A new secret stash for “fileless” malware

- Denis Legezo

In February 2022 we observed the technique of putting the shellcode into Windows event logs for the first time “in the wild” during the malicious campaign. It allows the “fileless” last stage Trojan to be hidden from plain sight in the file system. Such attention to the event logs in the campaign isn’t limited to storing shellcodes. Dropper modules also patch Windows native API functions, related to event tracing (ETW) and anti-malware scan interface (AMSI), to make the infection process stealthier.

Besides event logs there are numerous other techniques in the actor’s toolset. Among them let us distinguish how the actor takes initial recon into consideration while developing the next malicious stages: the C2 web domain name mimicking the legitimate one and the name in use belonging to the existing and software used by the victim. For hosting the attacker uses virtual private servers on Linode, Namecheap, DreamVPS.

One more visible common approach is the use of a lot of anti-detection decryptors. Actor uses different compilers, from Microsoft’s cl.exe or GCC under MinGW to a recent version of Go. Also, to avoid detection, some modules are signed with a digital certificate. We believe it is issued by the actor, because our telemetry doesn’t show any legitimate software signed with it, only malicious code used in this campaign.

Regarding last stage Trojans: the actor decided not to stick to just one – there are HTTP and named pipe based ones. Obviously besides the event logs the actor is obsessed with memory injection – lots of RAT commands are related to it and are used heavily. Along with the aforementioned custom modules and techniques, several commercial pentesting tools like Cobalt Strike and SilentBreak’s toolset are used.

Actually, as we don't have commercial versions of the latter it's hard to say which enumerated techniques came from the product and which are home-brewed. For sure, third-party code from GitHub is also in use: we registered at least BlackBone for legitimate processes in memory patching.

The infection chain

We started the research from the in-memory last stager and then, using our telemetry, were able to reconstruct several infection chains. What piqued our attention was the very targeted nature of the campaign and the vast set of tools in use, including commercial ones.

The variety of the campaign's techniques and modules looks impressive. Let us divide it into classes to technically describe this campaign. Actually, we need to cover the following sets of modules: commercial pentesting suites, custom anti-detection wrappers around them and last stage Trojans.

Commercial tool sets	SilentBreaks's toolset
	Cobalt Strike
Anti-detection wrappers	Go decryptor with heavy usage of the syscall library. Keeps Cobalt Strike module encoded several times, and AES256 CBC encrypted blob. We haven't previously observed Go usage with Cobalt Strike
	A library launcher, compiled with GCC under MinGW environment. The only possible reason for this stage is anti-detection
	AES decryptor, compiled with Visual Studio compiler
Last stage RAT	HTTP-based Trojan. Possible original names are ThrowbackDLL.dll and drxDLL.dll, but code is more complex than old publicly available version of SilentBreak's Throwback
	Named pipes-based Trojan. Possible original names are monolithDLL.dll and SlingshotDLL.dll. Based on file names there is a possibility that last stage modules are parts of a commercial Slingshot version

Once again, some modules which we consider custom, such as wrappers and last stagers, could possibly be parts of commercial products. So now after some classification we are ready to analyze modules one by one.

Initial infection

The earliest phase of attack we observed took place in September 2021. The spreading of the Cobalt Strike module was achieved by persuading the target to download the link to the .rar on the legitimate site file.io, and run it themselves. The digital certificate for the Cobalt Strike module inside is below (during the campaign with the same one, 15 different stagers from wrappers to last stagers were signed):

```
1 Organization: Fast Invest ApS

2 E-mail: sencan.a@yahoo.com

3 Thumbprint 99 77 16 6f 0a 94 b6 55 ef df 21 05 2c 2b 27 9a 0b 33 52 c4

4 Serial 34 d8 cd 9d 55 9e 81 b5 f3 8d 21 d6 58 c4 7d 72
```

Due to the different infection scenarios for all the targeted hosts we will describe just one of the observed ones. Having an ability to inject code into any process using Trojans, the attackers are free to use this feature widely to inject the next modules into Windows system processes or trusted applications such as DLP.

