

We define a forward reference as a data element in the payload of an ingress flow that can be used as the *ldi* of a future flow. It can range from a URL in a HTTP hyperlink to an IP-address in a DNS A-record. The adjective *forward* is used, to emphasize that the reference refers to the *ldi* of a future flow. It should not be confused with the *Referer* field in an egress HTTP request that refers *back* to the server that delivered the URL of the new request in a prior flow. For UDI detection all potential forward references in received payloads are stored in a list. The size of the list is limited by defining a maximum allowed validity time of unused forward references. In addition the complete list of references, received by one local computer can be cleared after a reboot of that computer.

Forward references are important in UDI detection, because they transfer the trust state from the destination of a prior flow to the destination of a new flow. This is illustrated in Figure 2.

5.2.3. THE UDI DETECTION ALGORITHM

The three stages of Figure 1 identify *ldi's* of trusted destinations. After the three stages, the remaining *ldi's* represent destinations that are not trusted and their associated flows are classified as anomalous. Algorithm 1 shows the complete detection procedure.

- *isEgress(X)* is true if X is an egress flow
- *IdentifyDestination(X)* extracts the *ldi* from flow X according to equation 1 or equation 3 for respectively non-resolver or resolver flows.
- *isLegitimate()*, *isReferenced(ldi)*, *isUserSubmitted()* are the tests of the three consecutive stages of Figure 1.

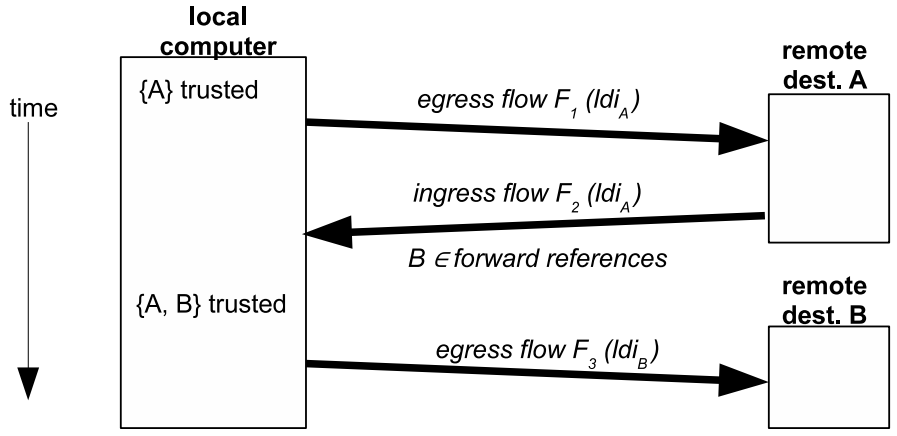

Figure 5.2: The remote destination B of egress flow F_3, identified by ldi_B is trusted, because it was referenced in a prior ingress flow F_2 of trusted destination A.

5

Algorithm 5.1 UDI detection algorithm

```
for each new flow X do
  if isEgress(X) then
    ldi = identifyDestination(X);
    if isLegitimate(ldi) or isReferenced(ldi) or isUserSubmitted(ldi) then
      X.Status = NORMAL;
    else
      X.Status = ANOMALOUS;
      signalAnomaly(X);
    end if
  else
    X.Status = getStatusOfAssociatedFlow(X);
    if X.Status = NORMAL then
      extractForwardReferences(X);
    end if
  end if
end for
```

- *getStatusofAssociatedFlow(X)* is NORMAL or ANOMALOUS, depending on the state of the associated egress flow of ingress flow(X).
- *extractForwardReferences(X)* will extract and store the forward references from trusted payloads.

5.2.4. DETECTION ERRORS

To elaborate UDI detection errors, we firstly introduce two classifications for *ldi's*.

1. A *malicious ldi* is the *ldi* of a destination that is used by a bot for a connection to

its C&C. All other *ldi's* are in this context *non-malicious*.

2. A *trusted ldi* is the *ldi* of a destination that is classified by the UDI detector as trusted. An egress flow with a *trusted ldi* is normal. An egress flows with an *untrusted ldi* is anomalous. An ingress flow inherits the normal or anomalous status from its associated egress flow.

In the ideal situation the UDI Detection algorithm will classify exactly all malicious *ldi's* as untrusted, and all non-malicious *ldi's* as trusted. However practical imperfections of the detector will introduce classification errors, resulting in False Negatives and False Positives. We treat here the two most important error sources: partial *ldi*-matching and selection of human-input features.

- *Partial* ldi-*matching*: The first and second stage of UDI detection use a list with respectively legitimate destinations and forward references. False Positives and False Negatives are directly related with the accuracy of the lists. An incomplete list increases the probability of false positives. To keep the list of legitimate destinations short and maintainable, it is convenient to use *partial matching*: only a part of the *ldi* has to match for classification as trusted. The match could be limited to the host-id of the *ldi* or even to just the domain-name of the host-id. A problem of partial matching, is the increased probability that malicious *ldi's* are erroneously classified as trusted, because of a partial match.

  The second stage uses a list of forward references, obtained from the payloads of ingress flows. Extracting complete forward references from payloads can be very complex. For example URLs in HTTP are often relative or dynamically composed from different elements in the payload by a client script. If the detector does not fully emulate the involved browser, this leads to missed forward references, that can cause false positives. Partial matching can also in this case reduce the false positives, however with an increased risk of false negatives, as in the first stage.

- *Selection of human-input features*: The heuristics to test the likeliness that the *ldi* is from human input, depends on a proper selection of features. Literature suggests many features to classify anomalous names [137] [85] [9] [82] [11], but non of the proposed feature sets is perfect, with false decisions as a result. A significant advantage of UDI detection is the removal of many non-human *ldi's*, such as references in prior flows, by the two preceding stages.

The complex design choices, make it difficult to derive a simple quantitative predictive model of the FPR and DR, as defined in Section 2.2.1. Instead we evaluated empirically the behavior of UDI detection and the resulting FPR and DR. Other causes of errors, related with detection evasion, are discussed in Section 5.5.

5.3. DETECTOR IMPLEMENTATION

We constructed a basic UDI detector as a proof of concept and evaluated its accuracy in experiments with real traffic. We implemented the UDI detection algorithm in our

framework CITRIC that we also used for evaluation of TFC detection. Two network interfaces make the system applicable as a bridge in a LAN and allow for real-time inspection with optional removal of bridged traffic by the detector. The bridge can also be configured as a stateful firewall. In addition to real-time detection, the captured traffic can be stored in pcap format for offline evaluation by CITRIC. The traffic is in this case captured by Gulp [112], a capture tool with a low probability of packet loss. Tables of forward references, *ldi's*, and flows are in CITRIC implemented with hash tables, to speed up the search for existing flows and *ldi's*. Tables, intermediate results, and detection decisions are extensively logged for later manual evaluation. Further details of CITRIC can be found in Appendix A. The source is publicly available [17].

5.3.1. PARTIAL *ldi*-MATCHING

To limit the complexity of payload parsing in this proof of concept, only the payloads of DNS and HTTP are inspected for forward references and partial *ldi*-matching is applied, as explained in Section 2.4. The extracted forward references are limited to the host-id part of Equation 1. The *ldi*-matching in the 2nd stage for DNS host names is limited to the TLD and at least 4 characters of the second level domain. If the second level domain is a well-known public suffix, such .co in .co.uk, 4 characters of the third level domain name are also included.

5.3.2. NAME-BASED CRITERIA

For the function *isUserSubmitted()* of Algorithm 1, we derived three name-based features from [9], [11], and [82] to test if an *ldi* origins directly from a human:

1. number of characters $\preceq$ C
2. number of non-letter characters $\preceq$ N
3. top level domain $\in$ {set of popular human-input TLDs}

The result of *isUserSubmitted()* is only true, if all three conditions are true. The optimal value of C, N and the set of popular human-input TLDs depends on the behavior of the local average user. In particular the set of TLDs depends on the nationality of the user and the location of computer. For example in The Netherlands the TLD *.nl* is popular, along with some international TLDs, such as *.com* and *.org*.

5.4. EXPERIMENTAL EVALUATION

The experimental evaluation of UDI detection has two objectives:

1. Determine the performance of practical UDI detection with different types of C&C traffic. The FPR is determined by feeding the UDI detector in a controlled environment with a defined set of real non-malicious traffic. The DR is determined by feeding the detector with various types C&C traffic, embedded in non-malicious traffic.
2. Demonstrate that each of the three stages contributes significantly in the reduction of the total FPR. In the case of a non-malicious flow, one of the stages has to

classify the *ldi* as trusted. If a stage does not classify an *ldi* as trusted, it is passed to the next stage. The fraction of remaining *ldi's*, passed from a stage x to the next stage, is expressed by H_x in Equation 4;

$$H_x = \frac{\#ldi_{stage\,x,not\,trusted}}{\#ldi_{stage\,x,non\,malicious}} \tag{5.4}$$

After the third stage the flows of the remaining untrusted *ldi's* are classified as anomalous. Since we assumed normal traffic, these are the False Positives. Equation 5 expresses the FPR (False Positive Rate).

$$FPR = H_t = H_1.H_2.H_3 \tag{5.5}$$

Although the three stages are derived from a coherent model that is based on the origin of *ldi's*, it is difficult to exclude hidden dependencies in sieving properties between the stages. This can result in a stage that does not a have a net contribution in the reduction of untrusted *ldi's*. By changing the sequence of the stages, the ratios per stage H_1, H_2 , and H_3 can change, by dependencies in sieving properties of the proceeding stages, but the overall FPR will not change. If one of the three stages has no net contribution, placement as the third stage will equal the ratio H_3 to 1.

5.4.1. CONTROLLED ENVIRONMENT

We evaluated False Positive behavior, True Positive behavior, and the dependency between stages, of the UDI detector with traces of both normal traffic and malicious C&C-traffic. All traffic was produced by, and captured from computers in a controlled environment:

- The normal traffic was generated by the use of popular applications and the visits to popular websites from computers with a freshly installed operating system and software.
- The C&C traffic was generated from computers, infected with real well-known botnet malware.

Due to corporate regulations and law, it is difficult in a large enterprise network to capture, store, and analyze traffic with complete payloads for experimental evaluation and review of the detection approach. Evaluation of UDI detection in just a small sample of an enterprise network is feasible, but results in a high risk that the traffic is very homogeneous with a limited number of different destinations. UDI-detection will then produce an optimistic False Positive behavior, caused by the fact that the production of new destinations is relatively low. In addition the probability of infection by various types of bots with sophisticated phone home traffic is small, resulting in unreliable information about the True Positive behavior. By testing in a controlled environment, we could evaluate UDI detection more accurately because:

- By the selection of a wide variety of legitimate applications, including many popular websites that result in abundant traffic to referred destinations, the False Positive behavior is evaluated under difficult conditions. This prevents a to optimistic estimation of the FPR.

- By the selection of different types of botnet traffic, combined with clean legitimate traffic, the malicious part of the traffic is precisely known, resulting in an accurate evaluation of the True Positive detection per type of C&C traffic.

5.4.2. EVALUATION OF FALSE POSITIVES AND STAGE CONTRIBUTION

For evaluation of the False Positive performance, traffic was generated by 40 selected cases of preinstalled applications and web applications, all commonly used by students of our university. Although some of the applications, used by students, are not expected to be present in a corporate environment, we chose for this selection, to test the detector under difficult conditions by a wide variety of applications. Examples of the cases are: Email with a stand-alone client and a webclient, participation in several social media, usage of Google Maps and Street View, planning of a journey by Dutch public transport, communication by WhatsApp, games and downloading. Depending on the case, the traffic was produced by a Windows 7, Linux, or Android device. The applied list of legitimate *ldi's* was kept as small as possible and consisted only of: the IP-addresses in the same local subnet, the domain names of OCSP servers of well-known certificate authorities, and the domain names of servers that were configured in the installed legitimate applications for automatic updates, home-pages, etc. DNS and browser caches of the evaluated systems were cleared before the experiment, to start synchronized with the UDI detector, as needed for a proper functioning of the referenced destination evaluation.

All collected traces were evaluated by the detector. The parameters of the function *isUsersubmitted()* were chosen: C=20, N=3 and *{.com, .org, .net, .nl, .uk, .de, .gov}* ∈ TLD. Two particular cases resulted in an excessive number of false positives ($FPR > 0.5$). They were isolated from the other traces and further manually examined. The first case was a download with Bittorrent. Since our implementation of UDI detection cannot extract the peer IP addresses of encrypted tracker information, all peer to peer connections were classified as anomalous. The second case was an Android game that connected continuously to different destinations. Since both cases are not representative for corporate usage, they were excluded from further FPR calculation. We will discuss these types of false positives and possible solutions in Section 5.

The traces of the remaining 38 cases contain 24362 flows with 54% HTTP, 8% of HTTPS, 36% DNS, and 2% of other traffic. Since all cases were produced with freshly installed software, we assume no C&C traffic. Consequently every flow, classified by the detector as anomalous, is regarded as a False Positive. The FPR was calculated by the fraction of the False Positives in the total number of flows and resulted over all 38 cases in an FPR of 0.0026 (64 False Positives in 24362 flows). Additional analysis revealed that web traffic to Content Delivery Networks and advertisement-related traffic were responsible for the majority of the false positives. The *ldi's* were referred in prior encrypted HTTPS payloads that could not be inspected by our implementation of the UDI detector. Nevertheless the number of False Positives caused by this shortcoming remained relatively low, because HTTPS-objects often share the same domain name as the referring web page. Also many entry pages are not encrypted but contain references of *ldi's* to https-objects. We will also further elaborate this in Section 5.5.

The individual effect of each of the three *ldi* evaluating stages is evaluated, by placing each particular stage in turn as the last stage and measuring the *ldi* Ratio, H, as defined in Equation 5.4. The results are shown in Table 1. All stages reduce the net FPR, since

Table 5.1: Measured *ldi* ratios of the last stage (H_3) and the combined preceding stages ($H_1.H_2$) for different sequences. L=Legitimate *ldi* state, R=Referenced *ldi* stage, U=Human Input stage.

Last Stage (H_3)	H_3	$H_1.H_2$
U	0,28	0,0097
R	0,0038	0,70
L	0,083	0,032

all H_3-values are significantly smaller than 1. This supports our model of the *ldi* origin in UDI detection. Table 1 also demonstrates that the referred *ldi* stage is the largest contributor to the reduction of False Positives, since its *ldi* ratio is the smallest. The small contribution of the name complexity filter, is caused by the fact that only a small number of flows have an *ldi* that is directly typed by the user.

5.4.3. EVALUATION OF TRUE POSITIVES

For analysis of True Positives, traces with a mixture of normal traffic and malicious command and control traffic were composed. This approach results in a well defined ground truth situation. The malicious traffic consisted of C&C traffic of well-known bot malware. Five botnet instances were selected to cover different types of C&C traffic:

1. HTTP-based C&C by Kelihos [35] with DNS and HTTP-traffic
2. Peer-to-peer-based C&C by Storm [66] with Kademlia-based UDP-traffic
3. Social medium-based C&C by Twebot [92] with HTTP and HTTPS web traffic to a Twitter timeline.
4. TOR-based C&C by TBOT [26] with TOR TCP traffic on port 9001
5. DNS-based C&C by Morto [87] with DNS traffic to *ms.jifr.co.cc*.

