RisePro stealer targets Github users in “gitgub” campaign

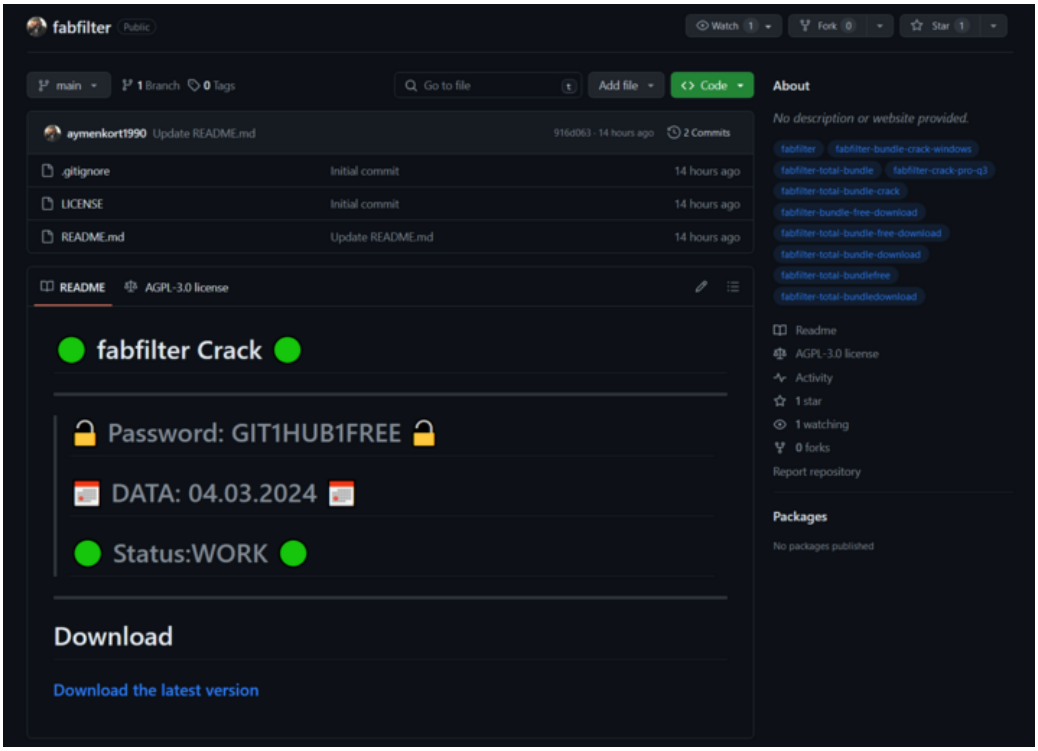
gdatasoftware.com/blog/2024/03/37885-risepro-stealer-campaign-github

RisePro resurfaces with new string encryption and a bloated MSI installer that crashes reversing tools like IDA. The "gitgub" campaign already sent more than 700 archives of stolen data to Telegram.

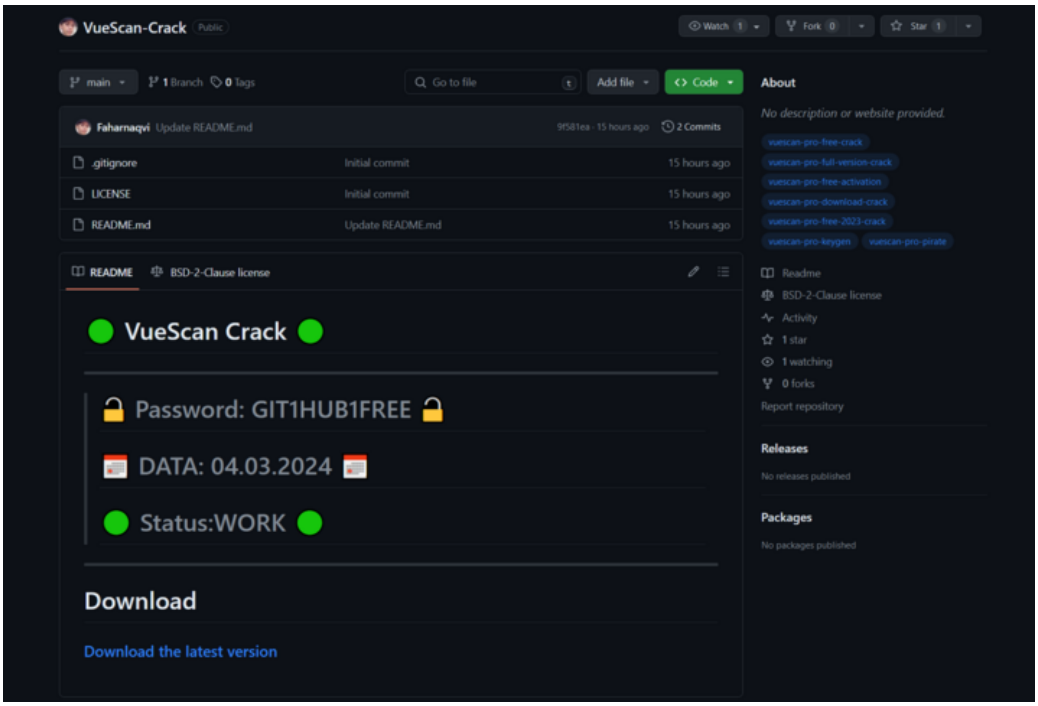
Following Arstechnica’s story about malicious Github repositories, we wrote a threat hunting tool to identify abused repositories. What caught our attention weren’t forks of existing repositories as described by Arstechnica, but newly created repos that lead to the same download link.