Keeping in mind truncated process injections, and even mimicking web domain registration, we could describe the attack process as quite iterative: initial recon with some modules and then preparation of additional attacks.

Commercial tool sets

Regarding the commercial tools, traces of SilentBreak and Cobalt Strike toolset usage in this campaign are quite visible. Trojans named ThrowbackDLL.dll and SlingshotDLL.dll remind us of Throwback and Slingshot, which are both tools in SilentBreak's framework, while the "sb" associated with the dropper (sb.dll) could be an abbreviation of the vendor's name.

Here we want to mention that several .pdb paths inside binaries contain the project's directory C:\Users\admin\source\repos\drx\ and other modules not named after Throwback or Slingshot, such as drxDLL.dll. However, encryption functions are the same as in the publicly available Throwback code.

Anti-detection wrappers

For the anti-detection wrappers, different compilers are in use. Besides MSVC, Go compiler 1.17.2 and GCC under MinGW have been used. Decryptors differ a lot; the features they contain are listed in the table below:

Anti-detection technique	**Usage**
Several compilers	The same AES256 CBC decryption could be done with Go and C++ modules
Whitelisted launchers	Autorunned copy of WerFault.exe maps the launcher into process address space
Digital certificate	15 files are signed with "Fast Invest" certificate. We didn't observe any legitimate files signed with it
Patch logging exports of ntdll.dll	To be more stealthy, Go droppers patch logging-related API functions like EtwEventWriteFull in self-address space with empty functionality
Keep shellcode in event logs	This is the main innovation we observed in this campaign. Encrypted shellcode with the next stager is divided into 8 KB blocks and saved in the binary part of event logs
C2 web domain mimicking	Actor registered a web domain name with ERP in use title

This layer of infection chain decrypts, maps into memory and launches the code. Not all of them are worth describing in detail, but we will cover the Go decryptor launcher for Cobalt Strike. All corresponding hashes are listed in the appendix.

Function names in the main package are obfuscated. Main.init decodes Windows API function names from kernel32.dll and ntdll.dll libraries (WriteProcessMemory and other functions) related to event log creation. Each of these names in the binary are base64-encoded four times in a row. Using WriteProcessMemory, the dropper patches with "xor rax, rax; ret" code the following functions in memory: EtwNotificationRegister, EtwEventRegister, EtwEventWriteFull, EtwEventWriteFull, EtwEventWrite.

In Main.start the malware checks if the host is in the domain and only works if it's true. Then it dynamically resolves the addresses of the aforementioned functions. The next stager is encrypted with AES256 (CBC mode), the key and IV are encoded with base64.

With such an approach, it requires the researcher to code some script to gather the encrypted parts of the next module. After decryption, to get the final portable executable, data has to be converted further.

Last stager types

Last stagers have two communication mechanisms – over HTTP with RC4 encryption and unencrypted with named pipes. The latter way is technically able to communicate with any network visible external host, but under Windows named pipes are built upon the SMB protocol, which would barely open for external networks. So these modules most probably serve for lateral movement.

