

THE DUQU 2.0

Technical Details

Version: 2.0 (9 June 2015)

CONTENTS

EXECUTIVE SUMMARY.....	3
INITIAL ATTACK.....	4
LATERAL MOVEMENT.....	4
ANALYSIS OF A DUQU 2.0 MSI PACKAGE.....	7
File properties.....	7
First Layer: ActionDLL (msi.dll).....	10
Second Layer: ActionData0.....	10
Third Layer: klif.dll.....	11
Attacking AVP.EXE.....	12
CTwoPENC.dll zero-day and KMART.dll.....	14
PAYLOAD CONTAINERS AND MIGRATION.....	15
Payload type "L".....	15
Payload run type "G".....	16
Payload run type "I".....	16
Payload run type "K".....	17
Payload run type "Q".....	17
PLATFORM PLUGGINABLE MODULES.....	17
PERSISTENCE MECHANISM.....	33
COMMAND AND CONTROL MECHANISMS.....	33
The "portserv.sys" driver analysis.....	35
SIMILARITIES BETWEEN DUQU AND DUQU 2.0.....	37
VICTIMS OF DUQU 2.0.....	42
ATTRIBUTION.....	43
CONCLUSIONS.....	44
REFERENCES.....	45

EXECUTIVE SUMMARY

Earlier this year, during a security sweep, Kaspersky Lab detected a cyber intrusion affecting several of its internal systems.

Following this finding, we launched a large-scale investigation, which led to the discovery of a new malware platform from one of the most skilled, mysterious and powerful groups in the APT world – Duqu. The Duqu threat actor went dark in 2012 and was believed to have stopped working on this project - until now. Our technical analysis indicates the new round of attacks include an updated version of the infamous ¹2011 Duqu malware, sometimes referred to as the step-brother of ²Stuxnet. We named this new malware and its associated platform “Duqu 2.0”.

Victims of Duqu 2.0 have been found in several places, including western countries, the Middle East and Asia. The actor appears to compromise both final and utilitarian targets, which allow them to improve their cyber capabilities.

Most notably, some of the new 2014-2015 infections are linked to the P5+1 events and venues related to the negotiations with Iran about a nuclear deal. The threat actor behind Duqu appears to have launched attacks at the venues for some of these high level talks. In addition to the P5+1 events, the Duqu 2.0 group has launched a similar attack in relation to the ³70th anniversary event of the liberation of Auschwitz-Birkenau.

In the case of Kaspersky Lab, the attack took advantage of a zero-day (CVE-2015-2360) in the WindowsKernel, patched by Microsoft on June 9 2015 and possibly up to two other, currently patched vulnerabilities, which were zeroday at that time.

1 <https://en.wikipedia.org/wiki/Duqu>

2 http://www.kaspersky.com/about/news/virus/2011/Duqu_The_Step_Brother_of_Stuxnet

3 <http://70.auschwitz.org/index.php?lang=en>

INITIAL ATTACK

The initial attack against Kaspersky Lab began with the targeting of an employee in one of our smaller APAC offices. The original infection vector for Duqu 2.0 is currently unknown, although we suspect spear-phishing e-mails played an important role. This is because for one of the patients zero we identified had their mailbox and web browser history wiped to hide traces of the attack. Since the respective machines were fully patched, we believe a zero-day exploit was used.

In 2011, we were able to identify Duqu attacks that used Word Documents containing an exploit for a zero-day vulnerability (CVE-2011-3402) that relied on a malicious embedded TTF (True Type Font File). This exploit allowed the attackers to jump directly into Kernel mode from a Word Document, a very powerful, extremely rare, technique. A similar technique and zero-day exploit (⁴CVE-2014-4148) appeared again in June 2014, as part of an attack against a prominent international organization. The C&C server used in this 2014 attack as well as other factors have certain similarities with Duqu, however, the malware is different from both Duqu and Duqu 2.0. It is possible that this is a parallel project from the Duqu group and the same zero-day (CVE-2014-4148) might have been used to install Duqu 2.0.

Once the attackers successfully infected one machine, they moved on to the next stage.

LATERAL MOVEMENT

In general, once the attackers gain access into a network, two phases follow:

- Reconnaissance and identification of network topology
- Lateral movement

In the case of Duqu 2.0, the lateral movement technique appears to have taken advantage of another zero-day, (CVE-2014-6324) which was patched in November 2014 with ⁵MS14-068 . This exploit allows an unprivileged domain user to elevate credentials to a domain administrator account. Although we couldn't retrieve a copy of this exploit, the logged events match the Microsoft detection guidance for this attack. Malicious modules were also observed performing a "pass the hash" attack inside the local network, effectively giving the attackers many different ways to do lateral movement.

Once the attackers gained domain administrator privileges, they can use these permissions to infect other computers in the domain.

To infect other computers in the domain, the attackers use few different strategies. In most of the attacks we monitored, they prepare Microsoft Windows Installer Packages (MSI) and then deploy them remotely to other machines. To launch them, the attackers create a service on the target machine with the following command line:

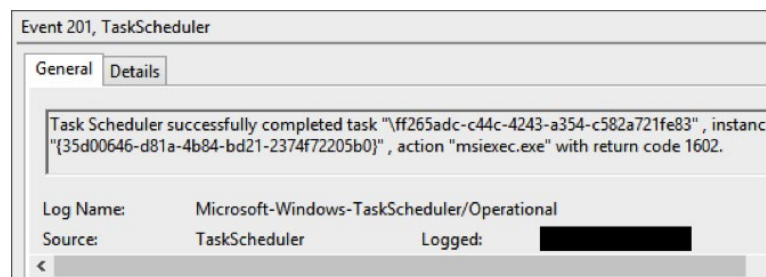
```
msiexec.exe /i "C:\\[...]\\tmp8585e3d6.tmp" /q PROP=9c3c7076-d79f-4c
```

4 <https://www.fireeye.com/blog/threat-research/2014/10/two-targeted-attacks-two-new-zero-days.html>

5 <https://technet.microsoft.com/library/security/MS14-068>

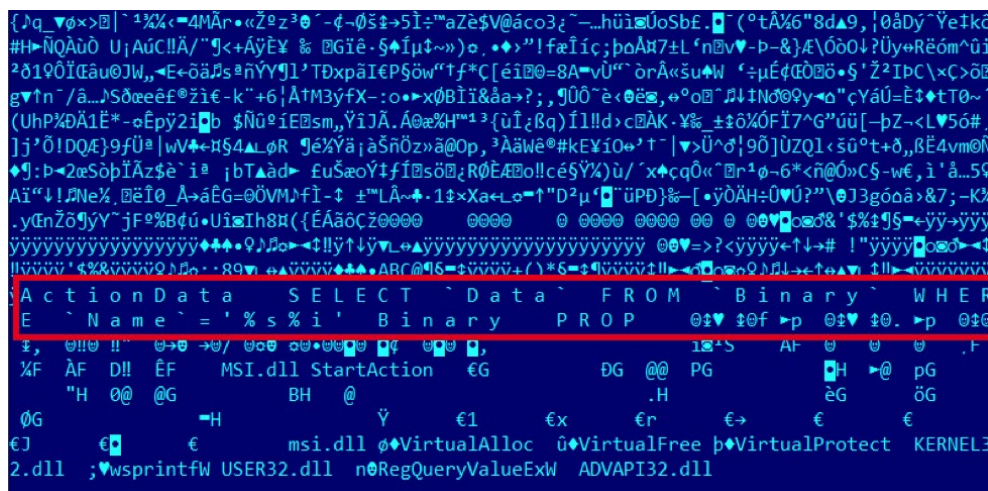
The PROP value above is set to a random 56-bit encryption key that is required to decrypt the main payload from the package. Other known names for this parameter observed in the attacks are "HASHVA" and "CKEY". The folder where the package is deployed can be different from case to case, depending on what the attackers can access on the remote machine.

In addition to creating services to infect other computers in the LAN, attackers can also use the Task Scheduler to start “msiexec.exe” remotely. The usage of Task Scheduler during Duqu infections for lateral movement was also observed with the 2011 version and was described by ⁶Symantec in their technical analysis.



"msiexec.exe" - Task Scheduler trace in the logs

The MSI files used in the attacks contain a malicious stub inside which serves as a loader. The stub loads the other malware resources right from the MSI file and decrypts them, before passing execution to the decrypted code in memory.



Malicious stub with query to load the other resources from the MSI file highlighted.

The encryption algorithms used for these packages differ from case to case. It's important to point out that the attackers were careful enough to implement unique methods, encryption algorithms and names (such as file names) for each attack, as a method to escape detection from security products and limit the ability of an antivirus company to find other infections once one of them has been identified.

So far, we've seen the following encryption algorithms used by the attackers:

- Camellia
- AES

6 http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/w32_dugu_the_precursor_to_the_next_stuxnet.pdf

- XTEA
- RC4
- Different multibyte XOR-based encryption

For compression algorithms, we've seen the following:

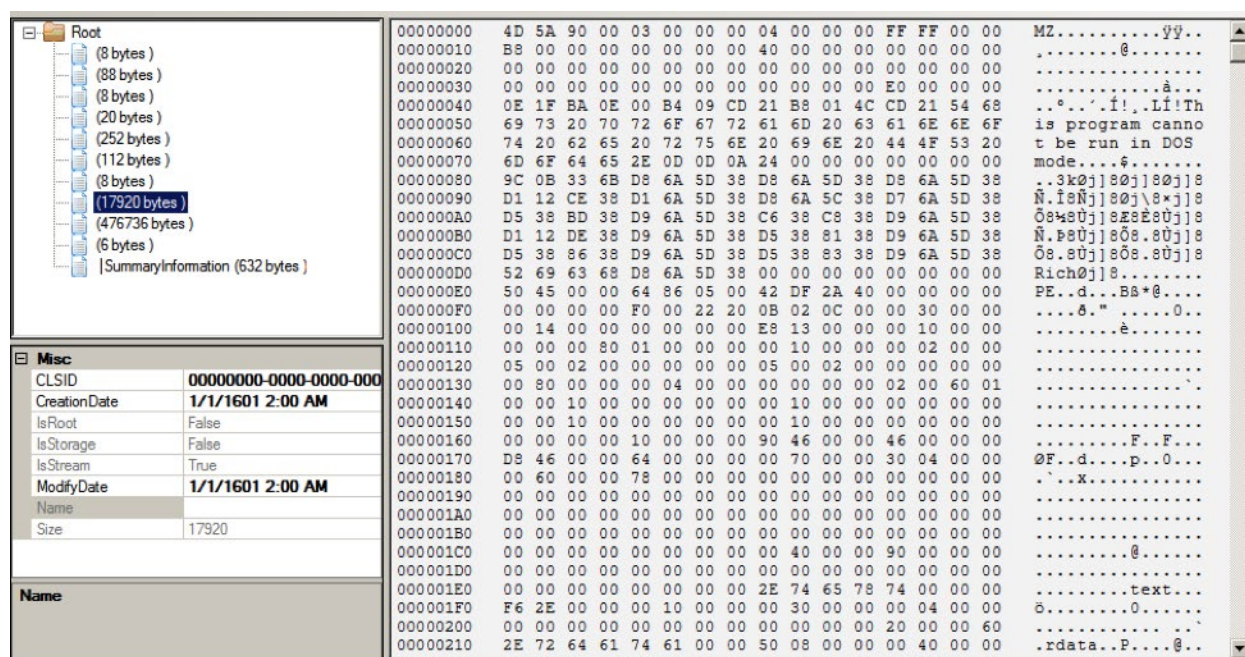
- LZJB
- LZF
- FastLZ
- LZO

In essence, each compiled attack platform uses a unique combination of algorithms that make it very difficult to detect.

The attackers can deploy two types of packages to their victims:

- "Basic", in-memory remote backdoor (~500K)
- Fully featured, C&C-capable, in-memory espionage platform (18MB)

These have similar structures and look like the following:



Malicious Duqu 2.0 MSI package.

In the screenshot above, one can see the loader (ActionDll: 17,920 bytes) and the main payload (ActionData0: 476,736 bytes). Upon execution, ActionDll is loaded and control is passed to its only export, StartAction.

The "basic" in-memory remote backdoor is pushed to computers inside the domain by the Domain Controller on a regular basis – almost like a worm infection. This gives the attackers an entry into most of the machines from the domain and if further access is needed, they can upload a more sophisticated MSI file that deploys tens of different plugins to harvest information.

A thorough description of the malware loading mechanism from the “basic” remove backdoor MSI can be found below.

ANALYSIS OF A DUQU 2.0 MSI PACKAGE

Filename: random / varies from case to case

MD5 (example, can vary): 14712103ddf9f6e77fa5c9a3288bd5ee

Size: 503,296 bytes

File properties

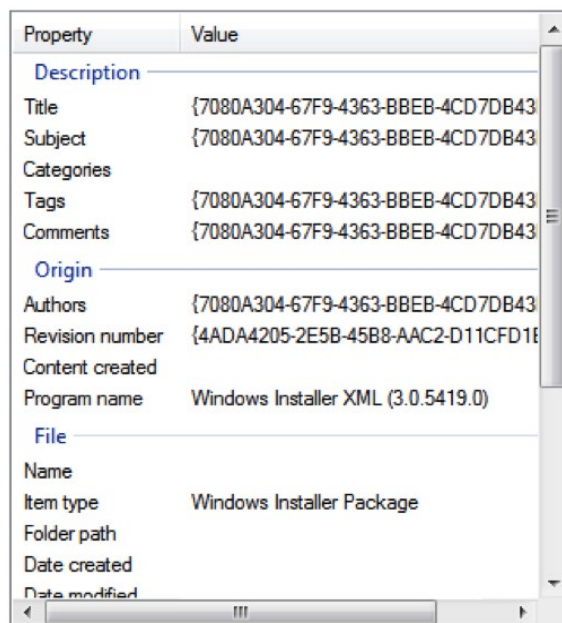
The MSI file has the following general properties:

- Composite Document File V2 Document
- Little Endian
- OS: Windows, Version 6.1
- Code page: 1252
- Title: {7080A304-67F9-4363-BBEB-4CD7DB43E19D} (randomly generated GUIDs)
- Subject: {7080A304-67F9-4363-BBEB-4CD7DB43E19D}
- Author: {7080A304-67F9-4363-BBEB-4CD7DB43E19D}
- Keywords: {7080A304-67F9-4363-BBEB-4CD7DB43E19D}
- Comments: {7080A304-67F9-4363-BBEB-4CD7DB43E19D}
- Template: Intel;1033
- Last Saved By: {7080A304-67F9-4363-BBEB-4CD7DB43E19D}
- Revision Number: {4ADA4205-2E5B-45B8-AAC2-D11CFD1B7266}
- Number of Pages: 100
- Number of Words: 8
- Name of Creating Application: Windows Installer XML (3.0.5419.0)
- Security: 4

It should be noted that MSI files used in other attacks can have different other properties. For example, we observed several other fields:

