SmokeLoader History | ThreatLabz

By ThreatLabz

Published: 2024-06-11 · Archived: 2026-04-05 15:14:35 UTC

2014: Ancient Modularizations

The following section covers new features and modifications to SmokeLoader version 2014, which include a multi-stage loading process, an updated algorithm for generating the bot ID, a separate encrypted C2 list, and more.

During the analysis of a sample dating back to 2014, we discovered the introduction of the string `s2k14` that is used as the name of a file mapping. We believe `s2k14` is a reference to SmokeLoader version 2014. The figure below shows this string referenced in the code.

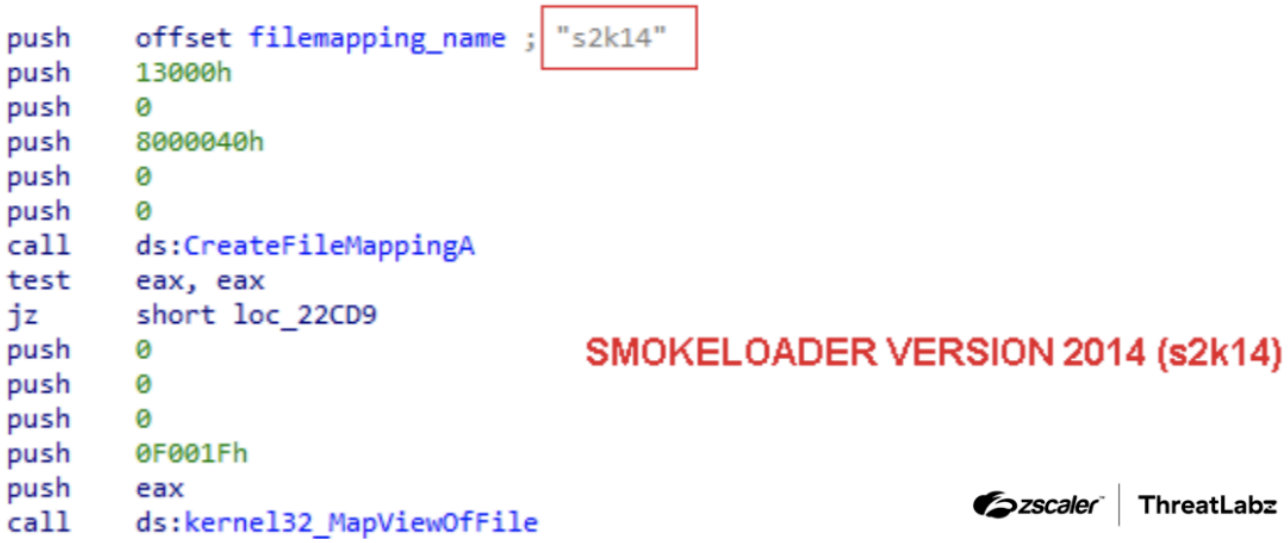

Figure 5: SmokeLoader version 2014 string for a file mapping name called `s2k14`.

One of the most interesting features in this version of SmokeLoader is the malware was split into several loading stages. The introduction of a stager component marked a significant change that became standard in all subsequent versions. Each version since 2014 consists of a stager, a main module, and plugins that implement additional features.

In SmokeLoader version 2014, the `./mods/` folder in the panel includes a `./mods/plugins` file that combines multiple plugins into a single file. The prior `./mods/grab` information stealing module from previous SmokeLoader versions was split into multiple stealing modules like an FTP/mail stealer module, browser stealer module, and keylogger module, and then packaged in this `plugins` file. This modification to the plugins persists from this version onward until the most recent version of SmokeLoader.

aPLib stager

The stager introduced in SmokeLoader version 2014 is quite simple. The stager performs the following actions:

- Decrypts a section of data using a single byte XOR key.
- Decompresses the data using aPLib.
- Maps the main module in a buffer allocated in the same process context.
- Executes some simple anti-analysis measures by checking the Process Environment Block (PEB) such as the `BeingDebugged` and `NtGlobalFlags` fields.

