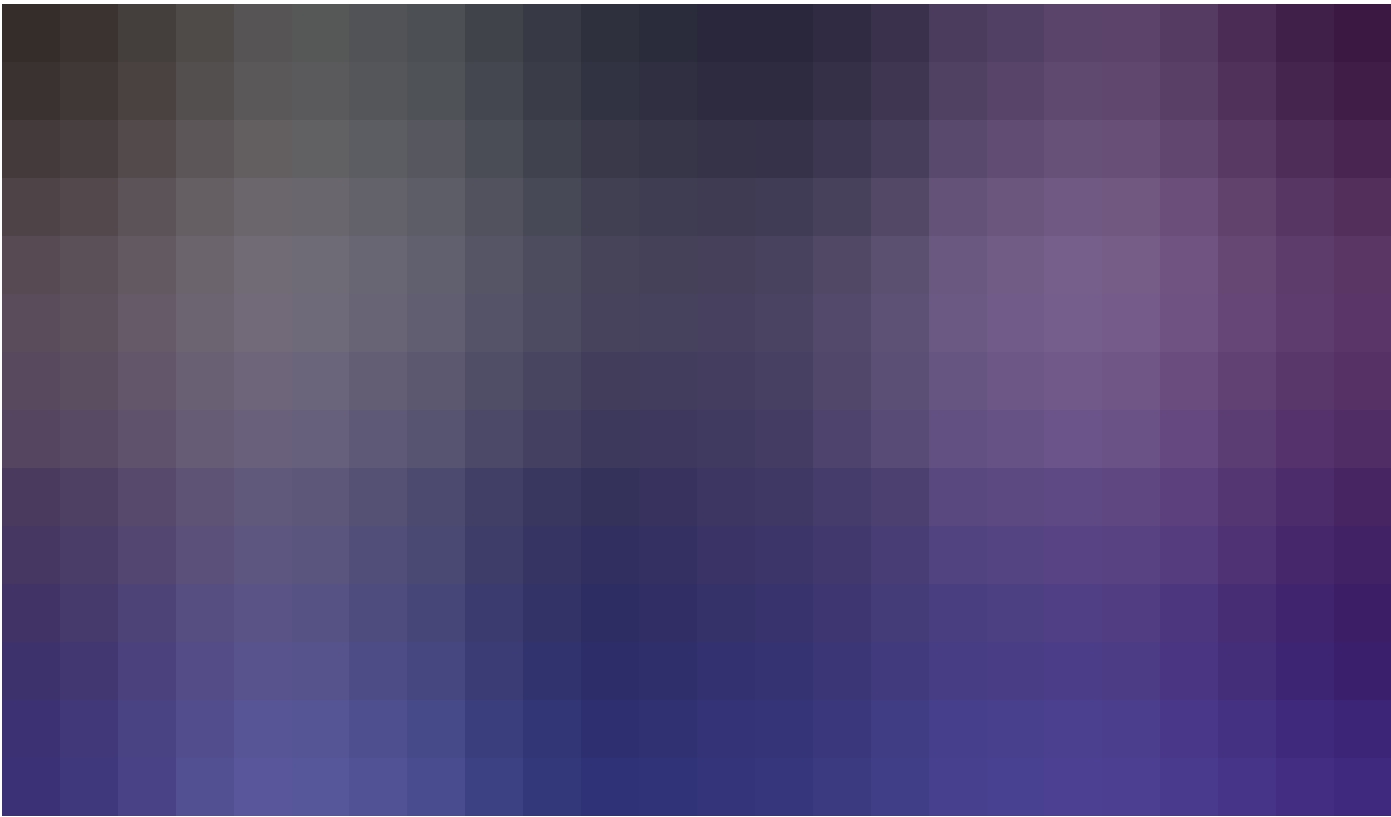


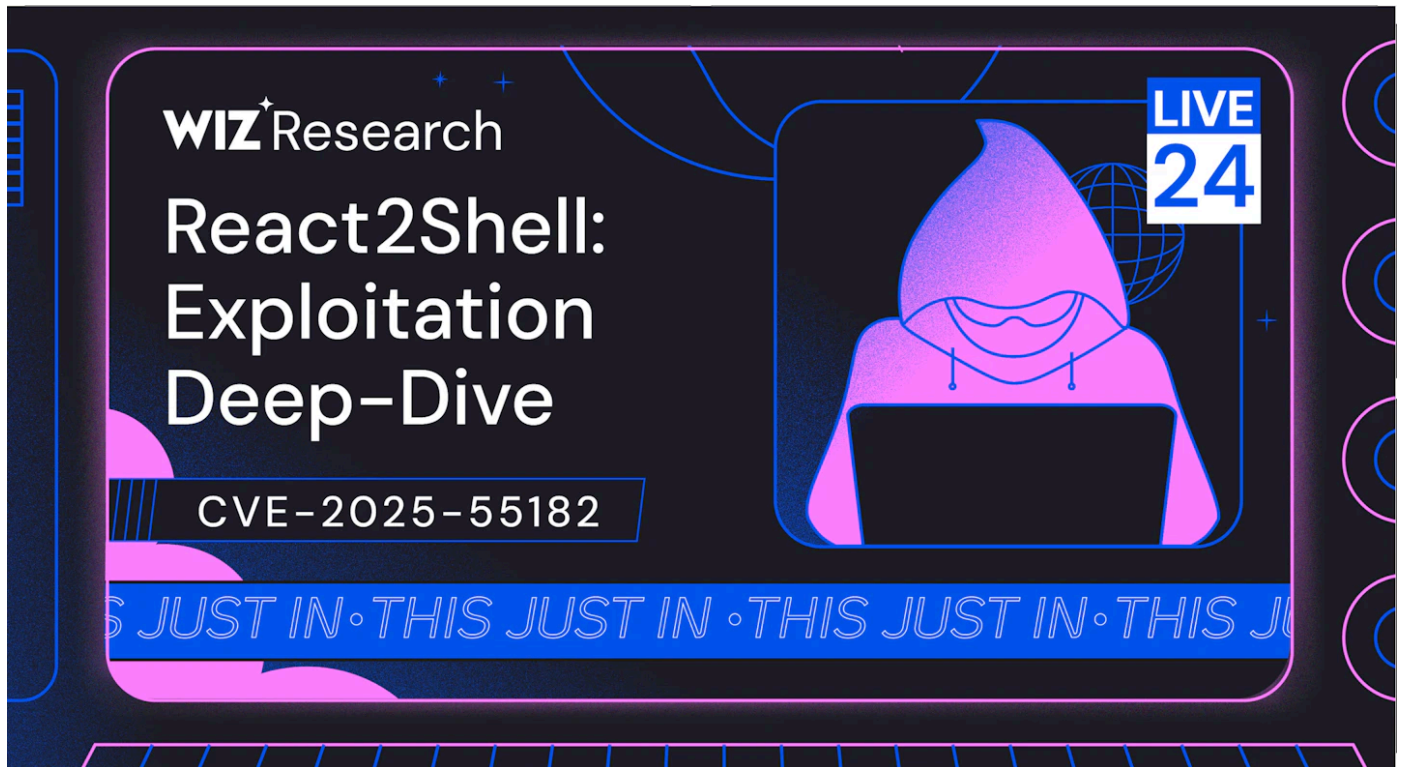
React2Shell Deep Dive: CVE-2025-55182 Exploit Mechanics

✦ wiz.io/blog/nextjs-cve-2025-55182-react2shell-deep-dive

Shir Tamari, Gili Tikochinski, Hila Ramati, Benjamin Read

December 8, 2025





Introduction

The disclosure of [CVE-2025-55182](#), a critical Remote Code Execution (RCE) vulnerability in React, has sent shockwaves through the industry. Dubbed "[React2Shell](#)," this vulnerability allows attackers to bypass security boundaries and execute arbitrary code on the server by exploiting improper input deserialization within React Server Components (RSC).

While initial reports have rightly focused on Next.js due to its massive popularity and default exposure, our research indicates the rabbit hole goes much deeper. This is not merely a framework-specific bug; it is a fundamental issue with how RSC payloads are handled, with implications reaching far beyond the Vercel ecosystem.