For the normal traffic we used the Top30 trace that was also used for the evaluation of TFC detection (Section 4.5). In the same way three of the five C&C traces were also used in the TFC detection experiments. The normal traffic the TOP30 trace was generated by visits to the 30 most popular global websites, derived from Alexa [2] and Google [53]. For each website visit, a typical functionality of the website was used, such as a login, a search, playing a clip, or a product selection. All traffic was captured and collected in one trace from a PC with a fresh Windows 7 installation with a Firefox browser, including popular plugins. Since we work ni this chapter with unidirectional flows instead of bidirectional flow all flow numbers are doubled. The trace contains 8358 flows (or 4179 biflows). Due to the popularity of the websites, the trace is a representative sample of web traffic. In addition popular websites truly test the effectiveness of the implemented referring *ldi* stage, because the received HTML objects contain a massive

number of forward references that cause auxiliary flows, such as media, advertisements, and mesh-ups. Missed references can result in false positives.

The applied list of legitimate *ldi's* was limited in this experiment to just the IP-addresses in the same local subnet and the domain names of OCSP servers of well-known certificate authorities.

For each C&C trace, the flows of one representative call-home effort, were manually isolated from captured traffic. With a self-developed tool we injected 10 copies of a C&C traffic sample in the normal traffic. Our tool modified the packet positions and timestamps of 10 injected C&C copies to spread the phone communication equally over the entire observation interval of the 8358 legitimate flows. In addition the tool modified IP-addresses and port numbers of the C&C traffic to create one consistent trace with both normal and C&C traffic, originating from the same computer, without conflicts in the used ephemeral TCP and UDP ports. Table 2 shows the number of measured True Positives and the resulting DR of the traces with injected C&C flows. For reference the trace without C&C traffic is also evaluated.

Table 5.2: Measured FPR and DR of UDI detection with 1 clean and 5 infected traces.

trace	phone home calls	malicious flows	TP	FP	DR	FPR
Top30 clean	0	0	0	16	-	0.0019
Top30+Kelihos	10	40	40	16	1	0.0019
Top30+Storm	10	20	20	16	1	0.0019
Top30+Twebot	10	60	0	16	0	0.0019
Top30+TBOT	10	20	20	16	1	0.0019
Top30+Morto	10	20	20	16	1	0.0019

All injected flows of Storm and TBOT are detected because the *ldi's* are not from known trusted applications, unreferenced, and bare IP-addresses. The phone home calls of Kelihos start with a DNS lookup of a hostname in the *.ru* domain. These DNS flows and the traffic to the resolved IP-address are therefore classified as anomalous, resulting in a DR of 1. The injected DNS-only C&C traffic by Morto is detected by the query of *ms.jifr.co.cc*. Similar to the lookup of the *.ru* domain, the *ldi* is not from a known legitimate application, neither from prior trusted traffic, nor from human input. The C&C traffic from Twebot is not detected, because the *ldi* is *Twitter.com*, which is a simple name that could have been entered by a human. Additionally *Twitter.com* is referred by other legitimate traffic. The inability to detect C&C traffic that uses popular hostnames, is caused by the partial *ldi*-matching that excludes the resource-id from the evaluation. A solution for this problem is proposed in the next Section.

5.5. EVASION OF UDI DETECTION AND SOLUTIONS

There are several ways for a bot to evade UDI detection. One approach is directly demonstrated in our experiments: by using a popular server as C&C, there is a high probability that the detector will classify the *ldi* as trusted. This is caused by partial *ldi*-matching, as demonstrated in our experiments with Twebot. The complete *ldi* of the C&C in our experiment was *Twitter.com/tlab32768*, including the timeline of a malicious account, but

due to partial *ldi*-matching, the malicious resource-id was omitted and only the host-id *Twitter.com* was evaluated. This resulted in classification as trusted, because other objects of *Twitter.com* were already referenced by prior flows and additionally *Twitter.com* can origin from user input by its low complexity. The solution is a complete *ldi* match instead of a partial. This requires two techniques:

1. *encrypted payload inspection* A significant part of modern traffic uses TLS, such as HTTPS. The encrypted payload prevents the extraction of resource-id's from egress flows, and forward references from ingress flows. Complete *ldi*-matching would then result in a large number of false positives. In our experiments this was reduced by partial *ldi*-matching. However a better solution is the insertion of an SSL/TLS-interception proxy. By the installation of a public-key certificate on clients in an organization, a trust relationship can be established between the observed computers and the proxy that enables decryption of TLS-traffic, without certificate warnings of browsers [71]. Resource id's and forward references can now completely be extracted from the decrypted traffic. This allows for complete *ldi*-matching and prevents false negatives by the use of popular hosts. The appliance of an SSL-interception proxy changes the detector from a passive into an active device, since traffic is intercepted, decrypted, and encrypted. Although this theoretically results in more exposure of the detector to bots, it is our belief that it will not increase significantly the risk of compromise.

2. *browser emulation* Modern websites use complex client scripts that can construct URL's by dynamically combining different elements from ingress payloads and even user input. This complicates the complete extraction of *ldi's* and forward references with an increased risk of false positives. As explained, partial *ldi*-matching is a simple solution for this, but introduces evasion possibilities by inaccurate matching. A solution is the extraction of complete forward references in the UDI-detector by emulation of the clients browser.

It is evident that both techniques include complex and processing-intensive payload analysis that requires further research.

UDI detection can also be evaded by completely different techniques:

1. *The botnet controls a trusted destination.* This is for the botnet a complex type of evasion because it has to recruit at least two instances: a local computer as the inside bot and an external host with a trusted destination. Additionally in case of a takedown it would be difficult to replace the C&C destination.

2. *The botnet acquires a hostname that can origin from human input.* This also raises problems for the botnet, since *human-friendly* hostnames are often occupied, and again in case of a takedown, the replacement is difficult. In our experiment we only used three simple static rules to classify *ldi's* from human input. The use of more features and machine learning can result in a more accurate classification that can adapt to specific situations.

3. *The botnet does not phone home.* Nagaraja et al. proposed a botnet that piggybacks C&C messages in pictures that were exchanged between members of social

media [91]. In this model there are no phone home connections, which makes it undetectable for UDI detection. However it requires the recruitment of at least two instances that exchange content by a popular social medium. Generally this is a difficult condition to achieve, with again problems in case of a takedown.

5.6. RELATED WORK

The combination of legitimate destination evaluation, referenced destination evaluation, and human input evaluation, distinguishes UDI detection from the other detection approaches.

5.6.1. WORK RELATED WITH FLOW ANALYSIS IN CONSECUTIVE STAGES

Detection of C&C traffic by flow-based analysis over several consecutive stages is a common approach. Strayer et al. propose a behavior-based detection system of IRC-based C&C [121]. Like our method they apply several consecutive stages to isolate the malicious traffic, with the first stage as a coarse filter to reduce processing in the other stages. However, unlike our *UDI* detection, the approach focuses on IRC and uses statistical flow-based and topological properties that depend on the presence of multiple infected bots.

Walsh et al. describe a first pass filter that uses simple statistical flow attributes to select flows for further analysis [133]. They only focus on this first filtering stage instead of a complete detector.

5.6.2. WORK RELATED WITH LOGICAL DESTINATION REFERENCING

The second stage of our UDI detector tests if the *ldi* of a new flow is referenced in the received payloads of prior flows. Zhang et al. propose CR-miner [140], a system that evaluates traffic dependencies between connections and user events, to determine malicious automatic traffic. CR-miner associates a new connection with earlier references and user input. In contrast to our method CR-miner is implemented in the observed computer itself, since it needs user and process properties for classification. This significantly increases the exposure level to potential malware. In addition CR-minor uses a different method for associating flows: the Referer field in the HTTP header of a new connection is used to determine if the flow was previously referenced by another flow. This method is only applicable for HTTP traffic that supports this field and it can be easily manipulated by malware, since it is produced by an application in a potentially infected computer. Our method is not sensitive for this type of tampering, because forward references are captured from payloads of ingress flows that origin from other computers and because *ldi's* cannot be manipulated, without changing the egress flow destination.

Burghouwt et al. use causal relationships between flows to detect botnet C&C traffic [20]. Instead of the destination, the direct cause of a flow determines if communication is initiated by malware. Unlike *UDI* detection this demands for the accurate measurement of the delay between certain events and induced new flows. Another difference is the required monitoring of user events by a software agent or a hardware device.

Whyte et al. present a detector of scanning worms by determining IP-addresses

that are not earlier seen in DNS-replies or received HTTP-data [135]. This can be seen as a special case of flow referral, that isolates flows with unreferenced destination IP-addresses, as is often seen with worms.

5.6.3. WORK RELATED WITH HUMAN INPUT EVALUATION

Several name-based properties of hostnames and URL's have been proposed, to detect malicious destinations. Since there is not a unique name-based property that can decisively classify anomalous names, the proposed techniques use multiple lexographical and non-lexographical features, often combined with machine learning. Alphanumerical frequencies are used in work of Yadav et al. [137] and Mc Grath et al. [85]. Length of hostnames and substrings is used in work of Mc Grath [85]. et al. and Bilge et al. [9]. The number of dots and other delimiters in URL's are used by Ma et al. [82] and Blum et al. [11]. Our human-input evaluation is different in two ways from other approaches. Firstly the human-input classification in UDI detection is preceded by a stage that removes all traffic with referred destinations. This reduces the FPR. Secondly most name-evaluating detectors produce a list of malicious names for blacklisting. UDI-detection can classify in real-time and allows for an immediate drop of an anomalous flow.

5.7. CONCLUSIONS

UDI detection detects different types of stealth C&C *phone home* communication in an enterprise network by the trustworthiness of contacted destinations. The destinations of egress traffic flows are classified as trusted or untrusted by three consecutive stages that evaluate the origin of the involved *ldi* (logical destination identifier).

With its capability of immediate detection of just a single C&C flow, this approach can complement existing network-based anomaly detection approaches in enterprise networks. The initial set of trusted destinations, which can be seen as specification-based knowledge, is feasible to maintain in a typical enterprise network.

Partial *ldi* matching allows for a relatively simple and feasible UDI-detector implementation. The results of experiments with C&C traffic of real bots and normal traffic support the detection approach with a low FPR and an accurate detection of various types of C&C traffic.

The evasion possibilities, discussed in Section 5.5, are all related with the partial ldi-matching in our proof of concept. UDI detection with complete *ldi* matching, will solve this problem, however its implementation is much more complex by the required SSL traffic interception, payload parsing by browser emulation, and the evaluation of more user related features by an appropriate machine-learning algorithm.

6

DETECTION OF BOTNET COMMAND AND CONTROL CHANNELS BY ANOMALOUS DEGREES OF DNS DOMAINS

DNS-based detection of Command and Control traffic is an attractive approach because DNS is a popular Command and Control component and its traffic is relatively easy to capture and evaluate.

In this chapter we introduce a new DNS-based detection approach, that detects botnet collusion by the degree of queried domains. An important property of our detection approach is scalability, which allows for observation of large groups of computers over multiple networks.

We evaluate evasion possibilities, derive a theoretical model of the expected degree distribution with its related False Positive Rate, and test the detector with captured Internet traffic.

This chapter is a revision of the paper with the same title published in the Proceedings of the ICITST2010 Conference [18].

6.1. INTRODUCTION

DNS-based detection of botnet Command and Control (C&C) traffic is an attractive detection component in enterprise networks. This can be explained by three factors:

- DNS is used in many botnet C&C infrastructures. By its popularity, DNS traffic does not raise suspicion. DNS supports *phone home* communication. This is important, because in enterprise networks most computers, including bots, are guarded by stateful firewalls and NAT, which only permit connections that initiate from inside to outside. Another advantageous C&C property of DNS is the availability of techniques that complicate repression of central C&C servers, such as DGA (Domain Generation Algorithm) [9] and IP fast-fluxing [93]. We will discuss DGA in Section Section 6.3.3.

- In an enterprise network it is relatively easy to capture all DNS traffic. One way to accomplish this, is to allow only DNS queries through an enterprise DNS server. Such a policy is easily enforced by internal firewall rules. This facilitates the inspection of all DNS traffic in or in the proximity of the enterprise server.

- The payload of DNS-traffic is relatively easy to evaluate, because it is relatively small, highly structured in predefined fields, and not encrypted.

We propose a new detection approach of C&C DNS traffic that evaluates the degree of queried domains. We define the *domain degree* as the number of different computers within a network that successfully query, by A or AAAA records, the IP-address of a hostname from a remote domain. Popular domains will have a relatively high degree. We assume that a botnet has significantly more bots than active C&C domains. This influences the degree of the malicious domains. If enough bots contact the same domain, its degree can raise to a detectable level. False positives, caused by the popularity of well known legitimate domains can be reduced by additional filtering. The measurement of the domain degree allows for a relatively easy aggregation over a very large enterprise network or a combination of multiple enterprise networks. Detection does not only identify C&C domains, but also the IP-addresses of bots that query the domains.

Section 6.2 of this chapter discusses related work. Section 6.3 elaborates the concept of domain degree, the accumulation over different networks, and the expected accuracy of the proposed detection approach. Section 6.4 compares measurements of the degree distribution of real traffic with theory. Section 6.5 discusses evasion techniques and countermeasures. Section 6.6 summarizes the most import results.

6.2. RELATED WORK

Many DNS-based detection approaches of botnet C&C or other malicious traffic have been proposed. We will discuss here important work that covers different types of approaches.

Zdrnja et al. propose passive monitoring of DNS traffic and relate it with historical information, to detect Spam domains. They describe anomalies, like non-existent domains, typo squatter domains, A-records with a high number of changing IP-addresses,

and record reputation. The approach uses anomalies in the content of the DNS-records and not domain degree [138].

Ramachandran et al. propose detection by the measurement of reconnaissance lookups in DNS-based blacklists[106]. This does not directly detect the C&C traffic of a botnet.

Choi et al. [23] describe a detection mechanism, called BotGAD. Like our approach, BotGAD searches groups of computers that contact periodically the same domain. BOTGAD collects IP-addresses of these computers per contacted domain, to construct groups and determines similarity between those groups in different time intervals. Unlike BotGAD, our detection system does not depend on the periodic measurements in different time intervals, nor the correlation between groups of IP-addresses of the DNS clients. The correlation of IP-addresses of all involved DNS-clients requires a processing effort that makes BotGAD less scalable than our approach.

Bilge et al. propose Expose, a detection system that uses 15 different features, derived from the DNS payload [9]. It can detect special domain names, that are generated by DGA (Domain Generation Algorithm). The feature set does not include IP-adresses of the querying bots, which is necessary for the determination of the domain degree.

Villamarin et al. propose the detection of botnet DNS traffic to non-existent domains by NX-replies, which is an indication of DGA-produced domain names [130]. Our approach does not depend on features that are direct indicators of DGA.

6.3. DETECTION BY DOMAIN DEGREE

Detection is accomplished by observing the DNS-traffic of a large group of potentially infected computers and counting per queried DNS domain the number of different computers that successfully resolve an IP-address. We assume that a malicious domain is queried by multiple infected bots, which results in an unexpected *popularity* of that domain. Of course the malicious domain will not be the only popular domain. By filtering well-known legitimate domains, we expect that malicious domains become distinguishable from the remaining domains by their degree. Before we explain the complete detection approach, we will first elaborate the distribution of the domain degree.