Then, the stager transfers execution to the main module, creates an instance of `svchost.exe`, and injects SmokeLoader into this newly created `svchost.exe` process using APC queue code injection.

The figure below shows how the 2014 version of the SmokeLoader aPLib stager works.

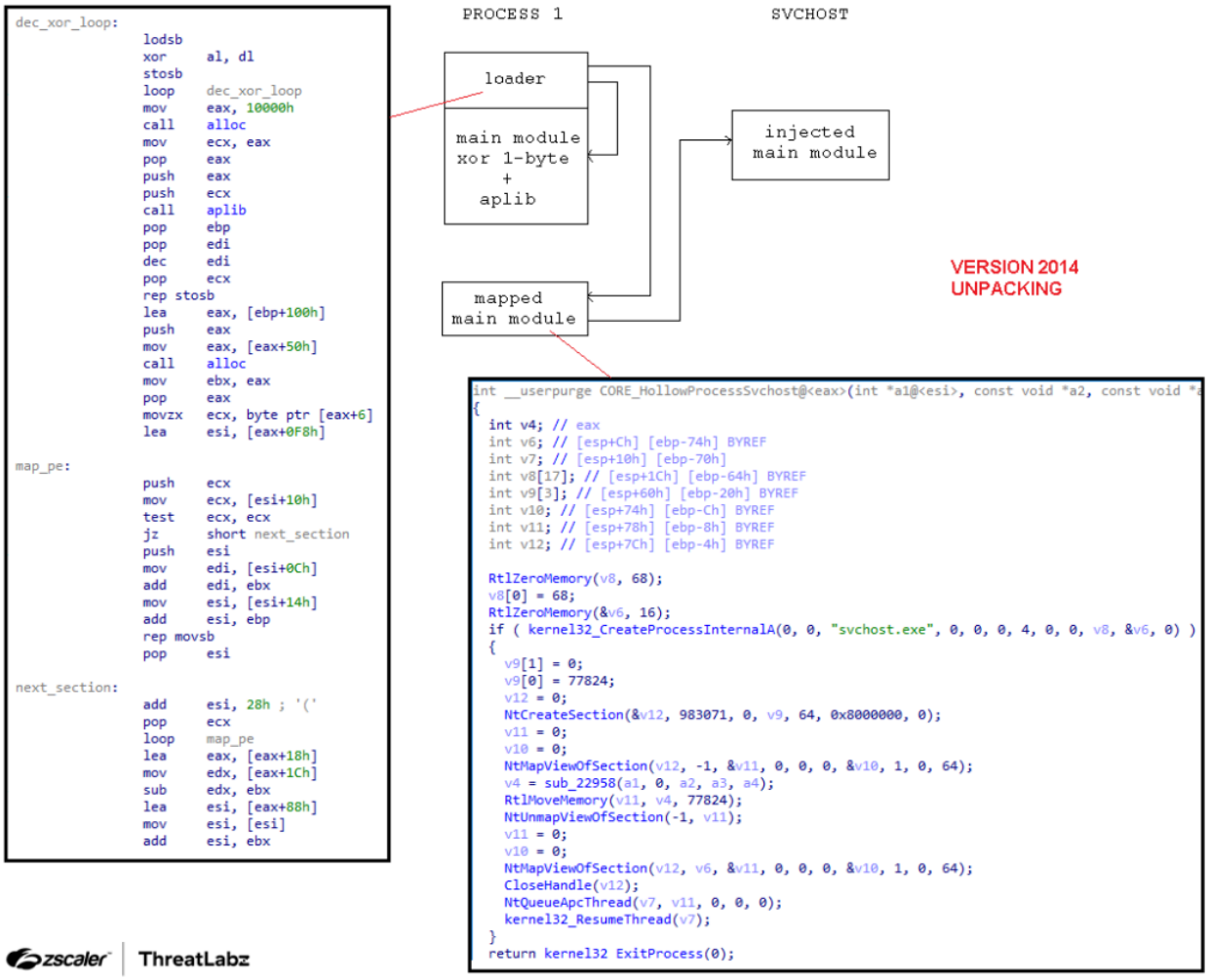

Figure 6: SmokeLoader version 2014 code unpacking and injecting into `svchost.exe`.