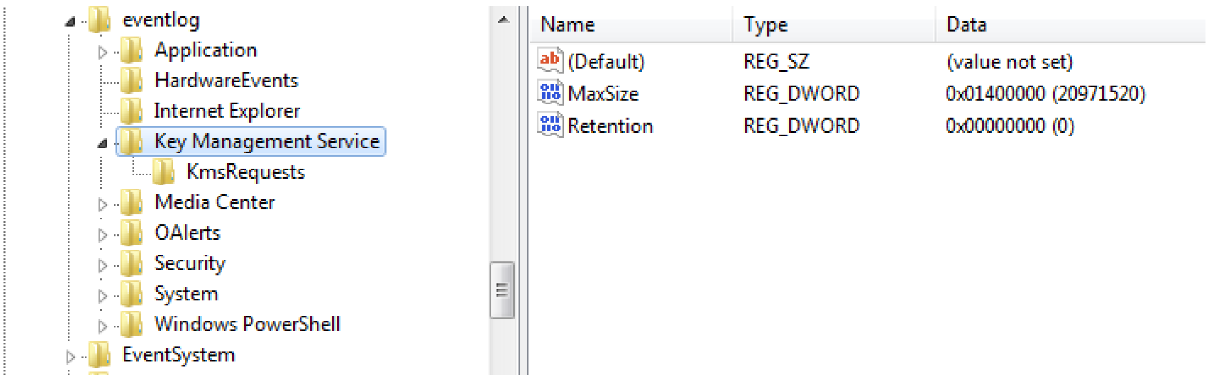
Feature	**HTTP-based trojan**	**Named pipes-based trojan**
C2 communication	Active connection to a randomly chosen C2 from a hardcoded list	Passive mode
Encryption	XOR-based, RC4	Plaintext
Self version in beacon	1.1	No
Natural language artifacts	Unused argument "dave"	No
Command set	Quite basic, 7 of them	More profound, 20 of them
Injection functionality	Yes and much in use	Yes and much in use
Quite unusual among the commands	Sleep time randomization: (random between 0,9 – 1,1) * sleep time	Get minutes since last user input

After this introduction into the set of malware, we will now describe the infection chain: dropper injection with Cobalt Strike pentesting suite.

Dropper in DLL, search order hijacking

We start custom module analysis from the wrapper-dropper dynamic library. This code is injected into Windows processes such as explorer.exe. At its single entry point after being loaded into the virtual address space of the launcher process, the dropper removes files created by previous stages or executions.

Firstly, the module copies the original legitimate OS error handler WerFault.exe to C:\Windows\Tasks. Then it drops one of the encrypted binary resources to the wer.dll file in the same directory for typical DLL search order hijacking. For the sake of persistence, the module sets the newly created WerFault.exe to autorun, creating a Windows Problem Reporting value in the Software\Microsoft\Windows\CurrentVersion\Run Windows system registry branch.



```
eventlog
    Application
    HardwareEvents
    Internet Explorer
    Key Management Service
        KmsRequests
    Media Center
    OAlerts
    Security
    System
    Windows PowerShell
EventSystem
```

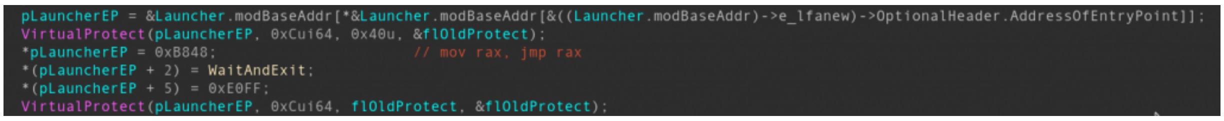
Name	Type	Data
(Default)	REG_SZ	(value not set)
MaxSize	REG_DWORD	0x01400000 (20971520)
Retention	REG_DWORD	0x00000000 (0)

The dropper not only puts the launcher on disk for side-loading, but also writes information messages with shellcode into existing Windows KMS event log

The dropped wer.dll is a loader and wouldn't do any harm without the shellcode hidden in Windows event logs. The dropper searches the event logs for records with category 0x4142 ("AB" in ASCII) and having the Key Management Service as a source. If none is found, the 8KB chunks of shellcode are written into the information logging messages via the ReportEvent() Windows API function (lpRawData parameter). Created event IDs are automatically incremented, starting from 1423.

Launcher in wer.dll

This launcher, dropped into the Tasks directory by the first stager, proxies all calls to wer.dll and its exports to the original legitimate library. At the entry point, a separate thread combines all the aforementioned 8KB pieces into a complete shellcode and runs it. The same virtual address space, created by a copy of the legitimate WerFault.exe, is used for all this code.



```
pLauncherEP = &Launcher.modBaseAddr[*&Launcher.modBaseAddr[&((Launcher.modBaseAddr)->e_lfanew)->OptionalHeader.AddressOfEntryPoint]];
VirtualProtect(pLauncherEP, 0xCui64, 0x40u, &flOldProtect);
*pLauncherEP = 0xB848;                    // mov rax, jmp rax
*(pLauncherEP + 2) = WaitAndExit;
*(pLauncherEP + 5) = 0xE0FF;
VirtualProtect(pLauncherEP, 0xCui64, flOldProtect, &flOldProtect);
```