Github repositories

We identified at least 13 such repositories belonging to a RisePro stealer campaign that was named “gitgub” by the threat actors. The repositories look similar, featuring a README.md file with the promise of free cracked software. Green and red circles are commonly used on Github to display the status of automatic builds. Gitgub threat actors added four green Unicode circles to their README.md that pretend to display a status alongside a current date and provide a sense of legitimacy and recency.



Repository on GitHub that lures users into downloading malware (click to enlarge)



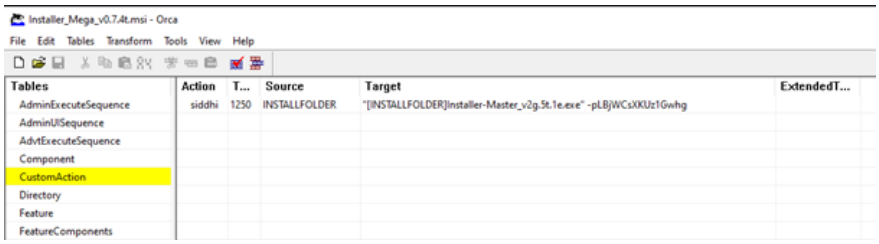
Similar "gitgub" repository but with a different title (click to enlarge)

The following repositories belong(ed) to the gitgub campaign:

- andreastanaj/AVAST
- andreastanaj/Sound-Booster
- aymenkort1990/fabfilter
- BenWebsite/-IObit-Smart-Defrag-Crack
- Faharnaqvi/VueScan-Crack
- javisolis123/Voicemod
- lolusuary/AOMEI-Backupper
- lolusuary/Daemon-Tools
- lolusuary/EaseUS-Partition-Master
- lolusuary/SOOTHE-2
- mostofakamaljoy/ccleaner
- rik0v/ManyCam
- Roccinhu/Tenorshare-Reiboot
- Roccinhu/Tenorshare-iCareFone
- True-Oblivion/AOMEI-Partition-Assistant
- vaibhavshiledar/droidkit
- vaibhavshiledar/TOON-BOOM-HARMONY

The download link is the same for all repositories, namely:

hxxps://digitalxnetwork[.]com/INSTALLER%20PA$$WORD%20GIT1HUB1FREE.rar



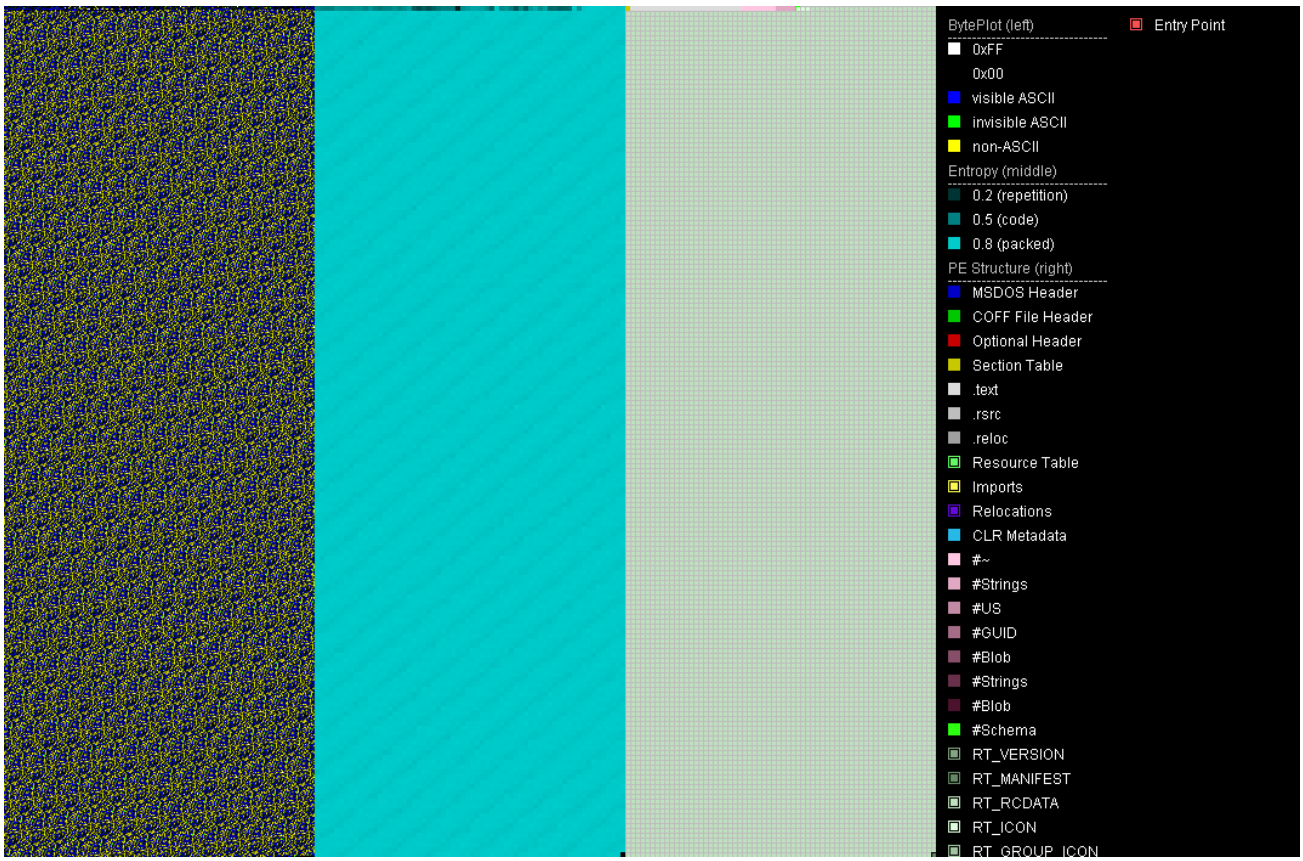
Installer_Mega_v0.7.4t.msi in Orca.exe (click to enlarge)

The unsuspecting user must unpack several layers of archives using the password "GIT1HUB1FREE" before they reach the first executable file named Installer_Mega_v0.7.4t.msi[2].

Orca.exe shows that this MSI unpacks the next stage by applying the password "LBjWCsXKUz1Gwhg". The resulting file is named Installer-Ultimate_v4.3e.9b.exe[3].

Installer-Ultimate_v4.3e.9b.exe

To complicate analysis, this file[3] is bloated to 699 MB, which causes IDA and ResourceHacker to crash. The visualization of the sample by PortexAnalyzer shows that the bloat is non-trivial. While many bloated files feature appended zero bytes, this file has high entropy and no overlay.



PortexAnalyzer visualization of bloated Installer-Ultimate_v4.3e.9b.exe

Knowing that the self-extracting archive from which we unpacked the sample compressed this file to 70 MB, we suspected a repeating pattern. Additionally, the visualization of PortexAnalyzer (see image above) shows a ripple in the high entropy area, which suggests the same.

After inspecting the area, we identified a repeating block of 0x1C0 bytes, followed by a unique byte block of size 0x2d. This repeating byte block allows the file to be compressed when archived while keeping the entropy of the unpacked, bloated file high.

```
17 53 18 2F 9F FC 12 BD 1A E9 7B 4A EB 53 CC 6A D3 05 E6 B3 9A FE 7C AF 05 52 05 73 2E 1D EB 14
56 12 84 56 BD DA E8 46 6A 7C 60 8C B3 8F 70 DE E8 70 F1 C2 71 C3 53 6E C8 F0 EB B0 12 32 86 00
88 5F D7 B2 66 F4 F4 80 22 28 45 12 62 08 D3 CB BE 49 CE CD 60 12 BA 6F 17 A0 0F B4 C2 2C D5 08
DF 3F F9 2E BA A0 C9 64 E3 AF 69 99 1D 5C E6 20 4D 1A 77 01 03 08 94 43 28 F3 F7 47 8D F7 55 FD
4E A7 65 99 D4 10 66 F9 8A D7 29 D2 76 24 F1 BA 60 3B BA 4F C5 8B 06 AB ED 3E C4 94 A7 FE 96 59
9B 6E 33 A1 EE 6D 47 66 F6 E2 F4 8C 41 89 1A 34 AD 0C 10 64 0C BA 27 AC 91 77 F4 08 B7 FF 5D AE
C5 D1 1A 8B 7B 12 93 64 B4 05 C0 2F 6F 49 F1 11 19 13 83 E1 0D D0 64 75 75 D6 5A E8 AA 42 C9 BE
A6 CD 16 27 2E 08 01 9E 3B 69 6F 27 D9 9B C1 62 A4 14 58 6F 45 00 44 5D 22 2A 29 1A DE 7E 1E 18
4A 95 AB 1F 26 97 07 97 2B 16 6E 54 E2 50 AB F2 24 99 6D 80 93 2C 9F DC 3A F5 08 37 BE C4 7B 4A
E1 BC 49 47 79 16 53 C5 9E 44 F1 7B AD 21 ED 7F 4E AF 17 2F 4C 6B 4C 5B 37 B2 74 80 0B 41 F8 F7
69 94 F0 E5 CE D8 01 DF B0 75 CB 39 84 50 7F E5 B5 87 3E E8 D6 DD F3 AB 3F B8 C9 0E 47 61 81 C8
1C 72 92 4F DE 19 00 96 AB FB 4C EA 51 47 9B 31 3F 42 1C 2A DF 90 DC CC 96 69 37 D5 BE B8 80 1C
F4 CA FC D0 8D 98 E2 81 16 D1 46 B7 C1 C4 36 AA C2 23 8E 08 9A 06 18 37 16 6E A2 D4 09 E9 DF D0
F2 8F 39 BA EC BB A2 77 20 C2 C6 16 2C 49 19 21 95 E5 1A 59 78 D8 61 64 9F 3E 11 76 CF 37 14 64
```