- Vendor: Microsoft or InstallShield
- Version: 1.0.0.0 or 1.1.2.0 or 2.0.0.0

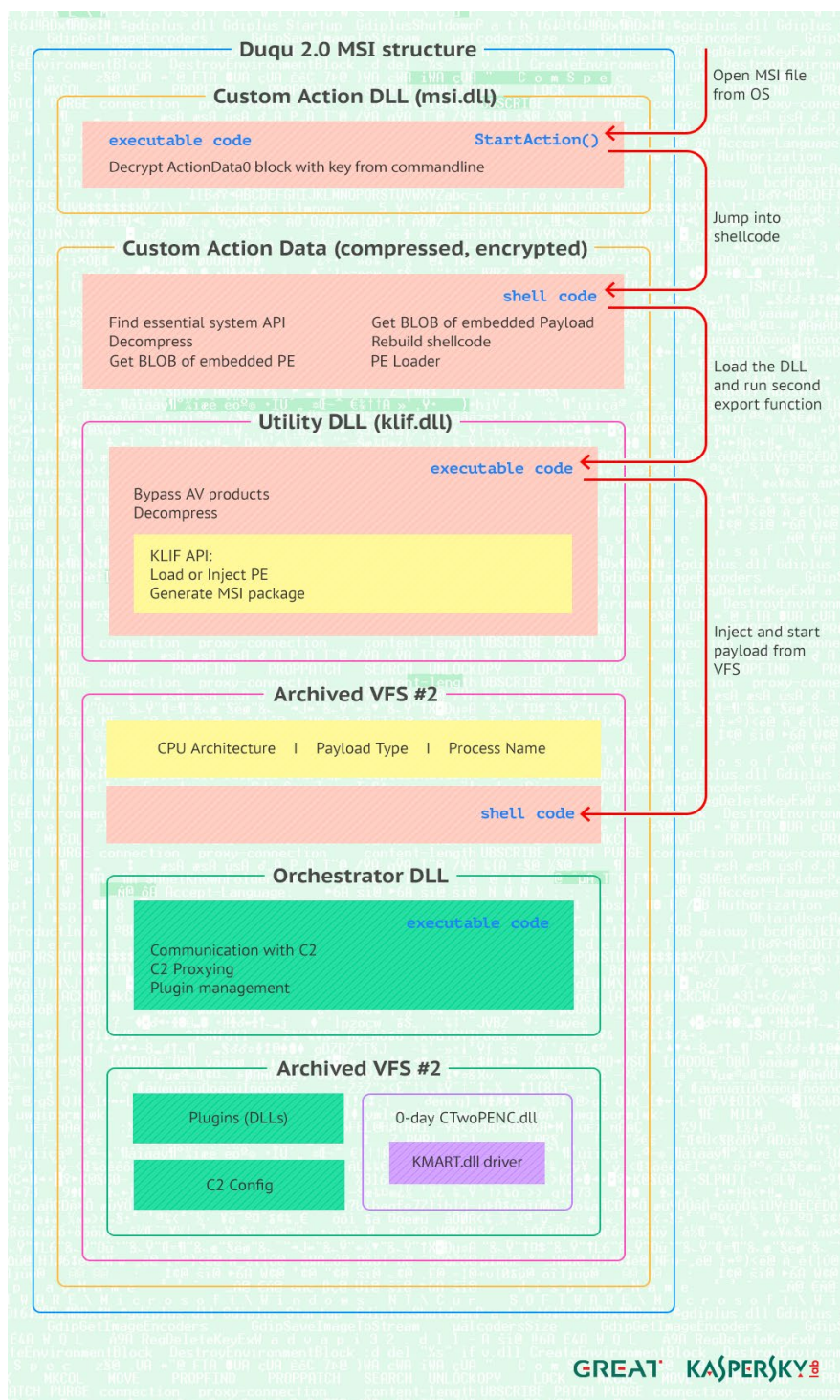
Some of these are visible via the Windows Explorer file properties dialog box:



There are two binary blocks inside this MSI package:

Tables	Name	Data
Binary	ActionDll	[Binary Data]
CustomAction	ActionData0	[Binary Data]
InstallExecuteSequence		
Property		

The first binary, called ActionDLL, is in fact a Windows PE DLL file, while the other one is a Camellia-encrypted and LZJB-compressed data payload (the encryption and compression algorithm vary from case to case). In fact, there are several layers of executable code embedded one into another as compressed or encrypted binary blocks. Here's a look at a Duqu 2.0 MSI package, with all its internal payloads:



We describe these components in more detail below.

First Layer: ActionDLL (msi.dll)

Original filename: msi.dll

MD5: e8eaec1f021a564b82b824af1dbe6c4d

Size: 17'920 bytes

Link time: 2004.02.12 02:04:50 (GMT)

Type: 64-bit PE32+ executable DLL for MS Windows

This DLL has only one export name called StartAction, which is called in the context of msiexec.exe process. When this function is called, it retrieves an MSI property called PROP and uses it as a decryption key for the bundled ActionData0 package:

```

mov     [rsp+hInstall], ecx
sub     rsp, 58h
mov     [rsp+58h+pcchValueBuf], 11h
lea     r9, [rsp+58h+pcchValueBuf] ; pcchValueBuf
lea     r8, [rsp+58h+szValueBuf] ; szValueBuf
lea     rdx, szName ; "PROP"
mov     ecx, [rsp+58h+hInstall] ; hInstall
call    MsiGetPropertyW
test     eax, eax
jz      short loc_180003BD2
xor     eax, eax
jmp     short loc_180003C49

```

Next, the code iterates over 12 possible payloads that have to be decrypted and started. The payloads are part of the MSI and may have the following names: ActionData0, ActionData1, ActionData2, etc.

The package described here contains only one payload named "ActionData0".

Second Layer: ActionData0

```

0000 AppClass      struct ;
0000 dwMagic        dd ? ; 0x72384263
0004 field_4        dd ?
0008 lstrcmplw      dq ?
0010 VirtualQuery   dq ?
0018 RtlAnsiStringToUnicodeString dq ?
0020 field_20       dq ?
0028 VirtualProtect dq ?
0030 VirtualAlloc   dq ?
0038 GetProcAddress dq ?
0040 RtlFreeUnicodeString dq ?
0048 MapViewOfFile  dq ?
0050 FlushInstructionCache dq ?
0058 VirtualFree    dq ?
0060 LdrLoadDll      dq ?
0068 ZwCreateSection dq ?
0070 ZwMapViewOfSection dq ?
0078 ZwUnmapViewOfSection dq ?
0080 FreeLibrary     dq ?
0088 CreateThread    dq ?
0090 WaitForSingleObject dq ?
0098 ZwClose         dq ?
00A0 GetSystemDirectoryW dq ?
00A8 ZwOpenSection   dq ?
00B0 GetExitCodeThread dq ?
00B8 ZwQuerySystemInformation dq ?
00C0 CreateFileW     dq ?
00C8 GetTickCount   dq ?
00D0 GetCurrentProcessId dq ?
00D8 GetCurrentProcess dq ?
00E0 ReadProcessMemory dq ?
00E8 DeviceIoControl dq ?
00F0 GetCurrentThreadId dq ?
00F8 GetModuleHandleW dq ?
0100 LdrUnlockLoaderLock dq ?
0108 LdrLockLoaderLock dq ?
0110 wsprintfW        dq ?

```

This binary chunk contains the main code, in compressed and encrypted format. It represents a composition of executable, position-independent code blocks mixed with embedded data objects. The code seems to be based on a framework and heavily uses helper structures that contain pointers to a set of system APIs and offsets to internal data blocks. Such structures are definitely a trademark of the developer. When they are initialized, one field (usually the first 4 bytes) contains a magic value that identifies the state and type of the structure.

Another trademark of the coder is the way to import system API by module and export name hashes. The hashing algorithm was found all over this and other layers of executable code. It's easily recognizable by two DWORD constants: **0x8A20C27** and **0x67F84FC6**.

Basically, the code in ActionData0 passes execution to an embedded executable, which we will refer by its internal name: "klif.dll". The execution is passed to the second exported function in table of exports of this DLL file. This disregards the export name and relies only on the order of functions in the table of PE export ordinals.

When this export function is called, a next stage helper structure pointer is passed to it, so that it can use some of the values set on the upper layer.

However, before passing execution to klif.dll, the code attempts alternative routes. First, it attempts to find the name of the following format "**api-ms-win-shell-XXXX.dll**", where "X" can be any decimal number. The name is valid if there is no module with such filename loaded into current process. The code attempts to iteratively find such name starting from api-ms-win-shell-0000.dll, api-ms-win-shell-0001.dll, api-ms-win-shell-0002.dll and so on. This may be a dependency to the Duqu platform component that is yet to be discovered.

Right after this, if the name was found, the code attempts to map a section kernel object by name, which is generated using a PRNG-based algorithm. The name of the section has the following template: "**\BaseNamedObjects\{XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXX}**", where "X" is any hexadecimal digit that is generated based on current system boot time. So far, the name of the section is "machine/boot time" dependent, which makes it unique but allows other processes or modules to locate such section if they use the same name generation algorithm. This section is accessed in different other parts of the code and modules. Let's refer to this section as OSBoot-section from now. Once the section name is generated the code tries to open such section and, if it is found, it takes some values from it and attempts to open a specific device and issue a number of IOCTL codes to the driver. The name of the driver device as well as IOCTL codes are located inside a section of the kernel mode driver KMART.dll that is described below.

The code developer has a preference for using sections to any other ways to access data. Another use of sections appears to be in mapping the part of code/data where klif.dll is embedded and then finding that section using a hardcoded magic QWORD number: **0xA1B5F8FC0C2E1064**. Once the section is found in address space of current process the code attempts to pass execution to it. This alternative execution route is not applicable to current MSI file package but simply exists in the code probably due to common code template used for building current MSI package. It may also be an indicator of another Duqu platform component that wasn't used in the attacks that we observed.

Third Layer: klif.dll

Original filename: klif.dll

MD5: 3fde1bbf3330e0bd0952077a390cef72

Size: 196'096 bytes

Link time: 2014.07.06 08:36:50 (GMT)

Type: 64-bit PE32+ executable DLL for MS Windows

Apparently, this file attempts to mimic some of legitimate names of Kaspersky Lab product components: "klif.sys". Although there is neither similarity in code nor in file information, the module uses Kaspersky Lab acronym in its export names: KLInit and KLDone.

When this DLL is loaded into a new process, it simply initializes internal structures, such as those providing pointers to the required system API.

The real payload of this module is located in the KLDone export function, which is second in the list of the export table. This export function is called from the previous code layer.

First, it makes sure that global application structure is initialized with essential functions from ntdll.dll, kernel32.dll and user32.dll. System API functions are imported using hashes of export names. The hashing algorithm is identical to the one described one layer above and uses the same magic constants: **0x8A20C27** and **0x67F84FC6**.

Next the code iterates through the list of running processes and hashes lowercase name of each process. The hash is compared to a hardcoded value of 0x3E3021CB, which is a hash for the **"avp.exe"** string.

Attacking AVP.EXE

If the "avp.exe" process is running, the module attempts to open the OSBoot-section as described before and tries to attack the avp.exe process. The attack starts from identifying the exact path to the installed Kaspersky Lab product by iterating through an array of hardcoded registry keys and values for the following products:

KES12	AVP15	AVP10	AVP8
KES11	AVP14.0.0	KES9	AVP7
KES10	AVP14	KES8	AVP6
AVP16.0.0	AVP13	AVP80	
AVP16	AVP12	AVP90	
AVP15.0.0	AVP11	AVP9	

The registry values queried by the module contain a filesystem path to the root directory where the corresponding product is installed. For example: **"C:\Program Files\Kaspersky Lab\Kaspersky Internet Security 15.0.0\"**

Once the registry key and value is found, the module confirms that avp.exe file is located in that directory. It does some basic file availability checks as well: makes sure that environment variables are resolved and the file can be opened for reading and it begins with 0x5A4D (magic "MZ" value of Windows executables).

After that, the module creates a new section and maps avp.exe as a file view to this section. To allow code execution, the attributes of this memory region are also changed accordingly. This allows the module to change bytes from the mapped avp.exe in memory. The module applies two patches to the mapped avp.exe in a quite unusual way using SSE2 CPU extensions. In fact, it patches just the old DOS PE header (less than 120 bytes). The patches are simple callback mechanisms that call arbitrary function passed as an argument.

Right after this, the module attempts to start a new thread that begins with one of the patched locations. So far, it looks like the new thread begins with a call originating from avp.exe module that does some other calls, i.e. jumping back to klif.dll. Apparently, this

attack was introduced to trick the avp.exe process into believing that further calls will be safe and trusted as the root of the call stack is coming from the legitimate avp.exe module. This is what we see further down in the code: the new thread instantly jumps from avp.exe back to klif.dll and tries to communicate with the Kaspersky Lab product minifilter driver, known as klif.sys.

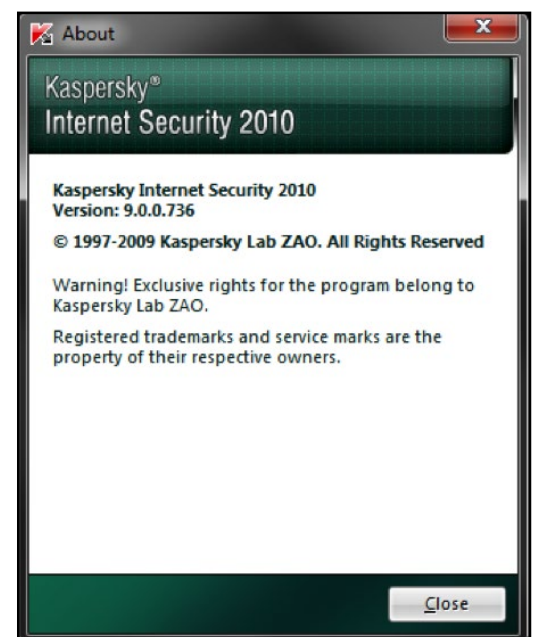
In fact, just before communicating with klif.sys, it opens OSBoot-section and gets an IOCTL code as well as the driver device name to send this IOCTL code to. The section and corresponding driver (KMART.dll) provide certain support to the current module. The code hashes current process name and verifies if it falls down into one of three hashes: **0x3E3021CB (avp.exe), 0xDE6D4DA0 (msiexec.exe), 0xB915B2D8 (rundll32.exe).**

If current process name hash is recognized, the module communicates with klif.sys by opening \KlifComm minifilter communication port and sending series of driver communication messages. According to analysis of the messages, this technique makes process or thread operations “invisible” to the klif interceptor. Such registered entity is considered to be trusted and its activity is removed from AV-scanning, process monitoring, firewall and other defense engines that are subscribed to intercepted events. In addition, the module requests support of the self-defense feature of Kaspersky Lab product, which is normally used to protect the software from aggressive malware which kills the security software processes using a number of techniques available from the OS. This of course guarantees that even a user with administrative privileges cannot stop such process.

Considering that this seemed to be an attack against Kaspersky Lab products, we did some additional tests of the products and found that current products verify the caller process by checking its custom digital signature. So far, without additional driver support, this technique should fail. Verification of the digital signature of the process that opened \KlifComm minifilter communication port was implemented in all Kaspersky Lab products since 2010. So far, this could affect only older products such as KIS2010, which was released by Kaspersky Lab in 2009 ►

It doesn't look realistic now that the attackers started implementing tricks against Kaspersky Lab products in 2009 or earlier. So we looked for another rational explanation and seem to have found it.

Such an attack doesn't normally work against our products because they verify that the caller process is legitimate by checking its custom digital signature. To bypass this, the Duqu 2.0 component named “KMART.dll” patches “klif.sys” in memory to bypass this check. The attack works because the attacker's “KMART.dll” is already running in kernel mode due to a vulnerability in the Windows kernel.



After sending the codes, the module proceeds to the next stage, which is process migration, described further below.

CTwoPENC.dll zero-day and KMART.dll

The third layer klif.dll performs a multitude of functions in order to ensure the survival of the malware in memory and bypass antivirus detections.

One important step is to get kernel level access. On 64-bit systems, one cannot simply load and run kernel mode code without a signed driver. While other attackers such as Equation or Turla chose to piggyback on third-party signed drivers, the Duqu 2.0 platform relies on a much more cunning trick.

One of the payloads bundled together with "klif.dll" is called "CTwoPENC.dll". This is a Windows kernel mode exploit (CVE-2015-2360) that allows them to run code with the highest privileges in the system. We recovered several versions of "CTwoPENC.dll", both for 32-bit and 64-bit versions of Windows, with the following compilation timestamps:

- 2014.08.25 01:20:04 (GMT)
- 2014.08.25 01:19:03 (GMT)
- 2014.07.06 09:17:03 (GMT)

Unlike other Duqu 2.0 modules, these timestamps appear to be legitimate. The reason for this remains unknown – perhaps the Duqu platform developers got this module from somebody else and forgot to patch its compilation timestamp.