6.3.1. DISTRIBUTION OF THE DOMAIN DEGREE

Figure 6.1 shows a network of three domains that are requested by an observed population of 8 computers. The word *network* does not refer here to computer networks, such as enterprise networks, but to a type of graph that represents associations between DNS-clients and requested domains. The nodes or vertices of the network represent the computers in the observed population and also the queried domains. The links or edges represent the DNS requests by the clients. From graph theory, we define the degree as the amount of computers in the observed population that successfully query a domain during a defined time interval. If C is the total number of observed computers, and k is the degree of a specific domain, the value of k will be limited by:

$$1 \leq k \leq C. \tag{6.1}$$

The degree is at least 1, because only domains are considered which are requested by at least one computer during the observation interval. If all computers visit the same

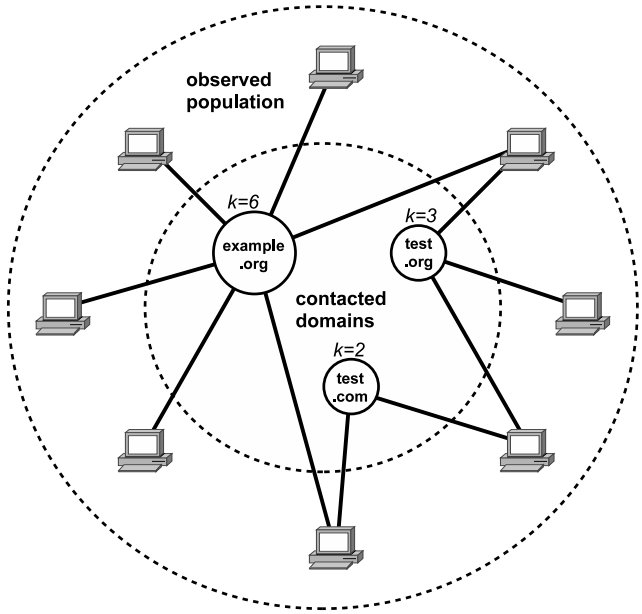

Figure 6.1: Network consisting of an observed population of computers and the domains, queried at least one time during some time interval.

domain, this domain will have a degree of C. In practice the degree of a domain is a statistical property, that is related with the number of observed clients, the observation time and popularity of the domain.

Barabási and Albert discovered that many networks are scale free with a power law decay in the distribution of the degrees [7]. One of the investigated networks was the World Wide Web, with the web pages as vertices and the links from other pages as edges. The degree distribution shows a power law decay, with a power of -2.1 which is typical for a scale free network. An explanation for this distribution is *preferential attachment* [7]. In the case of web pages this means that if a web page has a relatively high degree by its popularity, the degree will likely grow faster then the degree of less popular pages, because it is better known and easier to find. The World Wide Web is by far the most important way to navigate on the Internet to resources. Most URLs contain a domain name, hence making it plausible that the degree distribution of domains shows a similar power law decay. Therefore we propose the hypothesis that all domains, contacted by the observed computers, including bots and non-bots, will have a power law distribution in their degrees, estimated by:

$$\begin{aligned} P(k) &\approx A.k^{-\lambda} | 0 \leq k \leq C \\ P(k) &= 0 | k > C \end{aligned} \tag{6.2}$$

- C = the total number of observed computers
- λ= the power with a value close to -2
- A = a probability-normalizing constant, forcing the sum of all probabilities to 1.

6.3.2. DETECTION PROCESS

Our detection approach captures recursive DNS traffic between the resolvers (DNS clients) and the DNS server in an enterprise network. If a domain is requested for the first time and a valid reply is returned, the detector will store the domain name in a table. The IP-address of the requesting client is also stored in the table, associated with the requested domain. If the same computer successfully requests the domain again, the detector will not take any action, because the IP-address is already registered. However if another computer successfully requests the same domain, its IP-address will also be stored in the associated list. In this way the number of IP-addresses that belong to a particular domain, is the degree of that domain. The use of the same malicious domain by multiple bots will result in relatively high degree. Ideally if the degree is above a predefined threshold, the domain is classified as anomalous. The associated list of IP-addresses will also reveal all potential bots that queried the anomalous domain. Unfortunately there is a complication in this detection approach. Many popular legitimate domains have a degree that is significantly higher than the degree of a C&C domain. Adjusting the degree threshold to a value that detects the C&C domains, will cause a high number of False Positives by legitimate domains that have a higher degree. We define the FPR (False Positive Rate) as the ratio of False Positive domains to the total number of non-C&C domains. This is equivalent with the probability that a non-C&C domain d_i has a degree k_i above the threshold k_{th}:

$$FPR = \frac{FP}{TN+FP} = P(k_i > k_{th} | d_i \notin C\&C)\,. \tag{6.3}$$

FP = number of False Positives
TN+FP = number of True Negatives and False Positives = total number of non-C&C domains

Because we only consider non-C&C domains, the FPR of the detector equals the cumulative probability $P(k > k_{th})$ of the distribution of Equation 6.2:

$$FPR = P(k_i > k_{th} | d_i \notin C\&C) = \sum_{k=k_{th}+1}^{\infty} (P(k) = \sum_{k=k_{th}+1}^{C} A.k^{-\lambda}\,. \tag{6.4}$$

- k_{th} is the preset degree threshold. The detection of malicious domains that are only requested by a few bots, requires a very low k_{th}. This results in many false positives by popular non-malicious domains.
- C = the total number of observed computers
- λ= the power with a value close to -2
- A = a probability-normalizing constant, forcing the sum of all probabilities to 1.

To solve the presented problem we introduce the filtering of well-known domains. Some well-known domains, such as *Google.com* will have a very high degree, due to their popularity. With a list of such domains, they can be classified as normal, despite their high degree. There are several heuristical approaches to compose such a list:

1. *The use of a public list of well-known domains*: An example is the Alexa-ranking [1]. Alexa.com maintains a list of popular domains, measured by browser toolbars that are world-wide installed on a large group of computers. An API delivers the absolute ranking of a specific site. We assume that a malicious domain that is used by a limited number of sophisticated bots is not present in the ranking. Of course the botnet could use or hijack a very popular domain, such as the domain of a popular social medium. We will discuss this type of evasion and other types of evasion in Section 6.3.3.

2. *The use of a custom list of locally popular domains*: A custom list contains domains that have local popularity. These domains can be derived from network specific properties, such as the legitimate applications in the network or the nationality of its users. The limited diversity of computers and traffic in enterprise networks support this approach.

3. *The use of a baseline*: A special list of popular domains can be constructed by an initial measurement and analysis of DNS requests.

It is evident that the suggested filter approaches can also be combined, to identify and exclude more domains from the degree evaluation.

To obtain a detectable degree of a malicious domain, sufficient bots must be present in the observed network. By aggregation of observed degrees over multiple enterprise networks the average number of bots that visit the same domain will increase. Aggregation will also increase the degree of well-known popular domains, but these are removed by the proposed filter step.

Detection by degree distribution scales very well over a large enterprise network and even a large group of enterprise networks. Within each enterprise network it requires a monitoring system, consisting of a table of requested domains, with the associated IP-addresses of the requesting clients. The system can be implemented in the enterprise DNS server. The tables of different DNS servers or even different enterprise networks can be aggregated in a central system. Not all information has to be delivered to the central system: a list of domains with the degree of each domain is sufficient. In each network, different clients are contributing to the degree of a domain, hence the central system does not need to know the IP-addresses of the individual clients for correct accumulation of the degrees per domain. The necessary storage and associated processing in the central system will scale linearly with the number of contributing enterprise networks.

6.3.3. DETECTION EVASION

Most of the evasion-strategies proposed by Stinson et al. [119], such as encryption, packet- or flow-level noise, and even IP-churn are ineffective against our detector approach, because there is no dependency on traffic content, flow characteristics, or IP-addresses. However there remain several evasion possibilities:

- *The use of a C&C mechanism without DNS*. This evasion is trivial, because the detection observes the degree of DNS domains. However, the botmaster will loose

an attractive C&C option, because DNS provides a simple, popular, and robust service.

- *Long and irregular delays between attacks and the C&C communication.* This requires a large observation interval or a large number of observed hosts, to capture multiple visits to a C&C domain. If the observation interval is too short, it will result in a lower detection probability, because not every bot in the observed population contributes to the degree value of the C&C domain. However, this evasion technique will reduce the utility of a bot, because its control becomes less responsive. The calculation of degrees per DNS server and the easy limited size of aggregated information over different DNS servers allows for long observation intervals.

- *Multiple DNS domains.* It is in the advantage of the botmaster to use different C&C domains. If the bots spread their *phone home* contact over the domains, the degree of each domain will be small. In this way, the botmaster can keep its C&C domains below radar. The spread over several domains can be done in several ways:

  – *Creation of sub domains in a (second level) domain.* If the botmaster controls a domain, it is possible to create an almost unlimited number of subdomains. But this strategy is not effective against detection, because the detector can aggregate all subdomains to one high-level domain as shown in Fig. 6.2. This accumulates the degrees of the subdomains into one large value, that is easier to detect.

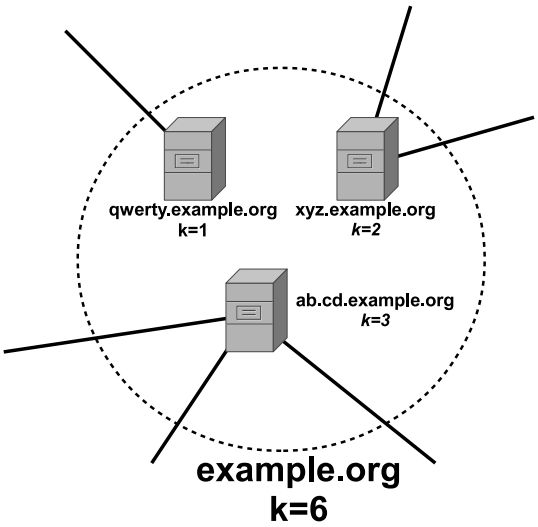

Figure 6.2: Example of the aggregation of many subdomains with small degree to one second-level domain with the accumulated degree.

- *Acquisition of many second-level domains*: Although this evades the described aggregation, it is not attractive for the botmaster, because the acquisition and management of second level domains involves costs. Hence the deployment of many of such domains reduces the utility of the botnet.
- *Creation of free dynamic DNS domains in multiple second level domains*: C&C domains can be created as a dynamic third level DNS domain that is offered for free by many DDNS providers. To avoid a suspicious high degree by aggregation in one DDNS domain, the domains can be spread over several second level DDNS names for example: *mybotdomain.dyndns.org*, *mybotdomain.no-ip.org*, etc. However, the creation and maintenance of a large number these free domains can be a burden, because many providers use request forms with CAPTCHAs, demand confirmation from unique email-addresses, and regular usage. In addition we expect that the queries of legitimate DDNS domains in an enterprise, if queried at all, are limited to a few specific subdomains that can be classified as normal by a predefined list.
- *Creation of URL shortening services*: URLs can be encoded by URL shortening services, such as *bitly.com* [10]. A botnet can use this service to contact a malicious domain or IP-address of the server by a legitimate well-known server of the shortening service. However, the shortening service delivers only the complete URL of the requested shortened URL. The bot still has to resolve the embedded malicious domain by DNS. The botnet can evade this second lookup, by shortening a URL that contains an IP-address instead of a malicious domain. However, the use of IP-addresses without DNS result in an anomaly, which can be detected by other techniques, such as UDI detection, as presented in Chapter 5.
- *Domain flux by DGA*: Domain flux is a technique that changes domain names instead of IP-addresses. Bots, like Conficker [102] and Torpig [120] use a DGA (Domain Generation Algorithm), to periodically generate a list of domains and attempt to contact the domains in the generated list. If the contact fails, they try another name in the list until success. The botmaster only has to registrate a small number of domains in the list, so there is not the problem of high costs. For the defender it is difficult to predict which domain is used. Since eventually the bots will only contact a small number of domains, our detection mechanism is not evaded. Domain flux also introduces vulnerabilities to a botnet. The high number of failed domain contacts is a detectable anomaly[138]. In addition, reverse engineering of the generation algorithm can result in the hijack of a botnet by claiming some of the unreserved domains. Stone et al. describe this method in [120].

6.4. EXPERIMENTAL EVALUATION

We evaluated the FPR of the detection approach with real traffic. In our experiments we have three objectives:

- Comparison of the degree distribution in a real network with the modeled scale free degree distribution of equation 6.2.

- Evaluation of the effectiveness of filtering popular domains.
- Estimation of the False Positive Rate as a function of the degree threshold k_{th}

We captured the DNS-traffic on the campus of a Dutch university in a part of the network that connects about 400 private computers of on-campus student housing with the Internet. The DNS traffic was offline analyzed by a set Perl scripts, we developed.

After 24 hours, the total number of computers with DNS-activity in the network was 351. Together they queried successfully 17908 different domains. We consider a captured DNS-reply with a valid A-record as a successful contact, assuming that a computer only resolves domains with the intention of a subsequent visit. It is likely that the trace will contain some C&C traffic. In our evaluation we neglect the effect of botnet traffic on the FPR measurements, by assuming that all traffic is non-C&C. As a result the measured FPR will be slightly higher than the real FPR, because a small fraction of the Positives are in fact True Positives. At the end of section 6.4.2 we will come back on this aspect.

6.4.1. MEASUREMENT AND COMPARISON OF THE DEGREE DISTRIBUTION

Figure 6.3 shows the probability distribution of the measured domain degrees. The double logarithmic scale reveals a straight line that indicates a power law decay of approximately -2.1. Also plotted, is the theoretical curve, based on equation 6.2 with λ=-2.1. The noise in the right part of the experimental curve is caused by the limited number of domains, with a high degree.

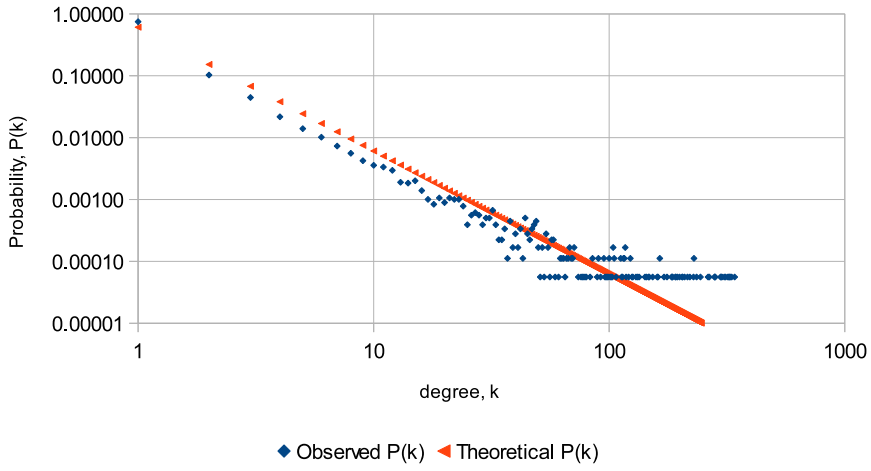

Figure 6.3: Graph of the observed and theoretical probability of domain degrees, with 351 observed computers that queried 17908 domains in 24 hours

Figure 6.4 shows the cumulative degree distribution, derived from the measured probability. In the same figure, the theoretical cumulative degree distribution, calculated by

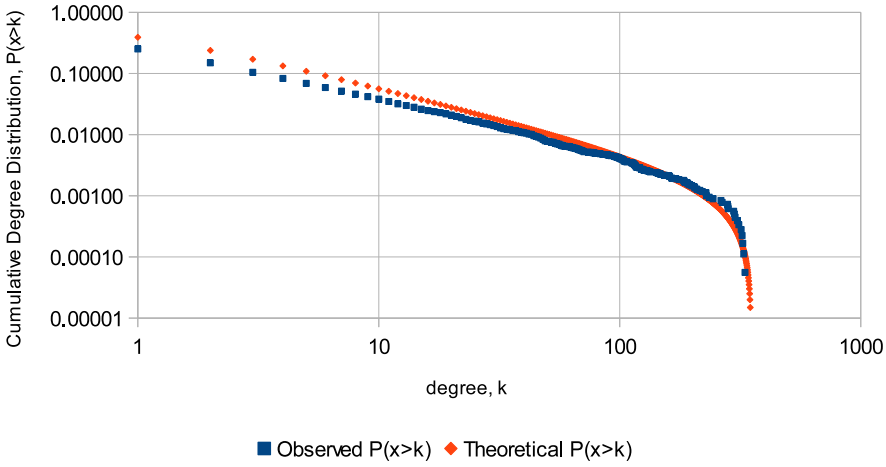

Figure 6.4: Graph of the observed and theoretical cumulative degree distribution of domain degrees, with 351 observed computers that queried 17908 domains in 24 hours.