(the hexadecimal values of the repeating block)

The bloat data resides in an RC_DATA PE resource named MICROSOFTVISUALSTUDIODEBUGGERI, which has a size of 0x2b85418f bytes. By removing this resource with CFF Explorer we successfully slimmed the file from 699 MB down to 3.43 MB.

Detect-it-Easy identifies an InnoSetup signature in this file, however, this signature turns out to be fake. The file is a .NET assembly.

The .NET streams are unusual: Firstly, there are two #Blob and #Strings streams even though there can only be one according to the Common Language Infrastructure (CLI) specification (see II.24.2.2, page 272). Secondly, there is a #Schema stream, which is not part of the CLI (see II.24.2.2, page 272). The three faulty streams have an invalid size of one byte and point to the same offset. This is likely an attempt to break or confuse .NET parsers.

```
Stream name    size          offset to BSJB   actual offset
-------------------------------------------------------------
#~             0x7045c       0xa0             0x1881fc
#Strings       0x3803c       0x704fc          0x1f8658
#US            0x7a0         0xa8538          0x230694
#GUID          0x10          0xa8cd8          0x230e34
#Blob          0xd708        0xa8ce8          0x230e44
#Strings       0x1           0xb63f0          0x23e54c
#Blob          0x1           0xb63f1          0x23e54d
#Schema        0x1           0xb63f2          0x23e54e
```

Output of PortexAnalyzer CLI showing the invalid .NET streams of Installer-Ultimate_v4.3e.9b.exe (click to enlarge)

```
ModuleRef
*********

1. Name: SwertiaPeridot.dll
2. Name: HogPeridot.dll
3. Name: DankeHog.dll
4. Name: KefsHog.dll
5. Name: GemworkGweed.dll
6. Name: WaspsCraver.dll
7. Name: CraverClash.dll
8. Name: PetsSeeable.dll
9. Name: ExertPets.dll
10. Name: WarmishClash.dll
11. Name: SeeableJuniper.dll
12. Name: SeeableWasps.dll
```

Output by PortexAnalyzer CLI shows the ModuleRef table with 727 references to DLLs

The ModuleRef table references 727 DLL files (see image on the right) which all seem to consist of two arbitrary dictionary words appended to each other, except for the last entry, which is kernel32.

The file is obfuscated with a version of .NET Reactor 6 and has virtualization enabled, which means deobfuscation of the loader's code requires to write a disassembler for the virtualized instructions.

During execution, the loader connects to hxxp://176.113.115(dot)227:56385/31522 and injects its payload[4] into either AppLaunch.exe or RegAsm.exe.

Injected RisePro payload

We identified the payload[4] as RisePro stealer version 1.6. The version number stems from the stealer's logs. We assume that this is the latest version of RisePro, as within the malware authors' publicly accessible Telegram channel, the most recent server updates are referred to as version 1.6.

The sample resolves its imports dynamically using import hashing with Fowler–Noll–Vo hash 1A.