"CTwoPENC.DLL" exploits a zero-day vulnerability in "win32k.sys" to gain kernel privileges while being run as an unprivileged user. It creates several windows with classes named "CPer", "Zero", "CTwo", "Vero" in several threads and manipulates the callback pointers.

```

v0 = GetProcessHeap();
v29 = HeapAlloc(v0, 8u, 0x8000u);
if ( v29 )
{
    v1 = GetProcessHeap();
    lpMem = HeapAlloc(v1, 8u, 0x4000u);
    if ( lpMem )
    {
        WndClass.lpfWndProc = DefWindowProcA;
        WndClass.lpszClassName = "CPer";
        if ( RegisterClassA(&WndClass) )
        {
            hWndNewParent = CreateWindowExA(0, "CPer", 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0);
            if ( hWndNewParent )
            {
                WndClass.lpszClassName = &ClassName;
                v9 = 0;
                do
                {
                    v25 = v9 + 1;
                    sub_100037FE(&ClassName, "^%d%s%d^", v9);
                    v10 = RegisterClassA(&WndClass);
                    *((_WORD *)lpMem + v9) = v10;
                    if ( !v10 )
                        break;
                    v11 = CreateWindowExA(0, &ClassName, &ClassName, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0);
                    v12 = (char *)v29 + 4 * v9;
                    *((_DWORD *)v12) = v11;
                    if ( !v11 || sub_100035AB(v11, &hWnd, 0) )
                        break;
                    v14 = *((_DWORD *)v12);
                    if ( Check64(v13) )
                }
            }
        }
    }
}

```

Part of the exploit code that registers window classes and creates new windows.

The main purpose of the exploit is to load a driver named "KMART.dll". If the exploit succeeds, it decompresses "KMART.dll" from the module's body and passes control

to it while running in kernel mode. Both 32-bit and 64-bit versions of the "KMART.DLL" exist. This driver implements a kernel-mode attack against the Kaspersky Lab products filter "klif.sys". It patches the import table addresses of system API functions PsGetCurrentProcessId, PsLookupProcessByProcessId and KlGetStringRef export of Kaspersky Lab's anti-virus driver klflt.sys. The intercepted functions are preceded by malicious code that impersonates the other malicious modules as coming from the process/file "avp.exe" (using the actual location of the anti-virus component), thus evading monitoring by the anti-virus software.

During testing, the zero-day exploit worked on all contemporary Windows versions (XP, Vista, 7, 8, 8.1 both 32-bit and 64-bit). This vulnerability has been patched by Microsoft on June 9, 2015."

PAYLOAD CONTAINERS AND MIGRATION

The module supports a number for different payloads provided in the form of containers following klif.dll in memory (located in PE overlay).

A payload container contains a short header describing target CPU architecture, payload type and target process name or hash. When the module uses some embedded code or file there is normally at least two variants of code bundled inside: 32 and 64-bit. Each payload run type is identified by a single character. Current module supports the following 5 payload run types: **g, i, k, l, q**. Seems like the developer used ordered alphabet characters, which means that there could be other payload run types which were removed from the code at some time. Currently the container located in the overlay of subject executable contains a "l" character as a payload run type, so let's describe it first.

Payload type "L"

First, the code looks for privileges from other processes. The process handle can be hardcoded or obtained by looking for currently running processes, hashing their names and comparing them to one of known hashes. The module includes the following hardcoded process hashes:

0x31283314 (services.exe), 0x1026D1BA (lsass.exe), 0x3BD2498C (svchost.exe).

If any of the recognized processes are found, the module opens process token and verifies that the following privileges are available:

- SeDebugPrivilege
- SeTcbPrivilege
- SeAssignPrimaryTokenPrivilege
- SeIncreaseQuotaPrivilege
- SeImpersonatePrivilege
- SeRestorePrivilege

If they are available the module duplicates process token and saves it into an internal structure for future use.

The module looks for one of the available security suites installed in current system and some generic processes if nothing was found. It attempts to find executable files of the following software:

- More than 20 different Kaspersky Lab products
- McAfee VSCore, SystemCore, HIP
- Symantec Internet Security
- Bitdefender
- ESET
- H+BEDV
- TrendMicro Network Security Component
- Rising AV
- Avast
- AVG 2012/2013/2014/2015
- F-Secure
- Qihoo 360 Security
- Generic Windows Processes:
 - %ProgramFiles(x86)%\Windows Defender\MsMpEng.exe
 - %SystemRoot%\syswow64\svchost.exe
 - %SystemRoot%\system32\lsass.exe
 - %SystemRoot%\system32\svchost.exe

When such an executable is found and eventually it should (at least svchost or lsass.exe should be present), it runs a new process from discovered location and replaces its executable code with own embedded into current executable in encrypted and compressed form.

If the code manages to find a security token from one of the processes with required security privileges, it will try to use this token when creating a new process using CreateProcessAsUser API. Otherwise it simply spawns new process instance.

The new process is started in suspended and debugged state. Just before starting the new process it disables DEP for the child processes to simplify code injection and data manipulation. So far, from the outside, it may look like a security software was started, while from the inside its code is completely different from legitimate module.

Payload run type "G"

This payload is almost identical to payload L described above, but the loader is simpler - it skips hijacking of privileged token. Code injection target is selected automatically depending on the availability of known processes.

Payload run type "I"

This payload is similar to payload G, but makes the loader parse payload header a bit differently: instead of process name string it takes only 4 bytes, which are supposedly used as a process hash. This seems to be a way to hide target process name and use only hash, however this hasn't been implemented completely in the module we analyzed.

Payload run type “K”

This payload is designed to run within the context of the current process. The code simply copies the code to be executed into separate memory and runs it in a dedicated thread. It blocks until thread finishes its execution.

Payload run type “Q”

This payload is identical to payload K described above but it doesn't block execution when a new thread is started. So far, the new code runs asynchronously.

After the payload container is opened and code migrated to another process, which can be elevated and protected from security software, the real malicious code is activated. In most cases, it is simple named pipe based backdoor that listens for incoming communications from the orchestrator. In rare cases, on selected machines, it can be heavy orchestrator module that communicates with command and control server, works as a bidirectional proxy and comes with a large bundle of secondary plugins.

PLATFORM PLUGGINABLE MODULES

In addition to the basic remote backdoor, the attackers deploy more sophisticated packages to domain controllers and to the victims of interest inside the LAN. These MSI packages can contain tens of different modules designed for various cyberespionage functions.

The fully featured packages are much larger than the basic remote backdoor – 18MB vs 500KB. They follow the same structure, with ActionDll and the loader mechanism, except they contain a lot more plugins to load and run.

During our analysis, we identified more than 100 variants of such plugins. A description of these plugins follows. To separate them, we used a virtual identifier based on the first two bytes of their MD5 sum.

03B7 – The main module of Duqu 2.0, orchestrator.

Implements multiple protocol handlers for C&C communication, can start an intermediate C&C proxy server with a self-signed HTTPS certificate. Starts the plugin framework, loads and manages all additional plugins.

It works via HTTP, HTTPS, SMB network pipes or direct TCP connection using a custom, encrypted protocol. Interaction via HTTP is concealed in JPEG or GIF files, similar to the 2011 version of Duqu. Request names, URLs and User-Agent strings may vary between attacks.

Additional known variants: 3026, 4F11.

0682 – Collects basic system information:

- List of running processes
- Active desktop and terminal sessions

Collected information is then transmitted to a named pipe provided by the caller.

Additional known variants: COB7

073C – Implements a **complete Windows socket-based transport**, both client and server side. Provides a class factory for the class that encapsulates various networking functions.

0872 – MSI CustomAction library that is activated when the malicious installer package is started by the Windows Installer. Loads the encrypted binary blob that contains actual malicious payload, decrypts and then executes it in memory.

The names in version information vary: **svcmsi_32.dll**, **msi3_32.dll**, **MSI.dll**, **msi4_32.dll**.

Encryption algorithms also vary: **Camellia 256**, **AES**, **XXTEA**.

The decryption key is extracted from an MSI parameter, possible names: **PROP**, **HASHVA**, **CKEY**.

The encrypted blob is searched by prefixes (can vary): **ActionData**, **CryptHashs**, **CADData**.

Both 32-bit and 64-bit versions are known. Additional known variants: 8D7C, 16EF, E6E5, 434C, 44BD, F708.

09A0 – 64-bit, **Exfiltrates file contents**, particularly searching for files matching these rules:

- *.inuse, *.html
- filename contains "data.hmi" or "val.dat"
- files from the /Int/HMI/ or /LG/HM/ folders.

```

Hml:                align 10h                                ; DATA XREF: sub_18001AC58+75010
                    unicode 0, <html>,0                      ; DATA XREF: sub_18001BF70:loc_18001C02110
Data_hmi:            unicode 0, <data.hmi>,0
IO+                  align 10h
Val_dat:             unicode 0, <val.dat>,0                  ; DATA XREF: sub_18001C5BC+1E10
IO+                  unicode 0, <val.dat>,0
IntHmi:              ; DATA XREF: sub_18001BF70+1F10
                    ; sub_18001C1F8+1E10
                    ; sub_18001C1F8:loc_18001C25110
IO+                  unicode 0, </Int/HMI/>,0
                    db 'L',27h,9,0
LgHm:                ; DATA XREF: sub_18001AC58+76410
IO+                  unicode 0, </LG/HM/>,0
qword_18003F988      dq 19DB1DED53E8000h                    ; DATA XREF: sub_180023DA0+13610
dword_18003F990      dd 1                                     ; DATA XREF: sub_180023DA0+1A310
                    align 8

```

File and directory names of interest for the 09A0 plugin.

Additional known variants: 8858

0AB8 – Provides 25 **functions for manipulating files and directories**:

- List files in directories
- Upload and download arbitrary files
- Read/write file contents

In several cases, the modules are looking specifically for directories named "\\int", "\\lg", "\\of\\md", "\\tl", "\\ak" and files with extensions ".part", ".manual", ".inuse".

```

AllDirs      db 'allDirs',0          ; DATA XREF: sub_10003A3A+32fo
Downloadable db 'Downloadable',0      ; DATA XREF: sub_10003C43+17fo
              align 10h
Dlg          db 'dLg',0              ; DATA XREF: sub_1000409E+18fo
Explorer_0   db 'Explorer',0         ; DATA XREF: sub_10004329+2Dfo
              align 10h
asc_10017210 db '\\',0              ; DATA XREF: sub_1000455A+18fo sub_100
              ; sub_10007882+77fo sub_10007882+88fo
              ; sub_10008FE8+18fo
              align 10h
              dd offset Int          ; "\\int"
              dd offset Lg          ; "\\lg"
              dd offset Tl          ; "\\tl"
              dd offset Ak          ; "\\ak"
              dd offset OfMd        ; "\\of\\md"
              dd offset Int          ; "\\int"

```

File and directory names of interest for the OAB8 plugin.

Additional known variants: A69E.

0B97 – Network transport. Implements API for connecting sockets and pipes provided by the caller. Additional variant: 56A2.

152B – Network and domain discovery.

- Enumerates all servers in the domain
- Tries to connect to remote registries and enumerate all users in "HKEY_USERS". Usernames are then converted to SIDs
- Enumerates all visible network shares

Additional known variants: A987

1C71 – In-memory storage. Receives and stores string data in file mappings.

2125 – Network infection module. Tries to acquire administrative credentials from the running processes and then connect to a target machine using Windows shares. The machine is infected with a malicious DLL backdoor, the target directory varies and may be in UPDROOT\$, ADMIN\$ or C\$. Uses custom SMB packets to identify the target OS version. The target malicious DLL is then started using a new service created with a name "**svcmSI_32@**". The module communicates with the target backdoor via Windows pipes. Additional variant: **DB65**. Instead of the malicious DLL, it uploads a new MSI package and then relies on **MSIEXEC.EXE** to start the MSI via a newly created service named "**msisvc_32@**", the target MSI name is generated randomly using a template "**tmp%x.tmp**".

```

A service was installed in the system.

Service Name: svcmSI_32@
Service File Name: msiexec /i %systemroot%\temp\tmp129deb0.tmp /q PROP=fa5d78ba-f17
Service Type: user mode service
Service Start Type: demand start
Service Account: LocalSystem

```

Example of a Windows event log (System) entry created at the moment of infection.

Administrative credentials are acquired by stealing a token from any running process that was started by a logged on user having domain administrator's rights. For that, it first

queries for the first 100 administrative usernames in the current domain directly from the domain controller.

```

; unicode 0, <\\%s>,0
align 4
; CHAR netname[]
netname:
; DATA XREF: sub_1000A90A+84f0 SelectConnectShare+381f0
; SelectConnectShare:loc_1000AEB4f0
; SelectConnectShare+47Af0
; unicode 0, <UPDR00T$>,0
align 10h
SAdminSystem32Msiexec_exe:
; DATA XREF: CheckAccessToMsiexec+59f0
; unicode 0, <%s\ADMIN$\system32\msiexec.exe>,0
align 10h
SAdminSyswow64Msiexec_exe:
; DATA XREF: CheckAccessToMsiexec+68f0
; unicode 0, <%s\ADMIN$\syswow64\msiexec.exe>,0
align 10h
; CHAR Admin[]
Admin:
; DATA XREF: SelectConnectShare+EFf0
; unicode 0, <ADMIN$>,0
align 10h
; char SC[]
SC:
; DATA XREF: SelectConnectShare+175f0
; unicode 0, <\\%s\%c$>,0
align 4
; char C_0[]
C_0:
; DATA XREF: SelectConnectShare+189f0
; unicode 0, <%c$>,0
; char C_1[]
C_1:
; DATA XREF: SelectConnectShare+1A3f0
; unicode 0, <%c:\>,0
align 4
dword_1000EA58 dd 0
; DATA XREF: SelectConnectShare+39Ef0
; char asc_1000EA5C[]
asc_1000EA5C:
; DATA XREF: SelectConnectShare+432f0
; NetUseAddMisc+2Df0 DoNetUseEnum+3Af0
; unicode 0, <\\%s\%s>,0
; char SIpc[]
SIpc:
; DATA XREF: NetUseAddIPC+36f0
; unicode 0, <\\%s\IPC$>,0
; wchar t Src

```

Remote locations used by the network infection module.

24B7 – Remote desktop administration. Acquires the screenshots of the desktop but also can send input to the desktop, move the mouse cursor. Both 32-bit and 64-bit variants exist. Additional known variants: 65BE, 7795, BDC8, DEE2

26D6 – Detects running network sniffers (**wireshark**, **tcpview**, **netstat**, **dumpcap**, **perfmon**); implements a **TCP server** and communicates via **network pipes**. Internal name: "P.dll".

2829 – Collects system information

- Monitors USB device attachment
- Collects USB drive history
- Enumerates network shares, windows captions, system routing tables
- Reads/writes encrypted files

2913 - WMI data collection

- Enumerate profiles with SIDs
- For each profile, extracts information from lnk files in the profile directory
- Enumerate processes via WMI (CIMV2), including terminated processes
- Extracts user information from available remote registries

Additional known variant: C776

29D4 - Service **msisvc_32@**; **DLL backdoor** that is used for network infection by module **2125**. Accepts commands via named pipe "Global\{B54E3268-DE1E-4c1e-A667-2596751403AD}". Both 32-bit and 64-bit variants exists.

