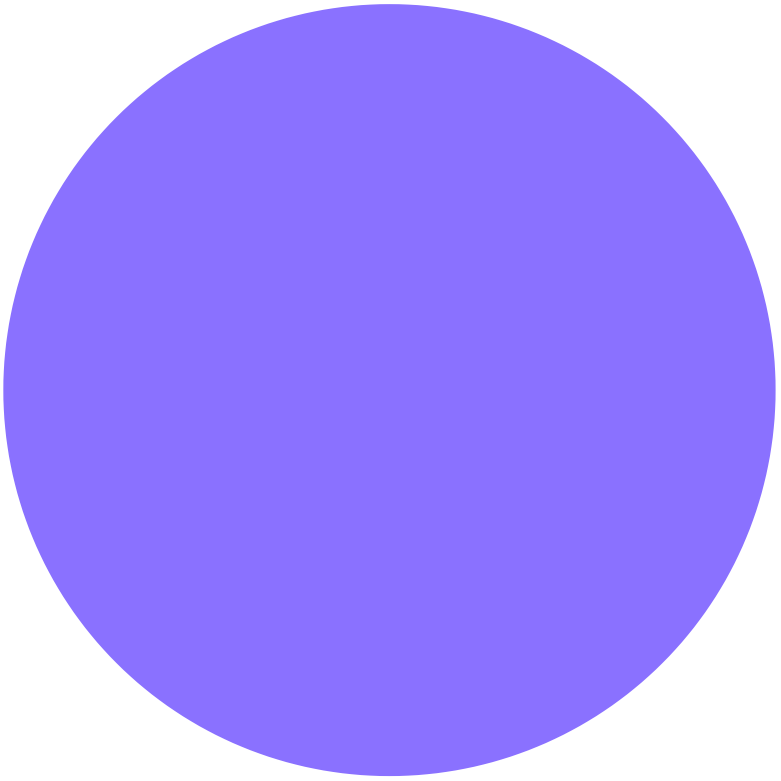
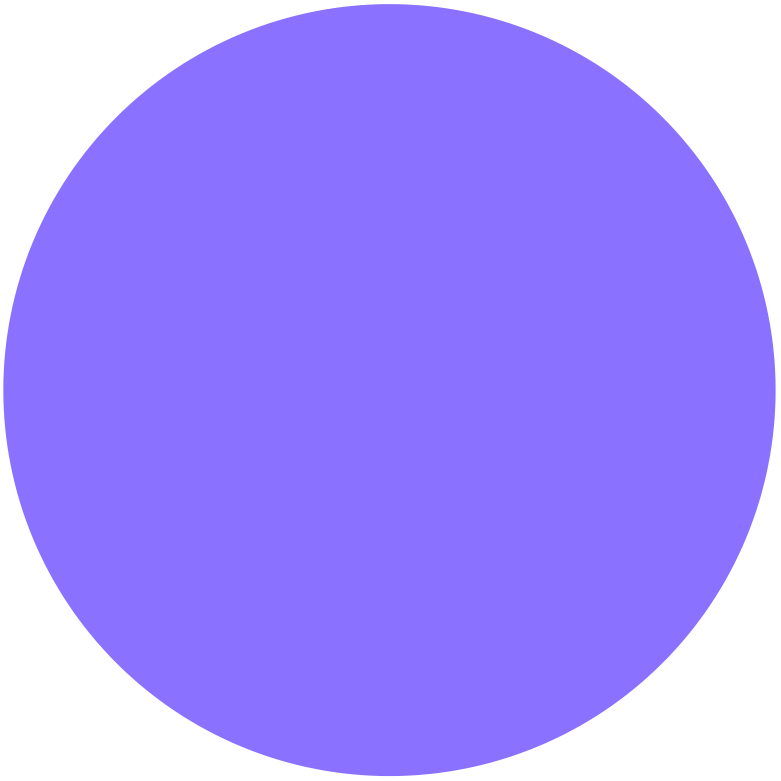
Ghost in the Zip | New PXA Stealer and Its Telegram-Powered Ecosystem

Archived: 2026-04-06 00:07:40 UTC

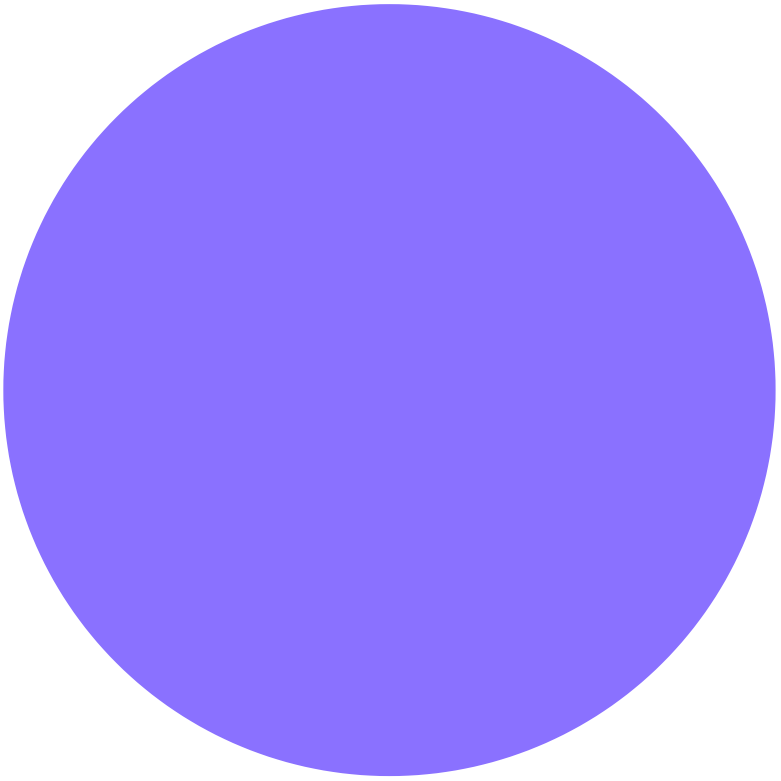
Executive Summary



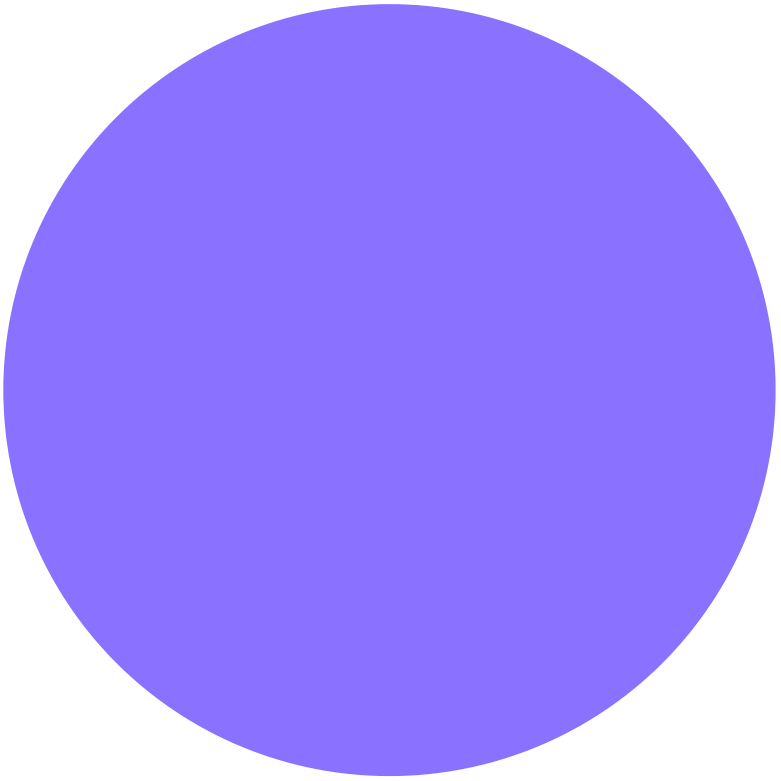
- Beazley Security and SentinelLabs discovered and analyzed a rapidly evolving series of infostealer campaigns delivering the Python-based PXA Stealer.



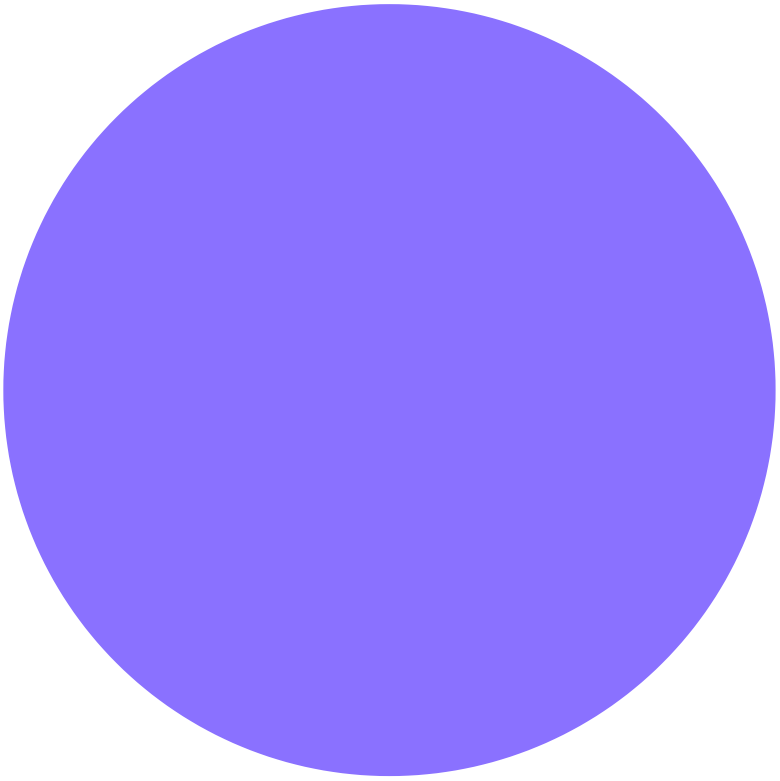
- This discovery showcases a leap in tradecraft, incorporating more nuanced anti-analysis techniques, non-malicious decoy content, and a hardened command-and-control pipeline that frustrates triage and attempts to delay detection.



- We identified more than 4,000 unique victim IP addresses in exfiltrated logs, with infected systems spanning at least 62 countries, most notably South Korea, the United States, the Netherlands, Hungary, and Austria.



- The stolen data includes over 200,000 unique passwords, hundreds of credit card records, and more than 4 million harvested browser cookies, giving actors ample access to victims’ accounts and financial lives.



- The threat actors behind these campaigns are linked to Vietnamese-speaking cybercriminal circles who monetize the stolen data through a subscription-based underground ecosystem that efficiently automates resale and reuse through the Telegram platform’s API.

Overview

In close partnership, Beazley Security and SentinelLabs have uncovered a large-scale, ongoing infostealer campaign built around the Python-based PXA Stealer. Initially surfacing in late 2024, this threat has since matured into a highly evasive, multi-stage operation driven by Vietnamese-speaking actors with apparent ties to an organized cybercriminal Telegram-based marketplace that sells stolen victim data.

Throughout 2025, these actors have continuously refined their delivery mechanisms and evasion strategies. Most notably, they’ve adopted novel sideloading techniques involving legitimate signed software (such as Haihaisoft PDF Reader and Microsoft Word 2013), concealed malicious DLLs, and embedded archives disguised as common

file types. These campaigns use elaborate staging layers that obscure their purpose and delay detection by endpoint tools and human analysts alike.

The final payload, PXA Stealer, exfiltrates a broad spectrum of high-value data–which includes passwords, browser autofill data, cryptocurrency wallet and FinTech app data, and more– to Telegram channels via automated bot networks. Our telemetry and analysis uncovered over 4,000 unique victims across more than 60 countries, suggesting a widespread and financially motivated operation that feeds into criminal platforms such as Sherlock. This data is then monetized and sold to downstream cybercriminals, enabling actors who engage in cryptocurrency theft or buy access to infiltrate organizations for other purposes.

This campaign exemplifies a growing trend in which legitimate infrastructure (e.g., Telegram, Cloudflare Workers, Dropbox) is weaponized at scale to both execute and monetize information theft, while simultaneously reducing the cost and technical overhead for attackers. As stealer campaigns become increasingly automated and supply-chain integrated, defenders must adjust to an adversary landscape defined not just by malware, but by infrastructure, automation, and real-time monetization.

Beazley Security would like to extend sincere thanks to our partners at SentinelOne for their instrumental collaboration and exceptional reverse engineering support during this investigation.

Background and Haihaisoft Sideloading

This cluster of PXA Stealer activity has been ongoing and active since late 2024, with some BotIDs being created as early as October, 2024. The general delivery mechanisms and TTPs have not changed. However the actors behind this cluster have continually pivoted to new sideloading mechanisms, along with updated Telegram C2 infrastructure.

During a wave of attacks occurring in April 2025, users were phished or otherwise lured into downloading a compressed archive containing a signed copy of the Haihaisoft PDF Reader freeware application along with the malicious DLL to be sideloaded. This component of the attack is responsible for establishing persistence on the target host via the Windows Registry, and retrieving additional malicious components, including Windows executable payloads hosted remotely on Dropbox. Various infostealers were delivered in this initial campaign, including LummaC2 and Rhadamanthys Stealer.

It was during the first wave that we also observed a change in TTPs: the threat actors shifted to updated Python-based payloads instead of Windows executables.

Attacks leveraging the updated Python-based payloads are initiated in the same manner: delivery of a large archive containing the signed copy of Haihaisoft PDF Reader, alongside the malicious DLL to be loaded.

Upon execution, the malicious DLL creates a .CMD script `Evidence.cmd` in the current directory, which orchestrates all subsequent steps in the attack chain. The .CMD script utilizes `certutil` to extract an encrypted RAR archive embedded inside a malformed PDF.

```
certutil -decode Documents.pdf LX8bzeZTzF5XSONpDC.rar
```

This command leads the Edge browser to open the PDF file, though this results in an error message as the file is not a valid PDF. Subsequently, the packaged WinRAR utility–masquerading as `images.png` –extracts an embedded RAR archive using decoded command lines. This process took several minutes and caused sandbox analysis to time out in several cases, which led to false negative results.

```
images.png  x -pS8SKXaOudHX78CnCmjawuXJAXwNAzVeK -inul -y LX8bzeZTzF5XSONpDC.rar
C:\Users\Public\LX8bzeZTzF5XSONpDC
```

This extracts several Python dependencies, including a legitimate Python 3.10 interpreter renamed `svchost.exe` and a malicious Python script named `Photos` , which are then executed. This step sets a Registry Run key to ensure the payload will run each time the computer starts.

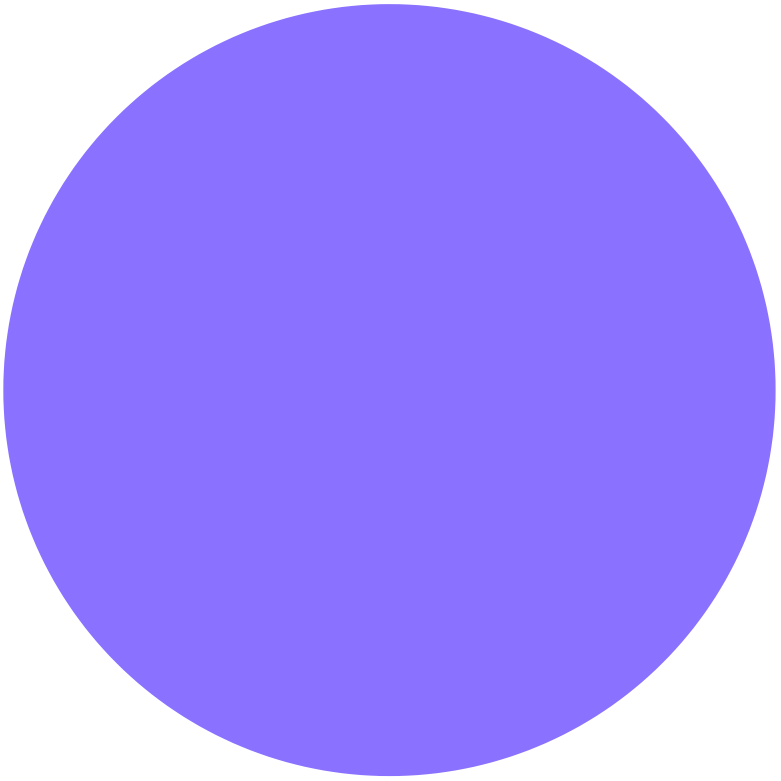
```
reg add "HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run" /v "Windows Update Service" /t REGSZ _/d "cmd.exe /c start \'
```

```
\"C:\Users\Public\LX8bzeZTzF5XSONpDC\svchost.exe\"C:\Users\Public\LX8bzeZTzF5XSONpDC\Photos" /f
```

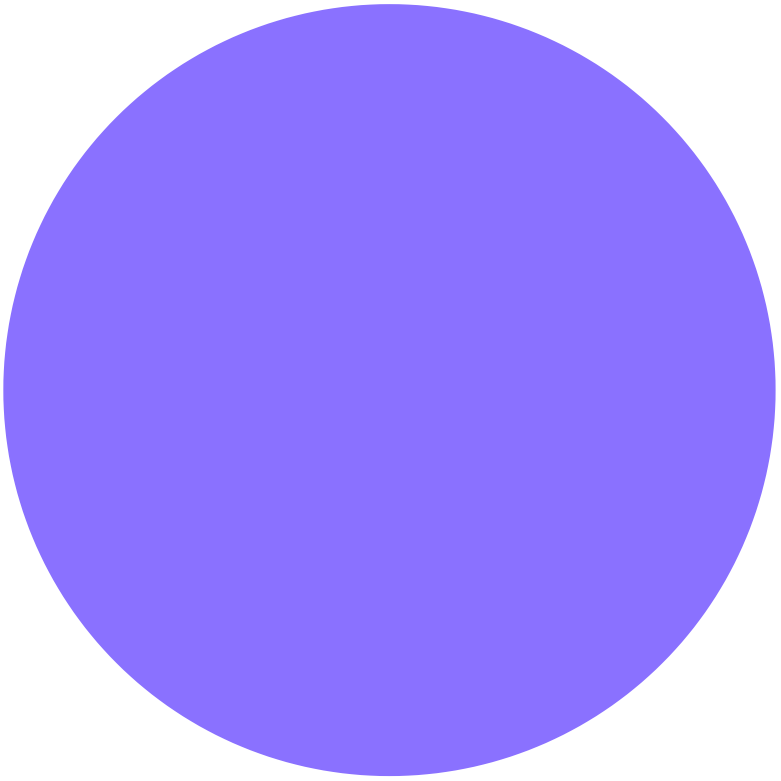
Evolved Infection Chain

In July 2025, Beazley Security MDR identified new activity that closely mirrored the infection chain and TTPs observed in the previous campaigns, but with several notable evolutions reflecting heightened operational maturity and ongoing innovation by the threat actors.