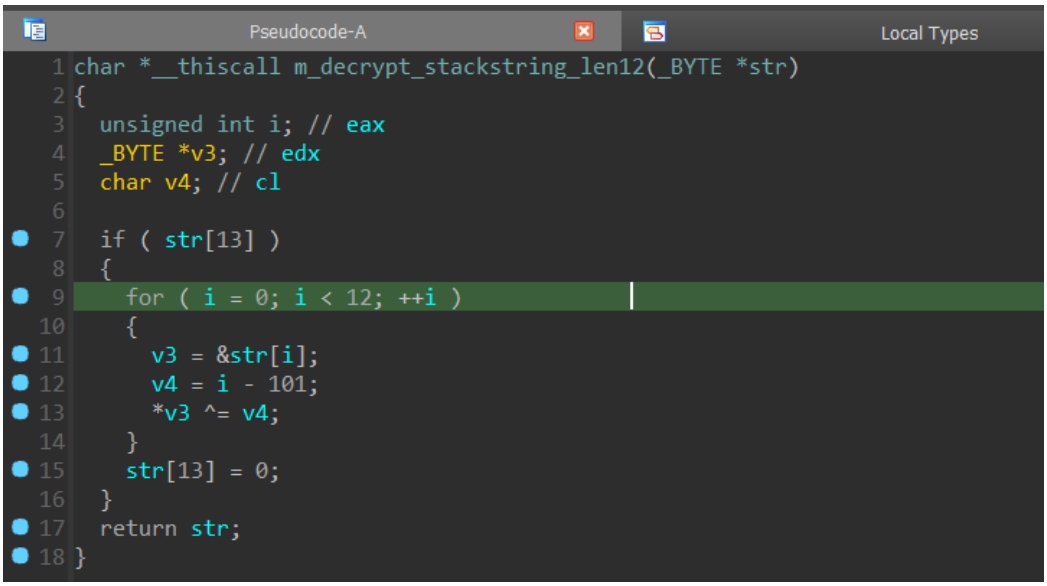
```
LABEL_80:
    ;
  }
  v88 = v84 + *(_DWORD *)(v86 + 32) + 4 * v87;
  while ( 1 )
  {
    v89 = *(_DWORD *)(v88 - 4);
    v88 -= 4;
    v90 = (char *)(v84 + v89);
    --v87;
    compHash = 0x811C9DC5;                      // FNV offset
    v92 = *v90;
    v93 = v90 + 1;
    if ( v92 )
    {
      do
      {
        ++v93;
        compHash = 0x1000193 * (compHash ^ v92);
        v92 = *(v93 - 1);
      }
      while ( v92 );
      if ( compHash == 0x53B2070F )             // LoadLibraryA
        break;
    }
    v84 = (int)m[3].Flink;
    if ( !v87 )
      goto LABEL_80;
  }
  if ( !((int (__cdecl *)(int *))(v573
                                + *(_DWORD *)(v573
                                            + *(_DWORD *)(v584 + 28)
                                            + 4 * *(unsigned __int16 *)
```

Import hash comparison for LoadLibraryA using FNV-1A

RisePro uses XOR obfuscated stack strings. A previous RisePro report by OALabs states that xorstr library has been used for RisePro's string obfuscation in the past, but it seems the encryption scheme has been changed in the meantime.

The library xorstr features "heavily vectorized compile time string encryption" and uses vpxor/pxor instructions to perform the XOR operations. A yara rule in the aforementioned OALabs report searches for pxor patterns in the binary. RisePro developers may have reacted to that public rule.

The new stack string decryption scheme uses one hardcoded decryption function for every string length that the stealer needs. All of these functions use the same underlying decryption algorithm but the length is hardcoded in the decryption loop.



RisePro's string decryption function for strings of length 12

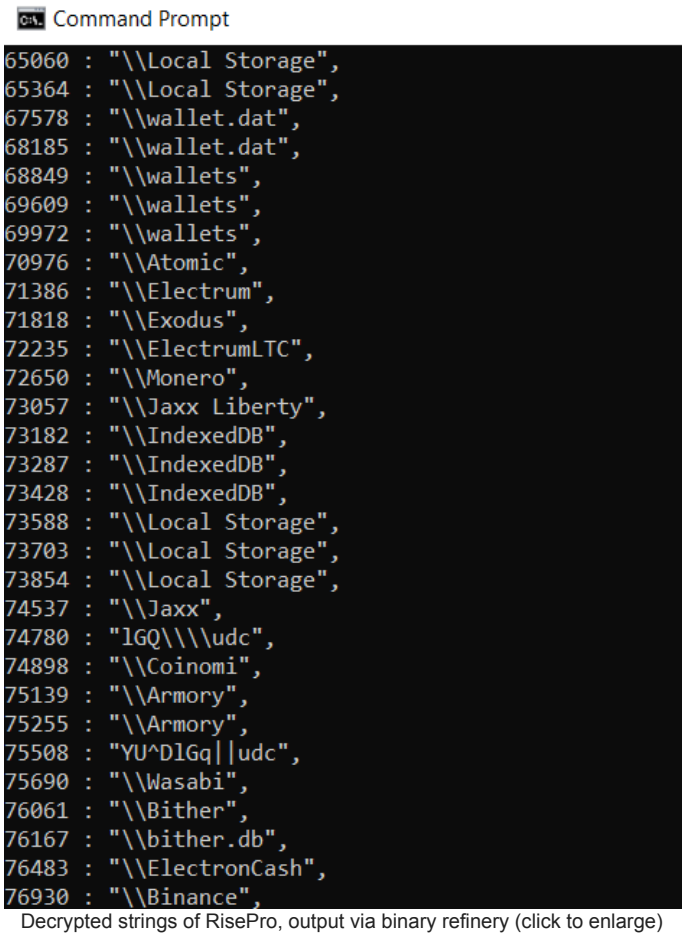
We decrypted the strings with a Binary Refinery snippet, which prints the decrypted strings and their offsets in a format usable for Python dictionaries.

```
emit sample |
        rex "\xC7(\x85|\x45).{4,8}\xC7(\x85|\x45).{50}" [|
                put o offset |
                rex "\xC7(\x85....|\x45.)(.{4})" {2} [|
                        nop ]|
                alu -s "0-101" --inc "B^S" |
                carve -n 4 printable |
                resub \\ \\ |
                resub \" \\\" |
                cfmt {o} : \"{}\", ]]
```