Additional known variants: 6F92, A505, D242

2B46 – Extensive collection of system and user information

- Domain controller's name
- List of users in the domain
- Administrators of the domain
- Enumerates domain trusts
- TCP tables
- UDP tables
- SNMP discovery (OS, parse all replies)
- USB drive history, mounted devices
- Installed programs
- Time zone
- OS install date
- **ODBC.ini**, **SQL Server** instance info, **Oracle** ALL_HOMES, **SyBase**, **DB2**, **MS SQL**, **MySQL** last connections
- DHCP/routing
- Network profiles
- Zero Config parameters
- Connected printers
- MRU list for **WinRAR**, **WinZip**, **Office**, **IE** typed URLs, mapped network drives, **Visual Studio** MRU
- Terminal Service Client default username hint
- User Assist history
- **PuTTY** host keys and sessions
- Logged on users
- Network adapter configuration
- **VNC** clients passwords
- Scan the network and identify OS using SMB packet

```

Hostname:                                ; DATA XREF: sub_10008AF1+16f0
        unicode 0, <HostName>,0
        align 4
Logfilename:                             ; DATA XREF: sub_10008AF1+27f0
        unicode 0, <LogFileName>,0
Portnumber:                              ; DATA XREF: sub_10008AF1+35f0
        unicode 0, <PortNumber>,0
        align 4
Portforwardings:                         ; DATA XREF: sub_10008AF1+4Af0
        unicode 0, <PortForwardings>,0
SSUSS:                                   ; DATA XREF: sub_10008AF1+CCf0
        unicode 0, <%s - %s:%u [ %s %s ]>,0
        align 4
SoftwareSimonTathamPuttySshhostkeys:    ; DATA XREF: sub_10008C2B+59Cf0
        unicode 0, <Software\SimonTatham\Putty\SshHostKeys>,0
        align 4
SoftwareSimonTathamPuttySessions:        ; DATA XREF: sub_10008C2B+5BAf0
        unicode 0, <Software\SimonTatham\Putty\Sessions>,0
SoftwareMicrosoftWindowsCurrentVersionExplorerComdlg32File: ; DATA XREF: sub_10008C2B+45f0
        unicode 0, <Software\Microsoft\Windows\CurrentVersion\Explorer\ComDlg>
        unicode 0, <32\filemru>,0
SoftwareMicrosoftWindowsCurrentVersionExplorerComdlg32Open: ; DATA XREF: sub_10008C2B+8Af0
        unicode 0, <Software\Microsoft\Windows\CurrentVersion\Explorer\ComDlg>
        unicode 0, <32\OpenSavePidlMRU>,0
SoftwareMicrosoftVisualstudio9_0FilemruList: ; DATA XREF: sub_10008C2B+D4f0
        unicode 0, <Software\Microsoft\VisualStudio\9.0\FileMRUList>,0
SoftwareMicrosoftVisualstudio9_0ProjectmruList: ; DATA XREF: sub_10008C2B+11Ef0
        unicode 0, <Software\Microsoft\VisualStudio\9.0\ProjectMRUList>,0

```

Some of the registry locations harvested by the module.

Additional known variants: A7F8

2BF7 - Utility DLL. Provides basic API for creating new MSI packages, loading and injecting arbitrary PE modules. Also responsible for loading the first level of the VFS inside the malicious MSI files. Both 32-bit and 64-bit versions exist.

Known names: "ntdll.dll", "klif.dll", "apiset.dll".

Additional known variants: 6DA1, 32DB, 8304, 9931, 9E60, A2D4, ABA9, B3BB, DC5F, DD32, F7BB

3395 – MS SQL discovery module. Module can send ARP packets to network and discover MS SQL Server ports. Additional functions are responsible for connecting and reading of remote registry contents.

35E9 – File system discovery.

- Enumerate network shares
- Enumerate local disks
- Traverse files system hierarchy and enumerate files; identify repase points

3F45 – Pipe backdoor. Opens a new globally visible named Windows pipe, receives and executes encrypted commands. The "magic" string that identifies the encrypted protocol is "ttttttt".

- Enumerates running processes
- Loads and executes arbitrary PE files

Both 32-bit and 64-bit versions exist.

Known pipe names:

- \\.\pipe\{AAFFC4F0-E04B-4C7C-B40A-B45DE971E81E} \\.\pipe\{AB6172ED-8105-4996-9D2A-597B5F827501}
- \\.\pipe\{0710880F-3A55-4A2D-AA67-1123384FD859} \\.\pipe\{6C51A4DB-E3DE-4FEB-86A4-32F7F8E73B99}
- \\.\pipe\{7F9BCFC0-B36B-45EC-B377-D88597BE5D78}, \\.\pipe\{57D2DE92-CE17-4A57-BFD7-CD3C6E965C6A}

Additional known variants: 6364, 3F8B, 5926, A90A, DDF0, A717, A36F, 8816, E85E, E927

4160 - Password stealer

- Extracts Google Chrome and Firefox login data
- LSA credentials

```

Localappdata:      align 4                ; DATA XREF: sub_1000401A+2Eio
0+                unicode 0, <%localappdata%>,0
                  align 4
Local:             ; DATA XREF: sub_1000401A:loc_10004087io
0                unicode 0, <local>,0
                  align 8
SGoogleChromeUserDataDefaultLoginData: ; DATA XREF: sub_1000401A+BDio
0+                unicode 0, <%s\Google\Chrome\User Data\Default\Login Data>,0
E+SelectUsername_valuePassword_valueOrigin_urlFromLogins db 'SELECT username_value,password_value,origin_url FROM logins',0
1+                ; DATA XREF: sub_10004158+59io
; _MEDIA_TYPE Unknown
Unknown:           ; DATA XREF: sub_10004511:loc_100045A9io
0+                dw 3Ch
                  unicode 0, <Unknown>
                  db 0

```

Data used to locate Chrome saved logins.

Additional known variants: B656

41E2 – Password stealer. 64-bit module. Extracts:

- IE IntelliForms history
- POP3/HTTP/IMAP passwords
- TightVNC, RealVNC, WinVNC3/4 passwords
- Outlook settings
- SAM, LSASS cache
- Windows Live, .Net Passport passwords

```

; CHAR Credenumeratw[]
+Credenumeratw db 'CredEnumerateW',0 ; DATA XREF: sub_BD6588+2Eio
                align 10h
; CHAR Credfree[]
Credfree       db 'CredFree',0 ; DATA XREF: sub_BD6588+3Eio
                align 20h
Microsoft_wininet: ; DATA XREF: sub_BD6588+C3io
+                unicode 0, <Microsoft_WinInet>,0
                align 10h
+Abe2869f9b474cd9A358C22904dba7f7 db 'abe2869f-9b47-4cd9-a358-c22904dba7f7',0
+                ; DATA XREF: sub_BD6588+D3io
                align 20h
WindowsliveName: ; DATA XREF: sub_BD6588+F6io
+                unicode 0, <WindowsLive:name>,0
                align 10h
_netPassport:   ; DATA XREF: sub_BD6588+A6io
+                unicode 0, <.Net Passport>,0
                align 10h
+_82bd0e679fea47488672D5efe5b779b0 db '82BD0E67-9FEA-4748-8672-D5EFE5B779B0',0
+                ; DATA XREF: sub_BD6588+B6io
                align 20h
A                db 'A',0 ; DATA XREF: sub_BD69C0+26io sub_BD69C0+
                align 10h
+ dword_BE1430 dd 20000010h ; DATA XREF: sub_BD84CC+121ir sub_BD84CC
                db 0
                db 0

```

References to information collected by the module.

Additional known variants: 992E, AF68, D49F

482F – Collects system information.

- Enumerates disk drives
- Gets list of running processes
- Extensive process information including uptime

- Memory information
- SID information

Additional known variants: F3F4

559B – Active Directory survey.

- Connects to the Active Directory Global Catalog ("GC:") using ADSI
- Enumerates all objects in AD
- Presents every entry in a human-readable format

```

v6 = ADsOpenObject(L"GC:", v5, v3, lu, &stru_100030C8, &ppObject);
*a3 = v6;
if ( v6 >= 0 )
{
    v7 = ADsBuildEnumerator((IADsContainer *)ppObject, &ppEnumVariant);
    *a3 = v7;
    if ( v7 >= 0 )
    {
        VariantInit(&pvarg);
        v8 = ADsEnumerateNext(ppEnumVariant, lu, &pvarg, &pcElementsFetched);
        *a3 = v8;
        if ( v8 < 0 || pcElementsFetched != 1 )
        {
            *v4 = -16;
        }
        else
        {
            *a3 = (*(int (__stdcall **)(LONG, IID *, int *))pvarg.lVal)(pvarg);
            VariantClear(&pvarg);
            if ( *a3 < 0 )
            {
                *v4 = -17;
            }
        }
    }
    else
    {
        *v4 = -15;
    }
}
else
{
    *v4 = -14;
}
if ( ppEnumVariant )
    ADsFreeEnumerator(ppEnumVariant);

```

Active Directory enumeration routine.

580C - Collects system and network information.

- Retrieves the domain controller name
- Enumerates all users and groups in the domain
- Collects Task Scheduler logs
- Collects disk information, removable device history
- Retrieves firewall policies
- Enumerates all named system objects
- Enumerates all system services

5B78 - Collects system information and utilities. One of the two exported functions has a name "GetReport".

- Enumerate running processes, extract tokens and SIDs, collect timing information
- Logon users using explicit credentials
- Impersonate users of running processes
- Build new 32-bit and 64-bit shellcode stubs using a hardcoded template

Both 32-bit and 64-bit versions exist.

Additional known variants: E8C7, EE6E.

5C66 – Encrypted file I/O, utilities

- File I/O operations: open/seek/read/write
- Manages compressed and encrypted temporary files

622B - Generate XML report about system using unique schema

- Computer name
- Windows directory
- Enumerates all logical drives
- Lists all files
- OS serial number
- Domain name
- Network adapter configuration: IP addresses, MAC, MTU, adapter list

```

S_info_xml:                                     ; DATA XREF: sub_1000B5DE+6D0
; unicode 0, <?s_info.xml>,0
GatherMetadataError:                          ; DATA XREF: sub_1000B5DE:loc_1000B71C0
; unicode 0, <Gather metadata error>,0
ArchiveErrorWriteFailed:                      ; DATA XREF: sub_1000B5DE+1230
; unicode 0, <Archive error: write failed>,0
ArchiveErrorEndFileFailed:                   ; DATA XREF: sub_1000B5DE:loc_1000B7460
; unicode 0, <Archive error: end file failed>,0
; align 4
unk_1000E1DC                                  ; DATA XREF: sub_1000B7F1+90
; db 0FFh
; db 0FEh ;
?xmlVersion1_0?:
; dw 3Ch
; unicode 0, <?xml version="1.0" ?>
; dw 3Eh, 0Ah, 0
; db 0
; db 0
SurveyresultXmlnsXsiHttpWww_w3_org2001XmlschemaInstan: ; DATA XREF: sub_1000B7F1+1A0
; dw 3Ch
; unicode 0, <SurveyResult xmlns:xsi="http://www.w3.org/2001/XMLSchema->
; unicode 0, <instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
; dw 3Eh, 0Ah, 0
; align 4
UniqueidCompnameSBootosserial08xUniqueidS: ; DATA XREF: sub_1000B7F1+350
; unicode 0, <
; dw 3Ch
; unicode 0, <UniqueID compname="%s" bootOsSerial="%08X" uniqueid="%s" >
; unicode 0, </>
; dw 3Eh, 0Ah, 0
Surveyresult:                                ; DATA XREF: sub_1000B5DE+EB0
; dw 3Ch
; unicode 0, </SurveyResult>
; dw 3Eh, 0Ah, 0
; align 4
True_0:                                       ; DATA XREF: sub_1000B843+2A0 sub_1000B843+AF0
; unicode 0, <true>,0
; align 4
; BoolValue False
False:                                       ; DATA XREF: sub_1000B843+220 sub_1000B843+B40
; unicode 0, <false>,0
; align 8
ParametersDirsonlySMaxdepthU:               ; DATA XREF: sub_1000B843+330
; unicode 0, <
; dw 3Ch
; unicode 0, <Parameters DirsOnly="%s" MaxDepth="%u">
; dw 3Eh, 0Ah, 0
; align 10h
TimefilterS:                                ; DATA XREF: sub_1000B843+500
; unicode 0, <
; dw 3Ch
; unicode 0, <TimeFilter %s />
; dw 3Eh, 0Ah, 0

```

XML tags used to generate the system report.

6302 - Utilities. Has internal name "d3dx9_27.dll". Executes timer-based events.

Additional known variants: FA84

669D – Utilities. Given a list of file names and directories, checks if they exist.

Additional known variants: 880B

6914 - Sniffer-based network attacks. Uses a legitimate WinPcap driver "npf.sys". Detects NBNS (NetBIOS protocol) requests of interest and sends its own responses:

- Responds to WPAD requests ("FHFAEBE" in NBNS packets)
- Sends responses to HTTP GET requests

The network filter is based on the BPF library. The payloads for the HTTP and WPAD responses are provided externally.

```

Str2      db 'GET ',0          ; DATA XREF: sub_1000565E+90f0
          align 4
DetectedGetRequestFromSToS: ; DATA XREF: sub_1000565E+F7f0
+         unicode 0, <Detected GET request from %s to %s>,0
          align 10h
NoMoreAttacksLeftNotResponding__ ; DATA XREF: sub_1000565E+11Cf0
+         unicode 0, <No more attacks left, not responding..>,0
          align 10h
SentResponsePacketToSForSAttacksLeftU: ; DATA XREF: sub_1000565E+21Af0
+         unicode 0, <Sent response packet to %s for %s (attacks left = %u)>,0
          align 10h

; char SubStr[]
+SubStr   db 'User-Agent: ',0    ; DATA XREF: sub_10005890+1f0
          align 10h
+Http1_12000kContentTypeTextHtmlConnectionCloseCon db 'HTTP/1.1 200 OK',0Dh,0Ah
          ; DATA XREF: sub_100058E2+F6f0
+         db 'Content-Type: text/html',0Dh,0Ah
+         db 'Connection: Close',0Dh,0Ah
+         db 'Content-Length: %d',0Dh,0Ah
+         db 'Accept-Ranges: none',0Dh,0Ah
+         db 'Cache-Control: no-cache, no-store, must-revalidate',0Dh,0Ah
+         db 'Pragma: no-cache',0Dh,0Ah
+         db 'Expires: Wed, 21 Jan 1995 11:56:08 GMT',0Dh,0Ah
+         db 0Dh,0Ah,0
          align 4
NotWpadRequest: ; DATA XREF: sub_10005852:loc_10005CBAf0
+         unicode 0, <Not WPAD request>,0
          align 10h
DetectedWpadRequestFromSToS: ; DATA XREF: sub_10005852+C0f0
+         unicode 0, <Detected WPAD request from %s to %s>,0
SentResponsePacket: ; DATA XREF: sub_10005852+150f0
+         unicode 0, <Sent response packet>,0

```

Fake HTTP response and related status messages.

6FAC - File API

- Get file size, attributes
- Securely delete a file
- Open/close/read/write file contents

Additional known variants: A7EE

7BDA – Collects system information

- Current state of AV and firewall protection using wscapi.dll API
- Detect if "sqlservr.exe" is running
- Computer name
- Workgroup info
- Domain controller name
- Network adapter configuration
- Time and time zone information
- CPU frequency

Additional known variants: EF2E

7C23 – Extracts metadata from documents and collects system information

- Computer name
- System volume serial
- Complete file API as in 6FAC

Searches for documents and archives and implements routines to extract all valuable information from them:

- E-mail messages: eml, msg
- Image files: jpg, jpe, jpeg, tif, tiff, bmp, png
- Multimedia files: wmv, avi, mpeg, mpg, m4a, mp4, mkv, wav, aac, ac3, dv, flac, flv, h264, mov, 3gp, 3g2, mj2, mp3, mpegts, ogg, asf. These are re-encoded with libffmpeg.
- Contents from PDF documents
- Microsoft Office: doc, docx, xlsx, pptx. Dedicated routines are called accordingly: "OfficeRipDoc", "OfficeRipDocx", "OfficeRipXlsx", "OfficeRipPptx". PPT slides are extracted and converted to a HTML digest of the presentation.
- Archives: gz, gzip, gzx3, zip, rar

Creates temporary files with extension ".fg4".