To prevent WerFault continuing its error handling process, the DLL patches the launcher's entry point with typical Blackbone trampolines

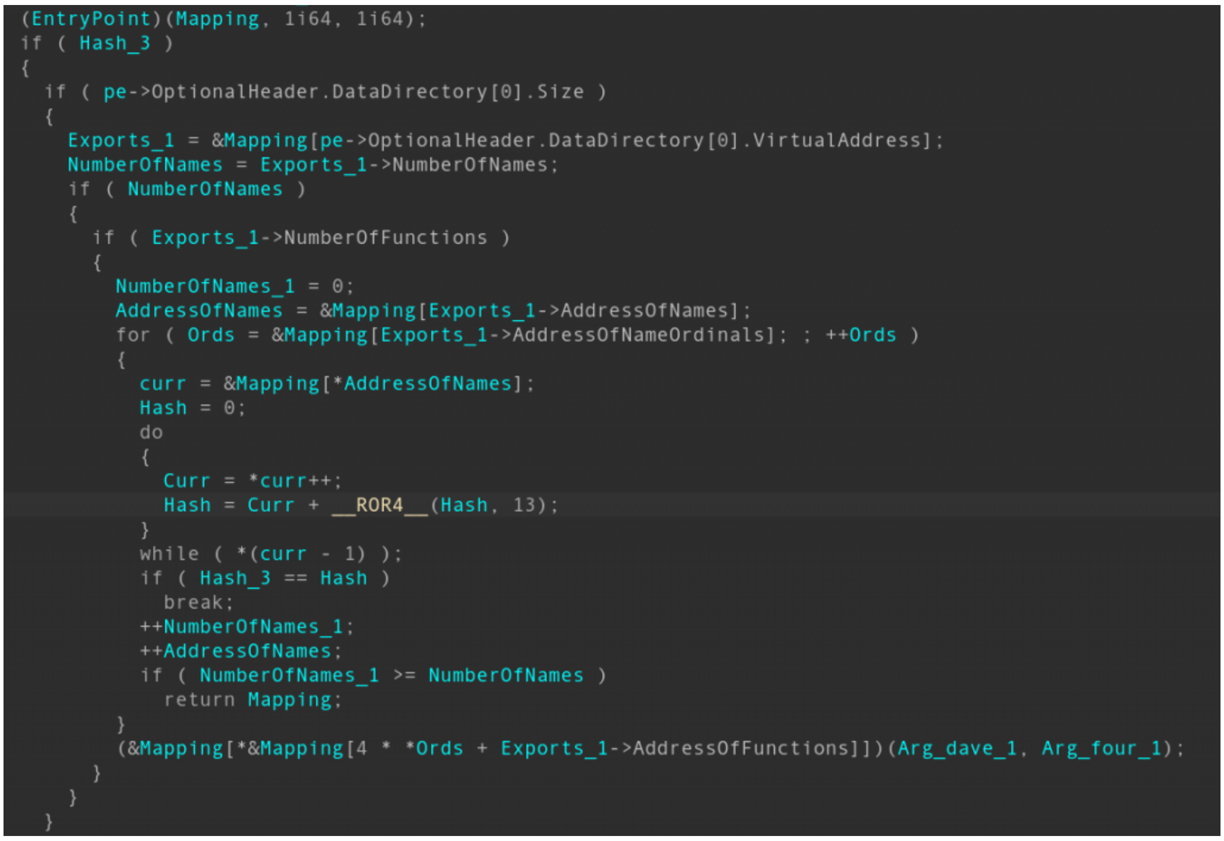
The way to stop the legitimate launcher's execution isn't traditional. In the main thread, wer.dll finds its entry point and patches it with a simple function. WaitAndExit() on the screenshot above would just call WaitForSingleObject() with the log gathering thread id and then exit, meaning no real WerFault.exe error handling code could ever be executed: the spoofed DLL mapped into its address space would block it.