In this post, we present our extensive research into React2Shell. We will move beyond the basic headlines to explore:

- **Active Exploitation:** Exclusive data from Wiz regarding what we are observing in the wild right now, from opportunistic cryptominers and "smash-and-grab" credential harvesting to sophisticated, persistent backdoors leveraging Sliver implants.
- **The Broader Scope:** Why frameworks like **Waku** and **Vite** (with RSC plugins) are also vulnerable, and how the "default open" nature of [Next.js](#) - specifically the `next-action` header - makes it a prime target.
- **The Technical Mechanics:** A deep dive into the deserialization logic, the self-referencing "gadget" chains, and how exactly the released PoC gets to RCE.

Whether you are patching a Next.js application, architecting a custom React server, or defending cloud-native workloads, understanding the mechanics of this exploit is critical. For detection and remediation guidance see our response [blogpost](#).

Exploitation In The Wild

We've observed a rapid wave of opportunistic exploitation of CVE-2025-55182, with attackers pivoting quickly from simple recon to hands-on-keyboard abuse in cloud-native environments. Most attacks target internet-facing Next.js applications and other containerized workloads running in Kubernetes and managed cloud services.

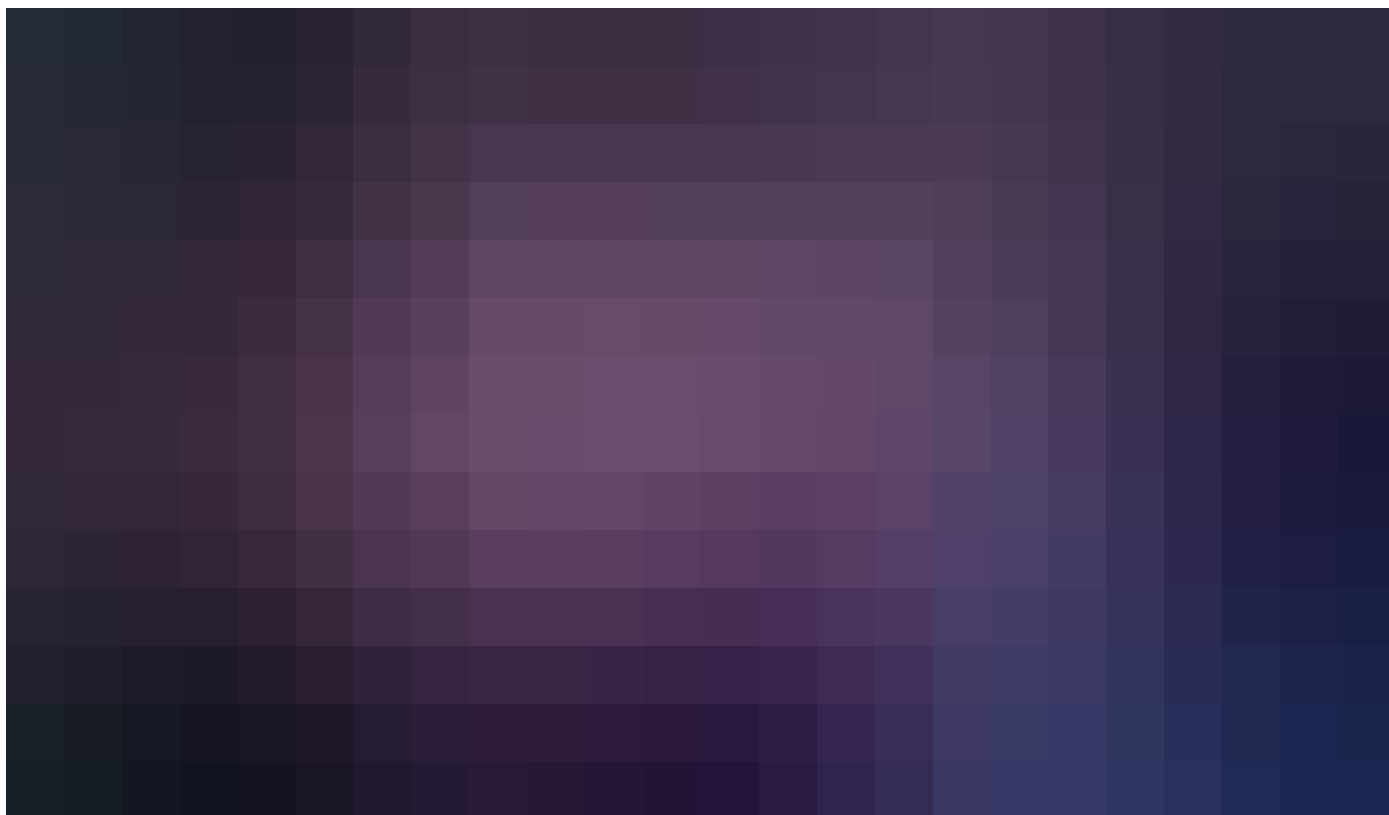


Diagram showing the different attacker payloads using React2Shell

Cloud-native initial access and recon

In multiple cases, attackers used the vulnerability to obtain an interactive shell inside application containers (for example, Next.js frontends running in Kubernetes/GKE). From there, we consistently see:

- Environment and identity discovery: `whoami`, `hostname`, environment variable dumps, and enumeration of `/etc/passwd`, often wrapped into one-liners and sent back to attacker-controlled infrastructure or oastify domains via `curl` or `nc`.

- DNS-based beaconing: extensive use of oast*. domains and similar callback infrastructure to confirm code execution and network egress from cloud workloads, often embedding application-specific hostnames and URLs into the callbacks to fingerprint the target environment while these might be noise from scanners and bug bounty hunters a lot of those commands sent environment variables to the oast.* domains, acting beyond what is needed for simple scanning.

In several incidents, we saw the exploit chain progress from simple connectivity checks to fully interactive reverse shells directly from the application runtime (Node.js / Next.js processes) to external command-and-control servers.