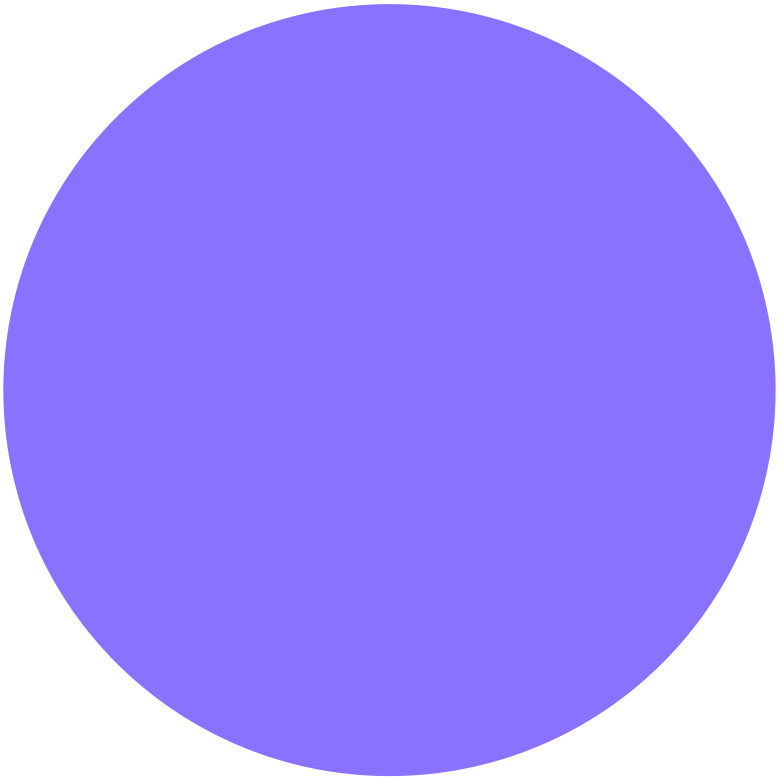
The large archive attached to the phishing lure contained:



- A legitimate, signed Microsoft Word 2013 executable



- A malicious DLL, `msvcr100.dll`, that is sideloaded by the Microsoft Word 2013 executable



- Additional files and later-stage payloads within a supporting directory named "_".

While similar to the April campaign, the July wave introduces more sophisticated file naming to increase evasion and leverages non-malicious decoy documents opened to ensure the user remains unsuspecting.

The Microsoft Word 2013 binary is renamed to appear to the user as a Word document:

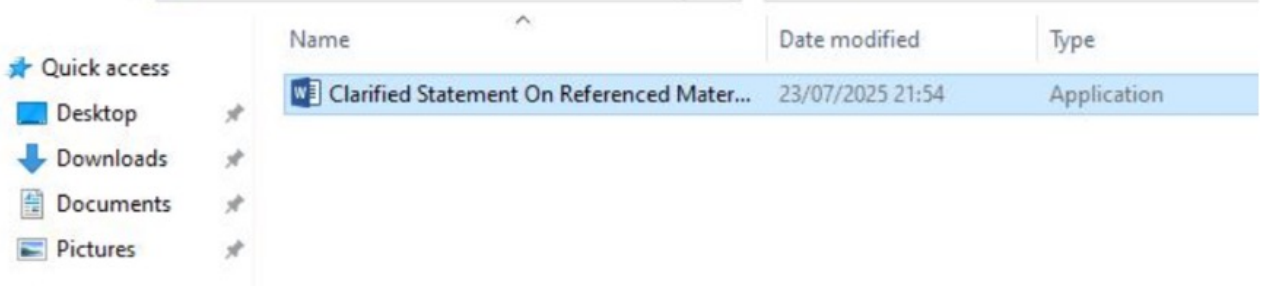


Figure 1 - Screenshot of renamed Word 2013 executable to lure the user

The other files extracted from the archive are hidden from the user in Windows Explorer but shown below:

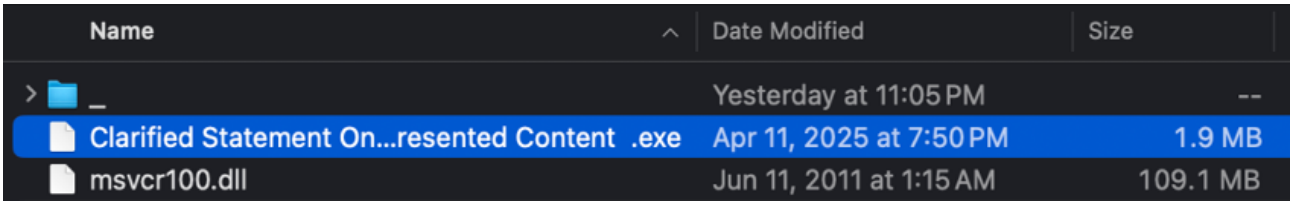


Figure 2 - Extracted contents of the archive, including hidden files

When the victim opens the Word executable, Windows loads the malicious `msvcr100.dll` since the OS searches for the filename in the local directory before system directories. The sideloaded DLL then launches a hidden instance of Command Prompt and begins a multi-stage chain of activity:

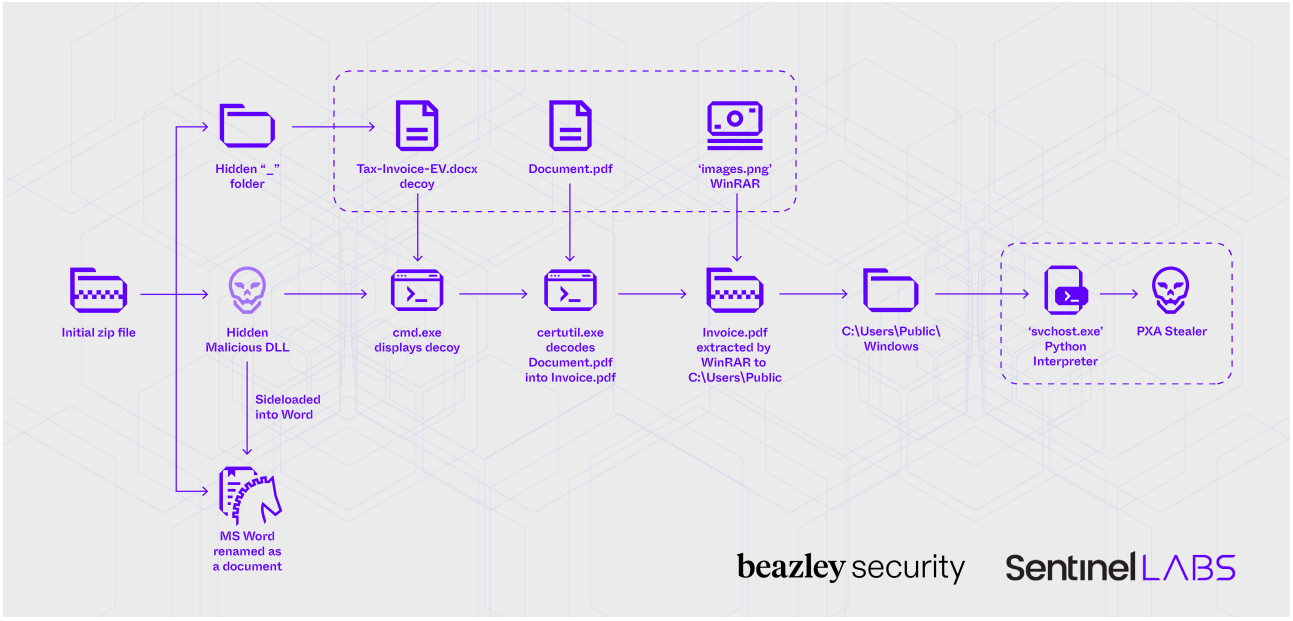


Figure 3 - Overview of the infection chain

First, Word launches a benign decoy document named `Tax-Invoice-EV.docx`, which displays a fake copyright infringement notice to the victim. We believe this document doubles as an anti-analysis feature by introducing a non-malicious file into the attack chain, which potentially wastes security analysts' time. The document lacks macros or other scriptable objects.

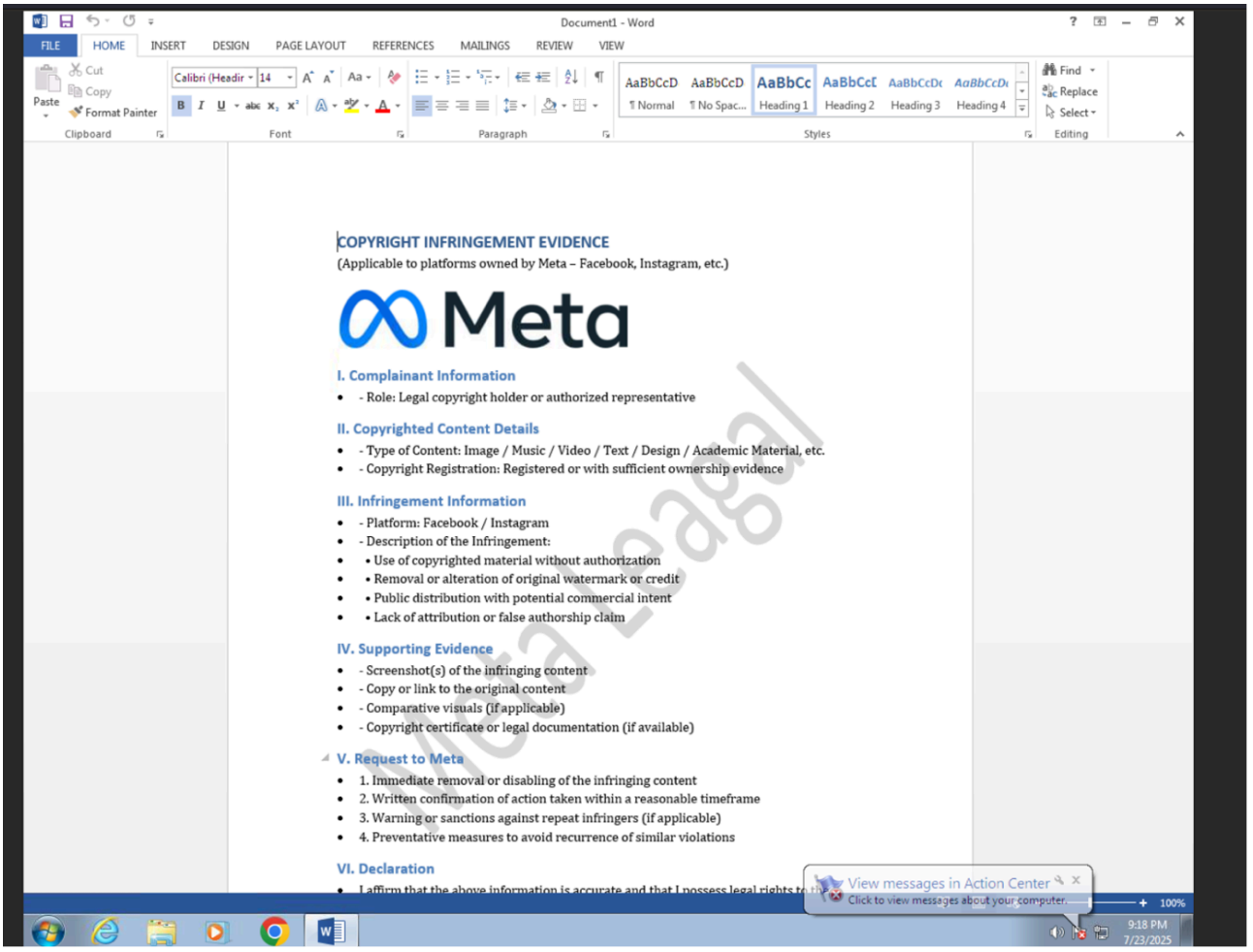


Figure 4 - Screenshot of the non-malicious decoy document

Next, like the previous activity, `certutil` is used to decode a file from the "-" folder into a new encrypted zip archive that is deceptively named with a PDF file extension, Document.pdf for example:

```
certutil -decode Document.pdf Invoice.pdf
```

Then, a legitimate WinRar executable also hosted in the "-" folder renamed `images.png` is used to unpack the archive:

```
images.png x -ibck -y -poX3ff7b6Bfi76keXy3xmSWnX0uqsFYur Invoice.pdf C:\\Users\\Public
```

The second archive contains a portable Windows Python interpreter, several Python libraries, and a malicious Python script. The Python interpreter is renamed to `svchost.exe` and launches a heavily obfuscated Python script again disguised as `images.png`, followed by the `$BOT_ID` argument.

```
start C:\\Users\\Public\\Windows\\svchost.exe C:\\Users\\Public\\Windows\\Lib\\images.png
```

Payload Analysis

The final payload is an updated version of PXA Stealer. PXA Stealer is a Python-based infostealer which first emerged in 2024. PXA is primarily seen in Vietnamese-speaking threat actor circles. The malware targets sensitive information including credentials, financial data, browser data and cookies, and cryptocurrency wallet details. As detailed below, a wide variety of applications and data types within these categories are supported by PXA Stealer. PXA Stealer is capable of exfiltrating data via Telegram, as has been observed in prior campaigns.

Similar to prior campaigns, the newly observed PXA Stealer payloads are capable of identifying, packaging, and exfiltrating data from an extensive list of applications and interfaces on infected systems. Exfiltration continues to be handled via Telegram, with specific Telegram BOT IDs and Tokens identified as tied to these more recent campaigns.

The new variant of PXA Stealer will enumerate Chromium/Gecko browsers, decrypt any saved passwords, cookies, stored personally identifiable information (PII), autofill data, and any authentication tokens. The infostealer will also attempt to inject a DLL into running instances of browsers such as Chrome, targeting Chrome's App-Bound Encryption Key to defeat the internal encryption schemes within Chrome. The DLL injected during the July campaign targets MSEdge, Chrome, Whale, and CocCoc browsers.

```
if ( v12 != 10 || (unsigned int)sub_180030760(v42, "msedge.exe", 10LL) )
{
  v49 = lpMultiByteStr;
  if ( v13 > 0xF )
    v49 = (LPSTR *)v25;
  if ( v12 != 10 || (unsigned int)sub_180030760(v49, "chrome.exe", 10LL) )
  {
    v50 = lpMultiByteStr;
    if ( v13 > 0xF )
      v50 = (LPSTR *)v25;
    if ( v12 == 11 && !(unsigned int)sub_180030760(v50, "browser.exe", 11LL) )
    {
      v51 = "coccoc";
LABEL_116:
      v53 = 6LL;
      goto LABEL_117;
    }
    v52 = lpMultiByteStr;
    if ( v13 > 0xF )
      v52 = (LPSTR *)v25;
    if ( v12 == 9 && !(unsigned int)sub_180030760(v52, "whale.exe", 9LL) )
    {
      v53 = 5LL;
      v51 = "naver";
```

Figure 5 - Browsers targeted by the injected DLL from the July campaign

The infostealer also grabs files from dozens of desktop cryptocurrency wallets, VPN clients, Cloud-CLI utilities, connected fileshares, as well as applications such as Discord, and much more.