Equation 6.4, is also plotted. The exponential drop in the right part is caused by the maximum degree of 351. The maximum relative error between the theoretical and experimental cumulative probability is 0,4. The correspondence between the experimental and theoretical graph supports our hypothesized scale free degree distribution. Table 6.1 shows the 10 domains with the highest observed degree. As expected, the domains are all well-known legitimate domains.

Since we neglect here the potential presence of real C&C traffic, the cumulative degree distribution equals the FPR, expressed by equation 6.4. The high degree of the well-known legitimate domains, makes detection unfeasible. For example a chosen threshold of 5 will cause an FPR of 0.08 or 1487 to False Positives of the 17908 domains.

6.4.2. THE EFFECT OF FILTERING POPULAR DOMAINS

The high FPR of the last Section, shows the necessity of the proposed identification of popular legitimate domains. We refer to this step as *filtering*, because the identified popular domains are excluded from further degree evaluation.

The effect of filtering well-known domains from our campus data set was evaluated by using the Alexa-ranking [1] to determine the popularity of a domain. Alexa.com maintains a list of popular domains, measured by a toolbar that is world-wide installed on a large group of computers. Of all observed domains in our data set 95% was ranked in the global top 1000000 list of Alexa. Figure 6.5 shows the probability distribution of the remaining domains. Figure 6.6 shows the resulting FPR, again assuming that all detected domains are legitimate domains. Compared to Figure 6.4, it shows that the filter stage significantly decreases the FPR. If we take again a threshold of 5 as an example, the resulting FPR is 0.0011 or 19 False Positives. Table 6.2 shows 10 detected domains

Table 6.1: The 10 domains with the highest domain degree in the captured DNS traffic.

Degree	Domain
341	google.com
331	doubleclick.com
327	google-analytics.com
324	msn.com
322	microsoft.com
319	live.com
312	atdmt.com
308	2mdn.com
301	hotmail.com
299	youtube.com

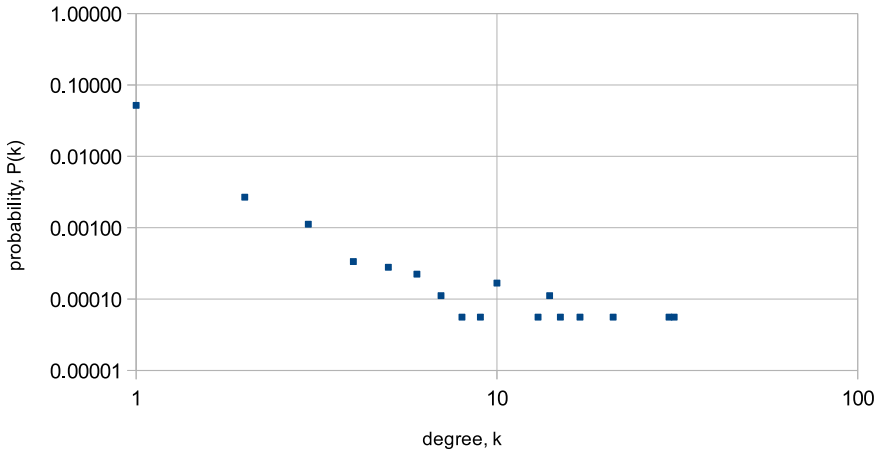

Figure 6.5: Graph of the observed probability of domain degrees after excluding well-known domains

with the highest degrees.

The domain names of the positives reveal that the number of False Positives can be further reduced by additional processing. For example the in-addr.arpa domain is used for DNS translation of IP-addresses to names, and therefore not part of the Alexa ranking. Another example is the presence of equal second level names with only different TLD's. Also the Alexa.com domain itself was not ranked by Alexa during the analysis of our experiments.

In our evaluation we neglected the presence of real C&C traffic. To evaluate this choice, we searched with other techniques for the presence of malicious domains:

- Manual analysis of the names of all DNS domains in the data set revealed DNS-queries with second level DGA names. The queried DGA-domains were not ac-

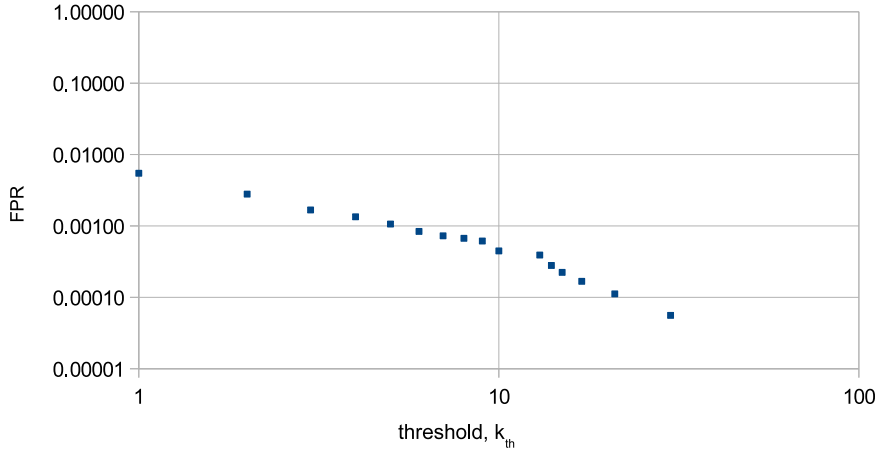

Figure 6.6: Observed FPR as function of the threshold k, after excluding well-known domains

tive, because none of the queries succeeded in a successful answer. Since the degree calculation of our detector is only based on successful connections, the DGA-domains were not identified by the detector. This is not a problem because this noisy type of C&C traffic can easily be detected by other methods [9][130].

- 29 of the domains in the data set, known to Alexa, were present in public blacklists [83]. Obviously the detector did not detect these domains, since they were removed by the Alexa filter.

- At least 1 of the 19 detected domains was suspicious. It was a DDNS domain with a DGA-like third level domain name. Since we had no access to the associated ob-

Table 6.2: The 10 domains with the highest domain degree in the captured DNS traffic after removal of the well-known popular domains by the Alexa.com ranking

Degree	Domain
31	creative-serving.com
30	neatoentertainment.com
21	alexa.com
17	respstatic.com
15	gtkg.net
14	gtkg.org
14	rbi-nl.com
13	crossmediafix.com
10	in-addr.arpa
10	msgpluslive-update.net

served client computers we could not verify if this was indeed C&C-related. The "catch" of only one suspicious domain in 19 detected domains seems disappointing, however, we cannot exclude the presence of other C&C domains in the detection result, because we could not extend our research to the observed machines.

The experimental results show that the addition of the Alexa filter results in a low False Positive Rate of only 19 domains over 24 hours at a detection threshold of 5. The results also show that this approach must be combined with other DNS-based approaches, such as DGA-analysis and blacklists. Since we do not know the exact ground truth situation in this experiment, it is difficult to draw conclusions about an *observed DR (Detection Rate)* of our detector in this experiment.

6.5. CONCLUSIONS

Domains, queried during a defined time interval by a observed group of computers, show a scale free degree distribution with a power law decay. Many domains are only queried by just one observed computer and a limited number of domains are queried by many observed computers.

If a network contains multiple bots that use DNS and query the same C&C domain, this domains will have a degree that is larger then 1. Evaluation of the domain degree of all queried domains can detect these domains if well-known legitimate domains are removed. An important success factor is the number of observed computers. A large number of observed computers will increase the probability of observing multiple bots visiting the same botnet.

This technique complements existing DNS anomaly detection approaches, such as queries to unusual or non-existing domain names. It will not only identify the names of malicious domains, but also the IP-addresses of bots that query the domains. Additionally there is no dependence on misuse-related knowledge, such as signatures or blacklists.

The capture of data and processing is relatively simple and allows for high scalability, when the results of multiple networks are combined, to span a larger group of observed computers. The exchanged information, which can be seen as a state of normal and malicious activity, as defined in Chapter 2, is very compact, because it is limited to only the queried domains and their locally measured degrees.

Experimental evaluation of domain degrees in real traffic, with the removal of well-known popular domains by the Alexa ranking, supports the feasibility of the described detection approach alongside other DNS-based detection approaches.

7

CONCLUDING REMARKS

This final chapter looks back at the research questions of the first chapter and the extent to which they are answered by the results, presented in this thesis. This chapter also looks forward to future work.

7.1. RESEARCH OBJECTIVES, SUBSEQUENT RESEARCH ACTIVITIES, AND RESULTS

Our research aimed for new network-based C&C detection approaches that anticipate specific properties of enterprise networks. In this chapter we will structure the results of the research activities by the research questions of Chapter 1.

1. What are distinctive properties of existing network-based C&C detection approaches and which enterprise-specific properties are potentially suitable for new detection approaches?
For a systematic overview and comparison of botnet countermeasures and botnet traffic detection in particular, we developed an ontology-based faceted classification in Chapter 2. Classification is conducted by identifying the appropriate class per facet. By a proper choice of the facets, the classification for each facet is highly independent from the classification by other facets, which facilitates the classification process. The underlying ontology can be represented by a class diagram that delivers a compact overview of all classes with the facets as semantic associations. The diagram can be refined with new classes by existing facets, new facets, or subfacets. With this approach we developed a generic ontology of botnet countermeasures (Figure 2.9) and a refined ontology of network-based C&C detection approaches (Figure 2.11). Both ontologies allow for the classification of highly diverse countermeasures in classes that accurately reflect the important distinctive properties of a particular countermeasure. The detection ontology classifies detection by three important facets:

1. *Features*, derived from the observations
2. *Knowledge*, used as a reference for normal or malicious instances
3. *State*, constructed by prior events and in some cases required for classification

We classified representative existing network-based detection approaches by this botnet detection ontology. The resulting overview reveals a number of frequently occurring properties in the existing approaches:

- Features are often limited to traffic headers and flow-related metadata
- Knowledge is often misuse related, consisting of signatures of known botnet traffic. In the case of anomaly-based traffic detection, the knowledge of normal traffic is often defined by a baseline for statistical analysis or by a training set for analysis by machine learning.
- Correlation between malicious traffic instances, such as C&C traffic and attack traffic from one or multiple hosts, is a popular anomaly-based approach. It requires a momentary state that is defined by preceding malicious traffic.

Our proposed classification of botnet detection allows for a systematic selection of classes and related properties that support the detection of sophisticated C&C traffic in typical enterprise networks. We identified four properties of enterprise networks that allow detection of infiltrated sophisticated bots with stealth C&C traffic:

1. Enterprise networks allow for comprehensive control and restriction of traffic.

2. Enterprise networks allow for detailed traffic observation, including Deep packet Inspection (DPI) and the decryption of TLS/SSL traffic by an interception proxy.

3. Most network-connected end-systems in an enterprise network, such as computers, printers, and mobile devices have limited hardware and software diversity and allow for comprehensive control.

4. Administrations of enterprise networks can and will take advanced countermeasures against recognized security risks, including significant changes in hardware, software, and procedures.

These properties enable specific classes per facet as shown by the ontology of Figure 2.11.

- *Detection Features*: DPI (Deep Packet Inspection) allows for detection features that are derived from transported application data. In addition, the limited hardware and software diversity and the ability of an enterprise to take advanced countermeasures against security risks, make it feasible to capture non-traffic features by special external hardware or by the host system of a virtualized environment, such as a hypervisor.

- *Knowledge and State*: The comprehensive control and restriction of traffic and the limited hardware and software diversity of end systems allow for anomaly-based detection by the specification of normal traffic. The observation of normal traffic in enterprise networks enables the construction of a momentary state that predicts potential future normal traffic.

2. How can botnets be detected in an enterprise network, anticipating the identified new characteristic properties?
In this thesis three new C&C detection approaches for enterprise networks are proposed. They anticipate the specific properties of an enterprise network by implementing the identified classes, obtained from research question 1. The approaches do not replace but complement traditional misuse- and anomaly-based detection approaches, including traffic detection by known botnet signatures and traffic detection by correlation of malicious traffic. The new detection approaches, proposed and evaluated in this thesis are:

1. *C&C anomaly detection by the identification of direct causes of a new traffic flow (TFC-detection, Chapter 3 and 4)*
   - Generic classification: B/N∧(H∨E)/R(Al/Ni/De(An))
   - Detection classification: De(Ho∧Tr(He∧Ap(DNS∧HTTP)∧Me)/An(St∧Sp)/No)
   The assumption is that C&C traffic is machine-generated and therefore not directly caused by human activity or other legitimate traffic. To demonstrate this we developed and implemented the OCS algorithm, that selects the optimal direct cause for each new traffic flow by its relation with prior traffic and user activity. The selection process includes both time measurements between events

and semantic relationships between different flows. All flows and causes can be organized in a Traffic Flow Causality graph or TFC graph as trees. The first flow of each tree is the root flow. The direct cause of this root flow is used for the classification of normal and anomalous traffic. Experiments with a proof of concept, implemented in CITRIC, and mixtures of captured clean diverse traffic and captured C&C traffic demonstrate that this approach can detect successfully single instances of various types of covert C&C traffic with a limited number of False Positives. The detection is real-time and does not require any preceding botnet traffic. If deployed in an IPS, detected C&C communication can be immediately and completely blocked. A limitation of the approach is that future bots can evade detection by piggybacking on certain events by synchronization. There are solutions for this problem, which we will discuss Section 7.2.

TFC-detection depends on the identified properties of enterprise networks. The limited diversity and comprehensive control of end systems in enterprise networks, combined with the ability to take advanced countermeasures, support the feasibility of described observation of keyboard and mouse activity. In addition, the ability of Deep Packet Inspection in enterprise networks enables the evaluation of semantic relationships. The comprehensive control and restriction of traffic reduces diversity in causal relationships, which increases the feasibility of the approach.

2. *C&C anomaly detection by the trustworthiness of visited destinations (UDI-detection, Chapter 5)*
   - Generic classification: B/N/R(Al/Ni/De(An))
   - Detection classification: De(Tr(He∧Ap(DNS∧HTTP)∧Me)/An(St∧Sp)/No)
   In this approach the origin of the destination identifier of each new traffic flow is determined. Examples of destination identifiers are IP-addresses, hostnames, or URLs. The assumption is that the destination identifier of botnet C&C traffic does not origin from well-known legitimate applications, prior traffic from trusted destinations, or human input. Similar to TFC-detection this approach evaluates semantic relations between traffic flows, by searching in the traffic payload for destination references of future traffic. However, unlike TFC detection, UDI-detection does not use time intervals to select direct causes. Similar to TFC-detection this approach evaluates user input but it does not require the observation of user input devices. UDI-detection evaluates the destination identifier of a traffic flow on its likeliness of being typed by a human. UDI-detection was tested in a similar way as TFC detection and successful real-time detection of single instances of various types of covert C&C traffic was demonstrated. The simplified implementation of UDI detection as a proof of concept in CITRIC resulted in limitations, related with C&C traffic to popular destinations, such as social media. We will discuss solutions as future work in Section 7.2. Similar to TFC-detection, UDI-detection anticipates the ability of Deep Packet Inspection in enterprise networks, combined with the comprehensive control and limited diversity of traffic and end systems.