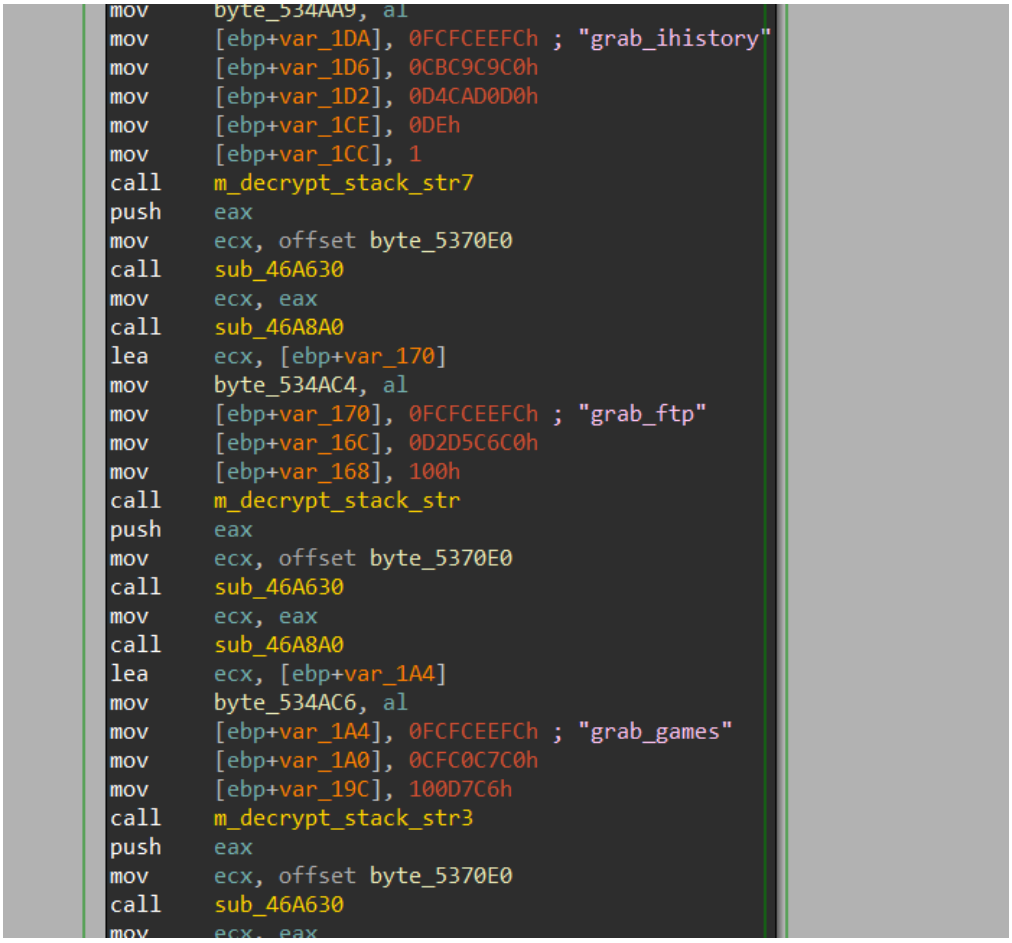
The snippet's regular expression matches a few unintended values, but it is not detrimental to set these as comments in the IDA database. We added the the output to the following IDAPython script.

```
import idc

decrypted_strings_dict = { ... }

for k, v in decrypted_strings_dict.items():
    base = 0x400000
    addr = k + base
    comment = '"' + v + '"'
    idc.set_cmt(addr, comment, 0)
    print("comment", comment, "set at", addr)
```

Address translation is not necessary here because the file offset and virtual addresses in the .text section are the same.



Decrypted strings of RisePro, output via binary refinery (click to enlarge)



IDA Pro with decrypted strings as comments (click to enlarge)

Network communication and configuration

The sample communicates with a C2 server in a manner similar to what was discovered in November 2023 by the Any.Run team.

We used a Python script that was developed by an Any.Run researcher to decrypt network data. The threat actors have not changed their approach and are still using primarily TCP port 50500.