Additional known variants: EB18, C091

```

_docx:                                ; DATA XREF: 10010508fo
        unicode 0, <.docx>,0
_pptx:                                ; DATA XREF: 10010514fo
        unicode 0, <.pptx>,0
_xlsx:                                ; DATA XREF: 10010520fo
        unicode 0, <.xlsx>,0
_zip:                                 ; DATA XREF: 1001052Cfo
        unicode 0, <.zip>,0
        align 4
_rar:                                  ; DATA XREF: 10010538fo
        unicode 0, <.rar>,0
        align 4
; const WCHAR Gdiplus_dll_0
Gdiplus_dll_0:                        ; DATA XREF: sub_1000AAAC
        unicode 0, <GdiPlus.dll>,0
; const WCHAR ImageJpeg
ImageJpeg:                            ; DATA XREF: sub_1000A8DC
        unicode 0, <image/jpeg>,0
        align 4
asc_10013978:                         ; DATA XREF: sub_1000AD7C
        unicode 0, <%s\\%s>,0
GatheringRarS:                        ; DATA XREF: sub_1000AD7C
        unicode 0, <Gathering Rar: %s>,0
Rar:                                  ; DATA XREF: sub_1000AD7C
        unicode 0, <Rar>,0
Rar_error_D:                          ; DATA XREF: sub_1000AD7C
        unicode 0, <RAR_ERROR_%d>,0
        align 4
; const WCHAR Ooxml
Ooxml:                                ; DATA XREF: sub_1000B25C
        unicode 0, <OOXML>,0
; const WCHAR String
String:                               ; DATA XREF: sub_1000B31C
        unicode 0, <>,0
        ; sub_1000B310+8Dfo sub_1
        ; sub_1000B310+11Ffo sub_
; const WCHAR Image
Image:                                ; DATA XREF: sub_1000B8AC
        unicode 0, <Image>,0
; const WCHAR Ffmpeg
Ffmpeg:                               ; DATA XREF: sub_1000B9BC
        unicode 0, <ffmpeg>,0
        align 4
RunningLibffmpegS:                   ; DATA XREF: sub_1000B9BC
        unicode 0, <Running libffmpeg: >

```

Part of the list of file extensions of interest and corresponding status messages.

8172 - Sniffer-based network attacks. Performs NBNS (NetBIOS protocol) name resolution spoofing for:

- WPAD requests
- Names starting with "SHR"
- Names starting with "3142" (log only)

```

+ DetectedShrRequestFromSToS: ; DATA XREF: SHRRequest+91fo
+   unicode 0, <Detected SHR request from %s to %s>,0
+   align 4
+ SentShrResponsePacket: ; DATA XREF: SHRRequest+122fo
+   unicode 0, <Sent SHR response packet>,0
+   align 10h
+ GotUnexpectedErrorWhileRunning: ; DATA XREF: SHRRequest:loc_10006FA5fo
+   unicode 0, <Got unexpected error while running>,0
+   align 4
+ DetectedLog3142C: ; DATA XREF: Log3142+40fo
+   unicode 0, <Detected Log: 3142%C>,0
+   align 4
+ DetectedLogS: ; DATA XREF: sub_10006D77+D5fo
+   unicode 0, <Detected Log: %S>,0
+   align 4
+ ; const WCHAR String
+ String: ; DATA XREF: sub_100072CB+Efo sub_1000
+   ; 100213FCfo
+   unicode 0, <services.exe>,0
+   align 4
+ ; char Str2[]
+ Str2 db 'GET ',0 ; DATA XREF: DetectReplyGET+71fo
+   align 10h
+ DetectedGetRequestFromSToS: ; DATA XREF: DetectReplyGET+E3fo
+   unicode 0, <Detected GET request from %s to %s>,0
+   align 4
+ NoMoreAttacksLeftNotResponding__: ; DATA XREF: DetectReplyGET+108fo
+   unicode 0, <No more attacks left, not responding..>,0
+   align 4
+ SentResponsePacketToSForUriSAttacksLeftU: ; DATA XREF: DetectReplyGET+213fo
+   unicode 0, <Sent response packet to >
+   dw 27h
+   unicode 0, <%s>
+   dw 27h
+   unicode 0, < for URI >
+   dw 27h
+   unicode 0, <%S>
+   dw 27h
+   unicode 0, < (attacks left = %u)>,0

```

Status messages related to the attack.

Additional feature: the module can build new shellcode blobs from hardcoded templates.

81B7 – Driver management

- Write driver to disk
- Start/stop driver
- Safely remove the driver's file from disk

Additional known variants: C1B9

8446 - Oracle DB and ADOdb client.

- Uses "oci.dll" API to access Oracle databases
- Extracts all available information from the database
- Also connects to ADOdb providers

```
Gj43koDdi:                ; DATA XREF: sub_10004D26+1Bfo
.      unicode 0, <GJ43KO-%dDI>,0
Table_04d_bin:            ; DATA XREF: sub_10005060+4Afo
.      unicode 0, <table_%04d.bin>,0
.      align 4
Table_bin:                ; DATA XREF: sub_1000519E+8Efo
.      unicode 0, <table.bin>,0
Db:                      ; DATA XREF: sub_10007AF0+BCfo
.      unicode 0, <DB>,0
.      align 10h
byte_100100F0 db 8 dup(0) ; DATA XREF: sub_1000579E+42fo sub_10006257+Flfo
.                                     ; sub_10007AF0+23fo
AlterSessionSetCursor_bind_capture_destinationOff: ; DATA XREF: sub_100059E7+59fo
.      unicode 0, <alter session set cursor_bind_capture_destination = off>,0
AlterSessionSetCursor_sharingForce: ; DATA XREF: sub_10006257+12fo
.      unicode 0, <alter session set cursor_sharing = force>,0
.      align 10h
AlterSessionSetNls_date_formatDdMmYyyyHh24MiSs: ; DATA XREF: sub_10006257+34fo
.      unicode 0, <alter session set nls_date_format=>
.      dw 27h
.      unicode 0, <dd/mm/yyyy hh24:mi:ss>
.      dw 27h, 0
.      align 8
BeginDbms_application_info_set_moduleSSEnd: ; DATA XREF: sub_10006257+12Afo
.      unicode 0, <BEGIN dbms_application_info.set_module(>
.      dw 27h
.      unicode 0, <%s>
.      dw 27h
.      unicode 0, <, >
.      dw 27h
.      unicode 0, <%s>
.      dw 27h
.      unicode 0, <); END;>,0
.      align 10h
BeginDbms_application_info_set_client_infoSEnd: ; DATA XREF: sub_10006257+15Efo
.      unicode 0, <BEGIN dbms_application_info.set_client_info(>
.      dw 27h
.      unicode 0, <%s>
.      dw 27h
.      unicode 0, <); END;>,0
AlterSessionSetCurrent_schemaS: ; DATA XREF: sub_10006257+19Efo
.      unicode 0, <ALTER SESSION SET CURRENT_SCHEMA = %s>,0
```

SQL queries and related data.

8912 – Encrypted file manipulation and collects system information

- Shared file mapping communication
- Write encrypted data to files
- Enumerate windows
- Enumerate network shares and local disks
- Retrieve USB device history
- Collect network routing table

Known mutex and mapping names:

- Global\{DD0FF599-FA1B-4DED-AC70-C0451F4B98F0} Global\{B12F87CA-1EBA-4365-B90C-E2A1D8911CA9},
- Global\{B03A79AD-BA3A-4BF1-9A59-A9A1C57A3034} Global\{6D2104E6-7310-4A65-9EDD-F06E91747790},
- Global\{DD0FF599-FA1B-4DED-AC70-C0451F4B98F0} Global\{B12F87CA-1EBA-4365-B90C-E2A1D8911CA9}

Additional known variants: D19F, D2EE

9224 – Run console applications. Creates processes using desktop “Default”, attaches to its console and redirects its I/O to named pipes.

92DB - Modified cmd.exe shell.

```

; wchar_t Else
Else:
    unicode 0, <ELSE>,0 ; DATA XREF: sub_410D11+108fo
    align 4

; wchar_t Date
Date:
    unicode 0, <DATE>,0 ; DATA XREF: sub_406D5C:loc_406E09fo 00420AC0fo
    align 4
    unicode 0, < :>

asc_42050C:
    unicode 0, <\\*>,0 ; DATA XREF: sub_4155FE+9fo sub_4155FE:loc_415822fo
    align 4

; const WCHAR Comspec
Comspec:
    ; DATA XREF: sub_408046:loc_4080C2fo
    ; sub_40BD53:loc_40BE00fo sub_40BD53+162fo
    ; sub_40DB6D+D2fo sub_41B01B+6Ffo
    unicode 0, <COMSPEC>,0

; wchar_t Rem
Rem:
    unicode 0, <REM>,0 ; DATA XREF: sub_410E7C+1Efo sub_4111A7+6Bfo 00420DA8fo

Chdir_0:
    unicode 0, <CHDIR>,0 ; DATA XREF: 00420A30fo

; wchar_t Cd_0
Cd_0:
    unicode 0, <CD>,0 ; DATA XREF: sub_406D5C+44fo 00420A18fo
    align 10h

Cmd_exe:
    ; DATA XREF: sub_40BD53+105fo sub_40BD53+D1fo
    ; sub_40BD53:loc_40BE7Ffo 0041071Cfo
    unicode 0, <\\CMD.EXE>,0
    align 4

Vol:
    unicode 0, <VOL>,0 ; DATA XREF: 00420C10fo

; const WCHAR Path
Path:
    ; DATA XREF: sub_40646D+39fo sub_40646D+70fo
    ; sub_40646D+83fo sub_408046+304fo sub_40BD53+53fo
    unicode 0, <PATH>,0
    align 4

; wchar_t Time
Time:
    unicode 0, <TIME>,0 ; DATA XREF: sub_406D5C:loc_406E34fo 00420AD8fo
    align 4

Set:
    unicode 0, <SET>,0 ; DATA XREF: 00420A90fo

```

Several CMD commands processed by the shell.

9F0D (64-bit), **D1A3**(32-bit) – **legitimate signed driver NPF.SYS** (WinPcap) distributed inside the VFS along with the plugins. It is used for sniffer-based network attacks.

A4B0 – Network survey

- Uses DHCP Server Management API (DHCPAPI.DLL) to enumerate all DHCP server's clients
- Queries all known DHCP sub-networks
- Searches for machines that have ports UDP 1434 or 137 open
- Enumerates all network servers
- Enumerates network shares
- Tries to connect to remote registries to enumerate all users in HKEY_USERS, converts them to SIDs

B6C1 - WNet API. Provides wrappers for the WnetAddConnection2 and WNetOpenEnum functions.

Additional known variants: BC4A

C25B – Sniffer based network attacks. Implements a **fake SMB server** to trick other machines to authenticate with NTLM.

- Implements basic SMB v1 commands

```

dword_10013340 dd 72h ; DATA XREF: sub_1000
off_10013344 dd offset smb_cmd_negotiate ; DATA XREF: sub_
dd 73h
dd offset SMB_COM_SESSION_SETUP_ANDX
dd 28h
dd offset SMB_COM_ECHO
dd 75h
dd offset SMB_COM_TREE_CONNECT_ANDX
dd 0A2h
dd offset SMB_COM_NT_CREATE_ANDX
dd 0A0h
dd offset SMB_COM_NT_TRANSACTION
dd 32h
dd offset SMB_COM_TRANSACTION2
dd 2Eh
dd offset SMB_COM_READ_ANDX
dd 08h
dd offset SMB_COM_WRITE
dd 2Fh
dd offset SMB_COM_WRITE_ANDX
dd 4
dd offset SMB_COM_CLOSE
dd 71h
dd offset SMB_COM_TREE_DISCONNECT
dd 74h
dd offset SMB_COM_LOGOFF_ANDX
dd 0

```

SMB commands handled by the module

- Pretends to have IPC\$ and A: shares
- Accepts user authentication requests
- Also handles HTTP "GET /" requests

```

; char Device[]
Device db '\Device\',0 ; DATA XREF: SelectAdapter+153i
align 10h
; char NtLm0_12[]
NtLm0_12 db 'NT LM 0.12',0 ; DATA XREF: smb_cmd_negotiate+9Ei
align 4
+challenge db 6Ch, 5Bh, 4, 86h, 0Dh, 0C2h, 0DBh, 0Eh, 0E4h, 65h, 51h, 0E5h, 0CDh, 0FEh
; DATA XREF: smb_cmd_negotiate+18Fi
db 4 dup(0)
SMB1 db 0 ; DATA XREF: SMB_COM_SESSION_SETUP_ANDX+7Bfi
Windows:
+ unicode 0, <Windows>
db 20h, 5, 0, 2Eh, 1, 3 dup(0)
Windows_0:
+ unicode 0, <Windows>
db 20h, 2, 4 dup(0)
LanManager:
+ unicode 0, <LAN Manager>
db 0
; wchar_t Ipc
Ipc: unicode 0, <IPC$>,0 ; DATA XREF: SMB_COM_TREE_CONNECT_ANDX+D5fi
align 10h
Ipc_0 db 'IPC:',0 ; DATA XREF: SMB_COM_TREE_CONNECT_ANDX+F0fi
align 4
A db 'A:',0 ; DATA XREF: SMB_COM_TREE_CONNECT_ANDX+11afi
Fat:
unicode 0, <FAT>,0
db 0

```

NTLM challenge and SMB server data

ED92 – File system survey

- Enumerates all local drives and connected network shares
- Lists files

EF97 – Filesystem utilities

- Enumerate files
- Create and remove directories
- Copy/move/delete files and directories
- Extract version information from files
- Calculate file hashes

Additional known variants: F71E

PERSISTENCE MECHANISM

The Duqu 2.0 malware platform was designed in a way that survives almost exclusively in memory of the infected systems, without need for persistence. To achieve this, the attackers infect servers with high uptime and then re-infect any machines in the domain that get disinfected by reboots. Surviving exclusively in memory while running kernel level code through exploits is a testimony to the technical prowess of the group. In essence, the attackers were confident enough they can survive within an entire network of compromised computers without relying on any persistence mechanism at all.

The reason why there is no persistence with Duqu 2.0 is probably because the attackers wanted to stay under the radar as much as possible. Most modern anti-APT technologies can pinpoint anomalies on the disk, such as rare drivers, unsigned programs or maliciously-acting programs. Additionally, a system where the malware survives reboot can be imaged and then analyzed thoroughly at a later time. With Duqu 2.0, forensic analysis of infected systems is extremely difficult – one needs to grab memory snapshots of infected machines and then identify the infection in memory.




However, this mechanism has one weakness; in case of a massive power failure, all computers will reboot and the malware will be eradicated. To get around this problem, the attackers have another solution – they deploy drivers to a small number of computers, with direct Internet connectivity. These drivers can tunnel traffic from the outside into the network, allowing the attackers to access remote desktop sessions or to connect to servers inside the domain by using previously acquired credentials. Using these credentials, they can re-deploy the entire platform following a massive power loss.

COMMAND AND CONTROL MECHANISMS

Duqu 2.0 uses a sophisticated and highly flexible command-and-control mechanism that builds on top of the 2011 variant, with new features that appear to have been inspired by other top class malware such as Regin. This includes the usage of network pipes and mailslots, raw filtering of network traffic and masking C&C traffic inside image files.

Inside a Windows LAN, newly infected clients may not have a C&C hardcoded in their installation MSI packages. Without a C&C, they are in “dormant” state and can be activated by the attackers over SMB network pipes with a special TCP/IP packet that contains the magic string “ttttttttttttttt”. If a C&C is included in the configuration part of the MSI file, this can be either a local IP address, which serves as a bouncing point or an external IP address. As a general strategy for infection, the attackers identify servers with high uptime and set them as intermediary C&C points. Hence, an infected machine can jump between several internal servers in the LAN before reaching out to the Internet.