Table 7.1: Overview of observed detection accuracy, C&C traffic types, and evasion possibilities of the proposed detection approaches.

Detection Approach	FPR	DR	C&C traffic	Evasion Possibilities
TFC detection	0.0017 [1]	>0.7	DNS/HTTP, P2P, Social Media	piggybacking on certain user and traffic events
UDI detection	0.0026 [2]	1	DNS/HTTP, DNS, P2P, TOR	C&C by popular Social Media, the use of user-friendly destination identifiers
Detection by domain degree anomalies	0.0011 [3]	-	DNS	the use of popular domains, continuous change of existing C&C domains

3. *C&C detection by anomalies in the distribution of the domain degree (Chapter 6)*
   - Generic classification: B/N/R(Al/Ni/De(An))
   - Detection classification: De(Tr(Ap(DNS))/An(St∧Sp)/Ma(Cc/Ho))
   The detector observes the payloads of DNS traffic and determines of each resolved domain the relative popularity, expressed by its degree, which is the number of end systems in an enterprise network that request successfully the IP-address of a domain. If the degree of a domain is unexpectedly high, it is regarded as an anomaly. A scale free model of the degree distribution is presented. Observation of diverse DNS traffic in a real network support this model. The scale free distribution shows that only a small number of well known domains are responsible for most false positives in this detection approach. This can be prevented by taking account of well known domains. The degree observation allows for easy accumulation over multiple enterprise networks. Detection is limited to C&C traffic that uses DNS to resolve the hostname of a C&C server. For identification of the individual bots the source address in the queries must be correct. This is easily enforced within an enterprise network. In addition, all end systems in an enterprise can be forced to use the internal official DNS service. This simplifies the observation and mitigates the risk of a botnet-colluding DNS-server.

3. What detection performance is expected from the newly identified detection approaches of question 2?
The FPR (False Positive Rate) and DR (Detection Rate) of the detection approaches are empirically evaluated. The FPR and DR are popular and appropriate conditional probabilities to express detection accuracy, because they do not depend on the ratio of C&C

[1] Traffic generated in a controlled environment. For details consult Chapter 4.5.
[2] Traffic generated in a controlled environment. For details consult Chapter 5.4.
[3] Normal traffic is DNS traffic, captured from a student housing network of 400 end systems. DR is not measured. For details consult Chapter 6.4.

traffic and normal traffic. Table 7.1 gives an overview of the observed detection performance, including the type of C&C traffic that is detectable by the approach and potential evasion possibilities. The detection approach of Chapter 3, which was only based on human activity, is omitted, because it is only an explorative step towards de TFC approach that includes observation of human activity. The observed values of the FPR and DR support the underlying models of the detection approaches, as presented and elaborated in the Chapters 3 to 6. In real enterprise networks the expected FPR can be significantly lower than the observed FPR in our experiments because the normal traffic of both the controlled environment and the student housing network is extremely diverse.

7.2. FUTURE WORK

This last section will give an overview of future work which is related to this study. This involves short and medium term work that directly arises from the research results and long term work that takes a much broader approach to the botnet problem.

7.2.1. SHORT TERM WORK

This study presents three new approaches to detect C&C traffic in enterprise networks. The ability to detect various types of C&C traffic with a relatively low FPR and in a different manner than existing approaches, allows for the implementation in practical Intrusion Detection Systems alongside other existing approaches. However with respect to TFC and UDI detection there are some important notes to make:

1. A practical implementation of TFC and UDI detection requires a TLS/SSL interception proxy as described in the Sections 4.6.1 and 5.5. Due to implementation complexity and time constraints, this proxy was not applied in the experiments. The resulting misses of forward references in the encrypted payloads were compensated by heuristics, such as the observation of HTTPS events in the case of TFC detection (Section 4.3.2) and partial LDI matching in the case of UDI detection (Section 5.3.1). However, analysis of the detection errors in our experiments indicates that a TLS/SSL proxy could significantly improve the FPR and DR of both TFC and UDI detectors. Moreover, the recent increase in popularity of encrypted traffic makes such a proxy not only an improvement but an important condition to obtain satisfying results. The commercial availability of TLS/SSL interception proxies makes the required effort to combine it with TFC and UDI detection more an engineering and implementation effort than a research effort.

2. Evasion by future botnets that are aware of the detection approaches is possible. As described, bots can evade TFC detection by synchronizing on certain events, bots can evade UDI-detection by using popular servers as intermediary, and bots can evade the DNS-based approach by not using DNS at all. The evasion problem, which in general applies to all existing countermeasures, does not make the presented approaches useless because (1) not all bots will immediately use the required evasion techniques and (2) evasion can be made more complex by combining methods that require different evasion approaches. The determination of

optimal detection combinations to improve the DR an impede evasion, involves both short term and long term applied research. The faceted classification of botnet countermeasures and detection approaches can contribute to such research because it allows for a systematic comparison of different countermeasures.

3. Practical implementation requires conditions that can only be met in certain enterprise networks. Most notably TFC detection requires a special device or agent to capture user events. TFC and UDI detection require a defined initial state of the observed computers which can be realized by a periodic reboot of the involved computers. TFC and UDI detection require knowledge of automatic legitimate traffic that is initiated from known software applications.

7.2.2. LONG TERM WORK

TFC and UDI detection have in common that each new traffic flow is associated with a state that is determined by elements of prior received traffic or user input. In the case of TFC detection this state requires knowledge of previous user events and traffic events as direct causes. In the case of UDI detection this state requires knowledge of received destinations. The assumption is in both cases that a computer or computer network is a deterministic causal system. With sufficient knowledge of past events a state can be determined that explains or even predicts all network activity in the near future. The present implementation of TFC and UDI detection does not collect enough knowledge of past events to completely determine such a state, however combination and extension of these methods can further approach this situation. We will describe the improvements step-by-step, starting from the present UDI-detection system:

1. Inspection of all traffic by TLS/SSL interception proxy as proposed in the short term work.

2. Accurate construction of the application state in the detector, especially by browser emulation. This results in the determination of the full ldi's of egress traffic and the extraction of all potential full forward references.

3. Instead of the statistical analysis of the name complexity of the ldi, the complete user input is evaluated. Potential forward references or parts of forward references (such as typed URLs or potential parts of URLs) are collected from the user input, similar to the references in received payloads.

4. In addition to the match between a forward reference and the ldi of a new flow, the time interval between the reception of a forward reference and its usage as ldi in a new flow, must meet certain conditions before a new flow is associated with the prior activity.

These modifications eliminate the need for error-prone heuristics, such as partial ldi-matching or the use of only time to make an association between a new traffic flow and observed activity. Piggybacking by synchronization with user or traffic events becomes very difficult, because in addition to synchronization the bot must also use an ldi that equals to one of the forward references of prior normal activity. Evasion by the use of

popular webservices as intermediary is also difficult, because instead of only the hostname or IP-address, the complete URL must have been referenced in prior legitimate activities. We belief that a significant research effort is required to obtain the knowledge for the outlined improvements.

Here, at the end of this dissertation on detection in enterprise networks, we want to briefly mention very different solution directions that aim for a complementary and structural long term solution of botnets, malware, and IT-related APTs in general. This directions include: attack prevention, forensics, analysis of the cybercrime-related economy, regulation, security by design, and offensive responses. Major research efforts in such directions, such as outlined in the National Cyber Security Research Agenda [13], are required. In the long term the proposed research can bring solutions that result in less dependency on the large variety of detective/repressive approaches. At least until that moment it remains wise to continue with improvements of existing detection/repression approaches and the quest for new detection/repression approaches.

REFERENCES

[1] Alexa.com. Alexa, The Web Information Company. http://www.alexa.com/topsites, 2010. Visited 12 May 2010.

[2] Alexa.com. Alexa, The Web Information Company. http://www.alexa.com/topsites, 2013. Visited 2 Mar 2013.

[3] R. Anderson, R. Böhme, R. Clayton, and T. Moore. Security economics and european policy. In *Managing Information Risk and the Economics of Security*, pages 55–80. Springer, 2009.

[4] R. Anderson, C. Barton, R. Böhme, R. Clayton, M. J. van Eeten, M. Levi, T. Moore, and S. Savage. Measuring the cost of cybercrime. In *The Economics of Information Security and Privacy*, pages 265–300. Springer, 2013.

[5] H. Asai, K. Fukuda, and H. Esaki. Traffic causality graphs: Profiling network applications through temporal and spatial causality of flows. In *Proc. of the 23rd International Teletraffic Congress*, pages 95–102. ITCP, 2011.

[6] A. Asosheh and N. Ramezani. A comprehensive taxonomy of ddos attacks and defense mechanism applying in a smart classification. *WSEAS Transactions on Computers*, 7(7):281–290, 2008.

[7] A.-L. Barabási and R. Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.

[8] P. Barford and V. Yegneswaran. An inside look at botnets. In *Malware Detection*, pages 171–191. Springer, 2007.

[9] L. Bilge, E. Kirda, C. Kruegel, and M. Balduzzi. Exposure: Finding malicious domains using passive dns analysis. In *Proc. of the Network and Distributed System Security Symposium 2011 , NDSS*. The Internet Society, 2011.

[10] Bitly.com. Bitly, URL Shortening Service. http://www.bitly.com, 2010. Visited 30 Dec 2010.

[11] A. Blum, B. Wardman, T. Solorio, and G. Warner. Lexical feature based phishing url detection using online learning. In *Proc. of the 3rd ACM workshop on Artificial Intelligence and Security*, pages 54–60. ACM, 2010.

[12] F. Boon, S. Derix, and H. Modderkolk. NSA infected 50,000 Computer Networks with Malicious Software. http://www.nrc.nl/nieuws/2013/11/23/nsa-infected-50000-computer-networks-with-malicious-software/, 2013. Visited December 2013.

[13] H. Bos, S. Etalle, and E. Poll. National cyber security research agenda ii. Technical report, ICT Innovatie Platform Veilig Verbonden, 2013.

[14] R. D. Boscovich. Microsoft, Europol, FBI, and Industry Partners disrupt Notorious ZeroAccess Botnet that hijacks Search Results, The Official Microsoft Blog. `http://blogs.technet.com/b/microsoft_blog/archive/2013/12/05/microsoft-europol-fbi-and-industry-partners-disrupt-notorious-zeroaccess-botnet-that-hijacks-search-results.aspx`, 2013. Visited December 2013.

[15] D. Brumley, L.-H. Liu, P. Poosankam, and D. Song. Design space and analysis of worm defense strategies. In *Proc. of the 2006 ACM Symposium on Information, Computer and Communications Security*, pages 125–137. ACM, 2006.

[16] J. Brustoloni, N. Farnan, R. Villamarín-Salomón, and D. Kyle. Efficient detection of bots in subscribers' computers. In *IEEE International Conference on Communications 2009, ICC'09*, pages 1–6. IEEE, 2009.

[17] P. Burghouwt. CITRIC at Github. `https://github.com/pb12/CITRIC`, 2014. Visited 31 Dec 2014.

[18] P. Burghouwt, M. Spruit, and H. Sips. Detection of botnet collusion by degree distribution of domains. In *Proc. of International Conference for Internet Technology and Secured Transactions 2010, ICITST2010*, pages 1–8. IEEE, 2010.

[19] P. Burghouwt, M. Spruit, and H. Sips. Towards detection of botnet communication through social media by monitoring user activity. In *Proc. of the 7th International Conference on Informations Systems Security 2011, ICISS2011*, pages 131–143. Springer, 2011.

[20] P. Burghouwt, M. Spruit, and H. Sips. Detection of covert botnet command and control channels by causal analysis of traffic flows. In *Proc. of the 5th International Symposium on Cyberspace Safety and Security 2013, CSS2013*, pages 117–131. Springer, 2013.

[21] P. Burghouwt, M. Spruit, and H. Sips. Detection of botnet command and control traffic by the identification of untrusted destinations. In *to be published in Proc. of the 10th International Conference on Security and Privacy in Communication Networks 2014, SECURECOMM2014*. Springer, 2014.

[22] D. Champagne and R. B. Lee. Scope of ddos countermeasures: taxonomy of proposed solutions and design goals for real-world deployment. Technical report, Princeton Univ. Tech. Report CE-L2005-007, 2005.

[23] H. Choi, H. Lee, and H. Kim. Botgad: Detecting botnets by capturing group activities in network traffic. In *Proc. of the 4th International ICST Conference on COMmunication System softWAre and middlewaRE 2009, COMSWARE'09*, page 2. ACM, ACM, 2009.

[24] M. Chuba and C. Claunch. What grid computing is really about. Technical report, Gartner research, 2006.

[25] Cisco. Enterprise campus 3.0 architecture: Overview and framework. Technical report, Cisco, 2011.

[26] Contagio. Skynet Tor botnet / Trojan.Tbot samples. `http://contagiodump.blogspot.nl/2012/12/dec-2012-skynet-tor-botnet-trojantbot.html`, 2013. Visited Jan 2013.

[27] E. Cooke, F. Jahanian, and D. McPherson. The zombie roundup: Understanding, detecting, and disrupting botnets. In *Proc. of the USENIX Workshop on Steps to Reducing Unwanted Traffic on the Internet, SRUTI'05*, Cambridge, CA, 2005. USENIX Association.

[28] S. Cranefield and M. Purvis. UML as an Ontology Modelling Language, 1999.

[29] W. Cui, R. H. Katz, and W.-t. Tan. Design and implementation of an extrusion-based break-in detector for personal computers. In *Twenty-First Annual Computer Security Applications Conference 2005, ACSAC2005*. IEEE, 2005.

[30] Curl. curl and libcurl. `http://http://curl.haxx.se`, 2014. Visited 31 Dec 2014.

[31] D. Dagon, G. Gu, C. P. Lee, and W. Lee. A taxonomy of botnet structures. In *Twenty-Third Annual Computer Security Applications Conference 2007, ACSAC 2007*, pages 325–339. IEEE, 2007.

[32] Damballa. Advanced Persistent Threats (APT), 2012.

[33] C. R. Davis, J. M. Fernandez, S. Neville, and J. McHugh. Sybil attacks as a mitigation strategy against the storm botnet. In *Proc. of the 3rd International Conference on Malicious and Unwanted Software 2008, MALWARE'08*, pages 32–40. IEEE, 2008.

[34] D. de Graaf, A. F. Shosha, and P. Gladyshev. Bredolab: Shopping in the cybercrime underworld. In *Digital Forensics and Cyber Crime*, pages 302–313. Springer, 2013.

[35] DeependResearch. Trojan Nap aka Kelihos/Hlux. `http://www.deependresearch.org/2013/02/trojan-nap-aka-kelihoshlux-feb-2013.html`, 2013. Visited Feb 2013.

[36] R. Deibert and R. Rohozinski. Tracking ghostnet: Investigating a cyber espionage network. *Information Warfare Monitor*, page 6, 2009.

[37] D. E. Denning. An intrusion-detection model. *IEEE Transactions on Software Engineering,*, 2(2):222–232, 1987.

[38] C. J. Dietrich and C. Rossow. Empirical research of ip blacklists. In *ISSE 2008 Securing Electronic Business Processes*, pages 163–171. Springer, 2009.

[39] C. J. Dietrich, C. Rossow, and N. Pohlmann. Cocospot: Clustering and recognizing botnet command and control channels using traffic analysis. *Computer Networks*, 57(2):475–486, 2013.