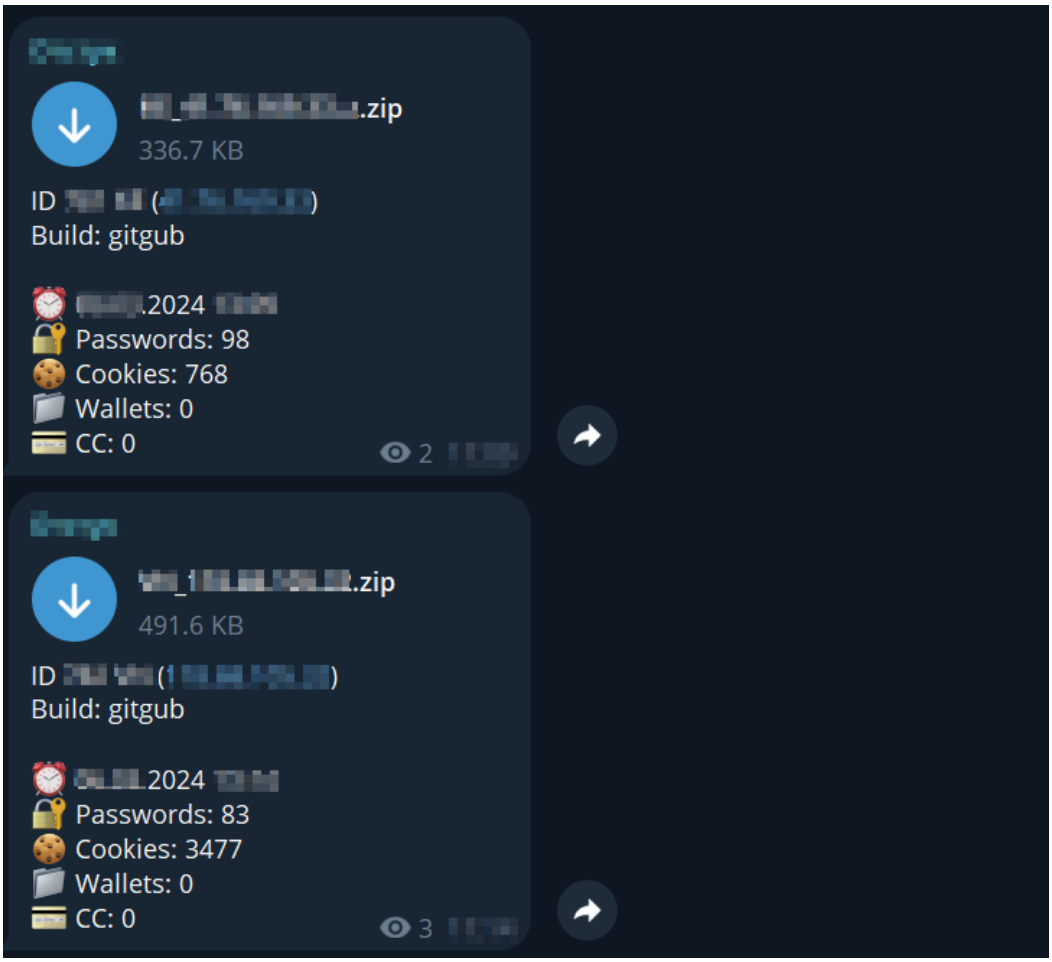
The configuration packet was especially interesting, as it contains information about grabber components, Telegram bot API token, and a Telegram message template.

{"code": "0x2719", "type": "SERVER_SEND_GRAB_CONFIG", "payload_size": 901, "payload": {"block_hwid_all": false,
"block_hwid_day": true, "block_ip_all": false, "external_access": true, "find_seeds": true, "ftp_upload": "", "grab_ds":
true, "grab_ftp": true, "grab_games": true, "grab_ihistory": true, "grab_messengers": true, "grab_screen": true, "grab_tg":
true, "grab_vpn": true, "grab_wallets": true, "merge_browser_data": false, "sapi_key": "[redacted]",
"socks5_password": "", "static_marks": "", "telegram_msg_template": "ID <%id%> <%country_code%> (<%ip%>)\nBuild:
<%build_name%>\n\n\u23f0 <%datetime%>\n\ud83d\udd10 Passwords: <%passwords%>\n\ud83c\udf6a Cookies: <%cookies%>
\n\ud83d\udcc1 Wallets: <%wallets%>\n\ud83d\udcb3 CC: <%cards%>\n<if_wallets>Wallets: <%wallets_list%></if_wallets>\n
<if_marks>Marks: <%marks_list%></if_marks>\n<add_log_file/>", "tg_ids": ["[redacted]", "[redacted]"],
"tg_notif_online_ids": ["[redacted]", "[redacted]"], "tg_token": "[redacted]",
"use_hvnc": true}},
TCP packet with JSON encoded configuration data

Another packet contained base64 encoded data that after decoding turned out to be a .zip archive with information from our analysis machine.