Credential harvesting and cloud metadata access

Multiple campaigns show a strong focus on cloud and developer credential theft:

- One actor used a reverse shell to systematically dump npm, AWS, Docker, Git, SSH and application `package.json` data from a Next.js server, all base64-encoded for exfiltration.
- A separate campaign executed a Base64-encoded script that:
- Scrapes environment variables for cloud and application secrets (e.g., `AWS`, `TOKEN`, `SECRET`, `PASS`, `DB_`).
- Recursively scans key filesystem paths (`/home`, `/root`, `/etc`, `/var/www`, `/opt`) for config and key material (`*.env`, JSON/YAML configs, SSH keys, etc.), while avoiding large or noisy files.
- Attempts to access the cloud instance metadata service at `169.254.169.254/latest/meta-data/iam/security-credentials/` to retrieve IAM credentials, indicating clear cloud-specific privilege escalation intent.

Another large, standalone shell script retrieved by attackers performs broad secret harvesting at scale:

- Walks `/root` and all `/home/*` directories.
- Targets common cloud/dev paths such as `.ssh`, `.aws`, `.kube`, `.config/gcloud`, and multiple cryptocurrency wallet locations.
- Captures environment, OS details, network interfaces, process list (`ps aux`), and network connections (`netstat -anpt`).
- Bundles all findings into a single report file and uploads it via HTTP POST to attacker-controlled infrastructure, using whichever tool is available (`curl`, `wget`, or `python`).

This behavior highlights a clear trend: attackers treat compromised containers as credential collection points for both cloud control planes and adjacent developer tooling.

Cryptominer deployment in containers

We observed several distinct cryptomining campaigns leveraging CVE-2025-55182, all targeting cloud workloads:

- One campaign dropped a UPX-packed XMRig variant and used custom infrastructure to distribute shell scripts that:
- Kill competing miners and processes.
- Attempt local privilege escalation (e.g., via CVE-2021-4034).
- Masquerade as plausible system processes (e.g., `systemd-devd`) to blend into container process lists.
- Another campaign simply pulled stock XMRig from GitHub, configured with attacker-controlled mining pools, and ran it from writable locations like `/tmp`.
- In more advanced activity, actors retrieved installer scripts from c3pool and executed them via an interactive shell, wiring the cryptominer to Monero wallets and using `nohup /var/tmp/crond` as a disguised persistence mechanism inside the container.

