SnatchLoader Reloaded | NETSCOUT

Archived: 2026-04-05 20:04:46 UTC

Executive Summary

SnatchLoader is a “downloader” malware—a type of malware that specializes in distributing (or loading) other malware onto infected computers. We first started seeing it in the wild around January 2017, but after a few months it went dormant. Recently, development of the malware has picked up again and we’ve seen updates as recently as last week. It is currently being used to load a banking trojan known as Ramnit. Additionally, it’s using an interesting feature known as “geo-IP blocking” so that only computers in certain geographical areas become infected. We have been able to determine that at a minimum the UK and Italy are being targeted, but the US, France, and Hong Kong are not.

Introduction

There was an interesting Twitter thread a couple of months ago about a spam campaign delivering, at the time, an unknown “downloader” malware—a type of malware that specializes in distributing other malware families. Based on our analysis we believe that it is an update to the downloader known as “SnatchLoader” which was briefly discussed on the KernelMode.info forum in January 2017 . As noted in that post, there seems to be some similarities between SnatchLoader and a third family known as H1N1 Loader—though a detailed code comparison was not performed. Its lineage aside, we haven’t seen any further discussions of SnatchLoader, so this post takes a look at the latest version that we’ve seen.

Samples

The sample referenced in the original Twitter thread is available on VirusTotal. However, most of our static analysis was performed on an updated version of the “core DLL” with a compilation date of 2017-10-04. This DLL is also on VirusTotal and was first seen there on 2017-10-11.

Windows API Calls

All calls to the Windows API are done at run time via function name hashing. The hashing algorithm is a combination of rotate left (ROL) and XOR operations. An example implementation in Python can be found on GitHub. Here is a list of some API function names and their corresponding hashes:

- RtlZeroMemory -> 0x6b6c652b
- CreateMutexW -> 0x43725043
- InternetConnectA -> 0x1d0c0b3e

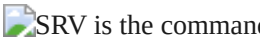
Static Config

A static config is stored encrypted in a PE section of the DLL--so far, we’ve seen two names for this section: .idata and .xdata.:


The first DWORD

The first DWORD of this section (0x99a8 in the screenshot) is used as a seed to a key generation function.

A Python implementation of this function is available on GitHub. The generated key is used with RC4 to decrypt the remaining data. The decrypted config can be separated into two chunks. The first chunk is XML-like and looks like this (whitespace has been added for readability):


SRV is the command and control

SRV is the command and control (C2) URL, TIME is the phone home poll interval in minutes, NAME is a campaign identifier (02.10 likely means October 2nd), and KEY is used to encrypt phone home communications. The second config chunk is an RSA certificate used for signature checking of downloaded data.

Command and Control

So far, all the C2 URLs we've observed are HTTPS. However, using a debugger, we can modify the communications to use HTTP and see what a phone home looks like in plaintext:


The POST data is encrypted

The POST data is encrypted using four layers:

1. RC4 using KEY from the config
2. Base64
3. Character substitutions
4. Split up into 64-byte chunks with "\r\n" delimiters

There are three character substitutions and they are reversible:

- + to –
- / to _
- . to =

The response data is encrypted similarly but without layer 4. Communications are broken up into four request types:

1. Get dynamic config
2. Send system information
3. Command poll
4. Send command results

Get Dynamic Config Request

The plain text version of the “get dynamic config” request looks like this:

```
req=0&guid=FCD08AEE3C0E9409&name=02.10&trash=ulbncmamlxwjakbnbmaklvvhamathrgsfrpbsfrfqeqpatisgsfrqbt
```

Its pieces are:

- req – request type
- guid – bot ID
- name – NAME from static config
- trash – random characters of random length

An example response looks like this:

```
SUCCESS|<CFG><SRV>https://lookmans[.]eu/css/order.php|https://vertasikupper[.]eu/css/order.php</SRV>
```

This response can be separated into two fields: the status field and the data portion. Here the status field is “SUCCESS” and the data portion is encapsulated in the “<CFG> block”—this config is called the DYNAMIC config in the code.

Send System Information Request

The second phone home request sends a bunch of system information and it looks like this:

```
req=1&guid=FCD08AEE3C0E9409&name=02.10&win=9&x64=1&adm=1&det=0&def=0&nat=1&usrn=SYSTEM&cmpn=JOHN-PC&
```

Its pieces are:

- req – request type
- guid – bot ID
- name – NAME from the config
- win – Windows version

- x64 – is 64-bit architecture
- adm – is admin
- det – anti-analysis related
- def – anti-analysis process name detected
- nat – has an RFC1918 IP address
- usrn – username
- cmpn – computer name
- uagn – user agent
- sftl – software listing from the Uninstall key in the registry
- prcl – process listing
- trash – random characters of random length

A response looks like this:

```
SUCCESS|
```

Command Poll Request

A command poll request looks like the “get dynamic config” request except the req number is 2. An example response looks like this:

```
SUCCESS|<TASK>20|1|2||MZ...\x00\x00</TASK>|
```

This response has two fields with the first being a status field and the second field being the data portion. The data here can be zero or more TASK blocks with the following fields:

- task ID
- command type
- command arg1 (e.g. file type)
- command arg2 (e.g. hash value)
- command data (e.g. an executable file or URL)

The main functionality of SnatchLoader is to download and load additional malware families so most of the command types and arguments are in support of doing that in various ways (executed normally, executed via rundll32, or injected into explorer.exe). In this example, the command is to extract the embedded executable file and execute it normally. Some of the other supported commands are:

- Plugin functionality (so far, we’ve only seen a Monero crypto currency mining plugin)
- Update config
- Update self

Send Command Results Request

The last phone home type is used to send the results of a command:

```
req=3&guid=FCD08AEE3C0E9409&name=02.10&results=&trash=pffebxmawlawigdawkifcymbxmawlgebxlawkifcymbxmh
```

It is similar to the “command poll” request except the req number is 3 and an additional parameter (results) has been added. There is no response content from the C2 for this request.

Geo-Blocking and Current Payload

An interesting characteristic of the C2 servers we’ve looked at so far is that they seem to be performing some sort of geo-blocking based on source IP addresses. While trying to interact with them via TOR or VPN exit nodes in the US, France, or Hong Kong the servers responded with “404 Not found” errors. But, using VPN exit nodes in the UK and Italy, the C2 responded affirmatively. In general, geo-blocking isn’t a novel feature, but it isn’t particularly common. At the time of writing, the analyzed SnatchLoader botnet was distributing Ramnit—an info stealing and banking malware. It has a compilation date of 2017-10-13 and is available on VirusTotal.

Conclusion

This post has been an overview of a downloader malware known as SnatchLoader. We can trace its origins as far back as January 2017 and it has been updated as recently as last week. It is being delivered via spam campaigns and based on geo-blocking functionality it looks to be targeting specific geographical areas. At the time of writing SnatchLoader is distributing the Ramnit malware family to at least the UK and Italy. Thanks much to Antelox, reOnFleek, XOR_Hex, mesa_matt, and kafeine for help with the geo-IP blocking, distributed payload, name origin, and general discussions on the family.

Source: https://www.arbornetworks.com/blog/asert/snatchloader-reloaded/