[40] C. Dixon, T. E. Anderson, and A. Krishnamurthy. Phalanx: Withstanding multimillion-node botnets. In *Proc. of the 5th USENIX Symposium on Networked Systems Design and Implementation, NSDI'08*, volume 8, pages 45–58. USENIX Association, 2008.

[41] D. Drummond. A new Approach to China. `http://googleblog.blogspot.nl/2010/01/new-approach-to-china.html`, 2010. Visited December 2013.

[42] N. Falliere, L. O. Murchu, and E. Chien. W32. stuxnet dossier version 1.4. Technical report, Symantec Corp., 2011.

[43] T. Fawcett. Roc graphs: Notes and practical considerations for researchers. *Machine Learning*, 31:1–38, 2004.

[44] M. Feily, A. Shahrestani, and S. Ramadass. A survey of botnet and botnet detection. In *Third International Conference on Emerging Security Information, Systems and Technologies 2009, SECURWARE'09*, pages 268–273. IEEE, 2009.

[45] R. Ferguson. The botnet chronicles. Technical report, Trend Micro, 2010.

[46] T. Fraser, L. Badger, and M. Feldman. Hardening cots software with generic software wrappers. In *proc. of the DARPA Information Survivability Conference and Exposition 2000, DISCEX'00*, volume 2, pages 323–337. IEEE, 2000.

[47] N. Friess, J. Aycock, and R. Vogt. Black market botnets. In *Proc. of the MIT spam Conference*. MIT, 2008.

[48] P. García-Teodoro, J. Díaz-Verdejo, G. Maciá-Fernández, and E. Vázquez. Anomaly-based network intrusion detection: Techniques, systems and challenges. *Computers & Security*, 28(1–2):18 – 28, 2009. ISSN 0167-4048.

[49] A. A. Ghorbani, W. Lu, and M. Tavallaee. *Network intrusion detection and prevention: concepts and techniques*, volume 47. Springer, 2009.

[50] F. Giroire, J. Chandrashekar, N. Taft, E. Schooler, and D. Papagiannaki. Exploiting temporal persistence to detect covert botnet channels. In *Proc. of the 12th International Symposium of Recent Advances in Intrusion Detection, RAID'09*, pages 326–345, Berlin, 2009. Springer.

[51] F. Giunchiglia and I. Zaihrayeu. Lightweight ontologies. In *Encyclopedia of Database Systems*, pages 1613–1619. Springer, 2009.

[52] J. Goebel and T. Holz. Rishi: Identify bot contaminated hosts by irc nickname evaluation. In *Proc. of the first USENIX Workshop on Hot Topics in Understanding Botnets, HOTBOTS'07*. USENIX Association, 2007.

[53] Google.com. Top 1000 sites - DoubleClick Ad Planner. http://www.google.com/adplanner/static/top1000/, 2013. Visited 2 Mar 2013.

[54] G. O. Gorman. Google Groups Trojan. http://www.symantec.com/connect/blogs/google-groups-trojan, 2009. Visited January 2011.

[55] GOVCERT.NL. Trendreport 2007, cyber crime in trends and figures. Technical report, GOVCERT.NL, 2007.

[56] T. R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199–220, 1993.

[57] G. Gu. *Correlation-based botnet detection in enterprise networks*. ProQuest, 2008.

[58] G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee. Bothunter: Detecting malware infection through ids-driven dialog correlation. In *Proc. of 16th USENIX Security Symposium, SECURITY'07*, page 12. USENIX Association, 2007.

[59] G. Gu, R. Perdisci, J. Zhang, W. Lee, et al. Botminer: Clustering analysis of network traffic for protocol- and structure-independent botnet detection. In *Proc. of the 17th USENIX Security Symposium SECURITY'08*, pages 139–154, Berkeley, CA, 2008. USENIX Association.

[60] R. Gummadi, H. Balakrishnan, P. Maniatis, and S. Ratnasamy. Not-a-bot: Improving service availability in the face of botnet attacks. In *Proc. of the 6th USENIX Symposium on Networked Systems Design and Implementation, NSDI'09*, Berkeley, CA, 2009. USENIX Association.

[61] N. Hachem, Y. Ben Mustapha, G. G. Granadillo, and H. Debar. Botnets: lifecycle and taxonomy. In *2011 Conference on Network and Information Systems Security, SAR-SSI*, pages 1–8. IEEE, 2011.

[62] I. Hamadeh and G. Kesidis. A taxonomy of internet traceback. *International Journal of Security and Networks*, 1(1):54–61, 2006.

[63] S. Hansman and R. Hunt. A taxonomy of network and computer attacks. *Computers & Security*, 24(1):31–43, 2005.

[64] Hewlett-Packard. Hp flexnetwork architecture, change the rules of networking. Technical report, HP, 2011.

[65] T. Holz, C. Gorecki, K. Rieck, and C. Freiling. Measuring and detecting fast-flux service networks. In *Proc. of Symposium on Network and Distributed System Security, NDSS'08*. The Internet Society, 2008.

[66] T. Holz, M. Steiner, F. Dahl, E. Biersack, and F. C. Freiling. Measurements and mitigation of peer-to-peer-based botnets: A case study on storm worm. In *Proc. of the first USENIX Workshop on Large-Scale Exploits and Emergent Threats, LEET'08*, volume 8, pages 1–9, Berkeley, CA, 2008. USENIX Association.

[67] IANA. tlds-alpha-by-domain. `http://data.iana.org/TLD/tlds-alpha-by-domain.txt`, 2014. Visited 31 Dec 2014.

[68] M. Iliofotou, P. Pappu, M. Faloutsos, M. Mitzenmacher, S. Singh, and G. Varghese. Network monitoring using traffic dispersion graphs (tdgs). In *Proc. of the 7th ACM SIGCOMM conference on Internet measurement, IMC'07*, pages 315–320. ACM, 2007.

[69] ITU-T. Background information on itu botnet mitigation toolkit. Technical report, ICT Applications and Cybersecurity Division, Policies and Strategies Department, SectorITU-T, 2008.

[70] A. K. Jain. Data clustering: 50 years beyond k-means. *Pattern Recognition Letters*, 31(8):651–666, 2010.

[71] J. Jarmoc and D. S. C. T. Unit. Ssl/tls interception proxies and transitive trust. In *Black Hat Europe 2012*. Blackhat, 2012.

[72] T. Karagiannis, K. Papagiannaki, and M. Faloutsos. Blinc: multilevel traffic classification in the dark. *ACM SIGCOMM Computer Communication Review*, pages 229–240, 2005.

[73] M. Karresand. Separating trojan horses, viruses, and worms-a proposed taxonomy of software weapons. In *Information Assurance Workshop, 2003. IEEE Systems, Man and Cybernetics Society*, pages 127–134. IEEE, 2003.

[74] E. Kartaltepe, J. Morales, S. Xu, and R. Sandhu. Social network-based botnet command-and-control: Emerging threats andcountermeasures. *Applied Cryptography and Network Security*, pages 511–528, 2010.

[75] D. G. Kendall. Stochastic processes occurring in the theory of queues and their analysis by the method of the imbedded markov chain. *The Annals of Mathematical Statistics*, pages 338–354, 1953.

[76] S. Khattak, N. Ramay, K. Khan, A. Syed, and S. Khayam. A Taxonomy of Botnet Behavior, Detection, and Defense, 2013.

[77] J. Kirk. Dutch team up with Armenia for Bredolab Botnet take down, 2010.

[78] R. Kohavi and F. Provost. Glossary of terms. *Machine Learning*, 30(2-3):271–274, 1998.

[79] A. Lazarevic, V. Kumar, and J. Srivastava. Intrusion detection: A survey. In *Managing Cyber Threats*, pages 19–78. Springer, 2005.

[80] W. Lee, W. Fan, M. Miller, S. J. Stolfo, and E. Zadok. Toward cost-sensitive modeling for intrusion detection and response. *Journal of Computer Security*, 10(1):5–22, 2002.

[81] A. Lelli. Trojan.Whitewell: What's your (bot) Facebook Status Today? http://www.symantec.com/connect/blogs/trojanwhitewell-what-s-your-bot-facebook-status-today, 2009. Visited December 2010.

[82] J. Ma, L. K. Saul, S. Savage, and G. M. Voelker. Beyond blacklists: learning to detect malicious web sites from suspicious urls. In *Proc. of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining 2009, KDD'09*, pages 1245–1254. ACM, 2009.

[83] Malwaredomainlist.com. Community driven malware domain list. http://www.malwaredomainlist.com, 2010. Visited 6 Jan 2010.

[84] McAfee. Protecting your critical assets, lessons learned from operation aurora. Technical report, McAfee, 2010.

[85] D. K. McGrath and M. Gupta. Behind phishing: An examination of phisher modi operandi. In *Proc. of the first USENIX Workshop on Large-Scale Exploits and Emergent Threats, LEET'08*, volume 8, pages 1–8, Berkeley, CA, 2008. USENIX Association.

[86] T. Micro. Taxonomy of botnet threats. *Whitepaper, November*, 2006.

[87] Microsoft.com. Worm:Win32/Morto.A. http://www.microsoft.com/security/portal/threat/encyclopedia/entry.aspx?Name=Worm:Win32/Morto.A, 2014. Visited April 2014.

[88] K. Miller and M. Pegah. Virtualization: virtually at the desktop. In *Proc. of the 35th annual ACM SIGUCCS fall conference*, pages 255–260. ACM, 2007.

[89] J. Mirkovic and P. Reiher. A taxonomy of ddos attack and ddos defense mechanisms. *ACM SIGCOMM Computer Communication Review*, 34(2):39–53, 2004.

[90] J. J.-D. Mol, J. A. Pouwelse, D. H. Epema, and H. J. Sips. Free-riding, fairness, and firewalls in p2p file-sharing. In *Proc. of the Eighth International Conference on Peer-to-Peer Computing, 2008, P2P'08.*, pages 301–310. IEEE, 2008.

[91] S. Nagaraja, A. Houmansadr, P. Piyawongwisal, V. Singh, P. Agarwal, and N. Borisov. Stegobot: a covert social network botnet. In *Information Hiding, 10th International Workshop*, pages 299–313. Springer, 2011.

[92] J. Nazario. Twitter-based Botnet Command Channel. http://asert.arbornetworks.com/2009/08/twitter-based-botnet-command-channel/, 2009. Visited Oct 2010.

[93] J. Nazario and T. Holz. As the net churns: Fast-flux botnet observations. In *Proc. of the 3rd International Conference on Malicious and Unwanted Software 2008, MALWARE2008.*, pages 24–31, Alexandria, VA, 2008. IEEE.

[94] N. F. Noy, D. L. McGuinness, et al. Ontology Development 101: A Guide to Creating your First Ontology, 2001.

[95] D. Olmedilla, O. F. Rana, B. Matthews, and W. Nejdl. Security and trust issues in semantic grids. In *Proc. of the Semantic Grid*, volume 5271, 2005.

[96] I.-V. Onut and A. A. Ghorbani. A feature classification scheme for network intrusion detection. *IJ Network Security*, 5(1):1–15, 2007.

[97] S. Ortolani. *Keylogger Detection and Containment*. PhD thesis, VU University Amsterdam, 2013.

[98] P. L. Overbeek, E. E. O. R. Lindgreen, and M. E. M. Spruit. *Informatiebeveiliging onder controle (Dutch Language)*. Pearson Education, 2005.

[99] R. Pilling. Global threats, cyber-security nightmares and how to protect against them. *Computer Fraud & Security*, 2013(9):14–18, 2013.

[100] D. Plohmann and E. Gerhards-Padilla. Case study of the miner botnet. In *4th International Conference on Cyber Conflict 2012, CYCON*, pages 1–16. IEEE, 2012.

[101] R. Pointer. Eggdrop. `http://www.eggheads.org/`, 1993. Visited December 2013.

[102] P. Porras, H. Saïdi, and V. Yegneswaran. A foray into conficker's logic and rendezvous points. In *Proc. of the second USENIX Workshop on Large-Scale Exploits and Emergent Threats: botnets, spyware, and more, LEET'08*, Boston, MA, 2009. USENIX Association.

[103] N. Provos. A virtual honeypot framework. In *Proc. of the 13th Conference on the USENIX Security Symposium, SSYM04*, San Diego, CA, 2004. USENIX Association.

[104] M. A. Rajab, J. Zarfoss, M. Fabian, and A. Terzis. My botnet is bigger than yours (maybe, better than yours): why size estimates remain challenging. In *Proc. of the 1st USENIX Workshop on Hot Topics in Understanding Botnets, BOTS'07*. USENIX Association, 2007.

[105] A. Ramachandran and N. Feamster. Understanding the network-level behavior of spammers. *ACM SIGCOMM Computer Communication Review*, 36:291–302, 2006.

[106] A. Ramachandran, N. Feamster, and D. Dagon. Detecting botnet membership with dnsbl counterintelligence. In *Botnet Detection*, pages 131–142. Springer, 2008.

[107] K. Rieck, G. Schwenk, T. Limmer, T. Holz, and P. Laskov. Botzilla: Detecting the phoning home of malicious software. In *Proc. of the 2010 ACM Symposium on Applied Computing, SAC2010*, pages 1978–1984. ACM, 2010.

[108] R. A. Rodríguez-Gómez, G. Maciá-Fernández, and P. García-Teodoro. Survey and taxonomy of botnet research through life-cycle. *ACM Computing Surveys (CSUR)*, 45(4):45, 2013.

[109] M. Roesch et al. Snort: Lightweight intrusion detection for networks. In *Proc. of the 13th USENIX Large Installation Systems Administration Conference, LISA'99*, volume 99, pages 229–238. USENIX Association, 1999.

[110] C. Rossow and C. J. Dietrich. Provex: Detecting botnets with encrypted command and control channels. In *Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 21–40. Springer, 2013.

[111] S. Saad, I. Traore, A. Ghorbani, B. Sayed, D. Zhao, W. Lu, J. Felix, and P. Hakimian. Detecting p2p botnets through network behavior analysis and machine learning. In *Ninth Annual International Conference on Privacy, Security and Trust 2011, PST*, pages 174–180. IEEE, 2011.

[112] C. Satten. Lossless gigabit remote packet capture with linux. Technical report, University of Washington, 2008. URL `http://staff.washington.edu/corey/gulp/`.

[113] C. Schiller and J. Binkley. *Botnets: The Killer Web Applications*. Syngress Publishing, Rockland MA, first ed. edition, 2007.

[114] H. Seybert and P. Reinecke. Statistics in focus 29/2013. Technical report, Eurostat, 2013.

[115] T. O. . Social. Special eurobarometer 404: Cyber security. Technical report, TNS Opinion & Social, 2013.

[116] S. M. Specht and R. B. Lee. Distributed denial of service: Taxonomies of attacks, tools, and countermeasures. In *Proc. of the ISCA 17th International Conference on Parallel and Distributed Computing Systems, PDCS*, pages 543–550. ISCA, 2004.

[117] N. Stakhanova, S. Basu, and J. Wong. A taxonomy of intrusion response systems. *International Journal of Information and Computer Security*, 1(1):169–184, 2007.

[118] A. Stavrou, D. L. Cook, W. G. Morein, A. D. Keromytis, V. Misra, and D. Rubenstein. Websos: an overlay-based system for protecting web servers from denial of service attacks. *Computer Networks*, 48(5):781–807, 2005.

[119] E. Stinson and J. Mitchell. Towards systematic evaluation of the evadability of bot/botnet detection methods. In *Proc. of the 2nd conference on USENIX Workshop on offensive technologies , WOOT'08*, Berkeley, CA, 2008. USENIX Association.