Several of these campaigns are multi-tenant, reusing the same script families and infrastructures (`anywherehost[.]site`, `inerna1[.]site`, and related hosts) across different customers and clusters.

Backdoors and post-exploitation frameworks

Beyond mining, at least one campaign deployed a fully featured backdoor:

- Attackers downloaded shell scripts from a dynamic DNS host that in turn fetched Sliver payloads (64-bit ELF binaries) and executed them from temporary locations.
- These Sliver implants communicated with an external C2 over IP infrastructure that reused TLS certificates and domains across multiple samples, indicating a dedicated attacker-controlled ecosystem rather than opportunistic commodity malware.

This represents a meaningful escalation from “smash-and-grab” cryptomining to long-term access and operator-driven post-exploitation inside cloud workloads.

Overall, the observed activity shows a clear pattern: attackers are using CVE-2025-55182 not just to run one-off commands, but to gain interactive, cloud-aware access to containerized workloads, aggressively harvest secrets, weaken local defenses, and monetize access through cryptomining and backdoor deployment.

The Vulnerability Extends Beyond Next.js

One important aspect we noticed was missing in other coverage of this Proof-of-Concept (PoC) is its applicability to other non-Next.js platforms that utilize React Server Components (RSC).

We believe Next.js was the primary focus for two main reasons:

- It is the most popular framework using the RSC feature.
- The RSC feature is enabled and exposed by default on all Next.js applications.

This second point is crucial, as it explains the PoC's potency: any Next.js application - whether or not it has Server Actions defined - has the vulnerable flow accessible simply by adding the `next-action` header to the HTTP request. There is no need to guess or find a correct action name; all payloads sent with this header are parsed.

Because of this ease of exploitation, other platforms where this vulnerable flow is reached differently may have been excluded from the initial conversation, but they remain extremely vulnerable.

In our internal research, we successfully executed code using this PoC (with minor adjustments) on both Waku and Vite (with the RSC Plugin). With only minor modifications to the PoC, we are confident that more frameworks are vulnerable and would require only very minor adjustments to be exploited as well.

In a custom "native" React setup, the decoding logic is only invoked if the developer has explicitly architected a server to support RSC. The vector requires the attacker to identify the specific, custom endpoint where the server manually calls the decoder function.

```
// VECTOR: Manual RSC Server Implementation
// The application explicitly imports and uses the vulnerable decoder.
import { decodeReply } from 'react-server-dom-webpack/server';

app.post('/my-rsc-endpoint', async (req, res) => {
  // The raw body is passed directly to the RSC decoder
  const args = await decodeReply(req.body);
});
```

After finding the endpoint to reach the server actions logic any application using the vulnerable react versions is susceptible to this exploit.

That means that detection of vulnerable servers must be done holistically both with dynamic scanners with the current PoCs but also with code and disk validation that can detect the vulnerable packages in lesser known platforms or "native" react implementations.

PoC Breakdown

Since there are already excellent online explanations of this vulnerability - for example, [Guillermo Rauch](#)'s (Vercel's CEO) - we will focus on explaining the main points briefly.

The vulnerability exploits improper input **deserialization of React Server Component (RSC) form data payloads**. By crafting a malicious payload, an attacker can achieve **Remote Code Execution (RCE)**.

Let's break down the proof-of-concept (PoC) payload, which relies on a complicated structure with several self-references:

```
const payload = {
  '0': '$1',
  '1': {
    'status': 'resolved_model',
    'reason': 0,
    '_response': '$4',
    'value': '{"then": "$3:map", "0": {"then": "$B3"}, "length": 1}',
    'then': '$2:then'
  },
  '2': '$@3',
  '3': [],
  '4': {
    '_prefix': 'console.log(7*7+1)//',
    '_formData': {
      'get': '$3:constructor:constructor'
    },
    '_chunks': '$2:_response:_chunks',
  }
}
```

Gadget Construction (Chunks 2, 3, and 4)

The payload works by replacing legitimate objects with attacker-controlled "gadgets" during deserialization, as the vulnerable code lacks type checking in crucial points.