To connect the the C&C servers, both 2011 and 2014/2015 versions of Duqu can hide the traffic as encrypted data appended to a harmless image file. The 2011 version used a JPEG file for this; the new version can use either a GIF file or a JPEG file. Here’s how these image files look like:

Duqu 2011 – JPEG	Duqu 2015 – GIF	Duqu 2015 - JPEG
 54x54 pixels	 11x11 pixels	 33x33 pixels

Another modification to the 2014/2015 variants is the addition of multiple user agent strings for the HTTP communication. The 2011 used the following user agent string:

- Mozilla/5.0 (Windows; U; Windows NT 6.0; en-US; rv:1.9.2.9) Gecko/20100824 Firefox/3.6.9 (.NET CLR 3.5.30729)

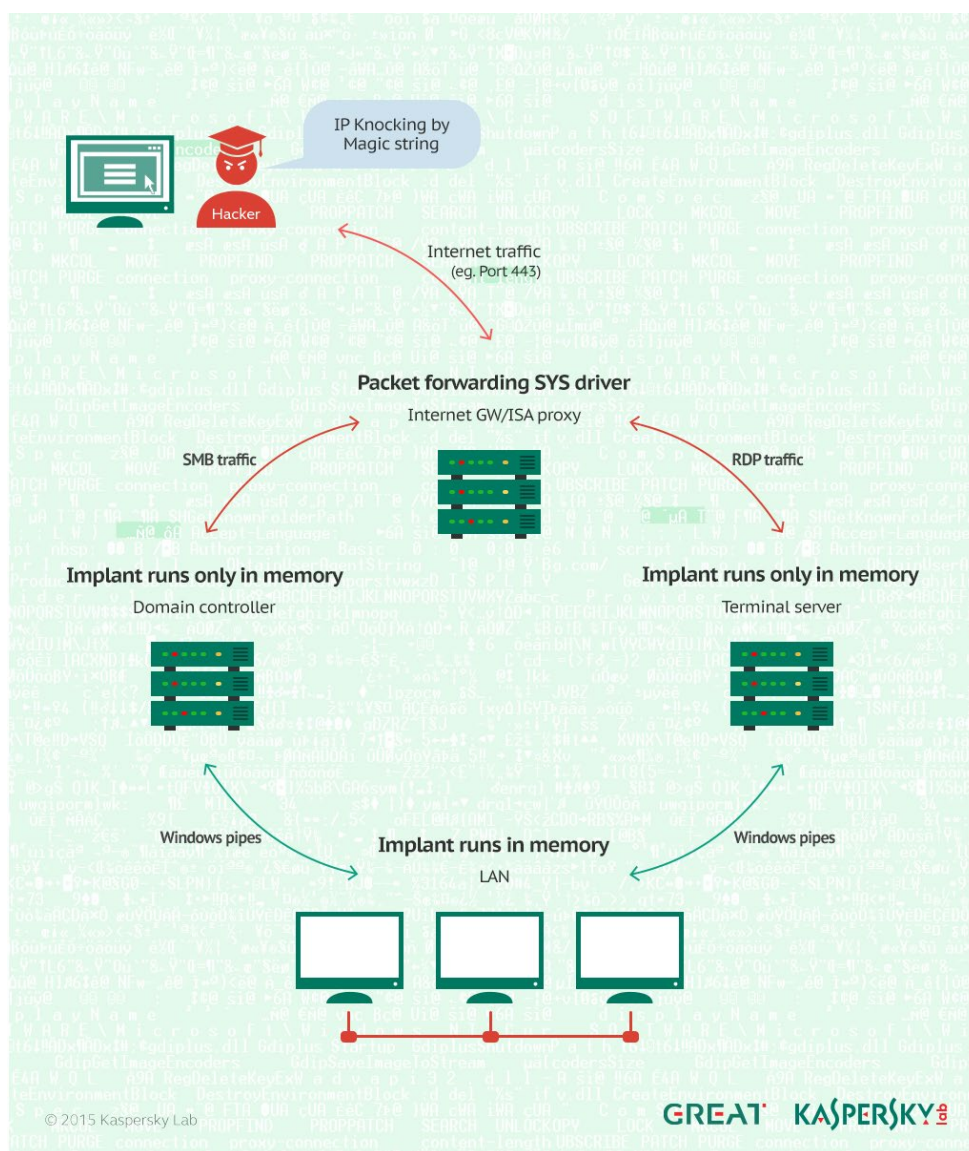
The new variants will randomly select an user agent string from a table of 53 different possible ones.

Another unusual C&C mechanism relies on driver files that are used to tunnel the C&C communications and attacker's RDP/SMB activity into the network. The attackers deploy such translation drivers on servers with direct Internet connectivity. Through a knocking mechanism, the attackers can activate the translation mechanism for their IPs and tunnel their traffic directly into the LAN. Outside the LAN, the traffic can be masked over port 443; inside the LAN, it can be either direct SMB/RDP or it can be further translated over fake TCP/IP packets to IP 8.8.8.8.

During our investigation, we observed several such drivers. A description can be found below.

The “portserv.sys” driver analysis

MD5: 2751e4b50a08eb11a84d03f8eb580a4e



Size: 14336

Compiled: Sat Feb 11 21:55:30 2006 (fake timestamp)

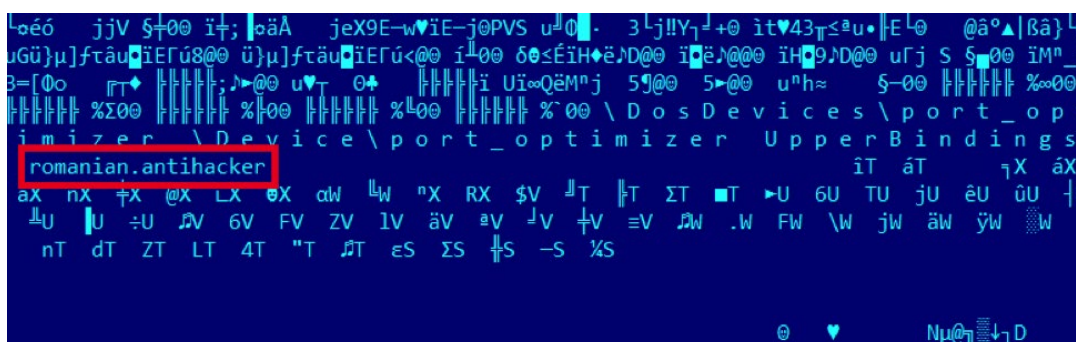
Internal name: termport.sys

Type: Win32 device driver (a 64 bit version is known as well)

This is a malicious NDIS filter driver designed to perform manipulation of TCP/IP packets to allow the attacker to access internal servers in the victim's infrastructure.

Upon startup, the filter driver hooks into the NDIS stack and starts processing TCP/IP packets.

To leverage the driver, the attacker first sends a special TCP/IP packet with the string **"romanian.antihacker"** to any of the hardcoded IPs belonging to infected server. In general, such servers are computers with direct Internet connectivity, such as a webserver or a proxy. The driver sees the packet, recognizes the magic string **"romanian.antihacker"** and saves the attacker's IP for later use.



Magic string used for knocking inside the driver.

When a packet comes from the attacker's IP (saved before), the following logic applies:

- Packet to server 1's IP on port 443, is redirected on port 445 (Samba/Windows file system)
- Packet from server 1's IP from port 445, is redirected to attacker's IP port 443
- Packet to server 2's IP on port 443 is redirected on port 3389 (Remote Desktop)
- Packet from server 2's IP from port 3389 is redirected to attacker's IP port 443

This effectively allows the attackers to tunnel SMB (remote file system access) and Remote Desktop into these two servers while making it look like SSL traffic (port 443).

These drivers allow the Duqu attackers to easily access servers inside the LAN from remote, including tunneling RDP sessions over Port 443 (normally SSL). It also gives them a persistence mechanism that allows them to return even if all the infected machines with the malware in memory are rebooted. The attackers can simply use existing credentials to log back into any of the servers that the driver is serving and can re-initialize the backdoors from there.

SIMILARITIES BETWEEN DUQU AND DUQU 2.0

The 2014/2015 Duqu 2.0 is a greatly enhanced version of the 2011 Duqu malware discovered by ⁷CrySyS Lab. It includes many new ideas from modern malware, such as Regin, but also lateral movement strategies and harvesting capabilities which surpasses commonly seen malware from other APT attacks.

Side by side:

	2011 Duqu	2014/2015 Duqu 2.0
Number of victims:	<50 (estimated)	<100 (estimated)
Persistence mechanism:	Yes	No
Loader:	SYS driver	MSI file
Zero-days used:	Yes	Yes
Main storage:	PNF (custom) files	MSI files
C&C mechanism:	HTTP/HTTPS, network pipes	HTTP/HTTPS, network pipes
Known plugins:	6	>100

There are many similarities in the code that leads us to conclusion that Duqu 2.0 was built on top of the original source code of Duqu. Those interested can read below for a technical description of these similarities.

⁷ <https://www.crysys.hu/publications/files/bencsathPBF11duqu.pdf>

One of the “trademark” features unique to the original Duqu was the set of functions that provide logging facilities. Unlike many other APTs, Duqu logs almost every important step of its activity but does it in a special way: there are no readable strings written to the log. Instead, a series of unique numbers identify every state, error, or message in the log. Comparing the functions that generate every log entry in Duqu and Duqu 2.0, we can conclude that they are almost identical:

<pre> 0000:10015F25 arg_C = dword ptr 14h 0000:10015F26 lpString2 = dword ptr 0Ch 0000:10015F27 arg_14 = byte ptr 1Ch 0000:10015F25 push ebp 0000:10015F26 mov ebp, esp 0000:10015F27 push edi 0000:10015F28 call ds:imp_GetLastError@0 ; GetLastError() 0000:10015F29 push [ebp+lpString2] ; lpString 0000:10015F32 mov edi, eax 0000:10015F34 call ds:strlenW 0000:10015F3A cmp eax, 400h 0000:10015F3F jnb short loc_10015F4C 0000:10015F41 push edi 0000:10015F42 call ds:imp_SetLastError@4 ; SetLastError(x) 0000:10015F48 xor eax, eax 0000:10015F4C jmp short loc_10015FAA ;----- 0000:10015F4C loc_10015F4C: ; CODE XREF: class17_ctor_from_string_and_date 0000:10015F4C push esi 0000:10015F4D call class17_ctor 0000:10015F52 mov esi, eax 0000:10015F54 test esi, esi 0000:10015F56 jnz short loc_10015F63 0000:10015F58 push edi 0000:10015F59 call ds:imp_SetLastError@4 ; SetLastError(x) 0000:10015F5F xor eax, eax 0000:10015F63 jmp short loc_10015FA9 ;----- 0000:10015F63 loc_10015F63: ; CODE XREF: class17_ctor_from_string_and_date 0000:10015F63 mov eax, [ebp+arg_0] 0000:10015F66 mov dword ptr ds:[_class17.int1 - _class17.int1][esi], eax 0000:10015F68 mov eax, [ebp+arg_4] 0000:10015F6B mov ds:[_class17.int2 - _class17.int1][esi], eax ; log entry 0000:10015F6E mov eax, [ebp+arg_8] 0000:10015F71 mov ds:[_class17.int3 - _class17.int1][esi], eax ; log entry 0000:10015F74 mov eax, [ebp+arg_C] 0000:10015F77 mov ds:[_class17.int4 - _class17.int1][esi], eax ; log entry 0000:10015F7A mov al, [ebp+arg_14] 0000:10015F7D mov byte ptr ds:[_class17.byte - _class17.int1][esi], al ; log 0000:10015F80 lea eax, (_class17.FileTime - _class17.int1)[esi] ; log entry 0000:10015F86 push eax 0000:10015F87 call ds:GetSystemTimeAsFileTime 0000:10015F8D cmp [ebp+lpString2], 0 0000:10015F91 jz short loc_10015FA0 0000:10015F93 push [ebp+lpString2] ; lpString2 0000:10015F96 lea eax, (_class17.string - _class17.int1)[esi] ; log entry 0000:10015F99 push eax 0000:10015FA0 call ds:strlenW 0000:10015FA0 loc_10015FA0: ; CODE XREF: class17_ctor_from_string_and_date 0000:10015FA1 push edi 0000:10015FA2 call ds:imp_SetLastError@4 ; SetLastError(x) 0000:10015FA7 mov eax, esi 0000:10015FA9 loc_10015FA9: ; CODE XREF: class17_ctor_from_string_and_date 0000:10015FA9 pop esi 0000:10015FAA loc_10015FAA: pop edi 0000:10015FAA pop ebp 0000:10015FAB retn 0000:10015FAC class17_ctor_from_string_and_date endp </pre>	<pre> 1002CDF5 arg_0 = dword ptr 8 1002CDF6 arg_4 = dword ptr 0Ch 1002CDF7 lpString = dword ptr 10h 1002CDF8 arg_C = byte ptr 14h 1002CDF5 push ebp 1002CDF6 mov ebp, esp 1002CDF7 push ecx 1002CDF8 push ecx 1002CDF9 push edi 1002CDE0 mov [ebp+var_8], edx 1002CDE1 mov [ebp+var_4], ecx 1002CDE2 call cGetLastError 1002CDE3 push [ebp+lpString] ; lpString 1002CDE4 mov edi, eax 1002CDE5 call cstrlenW 1002CDE6 cmp eax, 400h 1002CDE7 jnb short loc_1002CE24 1002CDE8 push edi 1002CDE9 call cSetLastError ; dwErrCode 1002CE00 xor eax, eax 1002CE02 jmp short loc_1002CE82 ;----- 1002CE24 loc_1002CE24: ; CODE XREF: Log+22j 1002CE24 push esi 1002CE25 call sub_1002CDBB 1002CE2A mov esi, eax 1002CE2C test esi, esi 1002CE2E jnz short loc_1002CE38 1002CE30 push edi 1002CE31 call cSetLastError ; dwErrCode 1002CE37 xor eax, eax 1002CE39 jmp short loc_1002CE81 ;----- 1002CE38 loc_1002CE38: ; CODE XREF: Log+39j 1002CE38 mov eax, [ebp+var_4] 1002CE39 mov [esi], eax 1002CE40 mov eax, [ebp+var_8] 1002CE43 mov [esi+4], eax 1002CE46 mov eax, [ebp+arg_0] 1002CE49 mov [esi+8], eax 1002CE4B mov eax, [ebp+arg_4] 1002CE4F mov [esi+0Ch], eax 1002CE52 mov al, [ebp+arg_C] 1002CE55 mov [esi+14h], al 1002CE58 lea eax, [esi+81h] 1002CE5B push eax 1002CE5C call cGetSystemTimeAsFileTime 1002CE5F cmp [ebp+lpString], 0 1002CE61 jz short loc_1002CE78 1002CE63 push [ebp+lpString] ; lpString2 1002CE66 lea eax, [esi+16h] 1002CE69 push eax 1002CE72 call cstrlenW 1002CE78 loc_1002CE78: ; CODE XREF: Log+74j 1002CE78 push edi 1002CE79 call cSetLastError ; dwErrCode 1002CE7F mov esi, esi 1002CE81 loc_1002CE81: ; CODE XREF: Log+44j 1002CE81 pop edi 1002CE82 loc_1002CE82: pop esi 1002CE83 mov edi, edi 1002CE85 pop ecx 1002CE86 retn 1002CE86 Log endp </pre>
--	---

The first generation of Duqu was also written in a very rare and unique manner. It was compiled with Visual Studio and while parts of it were definitely written in C++, the majority of its classes were not natively generated by the C++ compiler. After analyzing all the possible variants, we conclude that these classes were written in OO-C, the objective variant of the C language, and then somehow converted into a compilable C/C++ source. All these classes had a very specific feature: the virtual function table of every instance was filled “by hand” in its constructor. Interestingly, this is no longer the case for Duqu 2.0. The authors upgraded their compiler from Visual Studio 2008 (used in 2011) to Visual Studio 2013 and now use classes that look much more like native C++ ones:

<pre> 1000:10015F66 proc near ; CODE XREF: class17_ctor 1000:10015F66 class17_ctor ; class17_ctor_from 1000:10015F66 push 2092 1000:10015F67 call new 1000:10015F68 pop ecx 1000:10015F69 test eax, eax 1000:10015F6A jnz short loc_10015F6E 1000:10015F6C retn ;----- 1000:10015F6E loc_10015F6E: ; CODE XREF: class17_ctor+11j 1000:10015F6E mov ds:[_class17.copy_out_buffer - _class17.int1][eax], offset class17.copy_out_buffer 1000:10015F70 mov ds:[_class17.ctor_from_buffer - _class17.int1][eax], offset class17.ctor_from_buffer 1000:10015F72 mov ds:[_class17.dtor - _class17.int1][eax], offset generic_dtor ; log entry 1000:10015F74 retn 1000:10015F74 class17_ctor endp </pre>	<pre> 1002CDDA sub_1002CD98 endp SUBROUTINE 1002CDDA log_item_ctor proc near 1002CDDA mov ecx, 2084 1002CDDA call Alloc 1002CDDA test eax, eax 1002CDDA jnz short loc_1002CDE4 1002CDE4 retn ;----- 1002CDE4 loc_1002CDE4: ; CODE XREF: log_item_ctor+11j 1002CDE4 mov dword ptr [eax+820h], offset log_item_vtbl 1002CDE4 retn ;----- 1002CDE4 log_item_vtbl dd offset sub_1002CDE4 1002CDE4 dd offset copy_out_buffer ; DATA XREF: log_item_ctor:loc_1002CDE4 1002CDE4 dd offset ctor_from_buffer 1002CDE4 dd offset j_Free </pre>
---	---

On the left: the “hand-made” or “compiler-assisted” classed of OO-C in Duqu.
On the right: the same class in Duqu 2.0 has a native Vtable similar to native C++ one, however the offset of the pointer is not zero.