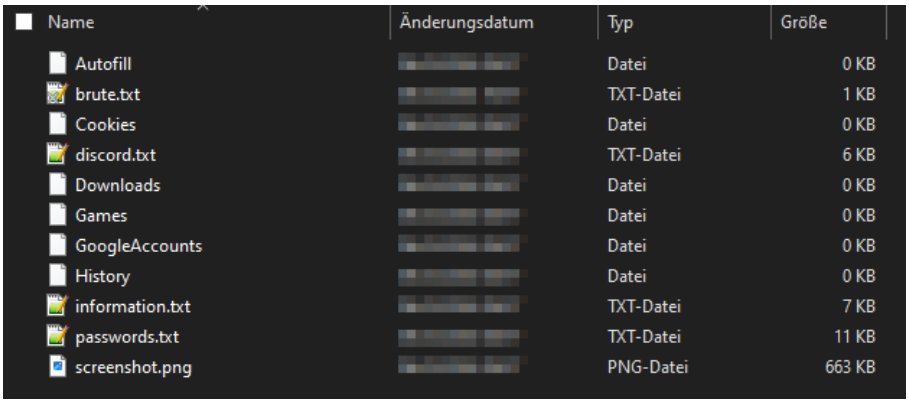
{"code": "0x271f", "type": "CLIENT_SEND_FILE", "payload_size": 468585, "payload": {"filename": "[redacted].zip",
"arg_1": "[redacted]", "arg_2": "gitgub", "file_bytes":
"UEsDBBQAAAgIAPtzZVgAAAAAAgAAAAAAAAAIAAAAQ29va2llc1wDAFBLAwQUAAAICAD7c2VY62LVpk4NAABfGQAAGAAAAAENvb2tpZXNcRWRnZV9EZWZhdWx0LnR4dLWYx67raHaFx8eAH8OTJtBiFjmoAXMWc5wcMOcgZvHprVt2t30q2bfqNqAhIXw7/Hutvf8eZ2OS/z0d+w+eUm3u4/bhWC73Ad1RkIRw6E5+UBrjfZIcBIIYhWEQx+IQQ4EUSoIsiv0HClLf/kGfyw/uwRAk8rfPrOGOGdqtOGpe89HcrtmNUq2+L5VXbexq1zJ2+OjK7WJtdg9cAX1dK0LN1SHWfi22FxZx2skRqNpZYowl9Qy2edTUGUkF1CJcU7CT4RM0IDBEeaISQzGB0RnNnIzIGcMYTkK2FB9JEIRJNl+diJ/+9u//9vekHso/DFNzJfZLCOeaHYaqzpa1mu6tlBy/n5eiBG8AdAAypMs7Z7i
TCP packet with stolen data

The data is exfiltrated to two Telegram channels. At the time of writing, both contained more than 700 messages with zip archives of stolen data. The combination of Telegram channel names and C2 IP addresses indicates a Russia-based operation.



Telegram channel with exfiltrated data archives

The malware collects a variety of valuable information. All unique passwords are stored in a file named “brute.txt”. In the file “password.txt” we discovered a big RISEPRO banner and the link to the public Telegram channel.



Exfiltrated data of one system (click to enlarge)

Contents of passwords.txt with RisePro stealer header (click to enlarge)

Indicators of compromise

Description	File name	SHA-256 / IP / URL
[1] original RAR archive	INSTALLER PA$$WORD GIT1HUB1FREE.rar	f52ba7d8a820d32c502c4aaef4bf9690fc0ca97b97c683b43057d82c06294257
[2] MSI unpacked from [1]	Installer_Mega_v0.7.4t.msi	0ff1e4724b5b02a034789e328531f04a660fd1bade2ad9e73c8b748e5f3e0753
[3] bloated file unpacked from [2]	Installer-Ultimate_v4.3e.9b.exe	492a71bf56d2e89d0b9c5d70ed6c37acea89c8134fa5ba623bce74b3f0fb189e
[4] RisePro payload injected to RegAsm.exe or AppLaunch.exe	memory only	b0e194ed54bafa753bda5761c1264b67a5c438ee7a9ed624a83be913f037dcbb
[5] manually debloated from [3]	Installer-Ultimate_v4.3e.9b.exe	059067376a6271fdead553b471ec899dec3662ec09bd5c3833911c87ea062e92
[6] contacted IP		176[.]113[.]115[.]227
[7] contacted IP		193[.]233[.]132[.]32
[8] download link of RAR archive [1]		hxxps://digitalxnetwork[.]com/INSTALLER%20PA$$WORD%20GIT1HUB1FREE.rar

from G DATA Security Lab
Virus-Analyst Team

- You are here:
- Blog (EN)
- RisePro stealer targets Github users in “gitgub” campaign