1. Chunks 2 and 3 (The Function() Gadget):

- '3': [] is a simple empty array.
- '4'._formData.get is pointed to **\$3:constructor:constructor**.
- An array's constructor is Array.
 - Array.constructor is the native **Function()** constructor.
 - This Function() constructor is the **primary RCE gadget**, as it behaves similarly to eval(), creating a callable function from a string of JavaScript.
- '4'._prefix holds the malicious JavaScript to be executed: 'console.log(7*7+1)//'.
- '2': points to `\$\$@3` this tells the de-serializer to treat the chunk as a **Promise** object, which causes our `.then` to be executed later.

2. Chunk 4:

3. This object mimics the internal Response object used by the deserializer, but with properties pointing to our gadgets:

1. `_formData.get` -> `Function()`
2. `_prefix` -> **Our Arbitrary Code**

4. **Chunk 1:**

1. This object is designed to look like a Chunk object, but it references our crafted Response object (`_response: $4`).

Exploit Execution Flow

The vulnerability is triggered during the deserialization logic so let's follow this logic step-by-step

1. **Initial Resolution:** Parsing begins with `'0': '$1'`, leading to the resolution of **Chunk 1**.
2. **Gadget Injection:** Resolving Chunk 1's ``_response`: '$4'` forces the deserializer to resolve Chunk 4. During this process, Chunk 4's `_formData.get` is set to the `Function()` gadget, and its `_chunks` property is set via `'$2:_response:_chunks'`, which is necessary to correctly link the object into the deserialization context.
3. **Promise Chain Trigger:** The core of the exploit lies in Chunk 1's value property, which is a nested JSON object designed to trigger a promise chain:
 1. `'value': '{"then":"$3:map","0":{"then":"$B3"},"length":1}'`
4. **RCE Trigger:** The innermost reference, `$B3`, is processed. This triggers the vulnerable code snippet:
JavaScript

```
case "B":
    return obj = parseInt(value.slice(2), 16), response._formData.get(response._prefix +
obj);
```

- The response object here is our crafted **Chunk 4**.
- The call becomes `Chunk_4._formData.get(Chunk_4._prefix + obj)`.
- This effectively executes: `Function( 'console.log(7*7+1)//' + obj )`.

1. **Final Execution:** The resulting anonymous function containing the attacker's code is returned up the promise chain and is ultimately called by the then resolutions, resulting in **code execution**.



Diagram Explaining the Execution Chain

How Wiz Can Help?

For a broader look at how Wiz helps teams identify, prioritize, and respond to vulnerabilities like this across their cloud environment, check out our latest post: [React2Shell \(CVE-2025-55182\): Everything You Need to Know About the Critical React Vulnerability](#).



Appendix IOC List

Type	Subtype	IOC
IP Address	C2	37.27.217.205
IP Address	C2	212.237.120.249
IP Address	Exfil	5.161.227.224
IP Address	Sliver C2	154.26.190.6

Type	Subtype	IOC
IP Address	Malware Host	45.32.158.54
IP Address	Malware Host	104.238.61.32
IP Address	Malware Host	193.34.213.150
IP Address	Malware Host	154.89.152.240
IP Address	Payload Host	172.245.79.16
IP Address	Stealer	47.84.82.8
IP Address	Stealer	8.222.213.56
IP Address	Miner Host	216.158.232.43
IP Address	Malware Host	185.229.32.220
Domain	Malware Infra	anywherehost.site
Domain	Malware Infra	inerna1.site
Domain	Malware Infra	ip.inovanet.pt
Domain	Sliver Infra	keep.camdvr.org
Domain	Sliver Cert	t.cnzzs.co
Domain	Loader Infra	ax29g9q123.anondns.net
Domain	Loader Infra	aws.orgserv.dnsnet.cloud.anondns.net
Domain	Exfil / SMTP	mail.wruff.de