The collected data is packaged into ZIP archives then exfiltrated to a specific Telegram bot via Cloudflare Worker relays. There are also conditions where the malware will reach out to external sources for additional Python payloads, such as 0x0[.]st, a Pastebin-like temporary file hosting resource. Other analyzed PXA Stealer payloads support stealing data from the following browsers:

360Browser	AVG	Chrome
360 Extreme Browser	Brave	Chromium
Aloha	Brave Nightly	CocCoc
Amigo	CCleaner	CryptoTab
Arc	Cent	Dragon
Avast	Chedot	Edge
Epic	Opera	Speed360
Ghost	Opera Crypto	SRWare
Iridium	Opera GX	Thorium
Liebao	QQBrowser	UR Browser
Liebao AI	Sidekick	Vivaldi
Maxthon	Slimjet	Wavebox

Naver	Sogou	Yandex

The malware targets the following list of cryptocurrency wallet related browser extensions:

Ambire	ExodusWeb3	SafePal Wallet
Aptos Wallet	Frame	Station Wallet
Argent X	Keystone Wallet	Sui Wallet
Atomic Wallet	Leather Bitcoin Wallet	Talisman Wallet
Backpack Wallet	Ledger Live	Tonkeeper Wallet
Bitapp	Leo Wallet	TON Wallet
Bitget Wallet	Magic Eden Wallet	Uniswap Extension
Bitski Wallet	MathWallet	Wallet Guard
Cosmostation Wallet	MyTonWallet	Zeal
Crocobit	OpenMask Wallet	Zeeve Wallet
Crypto.com	Portal DEX Wallet	Zerion
Edge Wallet	Pulse Wallet Chromium	
Equal	Quai Wallet	

User databases and configuration files for the following applications are targeted, many of which house sensitive data or cryptocurrency assets:

Armory	bytecoin	Electron Cash
Atomic	Chia Wallet	Electrum
Azure	Coinomi	ElectrumLTC
Binance	Daedalus Mainnet	Ethereum
Bitcoin Core	DashCorewallets	Exodus
Blockstream Green	Dogecoin	FileZilla
Guarda Desktop	Litecoinwallets	ProtonVPN
Jaxx Desktop	Monero	Raven Core
KeePass	MultiDoge	Telegram
Komodo Wallet	MyMonero	Wasabi Wallet
Ledger Live	OpenVPN	Zcash

The infostealer is also capable of targeting website-specific data. The malware includes the following list of sites, for which the stealer will attempt to discover and collect credentials, cookies and session tokens. The targeted sites are primarily financial, such as FinTech services or cryptocurrency exchanges:

ads.google.com	coinomi.co.nl	korbit.co.kr

adsmanager.facebook.com	coinone.co.kr	kraken.com
binance.com	coinplug.ng	kucoin.com
bingx.com	crypto.com	lbank.com
bitfinex.com	electrum.org	mexc.com
bitget.com	exodus.com	nami.exchange
bitgo.com	gate.com	okx.com
bitmart.com	gemini.com	paypal.com
bitunix.com	gopax.co.kr	probit.com
business.facebook.com	htx.com	upbit.com
bybit.com	huobi.com	whitebit.com
coinbase.com	hyperliquid.xyz	xt.com

The specific Telegram Bot Token, and associated Chat ID, identified in the samples from July are:

Telegram Bot Token: `7414494371:AAHsrQDkPrEVyz9z0RoiRS5fJKI-ihKJpzQ`

Telegram Chat ID: `-1002698513801`

Data is exfiltrated to Telegram via connection via Cloudflare workers. The specific Cloudflare DNS address is:

```
Lp2tpju9yrz2fklj.lone-none-1807.workers[.]dev
```

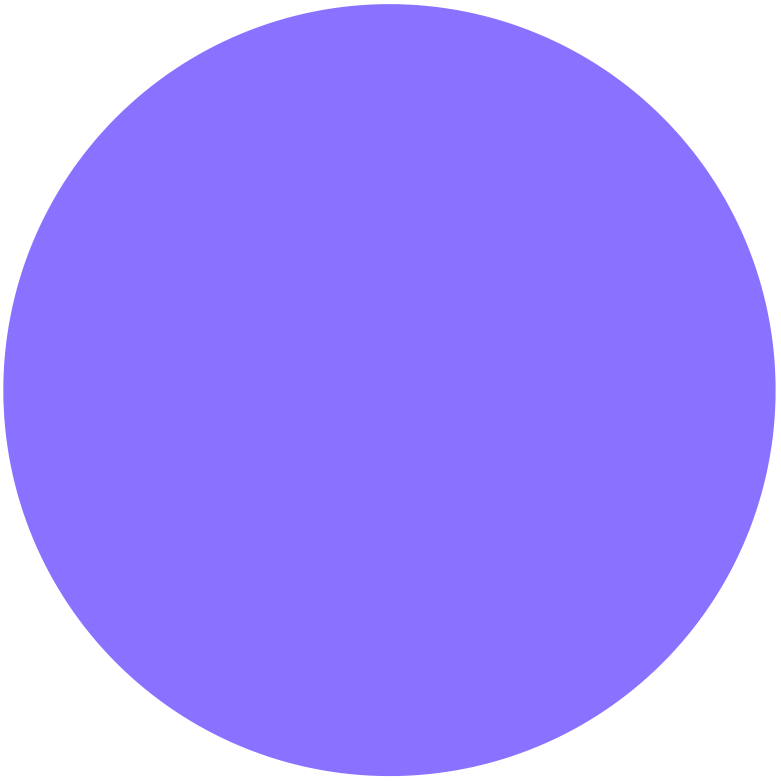
We reported this abuse of Cloudflare Workers to Cloudflare, and we thank their team for taking immediate action to disrupt this malicious infrastructure.

Each of the final PXA Stealer payloads corresponds to a Telegram Bot Token and ChatID combination. Each variant we analyzed is associated with the same Telegram Bot Token (`7414494371:AAHsrQDkPrEVyz9z0RoiRS5fJKI-ihKJpzQ`) although the ChatIDs vary. Additionally, there can be multiple ChatIDs, which correspond to a Telegram channel, tied to each payload. Each bot is tied to as many as 3 Telegram channels. One channel, typically denoted with the `New Logs` string, receives exfiltrated data contained in zip archives uploaded from victims' machines, along with log/ledger style data for each victims' exfiltrated data set. Specific entries also indicate the victim's geographic location, IP address and other contextual data.

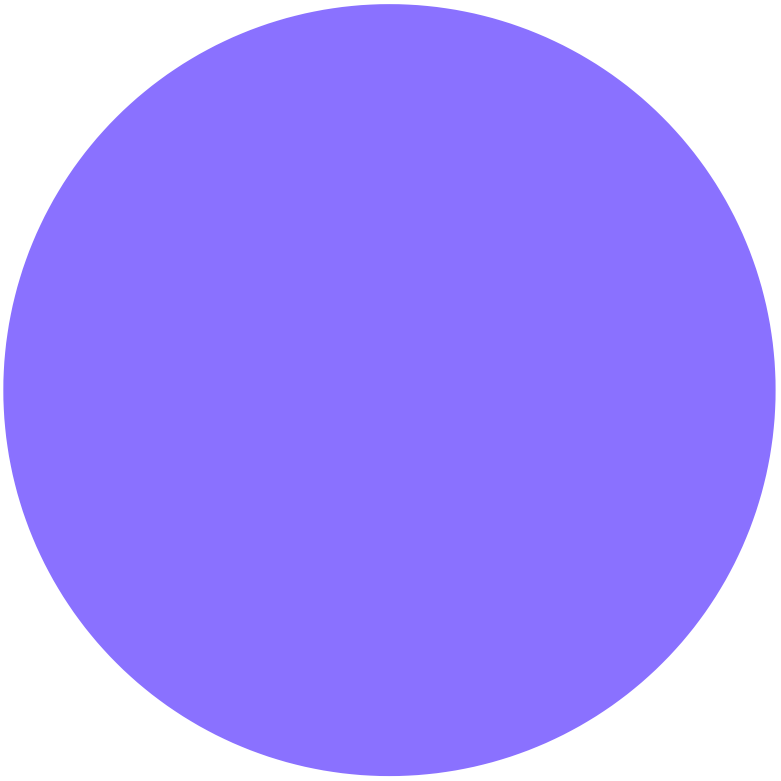
PXA Stealer log entries show counts for the types of data within:

```
CK:2868|PW:482|AF:606|CC:0|FB:1|Sites:4|Wallets:0|Apps:1
```

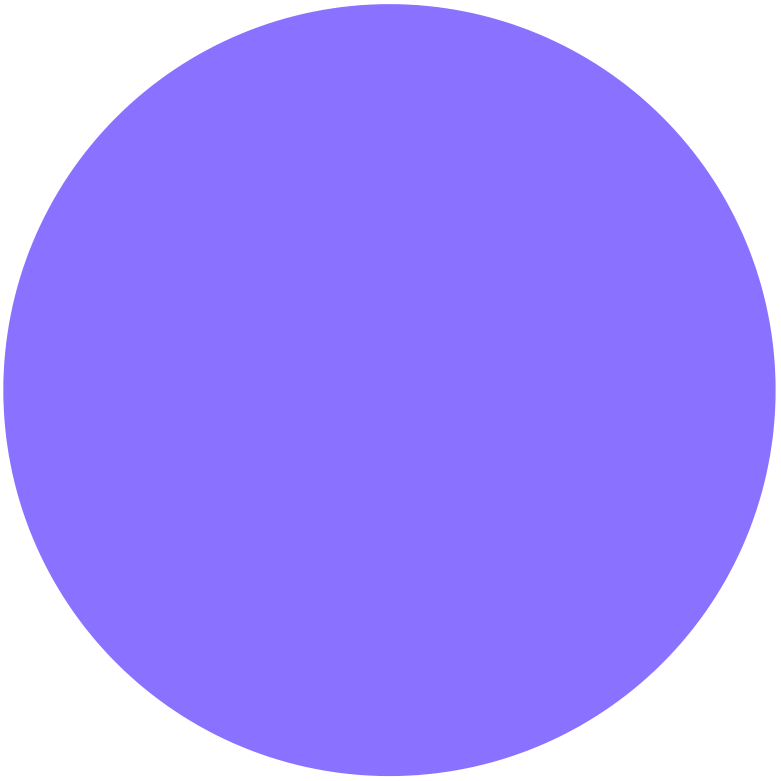
The stealer data types include:



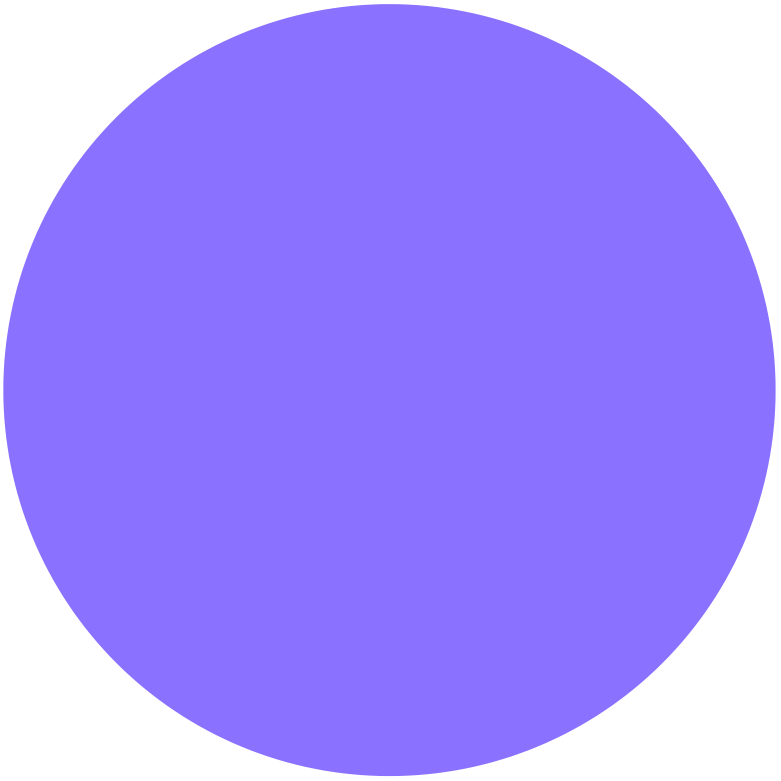
- CK=Cookies



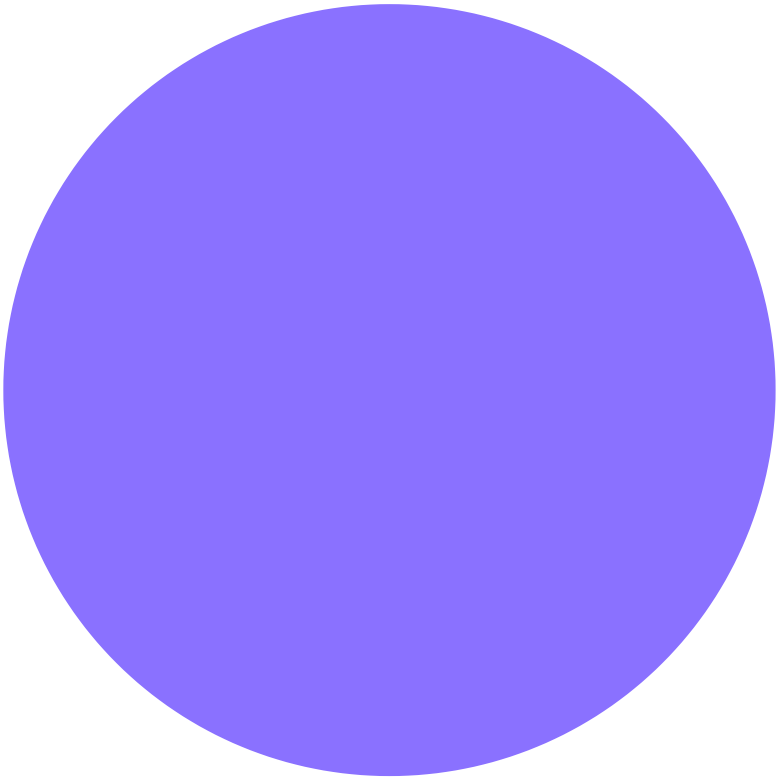
- PW = Passwords



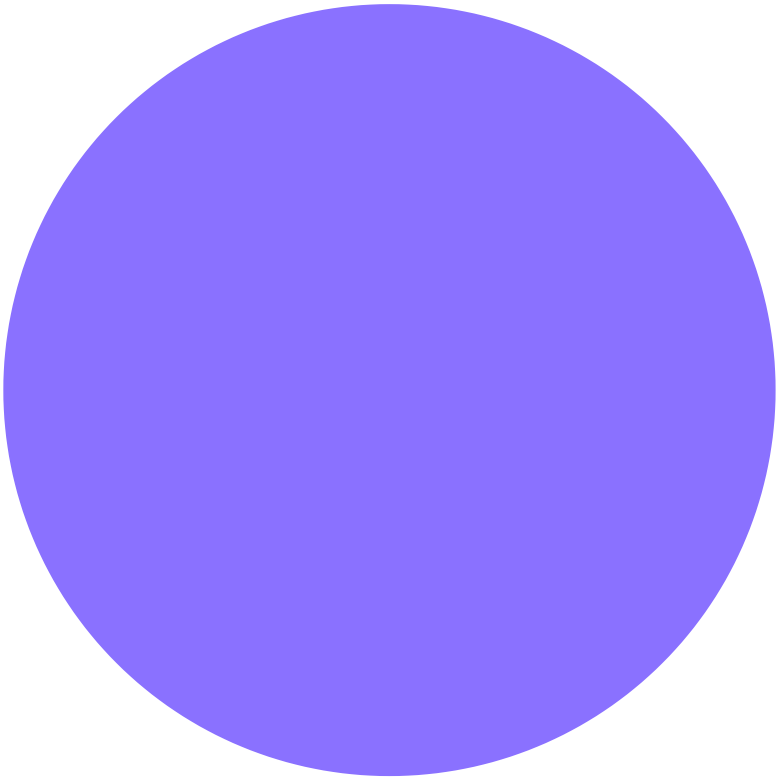
- AF = AutoFill data



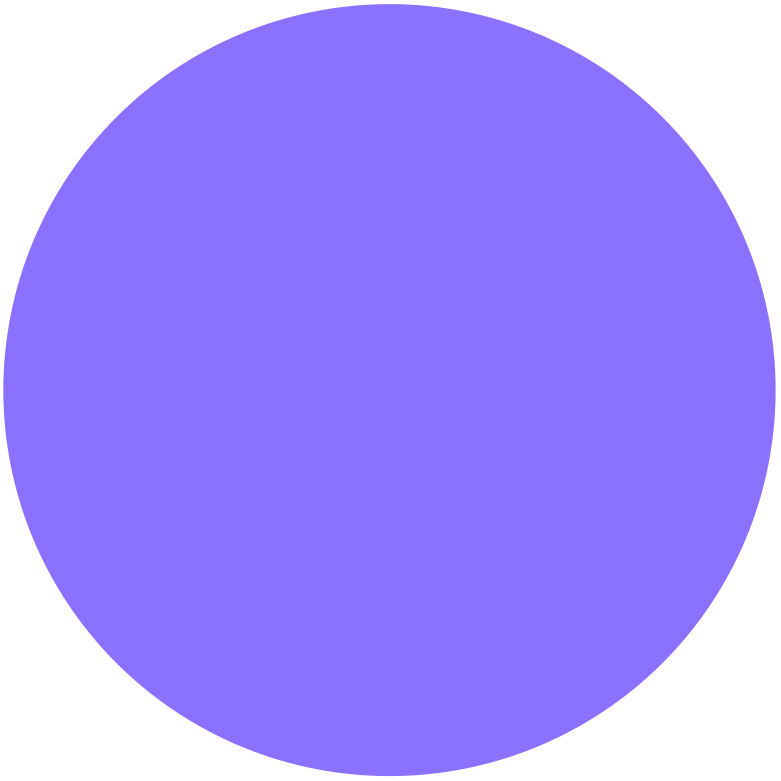
- CC=Credit Card data



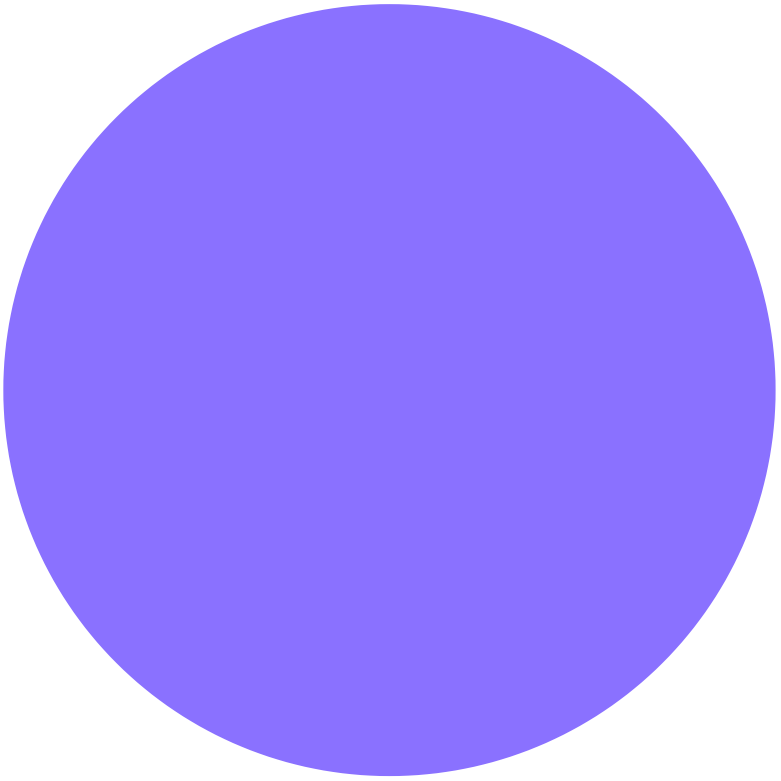
- FB= Facebook Cookies



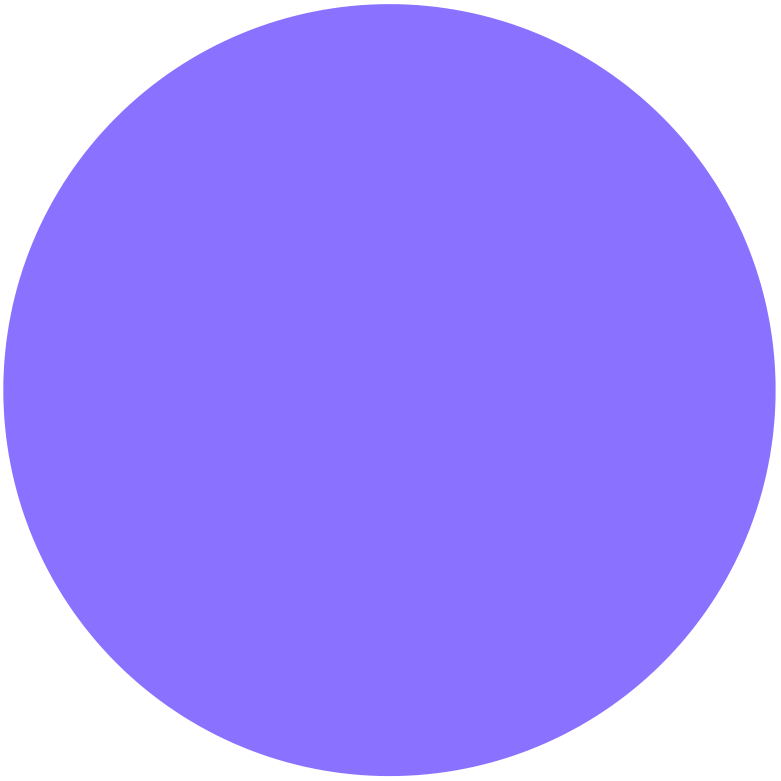
- TK= Authentication Tokens



- Sites = Domains / Site specific data



- Wallets = Crypto Wallet data



- Apps = Application specific data (ex: private messenger chat history and keys)

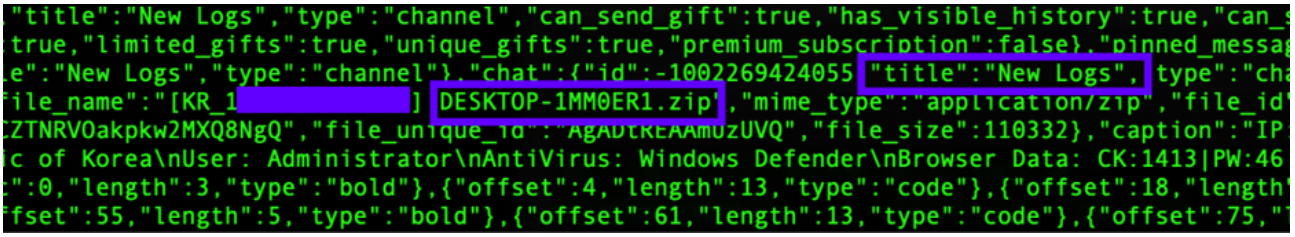
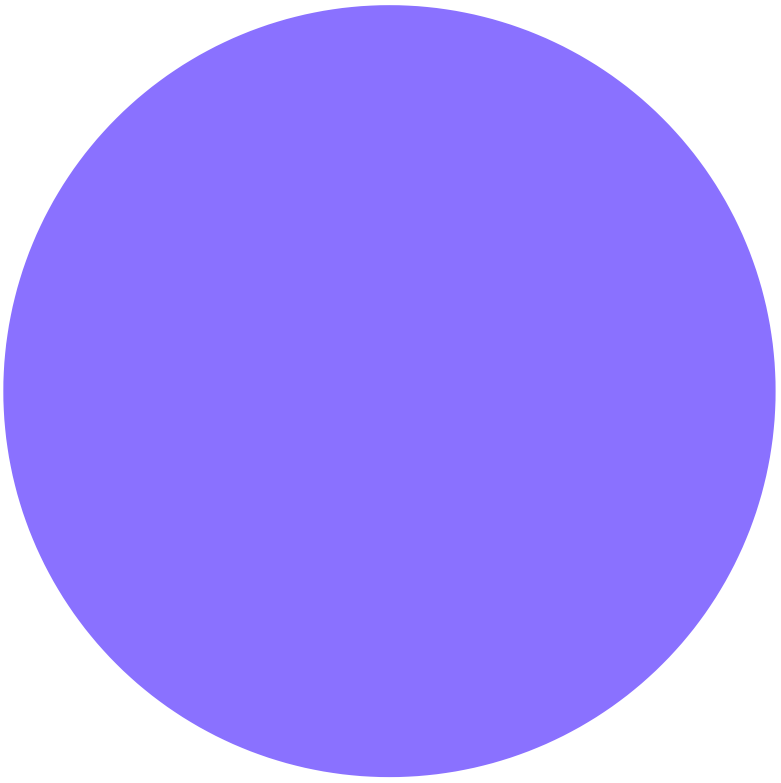


Figure 6 - Exfiltrated Victim Data from MRB_NEW_VER_BOT via PXA Stealer

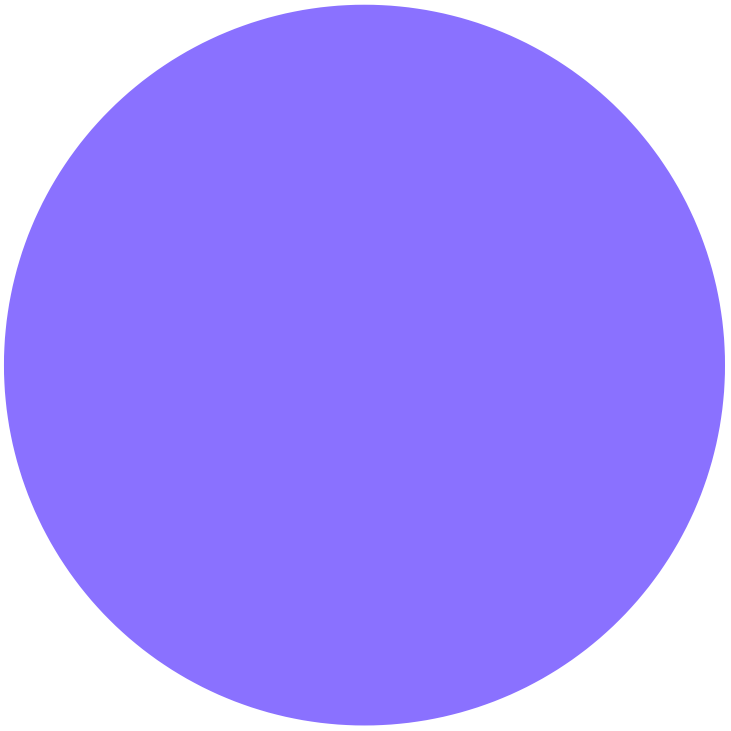
Each bot will also have an associated ‘Reset’ and ‘Notifications’ channel as well. The ‘Notification’ channels appear to allow operators to automate their communications process when new victim logs are uploaded or otherwise obtained. The ‘Reset’ channels appear to be used in similar manner to the ‘New Logs’ channels, storing newly exfiltrated victim data.

While all analyzed variants share the same Bot Token ID, we have observed multiple ChatIDs across the New Log/Reset/Notification combinations across this stealer's ecosystem. The observed Bots-to-ID sets include:

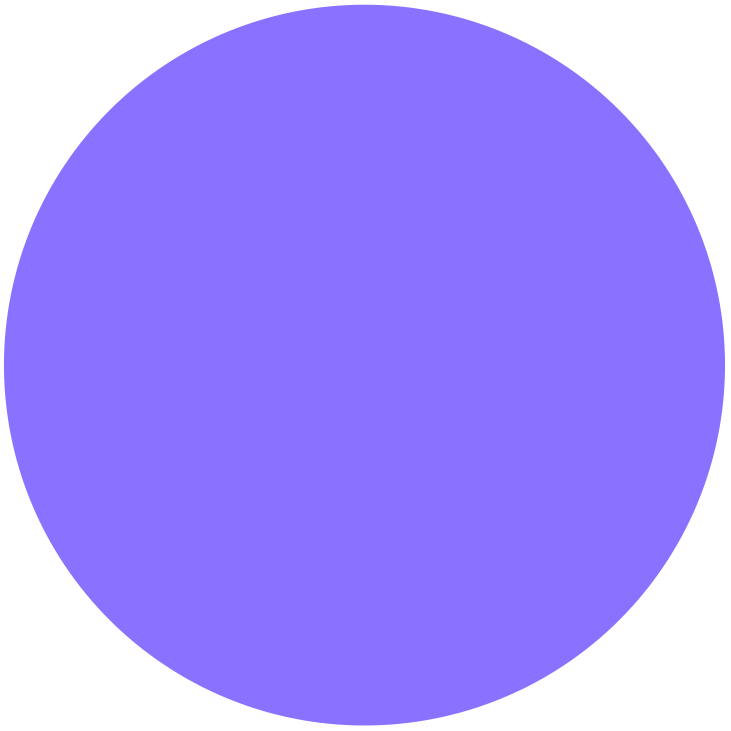
Telegram BotID `7414494371:AAHsrQDkPrEVyz9z0RoiRS5fJKI-ihKJpzQ`



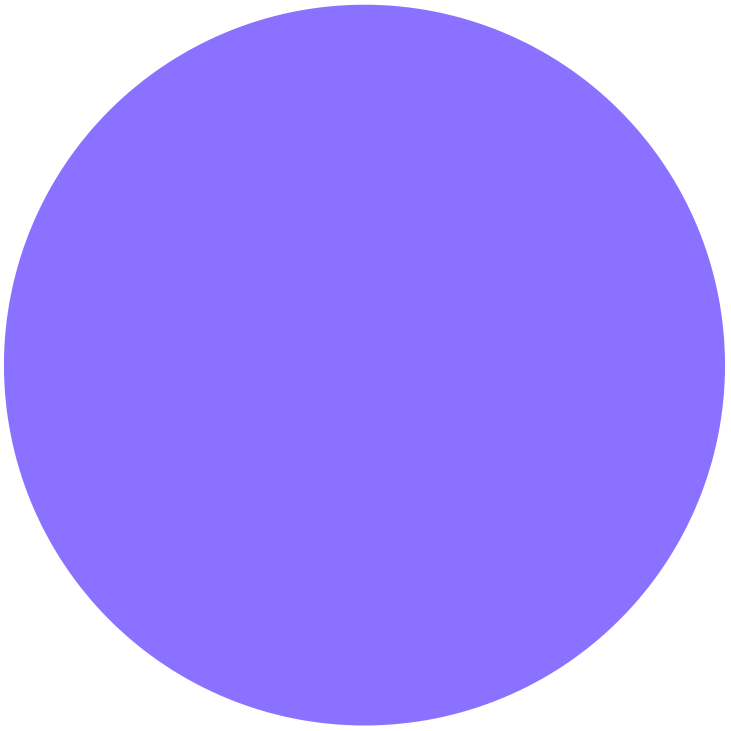
- James_New_Ver_bot (yd2sV / James)