The more concrete evidence of similarity can be found if we look for functions that actually use the logging facilities. The authors kept using the same unique numbers for identification of internal states, errors and function results. Networking functions are good candidates for comparison:

The image displays a side-by-side comparison of assembly code from Duqu 2011 (left) and Duqu 2015 (right). Both versions implement a logging function that calls `class17_ctor_from_string_and_date`. Red rectangles highlight identical parameter pushes in both versions, indicating that the same unique numbers were used for internal states, errors, and function results.

Duqu 2011 (Left):

```

0000:1000FD05 var_4_some_obj? = dword ptr -4
0000:1000FD05 arg_listen_address= dword ptr 8
0000:1000FD05 arg_accept_port = word ptr 0Ch
0000:1000FD05
0000:1000FD05 push ebp
0000:1000FD06 mov ebp, esp
0000:1000FD06 push ecx
0000:1000FD09 push ebx
0000:1000FD0A push esi
0000:1000FD0B push edi
0000:1000FD0C mov edi, eax
0000:1000FD0E call do_WSAStartup
0000:1000FD13 test eax, eax
0000:1000FD15 jz loc_1000FDCA
0000:1000FD18 push 5Ch ; dwBytes
0000:1000FD20 call new
0000:1000FD25 mov esi, eax
0000:1000FD27 pop ecx
0000:1000FD28 test esi, esi
0000:1000FD2A jz loc_1000FDCA
0000:1000FD30 push edi ; _DWORD
0000:1000FD31 call [edi+class_12.make_class11]
0000:1000FD34 pop ecx
0000:1000FD35 mov [esi+class_18.p_class11], eax
0000:1000FD38 test eax, eax
0000:1000FD3A jz loc_1000FDC3
0000:1000FD40 push 0 ; char
0000:1000FD42 push 0 ; lpString2
0000:1000FD44 push 0 ; int
0000:1000FD46 push 904D0561h ; int
0000:1000FD48 push 0B807043h ; int
0000:1000FD50 push 347DB92Ch ; int
0000:1000FD55 mov ebx, eax
0000:1000FD57 call class17_ctor_from_string_and_date
0000:1000FD5D push ebx
0000:1000FD5E call dword ptr ds:(class_11.logger_log - class_11.logger_log)
0000:1000FD60 or [esi+class_18.socket], 0FFFFFFFh
0000:1000FD67 add esp, 20h
0000:1000FD6A push [ebp+arg_listen_address] ; lpString2
0000:1000FD6D lea eax, [esi+class_18.listen_address]
0000:1000FD71 call ds:strcmpW ; lpString1
0000:1000FD77 mov ax, [ebp+arg_accept_port]
0000:1000FD7B word ptr [esi+class_18.listen_port_number], ax
0000:1000FD82 lea eax, [ebp+var_4_some_obj?]
0000:1000FD85 mov ecx, [ebp+var_4_some_obj?]
0000:1000FD86 push esi
0000:1000FD87 offset class18_listen_on_address
0000:1000FD8C push edi
0000:1000FD8D [esi+class_18.p_class12], edi
0000:1000FD90 [edi+class_12.exec_func], edi
0000:1000FD93 add esp, 10h
0000:1000FD96 test eax, eax
0000:1000FD98 jz short loc_1000FDC3
0000:1000FD9A mov edi, [esi+class_18.p_class11]
0000:1000FD9D push 1 ; char
0000:1000FD9F xor ebx, ebx
0000:1000FDA1 push ebx ; lpString2
0000:1000FDA3 push ebx ; int
0000:1000FDA5 push 569E08E9h ; int
0000:1000FDA8 push 0B807043h ; int
0000:1000FDD4 push 347DB92Ch ; int
0000:1000FDB2 call class17_ctor_from_string_and_date
0000:1000FDB7 push eax
0000:1000FDB8 edi, [esi+class_11.logger_log]
0000:1000FDB9 call [edi+class_11.logger_log]
0000:1000FDBB add esp, 20h
0000:1000FDBE cmp [ebp+var_4_some_obj?], ebx
0000:1000FDC1 jnz short loc_1000FDC5
0000:1000FDC3 loc_1000FDC3: ; CODE XREF: class18_ctor+35j; class18_ctor+93j
0000:1000FDC3 push esi
0000:1000FDC4 call class18_dtor
0000:1000FDC9 pop ecx
0000:1000FDCA loc_1000FDCA:
0000:1000FDCA xor eax, eax
0000:1000FDCC jmp short loc_1000FDC5
0000F105 1000FD05: class18_ctor [Synchronized with Hex View-1]

```

Duqu 2015 (Right):

```

1001B054 var_C = dword ptr -0Ch
1001B054 lpString2 = dword ptr -8
1001B054 var_2 = word ptr -2
1001B054 arg_0 = dword ptr 8
1001B054
1001B054 push ebp
1001B055 mov ebp, esp
1001B057 sub esp, 10h
1001B05A push ebx
1001B05B push esi
1001B05C push edi
1001B05D mov [ebp+var_2], dw
1001B061 mov [ebp+lpString2], ecx
1001B064 call cWSAStartup
1001B069 test eax, eax
1001B06B jz loc_1001B133
1001B071 mov ecx, 0A0h
1001B076 call Alloc
1001B07B mov edi, eax
1001B07D test edi, edi
1001B07F jz loc_1001B133
1001B085 mov ebx, [ebp+arg_0]
1001B088 push ebx
1001B089 mov ecx, [ebp+4]
1001B08C call dword ptr [ecx+5Ch]
1001B08F mov [edi+0Ch], eax
1001B092 pop ecx
1001B093 test eax, eax
1001B095 jz loc_1001B12C
1001B098 mov esi, [eax]
1001B09D mov edx, 0B807043h
1001B0A2 push 1
1001B0A4 push 0
1001B0A6 push 0
1001B0A8 push 904D0561h
1001B0AD mov ecx, 347DB92Ch
1001B0B2 call Log
1001B0B7 push eax
1001B0B8 push dword ptr [edi+0Ch]
1001B0BB call dword ptr [esi]
1001B0BD or dword ptr [edi+94h], 0FFFFFFFh
1001B0C4 lea eax, [edi+10h]
1001B0C7 add esp, 18h
1001B0CA push [ebp+lpString2] ; lpString2
1001B0CB push eax ; lpString1
1001B0CE call clstrcmpW
1001B0D4 mov ax, [ebp+var_2]
1001B0D8 lea ecx, [ebp+var_C]
1001B0DB push ecx
1001B0DE mov [edi+90h], ax
1001B0E3 mov [edi+8], ebx
1001B0E6 mov eax, [ebp+4]
1001B0E9 push edi
1001B0EA offset sub_1001B13C
1001B0EF push ebx
1001B0F0 call dword ptr [eax+8]
1001B0F3 add esp, 10h
1001B0F6 test eax, eax
1001B0F8 jz short loc_1001B12C
1001B0FA mov eax, [edi+0Ch]
1001B0FD mov edx, 0B807043h
1001B102 push 1
1001B104 push 0
1001B106 push 0
1001B108 mov esi, [eax]
1001B10A mov ecx, 347DB92Ch
1001B10F push 569E08E9h
1001B114 call Log
1001B119 push eax
1001B11A push dword ptr [edi+0Ch]
1001B11D call dword ptr [esi]
1001B11F add esp, 18h
1001B122 cmp [ebp+var_C], 0
1001B126 jz short loc_1001B12C
1001B128 mov eax, edi
1001B12A jmp short loc_1001B135
1001B12C loc_1001B12C: ; CODE XREF: sub_1001B054+41j; sub_1001B054+41j
1001B12C push edi
1001B12D call sub_1001B394
1001B132 pop ecx
1001B133 loc_1001B133:
1001B135 xor eax, eax
1001B135 loc_1001B135: ; CODE XREF: sub_1001B054+D6j

```

Implementation of the same networking function in Duqu and Duqu 2.0. Note the same unique numbers (in red rectangles) PUSHed as parameters to the logging function.

```

0000:000101733 push ebx
0000:000101734 push ebx
0000:000101735 push 0 ; flags
0000:000101736 push ecx ; len
0000:000101737 lea eax, [ebp+buf]
0000:000101738 push eax
0000:000101739 push [esi+class_19_socket] ; s
0000:000101740 call ds:recv
0000:000101741 mov ebx, eax
0000:000101742 test ebx, ebx
0000:000101743 jnz short loc_00010781
0000:000101744 mov edi, [esi+class_19_class1]
0000:000101745 push 4 ; char
0000:000101746 push eax ; lpString2
0000:000101747 push 0 ; int
0000:000101748 push 0F04EE80ah ; int
0000:000101754 loc_00010754: ; CODE XREF: class19_recv+9Ajl
0000:000101755 push 0B807043h ; int
0000:000101756 push 3470B82Ch ; int
0000:000101757 call class17_ctor_from_string_and_date
0000:000101758 push eax
0000:000101759 push edi
0000:000101760 add esp, 20h
0000:000101761 jmp short loc_000107E5
0000:000101772 loc_00010772: ; CODE XREF: class19_recv+1Cjl
0000:000101773 mov ebx, [esi+class_19_p_class8_input]
0000:000101774 push eax
0000:000101775 push [eax+class_8_get_available_size]
0000:000101776 pop ecx
0000:000101777 test ecx, ecx
0000:000101778 jnz short loc_00010738
0000:000101779 inc ecx
0000:00010177A leave
0000:00010177B retn
0000:000101781 loc_00010781: ; CODE XREF: class19_recv+39jl
0000:000101782 cmp ebx, -1
0000:000101783 jnz short loc_000107AF
0000:000101784 mov edi, ds:WSocketLastError
0000:000101785 call ebx ; WSAGetLastError
0000:000101786 jmp short loc_000107EE
0000:000101787 mov edi, ds:WSocketLastError
0000:000101788 call edi ; WSAGetLastError
0000:000101789 jmp short loc_000107EE
0000:000101790 mov edi, [esi+class_19_p_class11]
0000:000101791 push 10h
0000:000101792 push 0
0000:000101793 call ebx ; WSAGetLastError
0000:000101794 jmp short loc_0001075A
0000:0001017AF loc_000107AF: ; CODE XREF: class19_recv+71jl
0000:0001017B0 mov edi, [esi+class_19_p_class11]
0000:0001017B1 push 1 ; char
0000:0001017B2 push 0 ; lpString2
0000:0001017B3 push 0 ; int
0000:0001017B4 push 7BCE109h ; int
0000:0001017B5 push 0F807043h ; int
0000:0001017B6 push 3470B82Ch ; int
0000:0001017B7 call class17_ctor_from_string_and_date
0000:0001017B8 push eax
0000:0001017B9 push edi
0000:0001017BA add [edi+class_11_logger_log]
0000:0001017BB mov ebx, [esi+class_19_p_class8_input]
0000:0001017BC push ebx
0000:0001017BD lea ecx, [ebp+buf]
0000:0001017BE push ecx
0000:0001017BF push [eax+class_8_write_no_class]
0000:0001017C0 add esp, 2Ch
0000:0001017C1 test ecx, ecx
0000:0001017C2 jnz short loc_000107EE
0000:0001017E5 loc_000107E5: ; CODE XREF: class19

```

```

1001B882 push ebx
1001B883 esi
1001B884 push 0
1001B885 push esi
1001B886 lea eax, [ebp+buf]
1001B887 push eax
1001B888 push dword ptr [edi+30h] ; s
1001B889 call ds:recv
1001B890 mov ebx, eax
1001B891 test ebx, ebx
1001B892 jnz short loc_1001B8F5
1001B893 mov eax, [edi+2Ch]
1001B894 push 4
1001B895 push ebx
1001B896 push ebx
1001B897 mov esi, [eax]
1001B898 mov ecx, 0F04E80Ah
1001B899 loc_1001B8C0: ; CODE XREF: sub_1001B882+41j
1001B89A mov edx, 0B807043h
1001B89B mov ecx, 347D892Ch
1001B89C call Log
1001B89D push eax
1001B89E push dword ptr [edi+2Ch]
1001B89F call dword ptr [esi]
1001B8A0 add esp, 18h
1001B8A1 jmp short loc_1001B8C61
1001B8A2 loc_1001B8F5: ; CODE XREF: sub_1001B882+49j
1001B8A3 cmp ebx, 0FFFFFFFh
1001B8A4 jnz short loc_1001B8C25
1001B8A5 mov ebx, ds:WSAGetLastError
1001B8A6 call ebx ; WSAGetLastError
1001B8A7 cmp eax, 2739h
1001B8A8 jz short loc_1001B8C6C
1001B8A9 mov ebx, ds:WSAGetLastError
1001B8AA cmp eax, 2749h
1001B8AB jz short loc_1001B8C6C
1001B8AC mov eax, [edi+2Ch]
1001B8AD push 10h
1001B8AE push 0
1001B8AF mov esi, [eax]
1001B8B0 call ebx ; WSAGetLastError
1001B8B1 push eax
1001B8B2 mov ecx, 87840055h
1001B8B3 jmp short loc_1001B8B0B
1001B8B4 loc_1001B8C25: ; CODE XREF: sub_1001B882+46j
1001B8B5 mov eax, [edi+20h]
1001B8B6 mov edx, 0B807043h
1001B8B7 j push
1001B8B8 push 0
1001B8B9 push ebx
1001B8BA mov esi, [eax]
1001B8BB mov ecx, 347D892Ch
1001B8BC call Log
1001B8BD push eax
1001B8BE push dword ptr [edi+2Ch]
1001B8BF call dword ptr [esi]
1001B8C0 mov ecx, [edi+24h]
1001B8C1 lea eax, [ebp+buf]
1001B8C2 push ebx
1001B8C3 push edx
1001B8C4 push eax
1001B8C5 mov ecx, [eax+1Ch]
1001B8C6 call dword ptr [ecx]
1001B8C7 add esp, 24h
1001B8C8 test eax, eax
1001B8C9 jnz short loc_1001B8C6C
1001B8CA loc_1001B8C61: ; CODE XREF: sub_1001B882+71j
1001B8CB mov eax, [edi]
1001B8CC push edi
1001B8CD call dword ptr [eax+4]
1001B8CE pop ecx
1001B8CF mov eax, eax
1001B8D0 jmp short loc_1001B8C6F
1001B8D1 loc_1001B8C6C: ; CODE XREF: sub_1001B882+74j
1001B8D2 xor eax, eax
1001B8D3 inc ecx
1001B8D4 loc_1001B8C6F: ; CODE XREF: sub_1001B882+77j
1001B8D5 mov esi,

```

Duqu 2011

Duqu 2015

Another networking routine: after calling `recv()` to receive data from network, Duqu logs the results and possible network errors (obtained via `WSAGetLastError()`). Unique numbers in red rectangles are used to identify the current state of the networking routine.