Type	Subtype	IOC
Domain	Beacon	tr.earn.top
URL	Dropper	hxxp://anywherehost.site/xms/k1.sh?grep
URL	Dropper	hxxp://anywherehost.site/xms/kill2.sh
URL	Dropper	hxxp://anywherehost.site/xms/su
URL	Dropper	hxxp://anywherehost.site/xms/t1.ps1
URL	Miner	hxxp://anywherehost.site/xb/runner.zip
URL	Miner	hxxp://anywherehost.site/xb/systemd-devd.\$(uname -m)
URL	Dropper	hxxp://inerna1.site/xms/k1.sh
URL	Miner	hxxp://ip.inovanet.pt/systemprofile.zip
URL	Miner	hxxp://inerna1.site/xb/systemd-devd.x86_64
URL	Miner	hxxp://inerna1.site/xb/runner.zip
URL	Dropper	hxxp://inerna1.site/xms/t1.ps1
URL	Malware	hxxp://45.32.158.54/5e51aff54626ef7f/x86_64
URL	Malware	hxxp://193.34.213.150/nuts/bolts
URL	Malware	hxxp://193.34.213.150/nuts/x86
URL	Miner	hxxp://superminecraft.net.br:3000/sex.sh
URL	Miner	hxxp://216.158.232.43:12000/sex.sh
URL	Sliver Payload	hxxp://keep.camdvr.org:8000/BREAKABLE_PARABLE5
URL	Dropper	hxxp://154.89.152.240/check.sh
URL	Malware	hxxp://185.229.32.220:21642/2lt4de8wgl54wtjgo8/winds
URL	Sliver Dropper	hxxp://keep.camdvr.org:8000/d5.sh
URL	Sliver Payload	hxxp://keep.camdvr.org:8000/BREAKABLE_PARABLE10
URL	Stealer	hxxp://47.84.82.8/index

Type	Subtype	IOC
URL	Stealer Exfil	hxxp://47.84.82.8/upload
URL	Stealer	hxxp://8.222.213.56/index
URL	Malware	hxxp://104.238.61.32:8080/zold
URL	Loader	hxxp://ax29g9q123.anondns.net
URL	Beacon	hxxps://tr.earn.top/Log.php?id=
SHA1	Script	264e1a820b8b3bbd13325955f06aff2678c69935
SHA1	Script	20e1465fd07f0d4e19c299fb0d9af8e5ec1b21d2
SHA1	Script	6e43e26fa62dfa89fe8b016dc831a9ec44507af9
SHA1	Miner	d6e97c9783f0907f1ee9415736816e272a9df060
SHA1	Miner	732226c0966fe29116b147e893c35ce7df1c8f1a
SHA1	Miner	be86823d73a01266b096dab1628cfa2e4ca77265
SHA1	Malware	7fe3826fc7b90e20c9fe76a7891eff350d73b6b3
SHA1	Miner	7c8010d9ab6dfdc7a99aba7075a793260acbf2b8
SHA1	Stealer	91152e6ffe0474b06bb52f41ab3f3545ac360e64
SHA1	Stealer	5d368356bd49c4b8e3c423c10ba777ff52a4f32a
SHA1	Payload	34551bca762be99d732c0ced6ad8b0a2f7b11ad7
SHA1	Script	1ce4b6a89d2daa0cab820711d8424a7676ef5ff2
SHA1	Sliver	0972859984decfaf9487f9a2c2c7f5d2b03560a0
SHA1	Sliver	470ce679589e1c3518c3ed2b818516f27ccad089
SHA1	Dropper	c67e8aa881317cb32d7c36b2e3c0c5cfa21bf5e3
SHA1	Sliver	0972859984decfaf9487f9a2c2c7f5d2b03560a0
SHA1	Sliver	2937c58115c131ae84a1b2a7226c666f6a27ef88
SHA1	Script	1ce4b6a89d2daa0cab820711d8424a7676ef5ff2

Type	Subtype	IOC
Monero	Wallet	44VvVLU2Vmja6gTMbhNHAzc7heYTiT7V mQEXkjdaYo6K41WqH8qWw1CL8wKAAgz5 xLYT3XL3pb9KCUZS7PPZbzUGCCpZ9Ee
Monero	Wallet	42NTfUjbU3Gj536zubU7vpjfC7X9DPEC ciwbCXrrjBk5KqkJS1Xq4saVgQLP1yqU YHKzn7apt1p3W6mDWm87n3nwDEmWeSh

Tags

#Research	#Threat Intel
---------------------------	-------------------------------