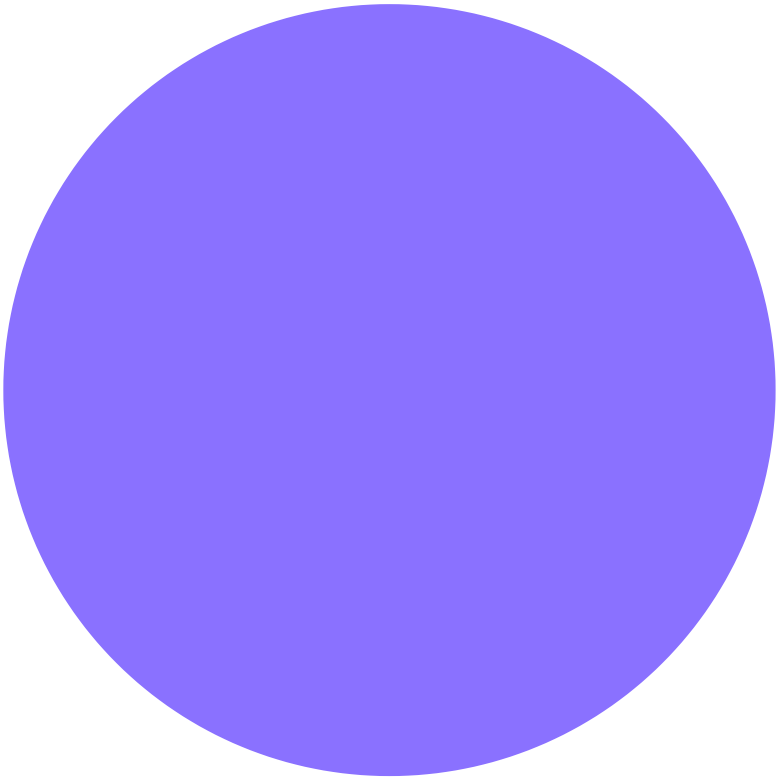
- James - New Logs



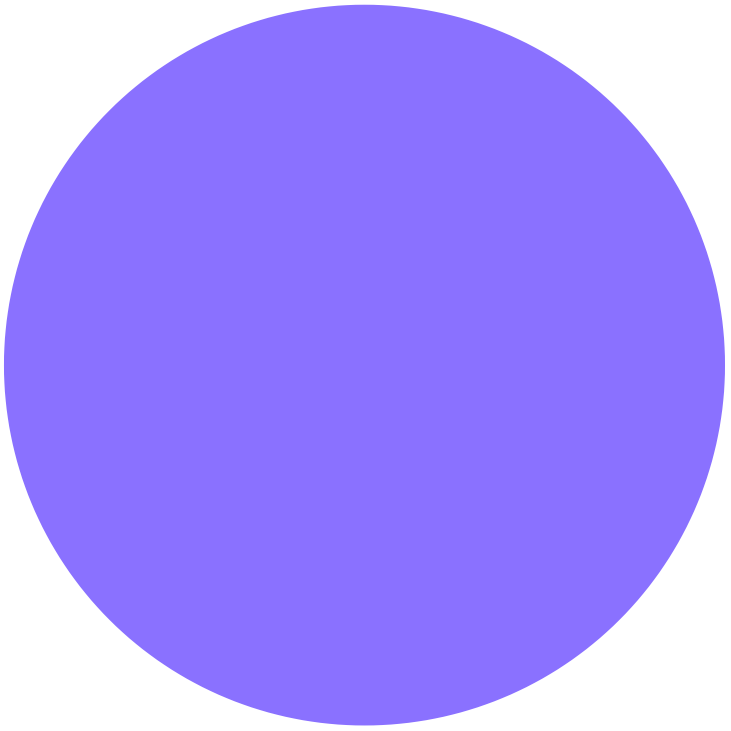
- James - New Logs Notification



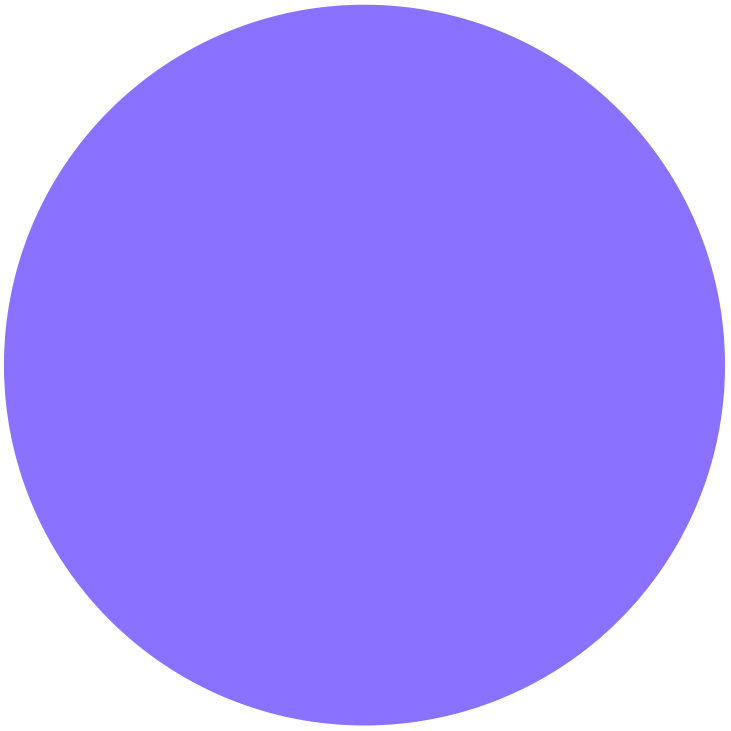
- James - Reset Logs



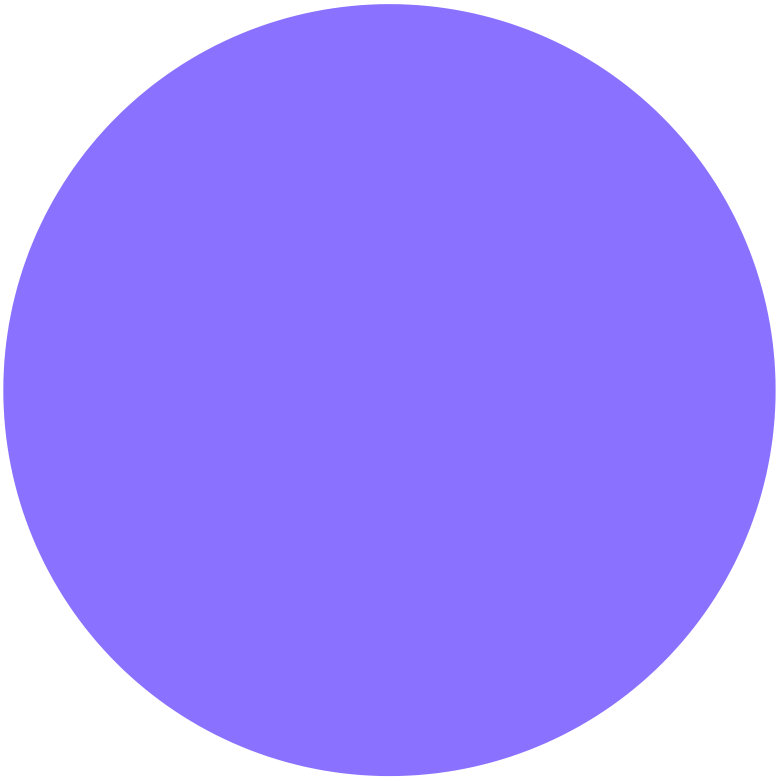
- DA_NEW_VER_BOT (qDTxA / DUC ANH)



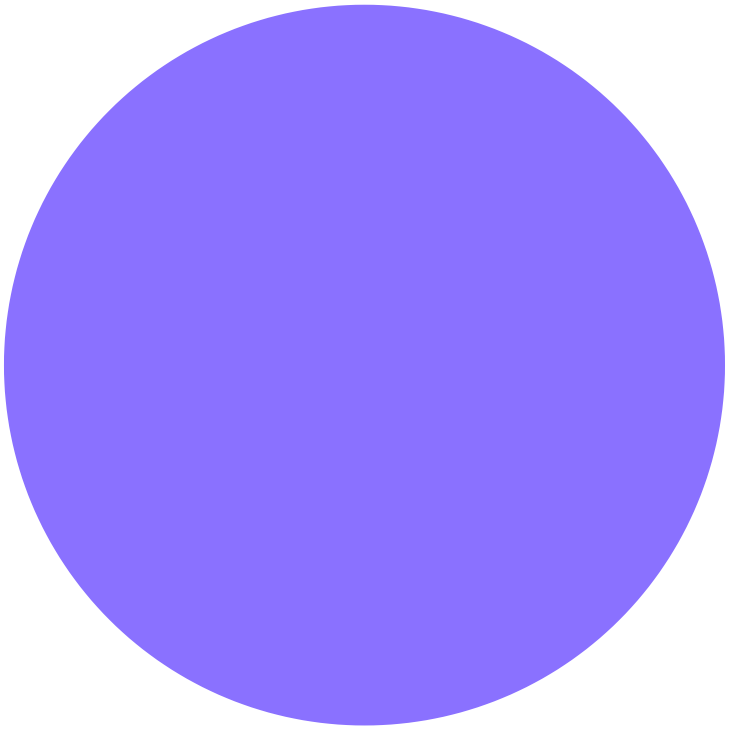
- New Logs - \u0110\u1ee9c Anh



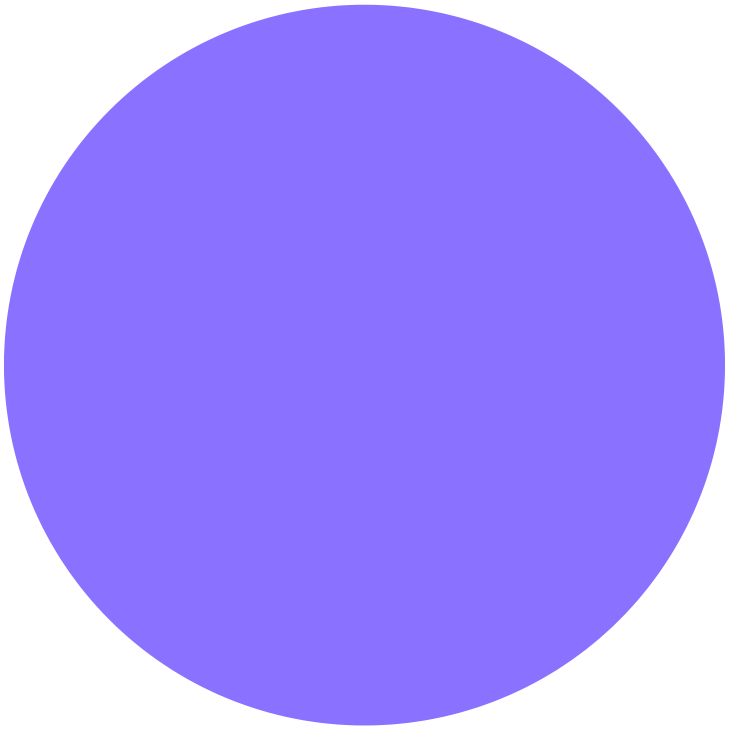
- Reset Logs - \u0110\u1ee9c Anh



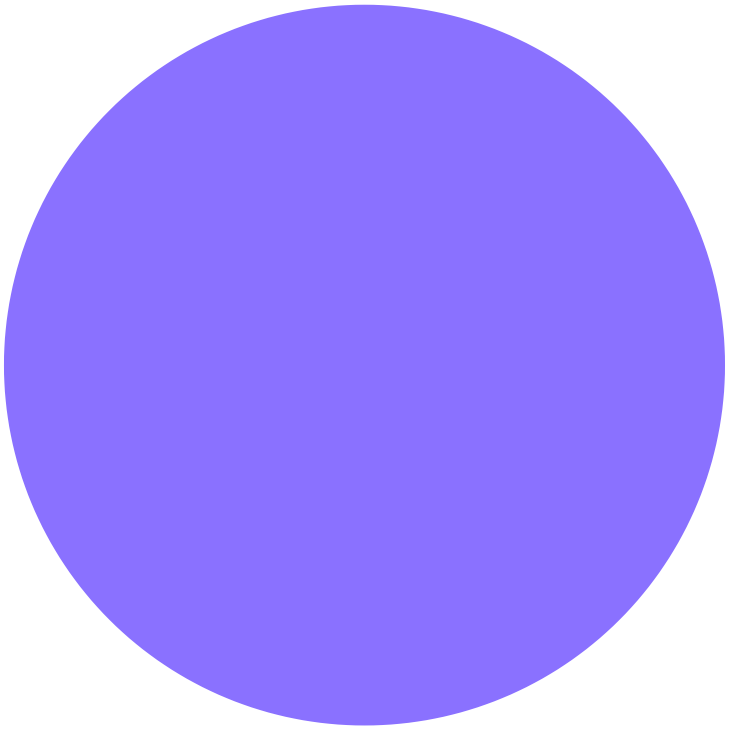
- MRB_NEW_VER_BOT (Plk1y / MRB_NEW)



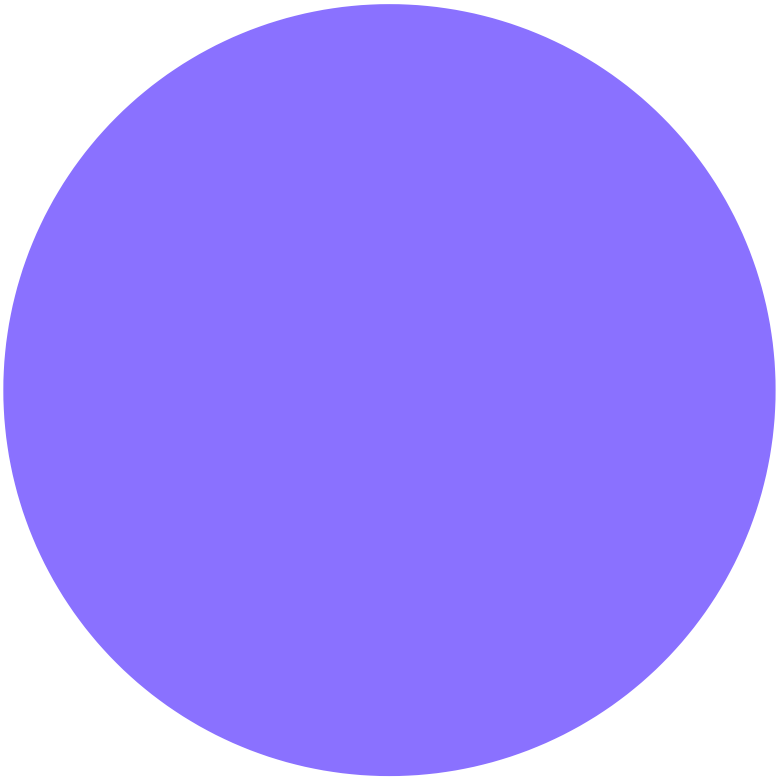
- New Logs



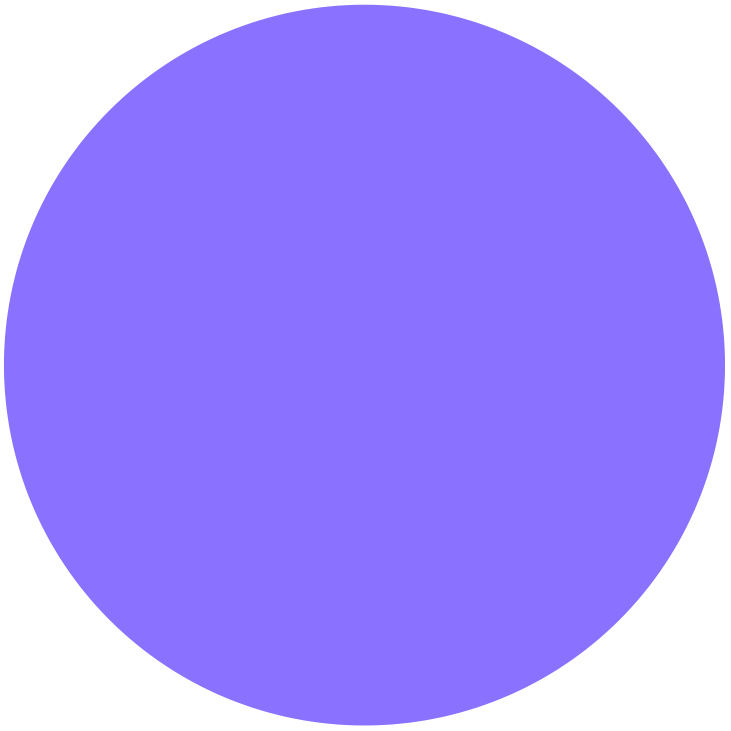
- Reset Logs



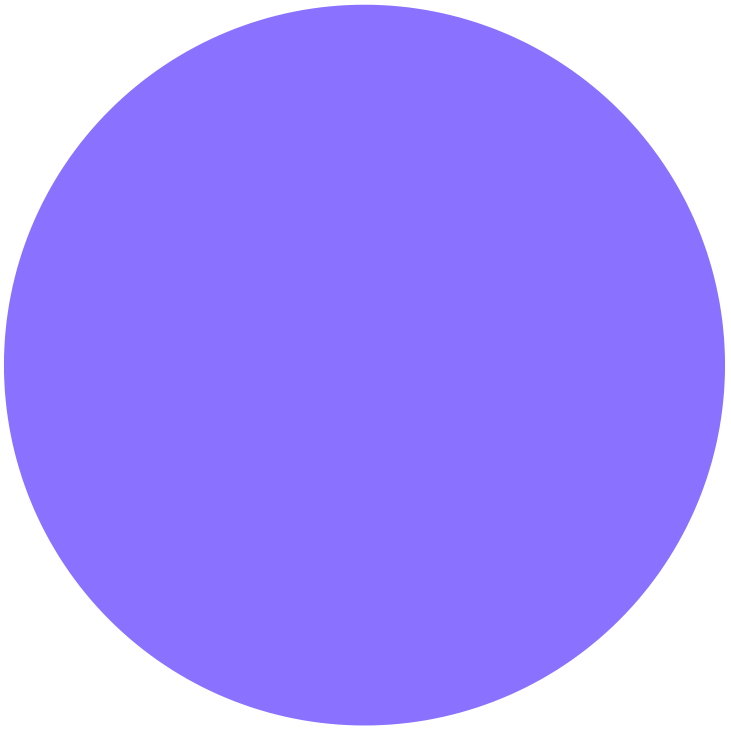
- Notify



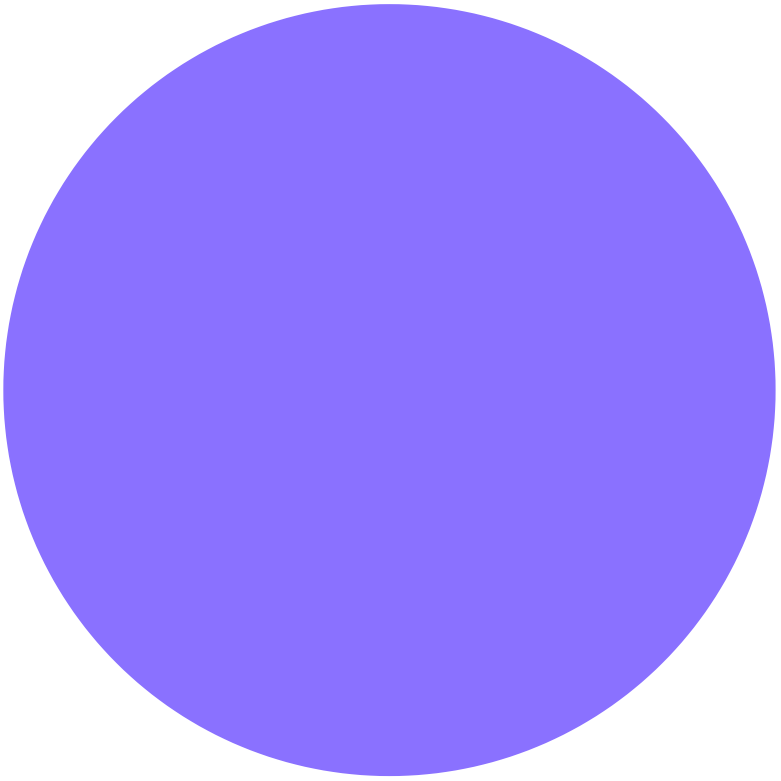
- JND_NEW_VER_BOT (5DJ0P / JND)