SmokeLoader's stager underwent significant evolution in subsequent versions, incorporating advanced obfuscation techniques and additional anti-analysis measures. Nevertheless, even in this version, we can observe the presence of rudimentary obfuscation tricks.

For instance, SmokeLoader employs non-polymorphic decryption loops to unravel layers of XOR encryption for functions that are invoked, as shown in the figure below.

```
                push    offset loc_401227
                mov     esi, [esp+4+var_4]
                mov     edi, esi
                push    43h ; 'C'
                pop     ecx

loc_401277:                             ; CODE XREF: start+11↓j
                lodsb
                xor     al, 47h         ; simple xor decrypt 0x401227
                stosb
                loop    loc_401277
                retn                    ; jmp to 0x401227
```

Figure 7: SmokeLoader version 2014 stager function decryption.

Persistence

The SmokeLoader seller has provided threat actors with an option to build a sample with (or without) persistence. SmokeLoader's approach to achieve persistence on the victim's system has undergone numerous changes over time. In earlier versions (2011-2017), SmokeLoader would leverage common Run registry keys and create a startup shortcut as a fallback option (if setting the registry values failed). In addition, SmokeLoader would establish two dedicated threads responsible for safeguarding the modified registry keys.

Bot ID

In the initial version of SmokeLoader, dating from approximately 2011 to 2012, we observed that the bot ID was generated by taking a simple MD5 hash of the victim machine's computer name. Over time, the algorithm to generate the bot ID has undergone slight modifications. Notably, in version 2014, SmokeLoader employed a CRC32 and XOR based algorithm.

Starting from 2014, all versions of SmokeLoader calculate the ID using both the computer name and the volume information, as shown in the figure below.

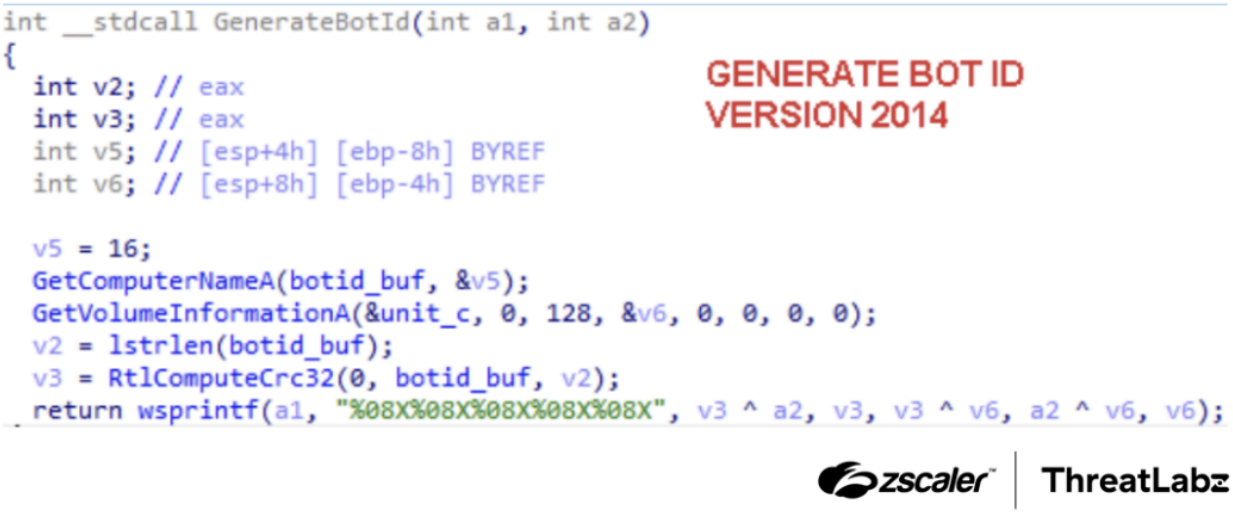

Figure 8: SmokeLoader version 2014 bot ID generation.