[120] B. Stone-Gross, M. Cova, L. Cavallaro, B. Gilbert, M. Szydlowski, R. Kemmerer, C. Kruegel, and G. Vigna. Your botnet is my botnet: analysis of a botnet takeover. In *Proc. of the 16th ACM conference on Computer and communications security, CCS2009*, pages 635–647. ACM, 2009.

[121] W. T. Strayer, D. Lapsely, R. Walsh, and C. Livadas. Botnet detection based on network behavior. In *Botnet Detection*, pages 1–24. Springer, 2008.

[122] Symantec. Istr, internet security report 2013. Technical report, Symantec, 2013.

[123] K. Taylor. An Analysis of Computer Use across 95 Organisations in Europe, North America and Australasia. Technical report, Wellnomics, 2007.

[124] Tcpdump. TCPDUMP/LIBCAP public repository. `http://tcpdump.org`, 2014. Visited 31 Dec 2014.

[125] V. Thomas and N. Jyoti. Bot countermeasures. *Journal in Computer Virology*, 3(2): 103–111, 2007.

[126] H. Tiirmaa-Klaar, J. Gassen, E. Gerhards-Padilla, and P. Martini. Botnets, cybercrime and national security. In *Botnets*, pages 1–40. Springer, 2013.

[127] B. H. Trammell and E. Boschi. RFC5103: Bidirectional flow export using ip flow information export (IPFIX). Technical report, Internet Engineering TaskForce, 2008.

[128] M. J. van Eeten, H. Asghari, J. M. Bauer, and S. Tabatabaie. Internet service providers and botnet mitigation. Technical report, Delft University of Technology, 2011.

[129] J. Venn. On the diagrammatic and mechanical representation of propositions and reasonings. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 10(59):1–18, 1880.

[130] R. Villamarín-Salomón and J. C. Brustoloni. Identifying botnets using anomaly detection techniques applied to dns traffic. In *The 5th IEEE Consumer Communications and Networking Conference 2008, CCNC2008*, pages 476–481. IEEE, 2008.

[131] N. H. Vo and J. Pieprzyk. Protecting web 2.0 services from botnet exploitations. In *Proc. of the 2nd Workshop on Cybercrime and Trustworthy Computing , CTC'10*, Washington, DC, 2010. IEEE.

[132] L. Von Ahn, M. Blum, N. Hopper, and J. Langford. Captcha: Using hard ai problems for security. In *Advances in Cryptology-EUROCRYPT 2003*. Springer, 2003.

[133] R. Walsh, D. Lapsley, and W. T. Strayer. Effective flow filtering for botnet search space reduction. In *Conference For Homeland Security, 2009. CATCH'09. Cybersecurity Applications & Technology*, pages 141–149. IEEE, 2009.

[134] N. Weaver, V. Paxson, S. Staniford, and R. Cunningham. A taxonomy of computer worms. In *Proc. of the 2003 ACM Workshop on Rapid Malcode, WORM2003*, pages 11–18. ACM, 2003.

[135] D. Whyte, E. Kranakis, and P. C. van Oorschot. Dns-based detection of scanning worms in an enterprise network. In *Proc. of the Network and Distributed System Security Symposium 2005 , NDSS*. The Internet Society, 2005.

[136] C. Wilson. Botnets, cybercrime, and cyberterrorism: Vulnerabilities and policy issues for congress. Technical report, Foreign Affairs, Defense, and Trade Division, United States Government, 2008.

[137] S. Yadav, A. K. K. Reddy, A. Reddy, and S. Ranjan. Detecting algorithmically generated malicious domain names. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pages 48–61. ACM, 2010.

[138] B. Zdrnja, N. Brownlee, and D. Wessels. Passive monitoring of dns anomalies. In *Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 129–139. Springer, 2007.

[139] H. R. Zeidanloo, M. J. Z. Shooshtari, P. V. Amoli, M. Safari, and M. Zamani. A taxonomy of botnet detection techniques. In *The 3rd IEEE International Conference on Computer Science and Information Technology 2010, ICCSIT*, volume 2, pages 158–162. IEEE, 2010.

[140] H. Zhang, W. Banick, D. Yao, and N. Ramakrishnan. User intention-based traffic dependence analysis for anomaly detection. In *The 2012 IEEE Symposium on Security and Privacy Workshops, SPW*, pages 104–112. IEEE, 2012.

A

APPENDIX: CITRIC

This appendix gives an overview of the CITRIC framework. We developed this framework as a tool for the experimental evaluation of TFC and UDI detection.

A.1. AN OVERVIEW OF CITRIC

CITRIC(Causal Inspection to Recognize Illegal Communication) is a detection framework, developed for the experimental evaluation of TC and UDI detection. It can analyze live traffic from an interface or offline from a pcap file. Its most important function is associating new traffic flows with previous traffic flows or user events by the TFC and UDI algorithms. Malicious traffic is detected by CITRIC in the experiments, as described in Chapter 4 and 5. In addition it produces a rich set of analytical information about the classification process and related traffic properties. CITRIC is written in C++. The main tool is the detection and analysis tool, called CITRIC. Other included tools are: showFGG for postprocessing logfiles to produce graphical representations of TFC graphs, PCAP-Mix for advanced mixing of different traces, and scripts to automate multiple offline runs of CITRIC with different settings. A separate agent, installed on an observed computer, monitors user activity and sends mouse clicks and key strokes in UDP packets to CITRIC. CITRIC has been built and used for experiments on a 64 bit Ubuntu 12.04 Linux platform. With two network interfaces, the system can be inserted in a LAN as a bridge for live inspection of the forwarded traffic. Optionally CITRIC can drop traffic by firewall rule insertion in IPTables. In the experiments of Chapter 4 and 5 we use CITRIC offline with prerecorded traces.

To approach as much as possible realtime performance all data is stored in RAM during processing. Heap variables and objects are initialized, immediately after starting CITRIC. Log files are only written to persistent storage at the completion of CITRIC. During processing only one threat is active that initiates event-driven actions from a main event loop.

The amount of existing software that we could reuse was very limited for several reasons. To illustrate this we give here a few examples:

- Many implementations of network protocols, such as curl [30], assume that the software runs on a node that plays an active role in the communication (the client or server). Since CITRIC is passively listening to a dialog in which it is not participating, reuse requires a significant reprogramming effort of such solutions.

- We experienced problems with the TCPdump/Wireshark libraries [124]. It can not properly unzip HTTP chunks, there is no support for biflows, and it is not designed for real-time performance, resulting in packet loss during live analysis. Therefore we only used the pcap library for the basic capture of traffic interfaces and files.

- Some lesser known existing components were not mature enough to (re)use. We considered some components, but quickly came to the conclusion that reuse would cost to much effort. It is also very difficult to add extra code for extraction of the desired analytical information because this requires complete knowledge of the reused solution.

This forced us to build the framework almost completely from scratch. The reused libraries are limited to: pcap, zlib, pcre, and graphviz.

The CITRIC source with sample configurations and sample pcap files is made publicly available at *https://github.com/pb12/CITRIC* [17].

A.2. THE MOST IMPORTANT OBJECTS OF CITRIC

We discuss here the most important objects of CITRIC

- *PCAP and PacketAnalyzer*
  PCAP is a wrapper around the pcap library. If a new packet is available, either from a live interface or a file, PCAP will make the complete IP-packet available. PacketAnalyzer parses the IP, TCP, UDP and ICMP headers of each packet and extracts all fields that are relevant for CITRIC, including the pcap-added time-stamp. Valid packets are further processed by the FlowAggregator

- *FlowAggregator*
  FlowAggregator checks if the captured packet is a new flow by the 5-tuple $(Protocol, IP_{ingress}, IP_{egress}, Port_{ingress}, Port_{egress})$. [1]. The ingress or egress direction is determined by the IP-address of the observed computer. This IP-address is known to CITRIC. If the tuple has not been seen before, FlowAggregator stores the most important parameters in a flow object. The flows are bidirectional: both ingress and egress packets with swapped IP addresses and ports are considered to belong to the same flow. The direction of the composed bidirectional flow is defined as the direction of the first packet, hence an observed client computer will normally only produce egress flows. Each flow is stored as an object in a linked list that allows for fast insertion of new flows and fast deletion of expired flows. If a new flow is added, its direct cause and the origin of its egress IP address is searched by an object called CauseAnalyzer. When a packet of an existing flow

[1] In case of ICMP traffic the ID and Sequence fields are used instead of the port fields, to support ICMP echo traffic

arrives, its application payload is further examined by specialized helper objects. Currently a DNSHelper and HTTPHelper are implemented.

- *DNSHelper and HTTPHelper*
  The helpers are specialized objects that parse the DNS or HTTP payload. The objective is to find traffic-related events and forward references (IP-addresses, hostnames) to potential future traffic. The DNSHelper parses the DNS-answer to identify the IP-addresses and names in received A-records. Since DNS is a highly regular and relatively simple protocol, the parsing process is a straightforward selection of the appropriate fields, as defined by the DNS protocol. DNS analysis results are stored in a list of DNS objects that represents the DNS cache of CITRIC. Discovered forward references are submitted as DNS events with timestamps and additional data to an Eventcollector that manages and stores all events. HTTPHelper works in a similar way as DNSHelper for HTTP but the parsing process is substantially more difficult. We will discuss the details in Section A.4. The HTTP events that signal an identified forward reference are called URLevents. In addition, HTTPHelper can signal HTTP and HTTPS events that have no forward reference, but indicate completion of an answer. The HTTP and HTTPS events support the heuristics of TFC-detection for the cases that forward references are missed. All events are submitted to the EventCollector.

- *CauseAnalyzer*
  This object implements the TFC and UDI detection algorithm. If a new flow is detected, a search is conducted for a matching direct cause or ldi reference by the CauseAnalyzer by the TFC or UDI algorithms. The direct causes are not only traffic events, but also user events, obtained from the special UDP traffic of the remote user agent. For UDI detection the ldi of root flows is examined on length, format(IP or name), the number of special symbols, and the TLD.

  CauseAnalyzer also collects and logs a large number of additional information, related with the classification process. The classification results can trigger alarm messages and optionally firewall actions. Pointers to direct causes and the related parents are stored as attributes in the related Flow object. Each flow without a direct cause or reference is classified as root flow. Root flows, not caused by user events or with complex names are classified as anomalous, as described by respectively TFC and UDI detection. The root flow and its descendants will be associated with a Tree object that collects related statistics.

A.3. THE STORAGE OF FLOWS, DNS RECORDS AND EVENTS

For each captured IP packet a search is made in the list of existing flows, to find out if a flow with the same 5-tuple (or swapped 5-tuple) already exists. Since the number of flows can grow very large, efficient adding, removing and especially searching of flows is important. This is achieved by three techniques:

- For fast creation of new flow and deletion of closed or expired flows, the flows are organized in doubly linked lists that allow for easy insertion and removal, without sorting steps.

- To prevent malloc-related delays during packet processing, the complete storage is allocated as a large array of of flow objects during initialization of CITRIC. Only the valid flows are incorporated in a linked list.

- To increase the search speed, a hash table is used that divides the flows over 1024 different and relatively small linked lists, instead of one large linked list. The hash is calculated by a bitwise XOR of the last 10 bits of the source IP, destination IP, source Port, and destination Port. We choose this hash because the calculation requires minimal time and the hash values approach a normal distribution if enough different destinations are involved. A hashtable maps the hash to the first flow of the appropriate linked list of flows. This reduces the average search time by a factor 1024.

The storage of DNS goes in a almost similar way. For each resolved IP-address, a hash is constructed by the 10 least significant bits, to increase search speed. The storage of events is slightly different:

- Events are stored in ringbuffers instead of a linked list. A linked list is not necessary in this case, because events arrive and expire chronologically. If the buffer is large enough to store all events of the related maximum time window, there is no problem that new events overwrite the oldest events. The ringbuffer allows for a search back in time, starting from the newest event.

- The ringbuffers are created during initialization of CITRIC, to prevent malloc-related delays.

- To increase the lookup speed, a hash table is used that divides events over 256 different ringbuffers. The hash is here calculated by a modulo-256 bytewise summation over the IP-address or second level domain name.

The size of the memory that has to be reserved for storage of flows and events depends on the amount of time we want to observe. CITRIC uses an average of 2kB per bidirectional flow, including related events, DNS-cache, analysis data etc. The measured average processing time per packet is 9 us (processing on 1 Intel E7 core of 3.2GHz).

A.4. SEARCHING IN THE HTTP PAYLOAD

Searching forward references (IP-address or hostname) in HTTP is relatively difficult. There are several causes:

- HTTP is used for the transport of a high variety of payload types. Some character-based payloads, such as html and javascript, can contain forward references. Other payloads, such as an image, do normally not contain forward references.

- CITRIC has to reconstruct as a passive listener the HTTP dialog between client and server. This is complex because the exact response of a HTTP client to a received payload is difficult to predict. One of the reasons is the existence of different versions of html, javascript and other payload types. In addition there are

many browser engines that work differently and even the same browser can response differently, depending on the platform. Moreover, the HTTP server will also respond differently to different types of browsers.

- There are many features in HTTP that complicate the search:
  - HTTP1.1 supports multiple request per bidirectional flow
  - Most HTTP character-oriented responses are compressed by GZIP or DEFLATE
  - Most HTTP character-oriented responses are encoded in chunks

  Decompression and interpretation of chunk encoding is not trivial. We observed differences and irregularities, depending on the used webbrowser and the visited webserver.

- The HTTP answer is normally spread over multiple packets. Modern browsers can prefetch data of URLs, received in HTTP answers packets, even before the answer is complete. This means that CITRIC must do the same.

- Forward references are sometimes difficult to recognize. There is no fixed format or label that exclusively marks forward references.

To solve the presented difficulties, without the need for a complete browser emulation, CITRIC implements a hierarchical FSM (Finate State Machine) for each HTTP flow, to efficiently search for forward references.

- One FSM follows the HTTP dialog per flow by observing the packet size of ingress and egress HTTP packets. It can exactly determine the start, continuation and end of an HTTP answer. If an answer starts or continues, the FSM will call a routine that actively searches for forward references. If the answer finishes or exceeds a maximum size, the FSM transitions to a state that waits for a new answer. If the volume suddenly decays, the FSM generate an HTTPS or HTTP event without forward references.

- The search routine uses its own FSM to process the HTTP answer. The FSM describes the state of the HTTP answer parsing in two hierarchies. The most important primary states are HEADER, BODY and PARSED to respectively indicate the states of HTTP header parsing, payload parsing, and completion of parsing.

- The HEADER state has substates, related with the momentary field that is received.

- The BODY state has substates, related with the momentary reception of a potential reference. The most important state are IDLE, CHAR_RECEIVED and DOT_RECEIVED. The IDLE state indicates that the previous received character is not allowed in the allowed character set of DNS (letters, numbers, dash, or dot). The CHAR_RECEIVED state indicates that the last received byte or bytes are letters, numbers, or a dash. The DOT_RECEIVED state indicates that the last received

char is a dot. After the transition of CHAR_RECEIVED to another state, a routine is called that tests if the last received characters match a valid TLD (Top Level Domain as defined by the IANA [67]), preceded by a dot and DNS-allowed characters. In this way all potential hostnames are discovered.

The processing of the payload and updating of the states is done byte-by-byte. The described FSMs and a buffer with the 256 most recently received bytes represent the complete state that CITRIC requires to parse an HTTP answer that is fragmented over multiple packets. The parsing process is completely independent of the size of each fragment! In CITRIC the processing is even more complex than described here, with additional states, to support compression and chunking. For these details we refer to the source code [17].