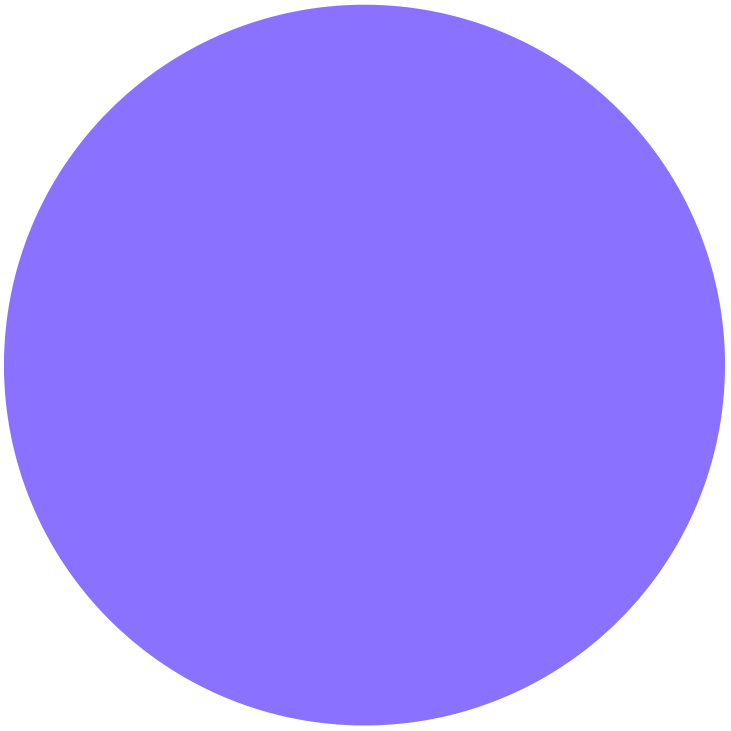
- JND - New Logs



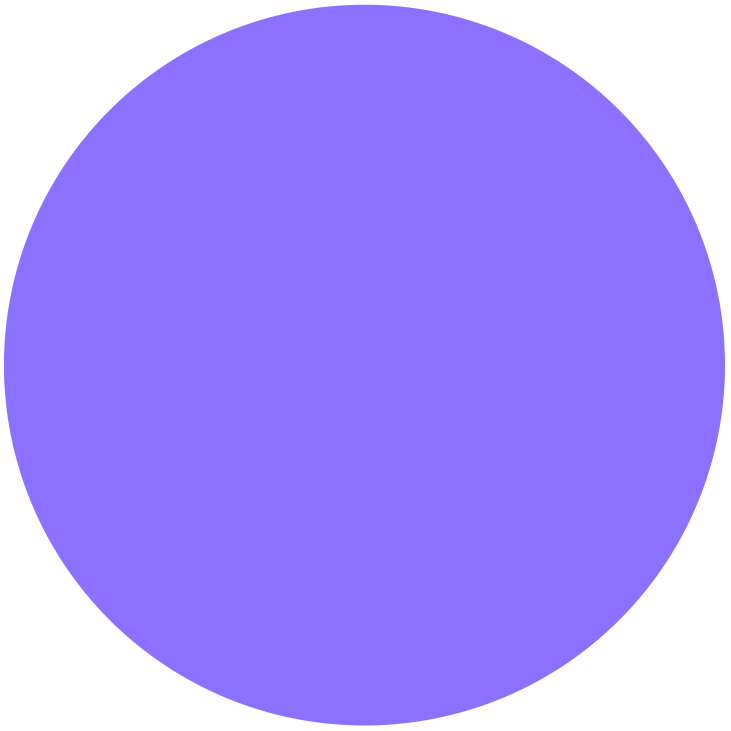
- JND - Reset Logs



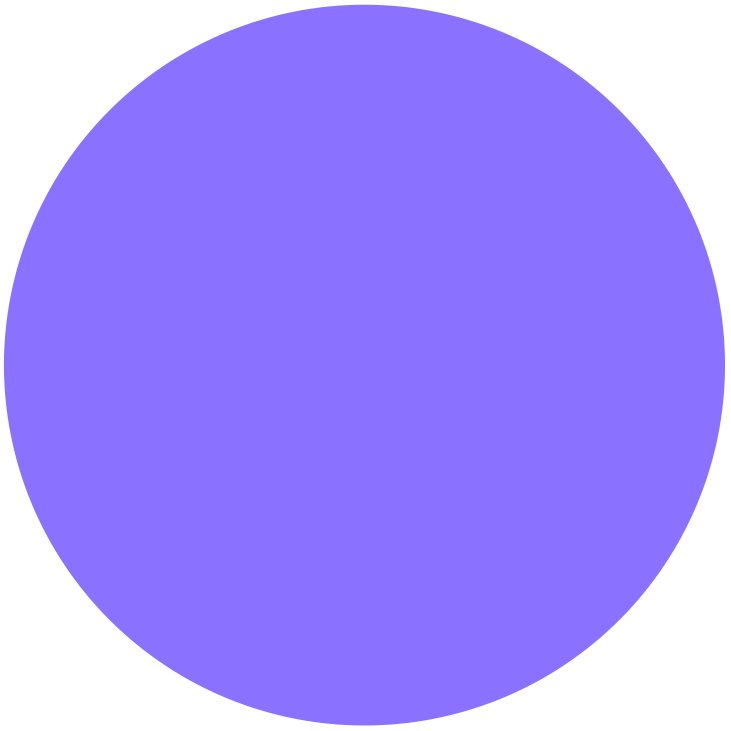
- AND_2_NEW_VER_BOT (oaCzj / ADN 2 / Adonis)



- Adonis - New Logs



- Adonis - Reset Logs



- New Log Notification

The encompassing Telegram ID is connected to a Bot that has the following properties:

Username: `“Logs_Data_bot”`

Firstname: `\u0412\u0418\u0414\u0415\u041e \u0421 \u041b\u0410\u0419\u041a\u0410`

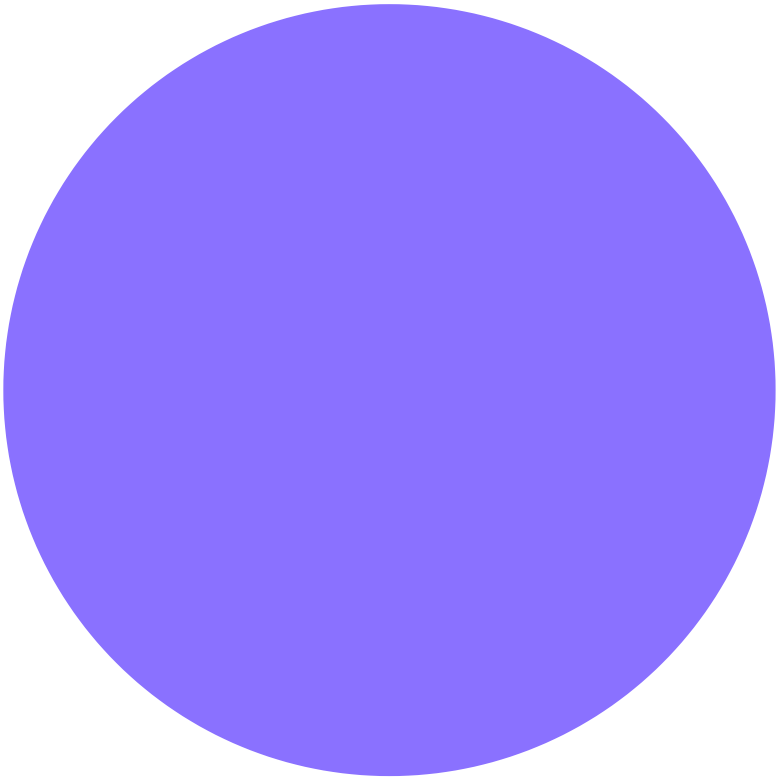
Lastname: `(nul)`

The firstname field on this bot decodes to a string of Cyrillic text “ВИДЕО С ЛАЙКА”. This roughly translates to ‘Video for/with/of Laika,” though the significance of this string is unclear.

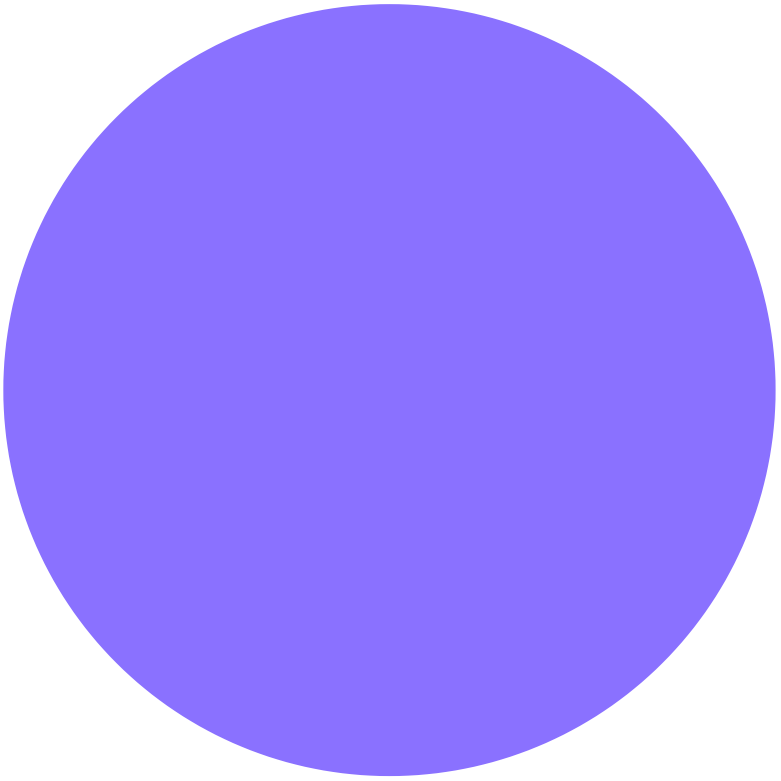
Telegram Abuse and Attribution

The later-stage dropper component is responsible for parsing target Telegram URLs based on a string gathered from a prescribed Telegram ChatID. This string is then combined with the base URL for either `paste[.]rs` or `0x0[.]st` to retrieve the next batch of obfuscated Python code.

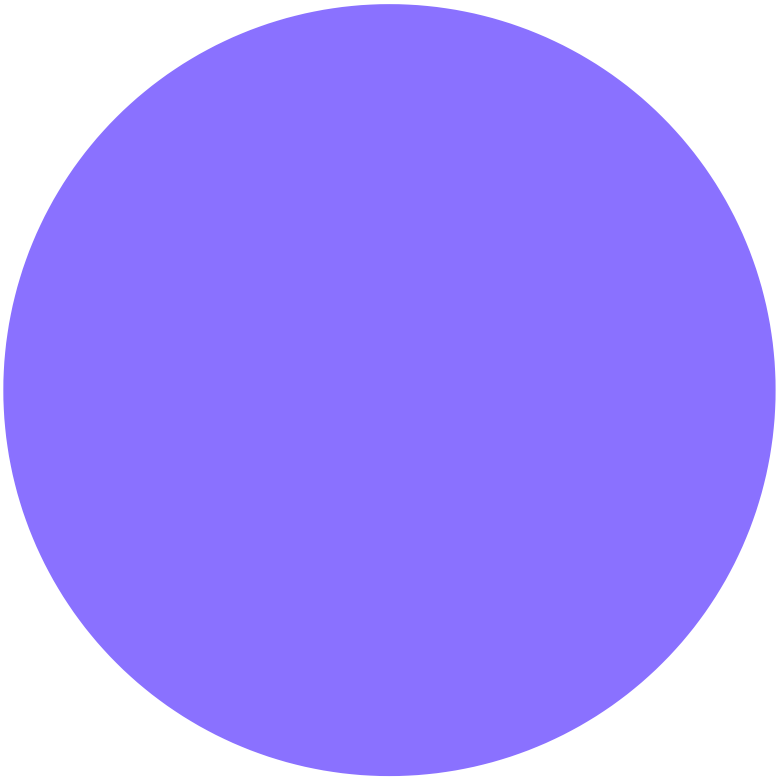
Multiple identifiers were observed across the multitude of analyzed samples. The most prominent we observed are:



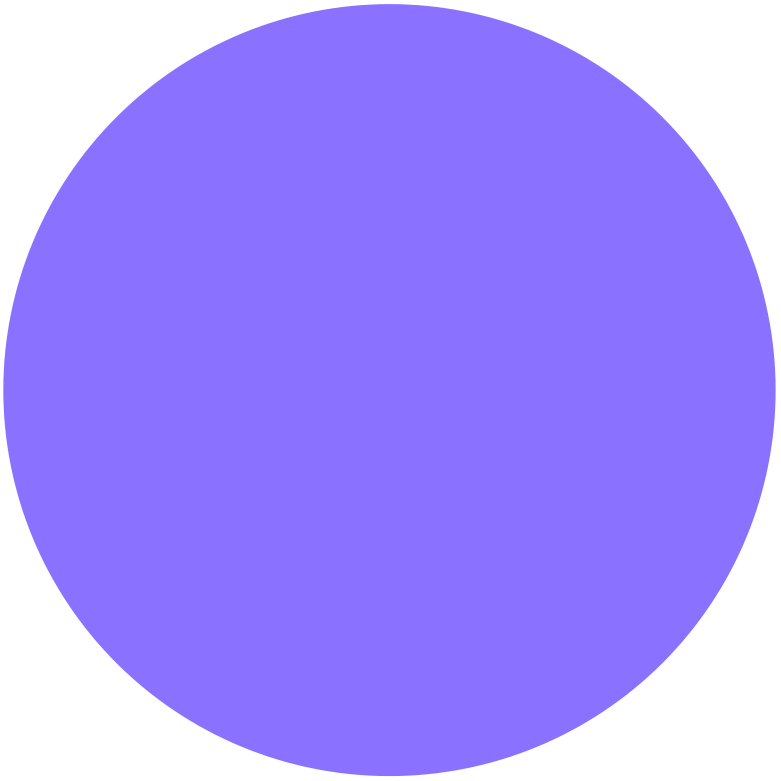
- ADN_2_NEW_VER_BOT



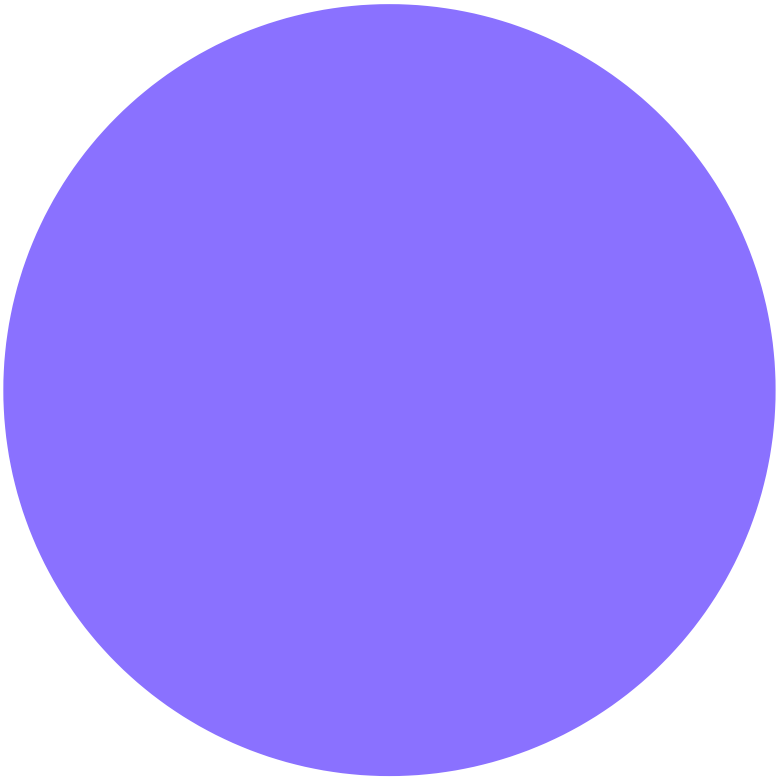
- DA_NEW_VER_BOT



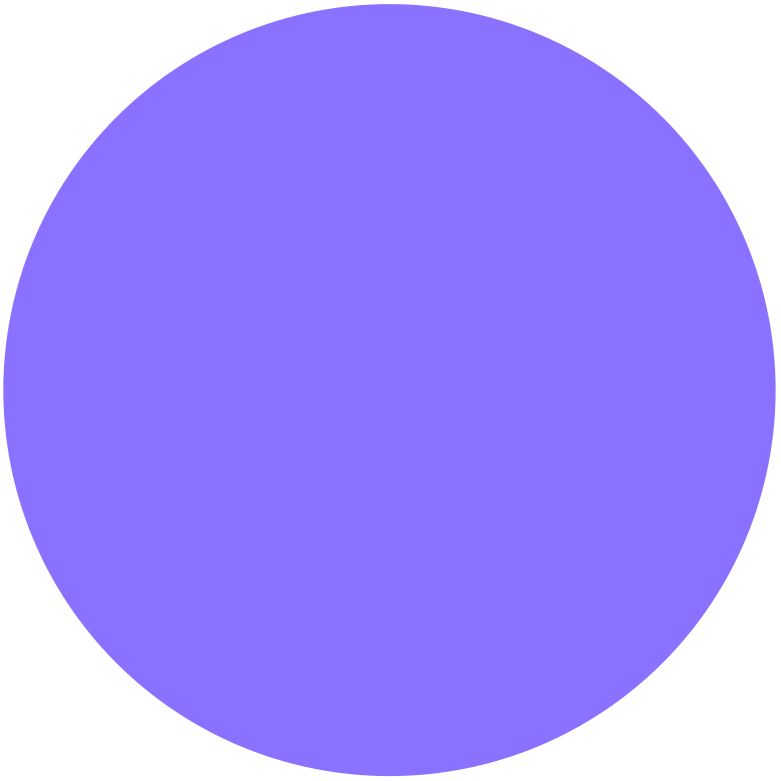
- JAMES_NEW_VER_BOT



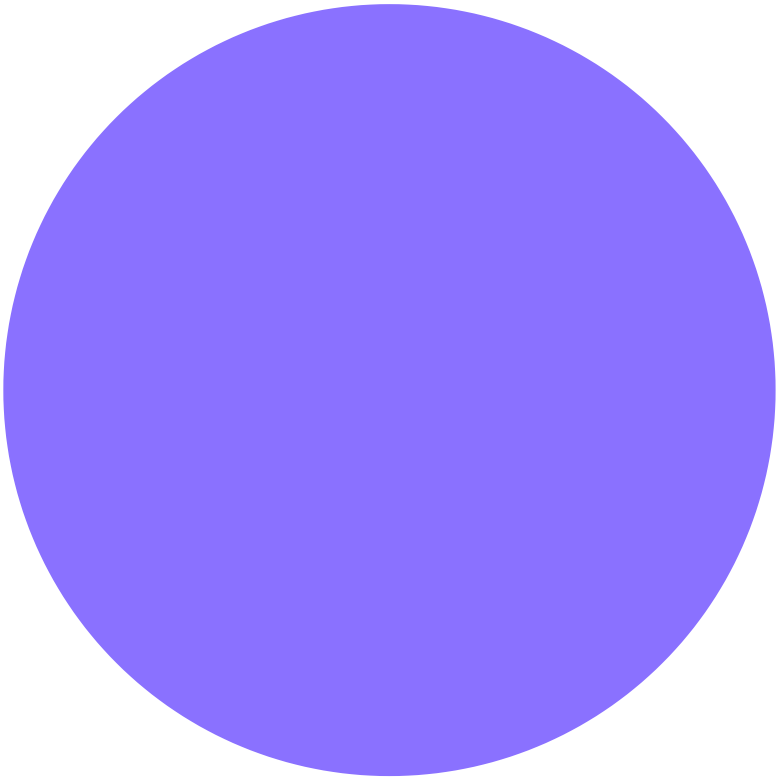
- JND_NEW_VER_BOT



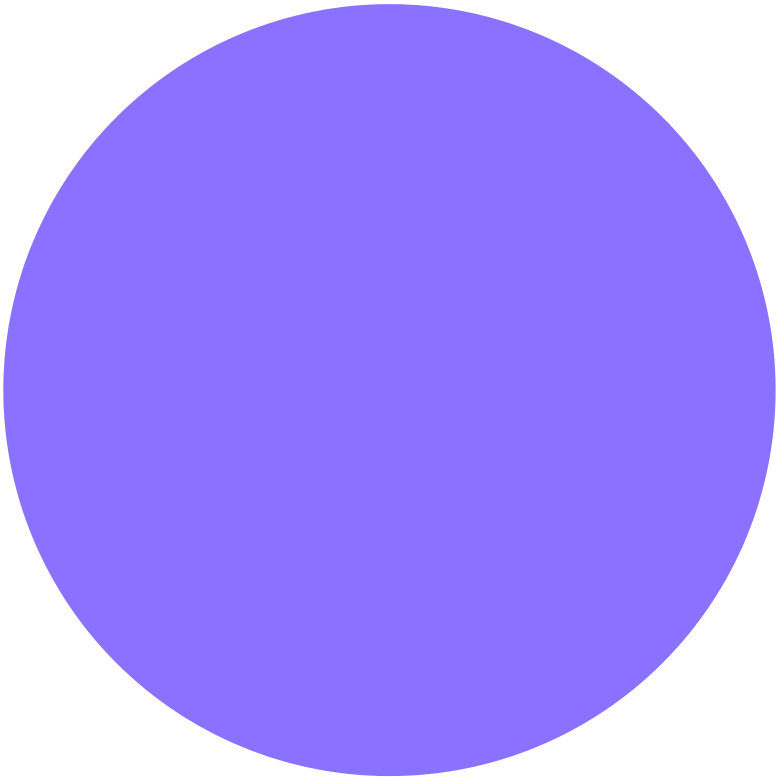
- MR_P_NEW_VER_BOT



- MR_Q_NEW_VER_BOT



- KBL_NEW_VER_BOT



- MRB_NEW_VER_BOT

These identifiers are visible within the commands launched by the side-loaded DLL described above.

```
cmdline: cmd /c cd _ && start Tax-Invoice-EV.docx && certutil -
decode Document.pdf Invoice.pdf && images.png x -ibck -y -
poX3ff7b6Bfi76keXy3xmSWnX0uqsFYur Invoice.pdf C:\\Users\\Public
&& start C:\\Users\\Public\\Windows\\svchost.exe
C:\\Users\\Public\\Windows\\Lib\\images.png MR_Q_NEW_VER_BOT &&
del /s /q Document.pdf && del /s /q Invoice.pdf && exit && exit
```

Each of these `_NEW_VER_BOT` identifiers corresponds to a Telegram User ID. The profile names resemble a bot, but are actually user accounts:

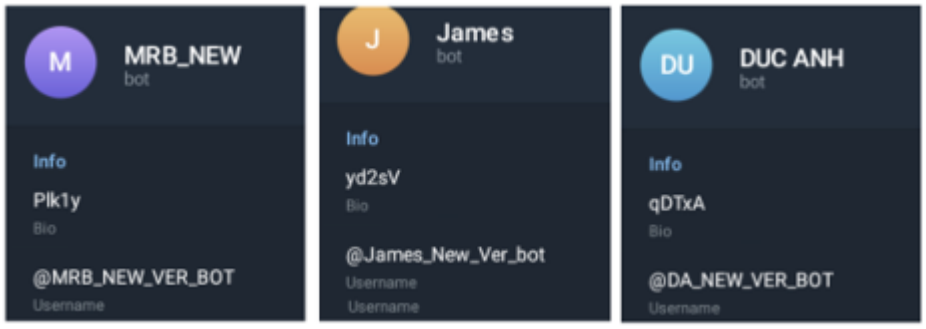


Figure 7 - Bio and Info fields from Telegram profiles masquerading as bots

When retrieving files from `paste[.]rs`, the corresponding strings are concatenated with the `hxxps://paste[.]rs` or `hxxps://0x0[.]st` prefix, which constructs the full download URL hosting another payload.

paste.rs/yd2sV

```
def _JUYNSA54ELD79SVTNAY5UZWCC():
        _V6120EU = "TMMNR2KVQWAWO8"
        _J6S1P0Q11N6 = 2398947243
        _YK5YDUK = "8BQ7ES2DOKD6"
        _YI373RE9LHSIF5I = "BCY9OQMR"
        _V8Y2LTLLQ2W = "68D5HY0KPEEYA"
        _RWJ743QF0I56H = 4540913849
        _91S5LA9UB = "94H2P2L7Q"
        _1F9L7SD2NSHAZNB = 3021720590
        _NCK05HBM = "FJ3MBQ00"
        _HU8G40UIB = 1161468254
        _I1DJ38 = 3713444404
        _6ZJQ4 = "A2KLNDNCIHF29"
        _NJ4ACQK = "LCX5ZPEAX"
        _9HOB25DJ4LQKYO = "NS354U0W93S5VSO19"
        _MHMCXJICD = "RA5Z4QF8I2Q8"
        _W899ZC = 7378578861
        _XV9TXZ5WM8JR = 8300444590
        _8860GCVRRMGJFXH = 1365423364
        _O28BNNP = "0DHEKR8GP"
        _EOPJ1JYMSL9O93X = "PW6ALGYK"
        _W1TH8C = 4634580416
        _VQXXDCL = 5118212194
        _KC1V7UXQKSK = 8501719841
        _FBZH9 = 4771199819
        _OMKQNWH3 = 2718170441
def _PICKDP88EGPM6Y9ZHSIOPREGB():
        _7D2PLJ7I6A = 9580085529
        _9KI9OW4K4N1S = 4825288407
        _O0J49XDCKF3H = "B12JTZPUHS"
        _CU7LEJ = "RE28HSC6UY6TF"
        _TLTAR6R0SM5D = 5087637022
        _KCS5YZQV = 5693074107
        _FETUGRX67OX0 = 1839519151
        _Z6UGHLROADOBQ65 = "AK3T41C5CK1"
        _SC4328OZRWHJZ = "GSVHA813"
        _4MECFQH8ZHJVJDB = 6708296684
        _AEIBAGJ6XX0UJ40 = 6572356386
        _BZ32WYWGB = 6587651516
        _LVJ60V9VYG9ED9P = "RD7M3WNRJS3WE3"
```

Figure 8 - Obfuscated Python code hosted on Paste[.]rs

Once downloaded, the obfuscated Python code is decoded and executed, delivering the Infostealer component of the attack.

The Telegram ChatID associated with the infostealer component of this attack is "@Lonenone." The "Lonenone" theme is also present in the Cloudflare Worker hostname `lp2tpju9yrz2fklj[.]lone-none-1807[.]workers[.]dev`. The profile display name contains an emoji of the Vietnam flag.

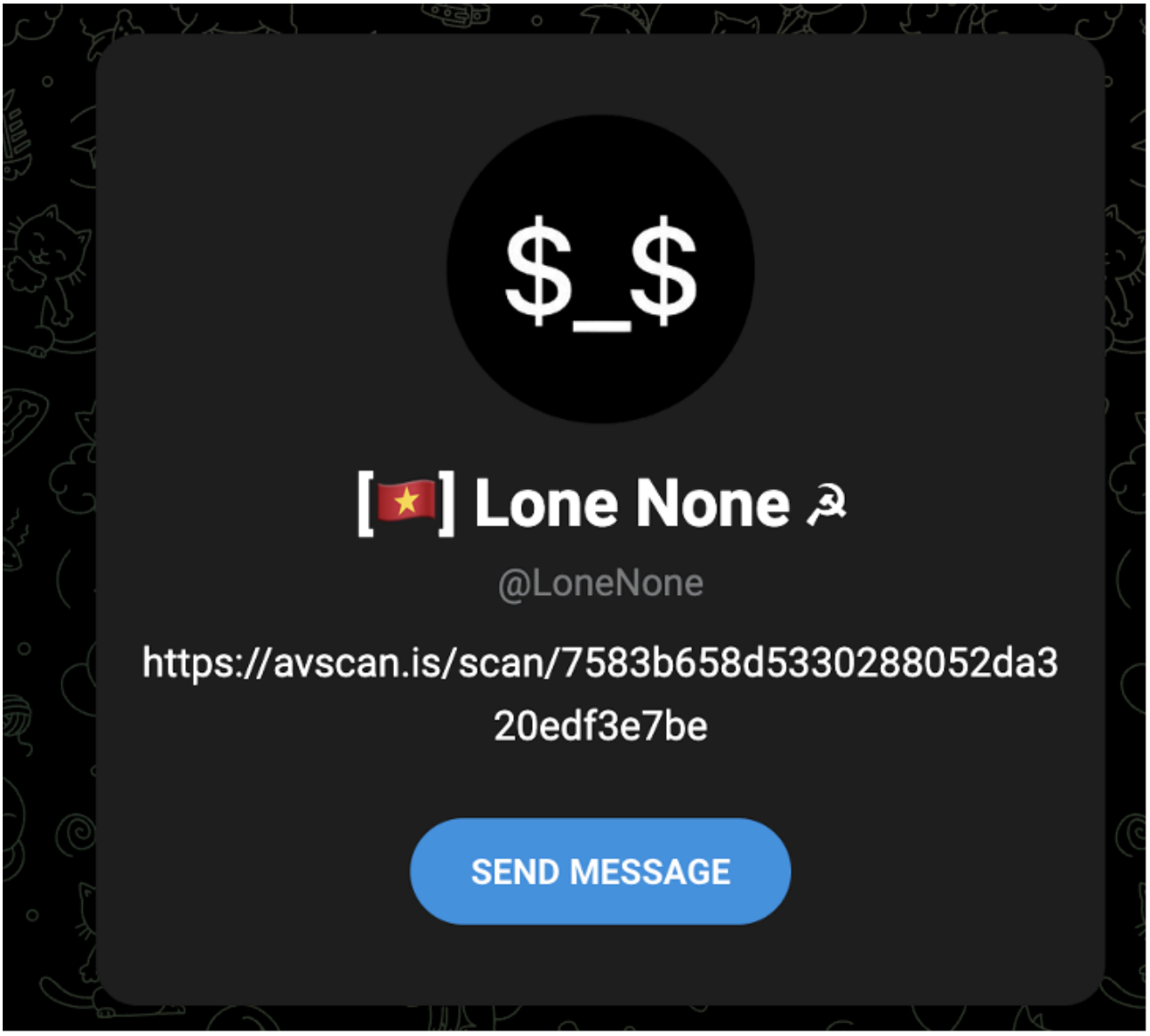


Figure 9 - Lone None Telegram ChatID.

```
    log_error(e)
GetIPD, Country_Code, IPV4 = GetIP()
Count = Counter()
AV_List = get_installed_av()
zip_data = io.BytesIO()
archive_path = os.path.join(TMP, f"[{Country_Code}_{IPV4}] {os.getenv('COMPUTERNAME', 'defaultValue')}.zip")
with zipfile.ZipFile(zip_data, 'w', zipfile.ZIP_DEFLATED, compresslevel=9) as zip_file:
    zip_file.comment = f'Time Created: {creation_datetime}\nContact: https://t.me/LoneNone'.encode()
    for root, _, files in os.walk(Data_Path):
```

Figure 10 - Reference to LoneNone TG channel in decoded (July) infostealer

This Telegram ChatID/Account is associated with the same threat actor using PXA Stealer as previously described by Cisco Talos. It is worth noting that there are a number of other Vietnamese-language artifacts present in these stages of the malware. For example, the aforementioned Telegram BOT IDs show 'Duc Anh' ... aka "đức anh" as display names, which loosely translates to “brother”.

PXA Stealer uses the BotIDs (stored as TOKEN_BOT) to establish the link between the main bot and the various ChatID (stored as CHAT_ID). The ChatIDs are Telegram channels with various properties, but they primarily serve to host exfiltrated data and provide updates and notifications to the operators.

PXA Stealer transmits data via HTTP POST requests to the Telegram API. Everything is handled via HTTPS, thus there is no visible Telegram process or self-contained client producing the traffic. This is one of PXA stealer’s methods of hiding exfiltration traffic from potential analysis or detection.

Prior to transferring the exfiltrated data, the stealer packages stage data into an archive using the following naming convention where CC=Country Code:

```
  [CC_IPADDRESS]_HOSTNAME.zip
(ex: [RU_123.45.67[.]89]DESKTOP-VICTIM.zip)
```

The main BotID (`7414494371:AAHsrQDkPrEVyz9z0RoiRS5fJKI-ihKJpzQ`) includes a reference to `probiv[.]gg` in the Bot metadata:

```
":[{"command":"start","description":"probiv.gg \u0437\u0430\u043f\u043e\u043c\u043d\u0438 \ud83d\udd25"}
```

```
Probiv[.]gg
```

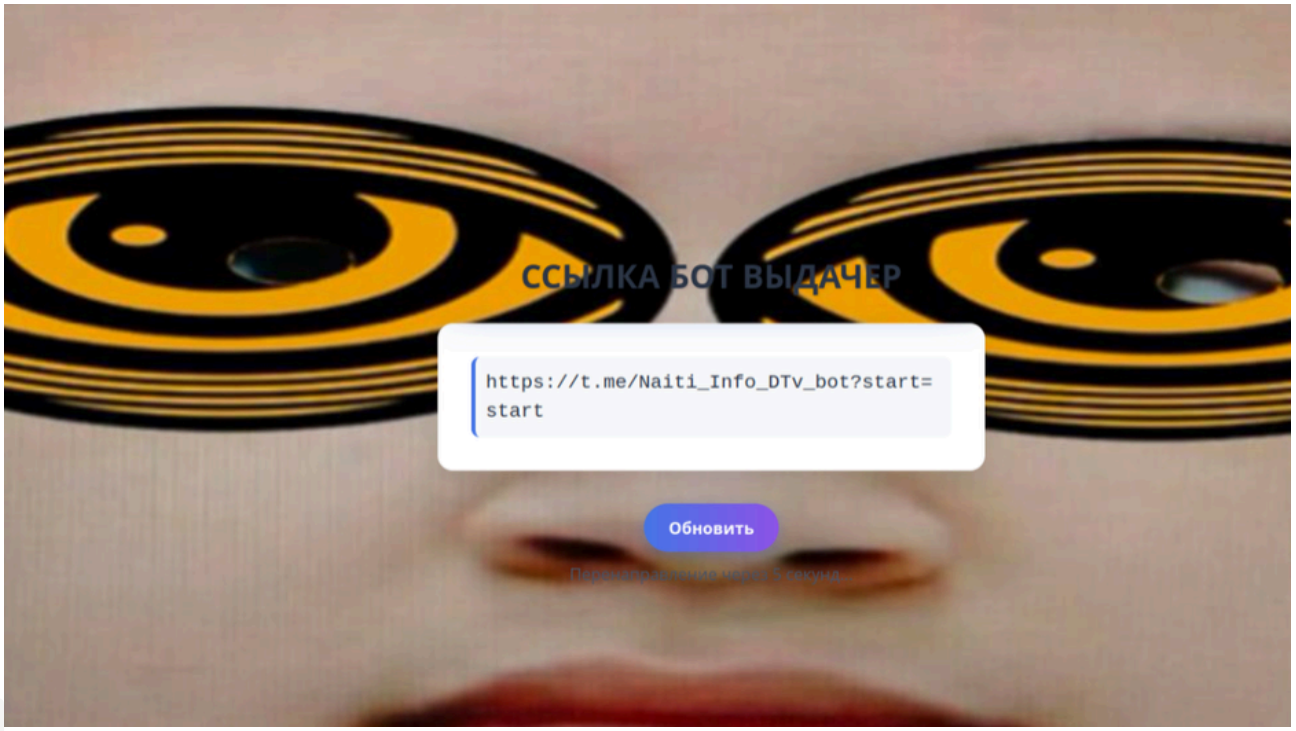


Figure 11 - Telegram redirect on probiv[.]gg

The redirect leads to the Telegram landing page for `SherLock1u_BOT`, a provider of stolen data, and the automated services to search for specific data types or sets.