Anti-analysis tricks and plain text strings

Interestingly, SmokeLoader version 2014's main module stored the malware's strings in plaintext as shown in the figure below. This is a departure from the initial version where the strings were encrypted.

```
aSbiedll        db 'sbiedll',0          ; DATA XREF: seg000:off_250E0↓o
aDbghelp        db 'dbghelp',0
aQemu           db 'qemu',0             ; DATA XREF: seg000:off_250E8↓o
                align 4
aVirtual        db 'virtual',0
aVmware         db 'vmware',0           ; DATA XREF: seg000:000250F0↓o
                align 4
aXen            db 'xen',0              ; DATA XREF: seg000:000250F4↓o
aZoneIdentifier db ':Zone.Identifier',0 ; DATA XREF: seg000:off_250F8↓o
                align 4
aMozilla40      db 'Mozilla/4.0',0      ; DATA XREF: seg000:off_250FC↓o
aCmdGetloadLogi db 'cmd=getload&login=',0
                                        ; DATA XREF: seg000:off_25100↓o
                align 4
aHttpWwwMsnCom  db 'http://www.msn.com/',0
                                        ; DATA XREF: seg000:off_25104↓o
aGetSHttp11User db 'GET /%s HTTP/1.1',0Dh,0Ah
                                        ; DATA XREF: seg000:off_25108↓o
                db 'User-Agent: %s',0Dh,0Ah
                db 'Host: %s',0Dh,0Ah
                db 'Connection: close',0Dh,0Ah
                db 0Dh,0Ah,0
                align 4
aPostSHttp11Use db 'POST /%s HTTP/1.1',0Dh,0Ah
                                        ; DATA XREF: seg000:off_2510C↓o
                db 'User-Agent: %s',0Dh,0Ah
                db 'Host: %s',0Dh,0Ah
                db 'Connection: close',0Dh,0Ah
                db 'Content-Length: %d',0Dh,0Ah
                db 'Content-Type: application/x-www-form-urlencoded',0Dh,0Ah
                db 0Dh,0Ah,0
aFile           db '&file=',0           ; DATA XREF: seg000:off_25110↓o
```

Figure 9: SmokeLoader version 2014 plaintext strings.

This is the first version of the malware that searches for `sbiedll`, `dbghelp`, `qemu`, `virtual`, `vmware`, and `xen` strings to check for libraries and processes related to malware analysis environments. If an analysis environment is detected, SmokeLoader terminates itself to evade detection.

Encrypted C2s

The algorithm used to decrypt the list of encrypted C2s is one of the SmokeLoader components that has undergone the most changes across versions.

While the strings in the main module of SmokeLoader version 2014 are stored in plaintext, the list of C2 servers is encrypted. To decrypt the list, a custom XOR-based decryption algorithm is employed. The code contains a table of pointers, with each pointer referencing a string that is prepended by a byte that is used as an XOR key, which is

followed by three unused bytes. The next byte is the size of the encrypted C2 URL and the remaining bytes are the encrypted C2 URL. The Python code below demonstrates the C2 decryption algorithm used in SmokeLoader version 2014.

```
def smoke2014_c2_xor_decrypt(enc_data):
  key = enc_data[0]
  size = enc_data[4]
  enc = enc_data[5:]
  dec = b''
  for i in range(0, len(enc)-1, 2):
      dec += (0xff&((enc[i] ^ key) -
             (enc[i+1] ^ key))).to_bytes(1, 'little')
  return dec
```

Anti-C2 patching mechanism

An interesting addition to SmokeLoader version 2014 is the implementation of what appears to be a simple copy-protection mechanism as shown in the figure below.