Shellcode into Windows event logs

The launcher transmits control to the very first byte of the gathered shellcode. Here, three arguments for the next function are prepared:

- Address of next stage Trojan. It is also contained within the data extracted from the event logs
- The standard ROR13 hash of exported function name Load inside this Trojan (0xE124D840)
- Addresses of the string "dave" and constant "4", which become the arguments of the exported function, found by hash

The parsing of the next Windows portable executable to locate its entry point is quite typical. To make the next stage Trojan less visible, the actor wiped the “MZ” magic in its header. After calling the code at the Trojan’s entry point, the shellcode also searches for the requested export and invokes it.



```
(EntryPoint)(Mapping, 1i64, 1i64);
if ( Hash_3 )
{
  if ( pe->OptionalHeader.DataDirectory[0].Size )
  {
    Exports_1 = &Mapping[pe->OptionalHeader.DataDirectory[0].VirtualAddress];
    NumberOfNames = Exports_1->NumberOfNames;
    if ( NumberOfNames )
    {
      if ( Exports_1->NumberOfFunctions )
      {
        NumberOfNames_1 = 0;
        AddressOfNames = &Mapping[Exports_1->AddressOfNames];
        for ( Ords = &Mapping[Exports_1->AddressOfNameOrdinals]; ; ++Ords )
        {
          curr = &Mapping[*AddressOfNames];
          Hash = 0;
          do
          {
            Curr = *curr++;
            Hash = Curr + __ROR4__(Hash, 13);
          }
          while ( *(curr - 1) );
          if ( Hash_3 == Hash )
            break;
          ++NumberOfNames_1;
          ++AddressOfNames;
          if ( NumberOfNames_1 >= NumberOfNames )
            return Mapping;
        }
        (&Mapping[*&Mapping[4 * *Ords + Exports_1->AddressOfFunctions]])(Arg_dave_1, Arg_four_1);
      }
    }
  }
```

Besides searching for the entry point and calling it, the shellcode also searches for a Trojan export by hardcoded hash and runs the found function with arguments “dave” and “4”

HTTP Trojan

For last stagers we will be a bit more detailed than for auxiliary modules before. The C++ module obviously used the code from SilentBreak’s (now NetSPI’s) Throwback public repository: XOR-based encryption function, original file name for some samples, e.g., ThrowbackDLL.dll, etc. Let us start here with the aforementioned Load() exported function. It’s just like the patching of WerFault above (the function waits on the main Trojan thread) but it ignores any parameters, so “dave” and “4” are unused. It is possible this launcher supports more modules than just this one, which would require parameters.

Target fingerprinting

The module decrypts C2 domains with a one- byte XOR key. In the case of this sample there is only one domain, eleed[.]online. The Trojan is able to handle many of them, separated by the “|” character and encrypted. For further communications over plain HTTP, the Trojan chooses a random C2 from this set with user agent “Mozilla 5.0”.

The malware generates a fingerprinting string by gathering the following information, also separated by “|”:

- Values of MachineGUID from the SOFTWARE\Microsoft\Cryptography

- Computer name
- Local IP addresses obtained with GetAdaptersInfo
- Architecture (x86 or x64)
- OS version
- Whether the current process has SeDebugPrivilege

The fingerprinter also appends “1.1” to the string (which could be the malware version) and the sleep time from the current config.

Encrypted HTTP communication with C2

Before HTTP communications, the module sends empty (but still encrypted) data in an ICMP packet to check connection, using a hardcoded 32-byte long RC4 key. Like any other strings, this key is encrypted with the Throwback XOR-based algorithm.

If the ping of a control server with port 80 available is successful, the aforementioned fingerprint data is sent to it. In reply, the C2 shares the encrypted command for the Trojan’s main loop.