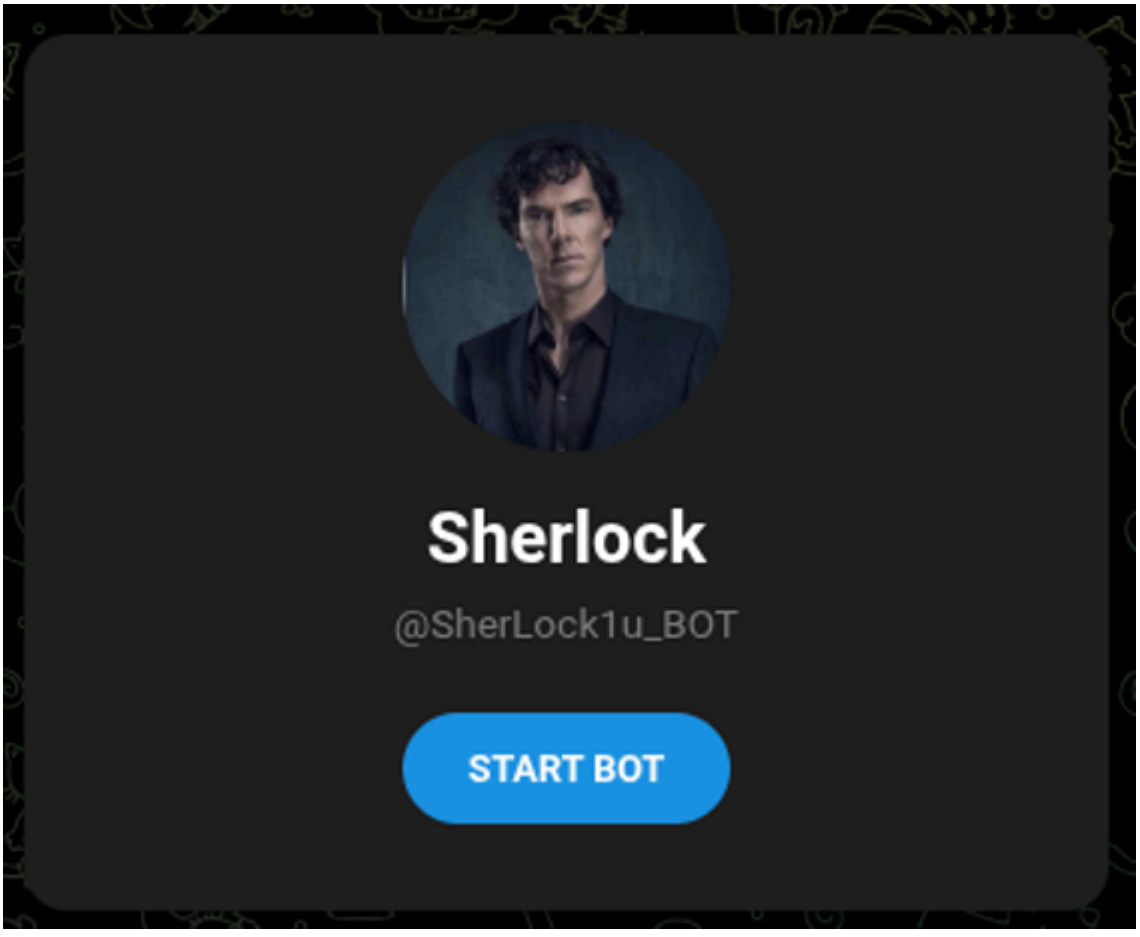


Figure 12 - SherLock1u_BOT

We also tracked activity from the bots since April indicating targeting of victims in South Korea. The following image shows details of exfiltrated data from one Korea-based victim by the `MRB_NEW_VER_BOT` ID.

Figure 13 - South Korea victim data uploaded to Telegram via PXA Stealer

Victimology

Our analysis uncovered details around victimology for several active BotIDs associated with the ongoing PXA Stealer campaign. Some of these Bots have been active since at least October 2024, and they continue to receive data from infected hosts to date.

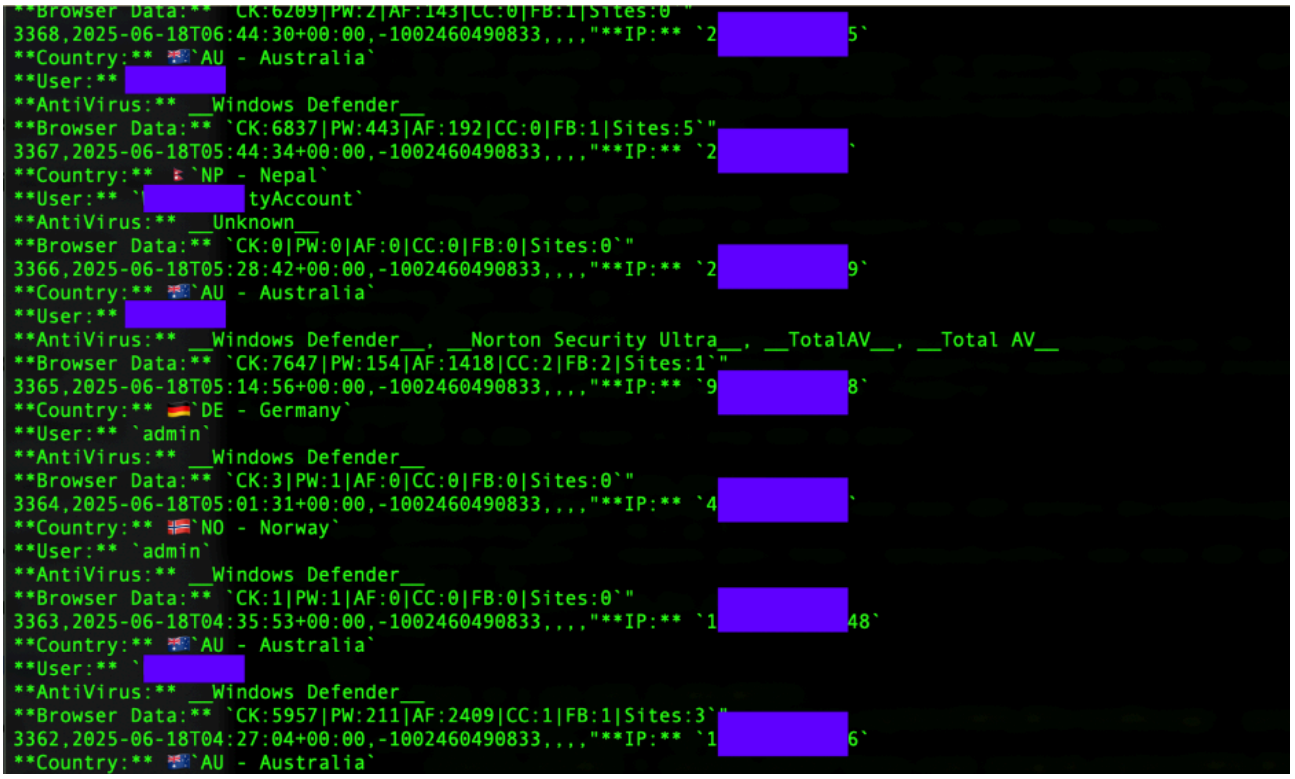


Figure 14 - Adonis (ADN_2_NEW_BOT) Victim records

The PXA Stealer logs contain victim IP addresses that indicate there are potentially more than 4,000 unique victims from 62 countries. The top targeted countries in the analyzed set are:

1. **1.**

   Republic of Korea (KR)

2. **2.**

United States (US)

3. **3.**

   Netherlands (NL)

4. **4.**

   Hungary (HU)

5. **5.**

   Austria (AT)

Some appear to favor specific locations, for example Adonis (ADN_2_NEW_VER_BOT) most heavily targets hosts in Israel and Taiwan, followed by KR and US.

Conclusion

The evolving tradecraft in these recent campaigns demonstrates that these adversaries have meticulously refined their deployment chains, making them increasingly more challenging to detect and analyze. The July 2025 attack chain in particular illustrates a highly tailored approach engineered to bypass traditional antivirus solutions, delay execution in sandboxes, and mislead SOC analysts who review process trees or EDR data by using byzantine delivery and installation methods.

This campaign's medley of legitimate applications and non-malicious decoy documents is designed to mislead users and SOC analysts alike. The actors reinforce this facade by naming a user-space folder to mimic the system directory Windows and disguising a Python interpreter as `svchost.exe` to blend into typical system activity. In parallel, they use files with familiar extensions, such as PNG and PDF, to conceal embedded WinRAR executables and ZIP archives, layering their evasion techniques to mislead users, investigators, and traditional detection technologies.

PXA Stealer, and the threat actors behind it, continue to feed the greater infostealer ecosystem. It is also important to note that PXA, along with similar stealers like Redline, Lumma, and Vidar, each produce data that can be neatly ingested into data monetization ecosystems. The sales-oriented services like Sherlock, such as Daisy Cloud and Moon Cloud, take data harvested by these stealers directly from the bots. The more mature services then normalize the sets of exfilterated data to make it 'sales-ready'. The idea behind leveraging the legitimate Telegram infrastructure is driven by the desire to automate exfiltration and streamline the sales process, which enables actors to deliver data more efficiently to downstream criminals. The developer-friendly nature of Telegram–combined with the company's laissez-faire attitude towards cybercrime–underscores the crucial role that Telegram plays in the holistic cybercriminal ecosystem.

Indicators of Compromise

SHA-1 Hashes

05a8e10251a29faf31d7da5b9adec4be90816238	First-Stage Dropper (archive)
06fcb4adf8ca6201fc9e3ec72d53ca627e6d9532	First-Stage Dropper (archive)
06fcb4adf8ca6201fc9e3ec72d53ca627e6d9532	First-Stage Dropper (archive)
0c472b96ecc1353fc9259e1b8750cdfe0b957e4f	First-Stage Dropper (archive)
1594331d444d1a1562cd955aefff33a0ee838ac9	First-Stage Dropper (archive)
1783af05e7cd52bbb16f714e878bfa9ad02b6388	First-Stage Dropper (archive)
185d10800458ab855599695cd85d06e630f7323d	First-Stage Dropper (archive)
23c61ad383c54b82922818edcc0728e9ef6c984d	First-Stage Dropper (archive)
23c61ad383c54b82922818edcc0728e9ef6c984d	First-Stage Dropper (archive)
345c59394303bb5daf1d97e0dda894ad065fedf6	First-Stage Dropper (archive)
345c59394303bb5daf1d97e0dda894ad065fedf6	First-Stage Dropper (archive)
37e4039bd2135d3253328fea0f6ff1ca60ec4050	First-Stage Dropper (archive)
3a20b574e12ffb8a55f1fb5dc91c91245a5195e8	First-Stage Dropper (archive)
3e9198e9546fa73ef93946f272093092363eb3e2	First-Stage Dropper (archive)
3f0071d64edd72d7d92571cf5e4a5e82720c5a9b	First-Stage Dropper (archive)

40795ca0880ea7418a45c66925c200edcddf939e	First-Stage Dropper (archive)
407df08aff048b7d05fd7636be3bc9baa699646d	First-Stage Dropper (archive)
44feb2d7d7eabf78a46e6cc6abdd281f993ab301	First-Stage Dropper (archive)
4528215707a923404e3ca7667b656ae50cef54ef	First-Stage Dropper (archive)
4528215707a923404e3ca7667b656ae50cef54ef	First-Stage Dropper (archive)
4607f6c04f0c4dc4ee5bb68ee297f67ccdcff189	First-Stage Dropper (archive)
48325c530f838db2d7b9e5e5abfa3ba8e9af1215	First-Stage Dropper (archive)
48d6350afa5b92958fa13c86d61be30f08a3ff0c	First-Stage Dropper (archive)
4dcf4b2d07a2ce59515ed3633386addff227f7bd	First-Stage Dropper (archive)
5246e098dc625485b467edd036d86fd363d75aae	First-Stage Dropper (archive)
540227c86887eb4460c4d59b8dea2a2dd0e575b7	First-Stage Dropper (archive)
5b60e1b7458cef383c45998204bbaac5eacbb7ee	First-Stage Dropper (archive)
612f61b2084820a1fcd5516dc74a23c1b6eaa105	First-Stage Dropper (archive)
61a0cb64ca1ba349550176ef0f874dd28eb0abfa	First-Stage Dropper (archive)
6393b23bc20c2aaa71cb4e1597ed26de48ff33e2	First-Stage Dropper (archive)

65c11e7a61ac10476ed4bfc501c27e2aea47e43a	First-Stage Dropper (archive)
6eb1902ddf85c43de791e86f5319093c46311071	First-Stage Dropper (archive)
70b0ce86afebb02e27d9190d5a4a76bae6a32da7	First-Stage Dropper (archive)
7c9266a3e7c32daa6f513b6880457723e6f14527	First-Stage Dropper (archive)
7d53e588d83a61dd92bce2b2e479143279d80dcd	First-Stage Dropper (archive)
7d53e588d83a61dd92bce2b2e479143279d80dcd	First-Stage Dropper (archive)
7e505094f608cafc9f174db49fbb170fe6e8c585	First-Stage Dropper (archive)
ae8d0595724acd66387a294465b245b4780ea264	First-Stage Dropper (archive)
b53ccd0fe75b8b36459196b666b64332f8e9e213	First-Stage Dropper (archive)
b53ccd0fe75b8b36459196b666b64332f8e9e213	First-Stage Dropper (archive)
bfed04e6da375e9ce55ad107aa96539f49899b85	First-Stage Dropper (archive)
c46613f2243c63620940cc0190a18e702375f7d7	First-Stage Dropper (archive)
c5407cc07c0b4a1ce4b8272003d5eab8cdb809bc	First-Stage Dropper (archive)
c9caba0381624dec31b2e99f9d7f431b17b94a32	First-Stage Dropper (archive)
ca6912da0dc4727ae03b8d8a5599267dfc43eee9	First-Stage Dropper (archive)

d0b137e48a093542996221ef40dc3d8d99398007	First-Stage Dropper (archive)
d1a5dff51e888325def8222fdd7a1bd613602bef	First-Stage Dropper (archive)
deace971525c2cdba9780ec49cc5dd26ac3a1f27	First-Stage Dropper (archive)
deace971525c2cdba9780ec49cc5dd26ac3a1f27	First-Stage Dropper (archive)
e27669cdf66a061c5b06fea9e4800aafdb8d4222	First-Stage Dropper (archive)
e27669cdf66a061c5b06fea9e4800aafdb8d4222	First-Stage Dropper (archive)
e9dfde8f8a44b1562bc5e77b965b915562f81202	First-Stage Dropper (archive)
f02ae732ee4aff1a629358cdc9f19b8038e72b7b	First-Stage Dropper (archive)
f02ae732ee4aff1a629358cdc9f19b8038e72b7b	First-Stage Dropper (archive)
f5793ac244f0e51ba346d32435adb8eeac25250c	First-Stage Dropper (archive)
f7bb34c2d79163120c8ab18bff76f48e51195d35	First-Stage Dropper (archive)
f8f328916a890c1b1589b522c895314a8939399c	First-Stage Dropper (archive)
f91e1231115ffe1a01a27ea9ab3e01e8fac1a24f	First-Stage Dropper (archive)
faf033dc60fed4fc4d264d9fac1d1d8d641af5e0	First-Stage Dropper (archive)
faf033dc60fed4fc4d264d9fac1d1d8d641af5e0	First-Stage Dropper (archive)

ff920aee8199733258bb2a1f8f0584ccb3be5ec6	First-Stage Dropper (archive)
3d38abc7786a1b01e06cc46a8c660f48849b2b5f	Side-loaded DLL
08f517d4fb4428380d01d4dd7280b62042f9e863	Encoded PDF (Archive)
1aa5a0e7bfb995fc2f3ba0e54b59e7877b5d8fd3	Python stealer
734738e7c3b9fef0fd674ea2bb8d7f3ffc80cd91	Python stealer
80e68d99034a9155252e2ec477e91da75ad4f868	Python stealer
ba56a3c404d1b4ed4c57a8240e7b53c42970a4b2	Python stealer
bd457c0d0a5776b43969ce28a9913261a74a4813	Python stealer
da210d89a797a2d84ba82e80b7a4ab73d48a07b1	Python stealer
dc6a62f0a174b251e0b71e62e7ded700027cc70b	Python stealer
533960d38e6fee7546cdea74254bccd1af8cbb65	Stage2 Python stealer
c5688fc4c282f9a0dc62cf738089b3076162e8c6	Stage2 Python stealer
c9a1ddf30c5c7e2697bc637001601dfa5435dc66	Stage2 Python stealer
4ab9c1565f740743a9d93ca4dd51c5d6b8b8a5b6	Browser Injection DLL

Domains

paste[.]rs	Code hosting site
0x0[.]st	Code hosting site
lp2tpju9yrz2fklj.lone-none-1807[.]workers[.]dev	Cloudflare Worker

URLs

hxxps://0x0[.]st/8nyT.py
hxxps://0x0[.]st/8dxc.py
hxxps://0x0[.]st/8GcQ.py
hxxps://0x0[.]st/8GpS.py
hxxps://0x0[.]st/8ndd.py
hxxps://0x0[.]st/8GcO.py
hxxps://0x0[.]st/8GsK[.]py
hxxps://paste[.]rs/yd2sV
hxxps://paste[.]rs/umYBi
hxxps://paste[.]rs/qDTxA
hxxps://paste[.]rs/Plk1y

hxxps://paste[.]rs/5DJ0P
hxxps://paste[.]rs/oaCzj
hxxps://www[.]dropbox[.]com/scl/fi/c1abtpif2e6calkzqsrbj/.dll?rlkey=9h1ar7wmsg407ngpl25xv2spt&st=mp7z58v2&dl=1

Source: https://labs.beazley.security/articles/ghost-in-the-zip-or-new-pxa-stealer-and-its-telegram-powered-ecosystem