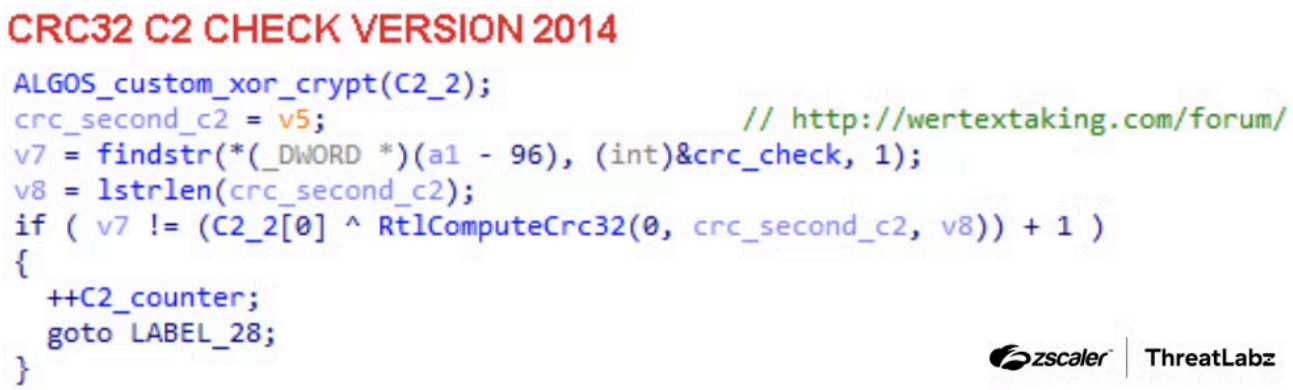

Figure 10: Simple copy-protection mechanism implemented in SmokeLoader version 2014 version.

The malware calculates the CRC32 value of the C2 URL string and compares it with a predefined expected value at various points in the code. This mechanism is likely designed to prevent other hackers from creating a builder that patches samples with new C2s, and therefore reduce potential sales of SmokeLoader.

Communication

The communication protocol for SmokeLoader version 2014 is similar to prior versions. SmokeLoader uses a simple text-based protocol that is encrypted and sent to a C2 server. The syntax for the protocol is the following:

```
arg1=value1&arg2=value2…&argN=valueN
```

SmokeLoader’s version 2014 panel recognizes the following arguments:

Argument	Description
`cmd`	The C2 panel accepts a set of commands. Depending on the specified command additional arguments must be specified.
`login`	This argument is the bot ID and its length must be 40 bytes.
`info`	Additional information given with some commands.
`ver`	Operating system version.
`bits`	Victim's Windows operating system architecture.
`file`	Mainly used together with the `getload` command to request updates or tasks.
`run`	Used in different commands for different purposes. For example, to ask for an update or to indicate that the update was successfully executed.
`port`	Used together with the `getsocks` command to specify the proxy's port.
`procname`	Process name included with the `procmon` command.
`doubles`	Additional flag that may be sent with the `getload` command.
`removed`	Additional flag that may be sent with the `getload` command.
`personal`	Additional flag that may be sent with the `getload` command.
`shell`	If the `getshell` command is sent to the server, this argument contains the results of the executed commands.

Argument	Description
`grab`	Used together with the `formgrab`, `ftpgrab`, and `keylog` commands. This argument contains the stolen information.
`filedata`	If a file is submitted, the `filedata` argument contains the content of the file.

Table 2: SmokeLoader version 2014 network protocol.

The panel is able to handle the following commands (i.e., the command name specified in the `cmd` argument):