Trojan commands

Code	Command features
0	Fingerprint the target again.
1	Execute command. The Trojan executes the received command in the new process and sends the result back to the C2.
2	Download from a URL and save to the given path.
3	Set a new sleep time. This time in minutes is used as a timeout if the C2 hasn’t replied with a command to execute yet. Formula for randomization is (random number between 0,9 – 1,1) * sleep time.
4	Sleep the given number of minutes without changing the configuration.
5	List processes with PID, path, owner, name and parent data.
6	Inject and run shellcode into the target process’ address space. To inject into the same process, the command argument should be “local”. Like the shellcode in the event logs, this one would run the provided PE’s entry point and as well as a specific export found by hash.
99	Terminates the session between trojan and C2.

Another Trojan in use during this campaign is named pipe-based and has a more profound command system, including privilege escalation, screenshotting, inactivity time measurement, etc. Here, we come to the infection chain end. We continue with another last stage Trojan type, which we observed injected into processes like edge.exe.

Named pipes-based Trojan

The Trojan location is C:\Windows\apds.dll. The original legitimate Microsoft Help Data Services Module library with the same name is in C:\Windows\System32. The main Trojan working cycle is in a separate thread. The malware also exports a Load() function, whose only purpose is to wait for a working thread, which is typical for this campaign’s modules.

First, the main trojan thread gets the original apds.dll and exports and saves it into an allocated new heap buffer right after the Trojan's image in memory. Then the Trojan edits the self-exported functions data in a way that allows it to call the original apds.dll exports through the crafted stubs like the following, where the address is the one parsed from the real apds.dll:

```
48B8<addr> MOV RAX,<addr>
FFE0 JMP RAX
```

This trampoline code is taken from the Blackbone Windows memory hacking library (RemoteMemory::BuildTrampoline function). DLL hijacking isn't something new, we have seen such a technique used to proxy legitimate functions many times, but recreating self-exports with just short stubs to call the original legitimate functions is unusual. The module then creates a duplex-named pipe, "MonolithPipe", and enters its main loop.

Work cycle

After the aforementioned manipulations with exported functions, the module lightly fingerprints the host with architecture and Windows version information. The Trojan also initializes a random 11-byte ASCII string using the rare constant mentioned, e.g., here in the init_keys function. The result serves as a unique session id.

The malware connects to the hardcoded domain on port 443 (in this case https://opswat[.]info:443) and sends POST requests to submit.php on the C2 side. HTTPS connection options are set to accept self-signed certificates on the server side. The C2 communication in this case is encrypted with an RC4 algorithm with the Dhga(81K1!392-!(43<KakjaiPA8$#ja key. In the case of the named pipes- based Trojan, the common commands are:

Code	Command features
0	Set the "continue" flag to False and stop working.
1	N/A, reserved so far.
2	Get time since the last user input in minutes.
3	Get current process information: PID, architecture, user, path, etc.
4	Get host domain and user account.
5	Impersonate user with credentials provided.
6	Get current process's available privileges.
7	Execute command with the cmd.exe interpreter.
8	Test connection with a given host (address and port) using a raw TCP socket.
9	Get running processes information: path, owner, name, parent, PID, etc.
10	Impersonate user with the token of the process with a provided ID.
11	List files in directory.
12	Take a screenshot.
13	Drop content to file.
14	Read content from file
15	Delete file.
16	Inject provided code into process with the given name.
17	Run shellcode from the C2.
18	N/A, reserved so far.

19	Run PowerShell script. During this campaign we observed Invoke-ReflectivePEInjection to reflectively load Mimikatz in memory and harvest credentials.

We have now covered the three layers of the campaign. Interestingly, we observed a Trojan with a complete command set as in the table above, but still using RC4-encrypted HTTP communications with the C2 instead of named pipes. The last stage samples look like a modular platform, whose capabilities the actor is able to combine according to their current needs.