A.5. SETTINGS AND LOGS

CITRIC is configured for specific experiments by a configuration file that contains:

- All event related time windows (TFC detection).
- Settings for statistical analysis of the ldi name (UDI detection).
- A switch between IDS and IPS mode (The IPS mode is only possible with CITRIC live running on a inserted Linux bridge, because it inserts firewall rules).
- A switch to add the DNS-ID in the flowtuple (required to distinguish DNS flows in some Windows implementations that use only one or a limited number of ephemeral ports for DNS).
- a whitelist of IP addresses and ranges
- a whitelist of domains and hostnames.

The choice between live mode or offline mode is done automatically by the name of the traffic source in the command line. If it ends with the .pcap extension it is assumed to be a prerecorded trace. In all other cases it is assumed to be a device. The output of CITRIC is delivered by 9 separate files that support extensive analysis:

- *<file>.stats:* Provides the aggregated results of the classification and detection process. This includes the applied settings, the total number of flows, packets by protocol, the number of events, identified direct causes, root flows and of course the anomalous flows.
- *<file>.log:* Log that provides all flows with cause, statistics, and tree classification (used during postprocessing for graphical representation).
- *<file>.tree:* Dump of all trees with statistical information and references to the root flow
- *<file>.event:* Dump of the event ringbuffers with the most recent captured potential events

- <*file*>*.dns:* Dump of the CITRIC DNS cache with timestamps and DNS flow info
- <*file*>*.flow:* Dump of all flows. It provides more detailed information than the .log file
- <*file*>*.http:* HTTP-specific information of the HTTP-flows
- <*file*>*.dnsstats:* Log with observed times between DNS causes and new flows for statistical analysis
- <*file*>*.urlstats:* Log with observed time between URL causes and new flows for statistical analysis

SUMMARY

Botnets play an important role in modern Internet-related cybercrime. A botnet consists of a group of infected computers, referred to as bots. The bots are remotely controlled and deployed in malicious activities, such as DDoS attacks, spam, and espionage. Clever design of the botnet C&C (Command and Control) infrastructure, combined with the adaptability of the bot and its attacks make botnets a universal cybercrime tool. This is reflected in the large number of discovered botnets and botnet-related incidents.

This dissertation aims to explore new and specialized C&C detection approaches for enterprise networks. An enterprise network is here defined as a computer network that is exclusively used by one organization. Specialized bots can attack enterprise networks with the aim to spy, to disrupt processes, or to manipulate processes and information. Detection of such bots and their C&C traffic is often difficult because they work below the radar of AntiVirus and intrusion detection systems, typically by the imitation of normal traffic, combined with the sparse and irregular production of C&C traffic. Detection is also difficult because bots that specifically target enterprise networks for espionage and manipulation do not produce noisy attack traffic, such as the traffic of typical DDoS attacks and spam distribution. Despite these detection difficulties, network-implemented detection is an attractive addition to host-implemented detection, because there is a minimal exposure to malware and a high independence of the observed platforms.

The first part of the research consists of an exploration of existing representative botnet countermeasures and C&C detection approaches. By a new ontology-based faceted classification of botnet countermeasures we achieve a systematic overview of existing botnet countermeasures and botnet traffic detection in particular. The faceted classification of detection methods also facilitates the translation of enterprise characteristics to specific properties that allow for specialized detection of the described covert C&C traffic. Enterprise networks differ from public networks by (1) their limited diversity in traffic types and connected end systems, often enforced by comprehensive control over the internal network, (2) the technical and juridical possibilities for Deep Packet Inspection, and (3) the preparedness of the network administration to put significant effort in mitigation of identified APTs (Advanced and Persistent Threats), related with espionage or sabotage. We derive from these differences a set of enterprise-specific detection properties, including Deep Packet Inspection and user activity as detection features, specification of normal traffic as detection knowledge, and a required detection state with no or minimal dependence on previous botnet traffic. Based on these derived enterprise-specific properties three new detection approaches are presented and evaluated.

The first detection approach, referred to as TFC detection (TFC=Traffic Flow Causality), detects C&C traffic by the direct causes of egress traffic. Two types of direct causes are observed: user events and traffic events. By measuring the time between user

activity and traffic, machine-generated traffic is distinguished from human-generated traffic. The identification of user events as direct causes for new traffic enables the detection of C&C traffic that uses services of popular social media. Traffic events are events, derived from received traffic, such as received DNS answers and received links in HTTP traffic. These events can also trigger new traffic. We introduce TFC Graphs (TFC=Traffic Flow Causality) that provide by the construction of trees an overview of traffic flows and their direct causes. In this way traffic flows of the same tree are associated with a root cause that is associated with normal traffic or C&C traffic. This approach results in real-time specification-based anomaly detection of various C&C traffic types. In addition it allows for offline forensic analysis of traffic. Experiments with CITRIC, a detection framework we have developed, demonstrate successful detection of different types of C&C traffic. Evasion of the detection by piggybacking on events is discussed in the evaluation.

The second approach, referred to as UDI-detection (UDI =Untrusted Destination Identification), detects C&C traffic by the estimation of the trustworthiness of egress traffic destinations. If the destination identifier of a traffic flow does not origin directly from: human input, prior traffic from a trusted destination or from a defined set of trusted applications, the destination is not trusted and its associated traffic is classified as anomalous. In contrast to TFC detection, UDI detection does not depend on time measurements and the observation of user events. This results in a less complex implementation. The experiments demonstrate successful detection of various types of C&C traffic. Similar to TFC detection, UDI detection allows for real-time repression of anomalous traffic. Evasion by the use of popular servers as intermediary is discussed in the evaluation.

The third approach detects DNS-based C&C traffic by the degree distribution of resolved DNS-domains. Domains with an unexpected high degree are classified as anomalous. In contrast to the two other approaches, this method requires a certain volume of C&C traffic, before it can detect the C&C traffic. As soon as a sufficient number of bots has resolved the same C&C domain, the domain is classified as a C&C domain and the requesting DNS clients are identified as bots. Since the required state only consists of domain names and the number of computers that queried a domain, this detection approach allows for scalable distribution over multiple enterprise networks. We derive a theoretical model of the expected degree distribution with the related False Positive Rate of the proposed detection approach. By the analysis of traffic, captured in a large network, we support the model and evaluate detection possibilities.

The ability of all three approaches to detect botnet C&C traffic differently from existing techniques and with a relatively low FPR allows for implementation in intrusion detection systems of enterprise networks alongside existing anomaly-based and signature-based detection approaches, to improve diversity. The combination of TFC and UDI detection and improvements of the event observation can further reduce evasion possibilities.

SAMENVATTING

Botnets spelen een belangrijke rol in de hedendaagse internet-gerelateerde cybercrime. Een botnet bestaat uit een groep geïnfecteerde computers, ofwel bots. Deze bots worden op afstand bestuurd en ingezet voor illegale activiteiten, zoals DDoS-aanvallen, spam en spionage. Slim ontwerp van de botnet C&C-infrastructuur (C&C=Command and Control) in combinatie met de aanpasbaarheid van bots en hun aanvallen, maken botnets tot een universeel instrument in cybercrime. Dit is terug te zien in het grote aantal ontdekte botnets en het grote aantal botnet-gerelateerde incidenten.

Dit proefschrift richt zich op de verkenning van nieuwe en gespecialiseerde vormen van C&C-detectie in bedrijfsnetwerken. Met een bedrijfsnetwerk wordt een netwerk bedoeld, dat exclusief gebruikt wordt door één organisatie. Gespecialiseerde bots kunnen bedrijfsnetwerken aanvallen met als doel te spioneren, bedrijfsprocessen te verstoren of bedrijfsprocessen en informatie te manipuleren. Detectie van dergelijke bots en hun C&C verkeer is vaak lastig omdat ze onder de radar blijven van Antivirus- en inbraakdetectie-systemen, in het bijzonder door imitatie van normaal verkeer, gecombineerd met beperkte en onregelmatige C&C communicatie. Detectie is ook lastig omdat bots die zich specifiek richten op bedrijfsnetwerken, geen luidruchtig aanvalsverkeer produceren, zoals bij DDoS-aanvallen en spam distributie. Ondanks deze detectieproblemen, is netwerk-geïmplementeerde detectie een aantrekkelijke toevoeging op host-geïmplementeerde detectie vanwege de minimale blootstelling aan malware en de onafhankelijkheid van de geobserveerde platforms.

Het eerste deel van het onderzoek bestaat uit een verkenning van bestaande representatieve botnetmaatregelen en C&C detectiemethodes. Door middel van een zelfontwikkelde ontologie-gebaseerde gefaceteerde classificatie van botnetmaatregelen wordt een systematisch overzicht verkregen van bestaande botnetmaatregelen en botnet verkeersdetectie in het bijzonder. De gefaceteerde classificatie van detectiemethodes faciliteert ook de vertaling van bedrijfsspecifieke kenmerken naar detectie-eigenschappen voor C&C verkeer. Bedrijfsnetwerken verschillen van publieke netwerken door (1) hun beperkte diversiteit in verkeerssoorten en aangesloten eindsystemen, vaak opgelegd door vergaande controle over het interne netwerk, (2) de technische en juridische mogelijkheden voor Deep Packet Inspection en (3) de bereidheid van de netwerkbeheerder om een grote inspanning te leveren voor het beperken van APT's (Advanced and Persistent Threats) op het gebied van spionage en sabotage. We leiden uit deze verschillen een aantal bedrijfsspecifieke eigenschappen af, waaronder: Deep Packet Inspection en gebruikersactiviteit als detectie features, specificatie van normaal verkeer als detectiekennis en een vereiste detectietoestand die niet of minimaal afhankelijk is van voorafgaand botnet verkeer. Op basis van deze eigenschappen worden vervolgens vier nieuwe detectiebenaderingen voorgesteld en uitgewerkt.

De eerste methode, TFC=detectie (TFC=Traffic Flow Causality), detecteert C&C verkeer door de directe oorzaak van uitgaand verkeer te bepalen. Directe oorzaken zijn:

gebruikers- en verkeersgebeurtenissen. Door de tijd te meten tussen gebruikersactiviteit en verkeer wordt machine-gegenereerd verkeer onderscheiden van mens-gegenereerd verkeer. Hierdoor kan o.a. C&C verkeer gedetecteerd worden, dat gebruik maakt van populaire sociale media. Verkeersgebeurtenissen worden afgeleid uit ontvangen verkeer, zoals DNS antwoorden en links in http verkeer. Deze gebeurtenissen kunnen ook nieuw verkeer veroorzaken. We introduceren TFC-graven (TFC=Traffic Flow Causality) die door de constructie van boomstructuren overzicht geven in verkeersstromen en hun directe oorzaken. Hiermee worden verkeersstromen binnen een boom gekoppeld aan een "root"-oorzaak, die geassocieerd wordt met normaal of botnet verkeer. Deze aanpak resulteert in real-time specificatie-gebaseerde anomaliedetectie van uiteenlopende soorten C&C verkeer. Daarnaast is ook offline forensische verkeersanalyse mogelijk. Experimenten met CITRIC, een zelfontwikkeld detectie en analyse framework, tonen succesvolle detectie aan van met zeer uiteenlopende soorten C&C verkeer. Ontwijking van de detectiemethode door middel van meeliften komen aan de orde bij de evaluatie.

De tweede methode, UDI detectie (UDI=Untrusted Destination Detection), detecteert C&C verkeer door de betrouwbaarheid te schatten van de bestemmingen van uitgaand verkeer. Als de bestemmings-identifier van een verkeersstroom niet afkomstig is uit menselijke input, uit eerder verkeer van een vertrouwde bestemming of uit een vastgestelde verzameling van vertrouwde legale applicaties, dan wordt de bestemming niet vertrouwd en het geassocieerde verkeer geclassificeerd als abnormaal. In tegenstelling tot TFC detectie, is er geen afhankelijkheid van tijdsmetingen of de directe observatie van gebruikersactiviteit. Dit resulteert in een eenvoudigere implementatie. Experimenten demonstreren succesvolle detectie van uiteenlopende soorten C&C verkeer. Net als TFC-detectie, biedt UDI-detectie de mogelijkheid voor real-time repressie van abnormaal verkeer. Ontwijking door gebruik te maken van populaire servers komt aan de orde bij de evaluatie.

De derde methode detecteert DNS-gebaseerd C&C verkeer door de graaddistributie van opgevraagde DNS domeinen te bepalen. Domeinen met een onverklaarbaar hoge graad worden geclassificeerd als abnormaal. In tegenstelling tot de twee andere detectiebenaderingen, vereist deze methode een bepaald volume aan botnetverkeer, voordat dit gedetecteerd kan worden. Zodra genoeg bots hetzelfde C&C domein hebben opgevraagd, zal het domein gedetecteerd worden en kunnen de betrokken DNS-clients als bots geïdentificeerd worden. Deze detectiemethode is geschikt voor schaalbare distributie over meerdere bedrijfsnetwerken. We leiden een theoretisch model af van de verwachte graaddistributie met de daaraan gerelateerde False Positive Rate van deze detectiemethode. Door middel van de analyse van verkeer dat gemeten is in een groot netwerk, wordt het model ondersteund en mogelijke C&C detectie geëvalueerd.

Het vermogen van alle drie de benaderingen om C&C verkeer te detecteren op een andere wijze dan bestaande methodes en met een relatief lage False Positive Rate, maakt toepassing mogelijk in intrusion-detectiesystemen van bedrijfsnetwerken, naast bestaande anomalie-gebaseerde en signature-gebaseerde methodes, waarmee de diversiteit vergroot wordt. De combinatie van TFC- en UDI-detectie en verbeteringen in de observatie van gebeurtenissen kan de mogelijkheden om detectie te ontwijken verder beperken.

CURRICULUM VITÆ

Pieter BURGHOUWT

29-06-1965 Born in Voorburg, The Netherlands.

EDUCATION

1977–1984 Grammar School
Chr. Gymnasium, Sorghvliet, The Hague

1986–1990 Bachelor in Electrical Engineering
The Hague University of Applied Sciences, The Hague

1990–1994 Master in Electrical Engineering
Delft University of Technology, Delft

LIST OF PUBLICATIONS

6. **P. Burghouwt, M. Spruit, H. Sips**, *Detection of Command and Control Traffic by the Identification of Untrusted Destinations*, to be published in Proc. of the 10th International Conference on Security and Privacy in Communication Networks 2014, SECURECOMM2014 (Springer LNICST, 2014).

5. **P. Burghouwt, M. Spruit, H. Sips**, *Detection of Covert Botnet Command and Control Channels by Causal Analysis of Traffic Flows*, in Proc. of the 5th Symposium of Cyberspace Safety and Security 2013, CSS2013 (Springer LNCS 8300, 2013), pp 117-131.

4. **P. Burghouwt, M. Spruit, H. Sips**, *Towards Detection of Botnet Communication through Social Media by Monitoring User Activity*, in Proc. of the 7th International Conference on Information Systems Security 2011, ICISS2011 (Springer LNCS 7093, 2011) pp 131-143.

3. **P. Burghouwt, M. Spruit, H. Sips**, *Detection of Botnet Collusion by Degree Distribution of Domains*, in Proc. of the International Conference for Internet Technology and Secured Transactions 2010, ICITST2010 (IEEE, 2010) pp 1-8.

2. **P. Burghouwt, H. De Goede, M. Spruit**, *Veilig Internetbankieren* (in Dutch), Informatiebeveiliging 8 (2007).

1. **P. Burghouwt**, *MFSK-demodulatie in het WISSCE-systeem* (in Dutch), Master's thesis, Delft University of Technology, Faculty of Electrical Engineering (1994).