Command (cmd)	Description

`getload`
 	
**with additional arguments*
	One of the main commands and differs depending on specified arguments:
	• **file**: If the `getload` command is included with the file argument, the panel handles the command in different ways depending on the value of the `file` argument:
	  ◦ If the `file` argument's value is `u` and the `run` argument is not set, the bot asks for an update. The server could return the update executable or a URL to download the update from.
	  ◦ If the `file` argument's value is `u` and the `run` argument is set, the bot confirms the successful execution of the update.
	  ◦ If the `file` argument's value is not `u` and the `run` argument is not set, the bot asks for the next task. The server tracks in its database the last task given to each bot. When this command is received, the server returns the next task (if any).
	  ◦ If the `file` argument's value is not `u` and the `run` argument is set, SmokeLoader confirms whether the last task was executed correctly.
	• **doubles:** If the bot specifies this argument together with the `getload` command, the panel sets the `doub` flag in the database for the associated victim ID. The purpose of this flag is unknown.
	• **removed**: The bot can specify a `removed` argument together with the `getload` command to confirm an uninstall request was received from the server. The panel deletes the bot ID from the database upon receipt.
	• **personal**: A `personal` argument can be given together with the `getload` command to ask for personal tasks. The argument should contain the ID of the personal task. If the configured task is set as `local` in the

Command (cmd)	Description
	database, the server would return a file in the response to be executed by the bot. Otherwise, a URL is provided and the bot downloads and executes the content from the given URL.
`getload` **without arguments*	If this command is received without arguments, it could be interpreted as a `hello` or `knock` query, and used by the bot to start the conversation with the server. When a server receives a `getload` command without arguments, the response could vary depending on the database configuration. • If there is a pending update for the bot, the server responds with the string " `Smku` ". • If there is a personal task for the bot, the server responds with the string " `Smki` ". • If there is a pending removal request for the bot, the server responds with the string " `Smkr` ". • In the remaining cases, the server responds with the total number of configured tasks followed by the configuration's rules for each plugin. Each configuration item starts with the " `|:|` " string followed by the name of the ruleset and the plugin's rules: ◦ `|:|socks_rules=` ◦ `|:|hosts_rules=` ◦ `|:|shell_rules=` ◦ `|:|fakedns_rules=` ◦ `|:|filesearch_rules=` ◦ `|:|procmon_rules=` ◦ `|:|ddos_rules=` ◦ `|:|keylog_rules=`
`getsocks`	Used by the bot to inform the server that it has enabled the SOCKS proxy feature. The server will attempt to validate the bot proxy is active by connecting to the bot's IP address on the specified port.
`gethosts`	Confirms that the hosts specified in the `hosts_rules` have been successfully spoofed.
`getshell`	The bot submits results from executed shell commands to the server. The shell argument contains the results.

Command (cmd)	Description
`formgrab`	Submits the results from the form grabber plugin to the server. The `grab` argument must contain a base64 encoded string that, once decoded, contains a comma separated list of grabbed data.
`ftpgrab`	Submits the results from the ftp grabber plugin to the server. The `grab` argument must contain a base64 encoded string with the stolen data.
`grab`	Submits stolen information to the server. The argument data contains the stolen information.
`avinfo`	Submits information about installed security products to the server. The information is submitted in the `info` argument. The submitted string is split by the delimiter `777`. The first substring contains information about the installed antivirus. The second substring contains information about installed firewalls.
`procmon`	If one of the processes configured with the rules in the `procmon_rules` configuration item is found on the victim machine, the bot notifies the server about the presence of the process, submitting the name of the process in the `procname` argument. The server could respond with a file to be executed or a URL to download the file from.
`ddos`	Confirms the DDoS attack configured with the `ddos_rules` has been successfully performed.
`keylog`	Submits the information captured by the keylogger plugin to the server. The grab argument must contain a base64 encoded string with captured data.
`getfilesearch`	If the HTTP query is application/bin, and the command is `getfilesearch`, the bot submits to the server the content of the files that were found by the `filesearch` plugin according to the rules specified in the `filesearch_rules` configuration item. The content of the file is given in the `filedata` argument.

Table 3: The commands supported by SmokeLoader’s 2014 C2 server.

SmokeLoader version 2012 sent commands and arguments as plaintext using HTTP GET requests. In version 2014, the network protocol was updated to send the command and argument data via HTTP POST requests. The POST request body consists of an initial `DWORD` containing the size of the data, followed by a `DWORD` that functions as an RC4 key required to decrypt the remaining data.

Explore more Zscaler blogs