Infrastructure

Domain	IP	First seen	ASN
eleed[.]online	178.79.176[.]136	Jan 15, 2022	63949 – Linode
eleed[.]cloud	178.79.176[.]136	–	63949 – Linode
timestechnologies[.]org	93.95.228[.]97	Jan 17, 2022	44925 – The 1984
avstats[.]net	93.95.228[.]97	Jan 17, 2022	44925 – The 1984
mannlib[.]com	162.0.224[.]144	Aug 20, 2021	22612 – Namecheap
nagios.dreamvps[.]com	185.145.253[.]62	Jan 17, 2022	213038 – DreamVPS
opswat[.]info	194.195.241[.]46	Jan 11, 2022	63949 – Linode
–	178.79.176[.]1	–	63949 – Linode

Attribution

The code, which we consider custom (Trojans, wrappers), has no similarities with previously known campaigns or previously registered SilentBreak toolset modules. Right now we prefer not to name the activity and instead stick to just “SilentBreak” given it is the most used among the tools here. If new modules appear and allow us to connect the activity to some actor we will update the name accordingly.

Conclusions

We consider the event logs technique, which we haven’t seen before, the most innovative part of this campaign. With at least two commercial products in use, plus several types of last-stage RAT and anti-detection wrappers, the actor behind this campaign is quite capable. There is the possibility that some of the modules we described here as custom ones are part of a commercial toolset as well. The code is quite unique, with no similarities to known malware. We will continue to monitor similar activity.

In the Targeted Malware Reverse Engineering training course, Kaspersky experts share its best and most valuable practices to build a safer world. Learn more about targeted malware with Denis Legezo and other GReAT experts at: https://kas.pr/bgy7

Indicators of Compromise

File Hashes (malicious documents, trojans, emails, decoys)

Dropper
822680649CDEABC781903870B34FB7A7
345A8745E1E3AE576FBCC69D3C8A310B

EF825FECD4E67D5EC5B9666A21FBBA2A
FA5943C673398D834FB328CE9B62AAAD

Logs code launcher
2080A099BDC7AA86DB55BADFFBC71566
0D415973F958AC30CB25BD845319D960
209A4D190DC1F6EC0968578905920641
E81187E1F2E6A2D4D3AD291120A42CE7

HTTP Trojan
ACE22457C868DF82028DB95E5A3B7984
1CEDF339A13B1F7987D485CD80D141B6
24866291D5DEEE783624AB51516A078F
13B5E1654869985F2207D846E4C0DBFD

Named pipes trojan and similar
59A46DB173EA074EC345D4D8734CB89A
0B40033FB7C799536C921B1A1A02129F
603413FC026E4713E7D3EEDAB0DF5D8D

Anti-detection wrappers/decryptors/launchers, not malicious by themselves
42A4913773BBDA4BC9D01D48B4A7642F
9619E13B034F64835F0476D68220A86B
0C0ACC057644B21F6E76DD676D4F2389
16EB7B5060E543237ECA689BDC772148
54271C17684CA60C6CE37EE47B5493FB
77E06B01787B24343F62CF5D5A8F9995
86737F0AE8CF01B395997CD5512B8FC8
964CB389EBF39F240E8C474E200CAAC3
59A46DB173EA074EC345D4D8734CB89A
A5C236982B0F1D26FB741DF9E9925018
D408FF4FDE7870E30804A1D1147EFE7C
DFF3C0D4F6E2C26936B9BD82DB5A1735
E13D963784C544B94D3DB5616E50B8AE
E9766C71159FC2051BBFC48A4639243F
F3DA1E157E3E344788886B3CA29E02BD

Host-based IoCs

C:\Windows\Tasks\wer.dll
C:\Windows\Tasks\WerFault.exe copy of the legit one to sideload the malicious .dll
Named pipe MonolithPipe
Event logs with category 0x4142 in Key Management Service source. Events ID auto increments starting from 1423.

PDB paths

```
C:\Users\admin\source\repos\drx\x64\Release\sb.pdb
C:\Users\admin\source\repos\drx\x64\Release\zOS.pdb
C:\Users\admin\source\repos\drx\x64\Release\ThrowbackDLL.pdb
C:\Users\admin\source\repos\drx\x64\Release\drxDLL.pdb
C:\Users\admin\source\repos\drx\x64\Release\monolithDLL.pdb
```