The code of the orchestrator evolved in many aspects since 2011. One of the notable differences is a huge list of HTTP User-Agent strings that are now used instead of a single hard-coded one:

Duqu 2

```
0000:1003B0B8 aContent-Type: 2 db 'Content-Type',0 : DATA XREF: class22
0000:1003B0C5 align 4
0000:1003B0C8 : char aImageJpeg_0[] align 4
0000:1003B0CB aImageJpeg_0 db 'image/jpeg',0 : DATA XREF: class22
0000:1003B0D3 align 4
0000:1003B0D6 aPost align 4 db 'POST',0 : DATA XREF: http_
0000:1003B0DA align 4
0000:1003B0DD aGet align 4 db 'GET',0 : DATA XREF: http_accept_request+5a5
0000:1003B0E1 align 4
0000:1003B0E4 : char sCn00B4E4[] align 4
0000:1003B0E8 asc_1003B0E4 db '/',0 : DATA XREF: http_accept_request+8f1
0000:1003B0EB align 4
0000:1003B0EE : char sUserAgent[] align 4
0000:1003B0F0 aUserAgent db 'User-Agent',0 : DATA XREF: http_accept_request+100007Cf0
0000:1003B0F3 align 8
0000:1003B0F6 : char aMozilla_Os[] align 8
0000:1003B0F9 aMozilla_Os_Ownd db 'Mozilla/5.0 (Windows; U; Windows NT 6.0; en-US; rv:1.9.2.9) Gecko/
0000:1003B0FE : 'Mozilla/5.0 (Windows; U; Windows NT 6.0; en-US; rv:1.9.2.9) Gecko/
0000:1003B101 : 9.1; Net-CLR 3.5.30729.0' : DATA XREF: http_accept_request+e42
0000:1003B104 align 4
0000:1003B107 aCookie align 4
0000:1003B10A db 'Cookie',0 : DATA XREF: http_accept_request+100007F1
0000:1003B10D align 10h
0000:1003B110 : char aConnection[] align 10h
0000:1003B113 db 'Connection',0 : DATA XREF: http_server_on_data_available+81
0000:1003B116 align 4
0000:1003B119 db 'Content-Length',0 : DATA XREF: http_server_on_data_available+18f
0000:1003B11B align 4
0000:1003B11E aContent-Type align 4 db 'Content-Type',0 : DATA XREF: http_server_on_data_available+271
0000:1003B121 align 4
0000:1003B124 aTransferEncoding db 'Transfer-Encoding',0 : DATA XREF: http_server_on_data_available+361
0000:1003B127 align 10h
0000:1003B12A db 'Cookie',0 : DATA XREF: http_server_on_data_available+451
0000:1003B12D align 4
0000:1003B130 : char String[] align 4
0000:1003B133 db 'String',0 : DATA XREF: http_server_on_data_available+731
0000:1003B136 align 4
0000:1003B139 aPost align 4 db 'POST',0 : DATA XREF: http_server_on_data_available+861
0000:1003B13C align 4
0000:1003B13F aGet align 4 db 'GET',0 : DATA XREF: http_server_on_data_available+1081
0000:1003B142 align 4
0000:1003B145 : char aOk[] align 4
0000:1003B148 db 'OK',0 : DATA XREF: http_server_on_data_available+10C1
0000:1003B14B align 4
0000:1003B14E : char aKeepAlive[] align 4
0000:1003B151 aKeepAlive db 'Keep-Alive',0 : DATA XREF: http_server_on_data_available+1601
0000:1003B154 align 4
0000:1003B157 : char aKeepAlive_0 align 4
0000:1003B15A align 4
0000:1003B15D aConnection_0 db 'Connection',0 : DATA XREF: http_server_on_data_available+17E1
0000:1003B160 align 4
0000:1003B163 aClose db 'Close',0 : DATA XREF: http_server_on_data_available+181
0000:1003B166 align 4
0000:1003B169 : char aConnection_1[] align 4
0000:1003B16C align 4
0000:1003B16F aConnection_1 db 'Connection',0 : DATA XREF: http_server_on_data_available+18A1
0000:1003B172 align 4
0000:1003B175 db '0',0 : DATA XREF: http_server_on_data_available+1A11
0000:1003B178 align 4
0000:1003B17B : char aContentLength_0[] align 4
```

[illegible]

Duqu 2011

Duqu 2015

The authors also modified the “magic” two-byte value that identifies encrypted network traffic: “SH” was replaced with a more neutral and harder to trace “WW”:

0000:10019F65 0000:10019F66 0000:10019F67 0000:10019F68 0000:10019F69 0000:10019F70 0000:10019F71 0000:10019F72 0000:10019F73 0000:10019F74 0000:10019F75 0000:10019F76 0000:10019F77 0000:10019F78 0000:10019F79 0000:10019F80 0000:10019F81 0000:10019F82 0000:10019F83 0000:10019F84 0000:10019F85 0000:10019F86 0000:10019F87 0000:10019F88 0000:10019F89 0000:10019F90 0000:10019F91	push 0 push 0Ch lea ecx, [ebp+cmd_header] push ecx push ecx push eax call [eax+class_6.read_and_cut_data] mov eax, 'HS' add esp, 10h cmp word ptr [ebp+cmd_header], ax jnz short fail mov ecx, dword ptr [ebp+cmd_header+2] lea eax, [esi+class_43.cmd] mov [ebx], ecx xor ecx, ecx cmp [ebp+cmd_header+6], cl push eax push [esi+class_43.class6_input] call r1	1002B7D5 1002B7D6 1002B7D7 1002B7D8 1002B7D9 1002B7DA 1002B7DB 1002B7DC 1002B7DD 1002B7DE 1002B7DF 1002B7E0 1002B7E1 1002B7E2 1002B7E3 1002B7E4 1002B7E5 1002B7E6 1002B7E7 1002B7E8 1002B7E9 1002B7EA 1002B7EB 1002B7EC 1002B7ED 1002B7EE 1002B7EF 1002B7F0 1002B7F1 1002B7F2 1002B7F3 1002B7F4 1002B7F5 1002B7F6 1002B7F7 1002B7F8 1002B7F9 1002B7FA 1002B7FB 1002B7FC 1002B7FD	push 0 push 0Ch push edx mov ecx, [eax+20h] push eax call dword ptr [ecx] mov eax, 'WW' add esp, 10h cmp [ebp+var_C], ax jnz short loc_1002B851 mov [ebp+var_A] push [ebp+var_A]
---	--	--	---

Duqu 2011

Duqu 2015

Code that verifies the “magic” value in network traffic.

The chars are swapped due to little-endianness of data in x86/64 architectures.

Both Duqu and Duqu 2.0 use special structures to identify the interfaces of their plugins. The orchestrator also has one for the “core” plugin that is compiled in its code. The newer version has a slightly bigger table, hence more functions, and a different notation for describing the plugin features. Special strings (i.e. “A888A8>@”) describe each function’s signature. The older Duqu had contained similar strings in binary (unreadable) form.

0000:1003477C 0000:1003477D 0000:1003477E 0000:1003477F 0000:10034780 0000:10034781 0000:10034782 0000:10034783 0000:10034784 0000:10034785 0000:10034786 0000:10034787 0000:10034788 0000:10034789 0000:1003478A 0000:1003478B 0000:1003478C 0000:1003478D 0000:1003478E 0000:1003478F 0000:10034790 0000:10034791 0000:10034792 0000:10034793 0000:10034794 0000:10034795 0000:10034796 0000:10034797 0000:10034798 0000:10034799 0000:1003479A 0000:1003479B 0000:1003479C 0000:1003479D 0000:1003479E 0000:1003479F 0000:100347A0 0000:100347A1 0000:100347A2 0000:100347A3 0000:100347A4 0000:100347A5 0000:100347A6 0000:100347A7 0000:100347A8 0000:100347A9 0000:100347AA 0000:100347AB 0000:100347AC 0000:100347AD 0000:100347AE 0000:100347AF 0000:100347B0 0000:100347B1 0000:100347B2 0000:100347B3 0000:100347B4 0000:100347B5 0000:100347B6 0000:100347B7 0000:100347B8 0000:100347B9 0000:100347BA 0000:100347BB 0000:100347BC 0000:100347BD 0000:100347BE 0000:100347BF 0000:100347C0 0000:100347C1 0000:100347C2 0000:100347C3 0000:100347C4 0000:100347C5 0000:100347C6 0000:100347C7 0000:100347C8 0000:100347C9 0000:100347CA 0000:100347CB 0000:100347CC 0000:100347CD 0000:100347CE 0000:100347CF 0000:100347D0 0000:100347D1 0000:100347D2 0000:100347D3 0000:100347D4 0000:100347D5 0000:100347D6 0000:100347D7 0000:100347D8 0000:100347D9 0000:100347DA 0000:100347DB 0000:100347DC 0000:100347DD 0000:100347DE 0000:100347DF 0000:100347E0 0000:100347E1 0000:100347E2 0000:100347E3 0000:100347E4 0000:100347E5 0000:100347E6 0000:100347E7 0000:100347E8 0000:100347E9 0000:100347EA 0000:100347EB 0000:100347EC 0000:100347ED 0000:100347EE 0000:100347EF 0000:100347F0 0000:100347F1 0000:100347F2 0000:100347F3 0000:100347F4 0000:100347F5 0000:100347F6 0000:100347F7 0000:100347F8 0000:100347F9 0000:100347FA 0000:100347FB 0000:100347FC 0000:100347FD 0000:100347FE 0000:100347FF 0000:10034800 0000:10034801 0000:10034802 0000:10034803 0000:10034804 0000:10034805 0000:10034806 0000:10034807 0000:10034808 0000:10034809 0000:1003480A 0000:1003480B 0000:1003480C 0000:1003480D 0000:1003480E 0000:1003480F 0000:10034810 0000:10034811 0000:10034812 0000:10034813 0000:10034814 0000:10034815 0000:10034816 0000:10034817 0000:10034818 0000:10034819 0000:1003481A 0000:1003481B 0000:1003481C 0000:1003481D 0000:1003481E 0000:1003481F 0000:10034820 0000:10034821 0000:10034822 0000:10034823 0000:10034824 0000:10034825 0000:10034826 0000:10034827 0000:10034828 0000:10034829 0000:1003482A 0000:1003482B 0000:1003482C 0000:1003482D 0000:1003482E 0000:1003482F 0000:10034830 0000:10034831 0000:10034832 0000:10034833 0000:10034834 0000:10034835 0000:10034836 0000:10034837 0000:10034838 0000:10034839 0000:1003483A 0000:1003483B 0000:1003483C 0000:1003483D 0000:1003483E 0000:1003483F 0000:10034840 0000:10034841 0000:10034842 0000:10034843 0000:10034844 0000:10034845 0000:10034846 0000:10034847 0000:10034848 0000:10034849 0000:1003484A 0000:1003484B 0000:1003484C 0000:1003484D 0000:1003484E 0000:1003484F 0000:10034850 0000:10034851 0000:10034852 0000:10034853 0000:10034854 0000:10034855 0000:10034856 0000:10034857 0000:10034858 0000:10034859 0000:1003485A 0000:1003485B 0000:1003485C 0000:1003485D 0000:1003485E 0000:1003485F 0000:10034860 0000:10034861 0000:10034862 0000:10034863 0000:10034864 0000:10034865 0000:10034866 0000:10034867 0000:10034868 0000:10034869 0000:1003486A 0000:1003486B 0000:1003486C 0000:1003486D 0000:1003486E 0000:1003486F 0000:10034870 0000:10034871 0000:10034872 0000:10034873 0000:10034874 0000:10034875 0000:10034876 0000:10034877 0000:10034878 0000:10034879 0000:1003487A 0000:1003487B 0000:1003487C 0000:1003487D 0000:1003487E 0000:1003487F 0000:10034880 0000:10034881 0000:10034882 0000:10034883 0000:10034884 0000:10034885 0000:10034886 0000:10034887 0000:10034888 0000:10034889 0000:1003488A 0000:1003488B 0000:1003488C 0000:1003488D 0000:1003488E 0000:1003488F 0000:10034890 0000:10034891 0000:10034892 0000:10034893 0000:10034894 0000:10034895 0000:10034896 0000:10034897 0000:10034898 0000:10034899 0000:1003489A 0000:1003489B 0000:1003489C 0000:1003489D 0000:1003489E 0000:1003489F 0000:100348A0 0000:100348A1 0000:100348A2 0000:100348A3 0000:100348A4 0000:100348A5 0000:100348A6 0000:100348A7 0000:100348A8 0000:100348A9 0000:100348AA 0000:100348AB 0000:100348AC 0000:100348AD 0000:100348AE 0000:100348AF 0000:100348B0 0000:100348B1 0000:100348B2 0000:100348B3 0000:100348B4 0000:100348B5 0000:100348B6 0000:100348B7 0000:100348B8 0000:100348B9 0000:100348BA 0000:100348BB 0000:100348BC 0000:100348BD 0000:100348BE 0000:100348BF 0000:100348C0 0000:100348C1 0000:100348C2 0000:100348C3 0000:100348C4 0000:100348C5 0000:100348C6 0000:100348C7 0000:100348C8 0000:100348C9 0000:100348CA 0000:100348CB 0000:100348CC 0000:100348CD 0000:100348CE 0000:100348CF 0000:100348D0 0000:100348D1 0000:100348D2 0000:100348D3 0000:100348D4 0000:100348D5 0000:100348D6 0000:100348D7 0000:100348D8 0000:100348D9 0000:100348DA 0000:100348DB 0000:100348DC 0000:100348DD 0000:100348DE 0000:100348DF 0000:100348E0 0000:100348E1 0000:100348E2 0000:100348E3 0000:100348E4 0000:100348E5 0000:100348E6 0000:100348E7 0000:100348E8 0000:100348E9 0000:100348EA 0000:100348EB 0000:100348EC 0000:100348ED 0000:100348EE 0000:100348EF 0000:100348F0 0000:100348F1 0000:100348F2 0000:100348F3 0000:100348F4 0000:100348F5 0000:100348F6 0000:100348F7 0000:100348F8 0000:100348F9 0000:100348FA 0000:100348FB 0000:100348FC 0000:100348FD 0000:100348FE 0000:100348FF 0000:10034900 0000:10034901 0000:10034902 0000:10034903 0000:10034904 0000:10034905 0000:10034906 0000:10034907 0000:10034908 0000:10034909 0000:1003490A 0000:1003490B 0000:1003490C 0000:1003490D 0000:1003490E 0000:1003490F 0000:10034910 0000:10034911 0000:10034912 0000:10034913 0000:10034914 0000:10034915 0000:10034916 0000:10034917 0000:10034918 0000:10034919 0000:1003491A 0000:1003491B 0000:1003491C 0000:1003491D 0000:1003491E 0000:1003491F 0000:10034920 0000:10034921 0000:10034922 0000:10034923 0000:10034924 0000:10034925 0000:10034926 0000:10034927 0000:10034928 0000:10034929 0000:1003492A 0000:1003492B 0000:1003492C 0000:1003492D 0000:1003492E 0000:1003492F 0000:10034930 0000:10034931 0000:10034932 0000:10034933 0000:10034934 0000:10034935 0000:10034936 0000:10034937 0000:10034938 0000:10034939 0000:1003493A 0000:1003493B 0000:1003493C 0000:1003493D 0000:1003493E 0000:1003493F 0000:10034940 0000:10034941 0000:10034942 0000:10034943 0000:10034944 0000:10034945 0000:10034946 0000:10034947 0000:10034948 0000:10034949 0000:1003494A 0000:1003494B 0000:1003494C 0000:1003494D 0000:1003494E 0000:1003494F 0000:10034950 0000:10034951 0000:10034952 0000:10034953 0000:10034954 0000:10034955 0000:10034956 0000:10034957 0000:10034958 0000:10034959 0000:1003495A 0000:1003495B 0000:1003495C 0000:1003495D 0000:1003495E 0000:1003495F 0000:10034960 0000:10034961 0000:10034962 0000:10034963 0000:10034964 0000:10034965 0000:10034966 0000:10034967 0000:10034968 0000:10034969 0000:1003496A 0000:1003496B 0000:1003496C 0000:1003496D 0000:1003496E 0000:1003496F 0000:10034970 0000:10034971 0000:10034972 0000:10034973 0000:10034974 0000:10034975 0000:10034976 0000:10034977 0000:10034978 0000:10034979 0000:1003497A 0000:1003497B 0000:1003497C 0000:1003497D 0000:1003497E 0000:1003497F 0000:10034980 0000:10034981 0000:10034982 0000:10034983 0000:10034984 0000:10034985 0000:10034986 0000:10034987 0000:10034988 0000:10034989 0000:1003498A 0000:1003498B 0000:1003498C 0000:1003498D 0000:1003498E 0000:1003498F 0000:10034990 0000:10034991 0000:10034992 0000:10034993 0000:10034994 0000:10034995 0000:10034996 0000:10034997 0000:10034998 0000:10034999 0000:1003499A 0000:1003499B 0000:1003499C 0000:1003499D 0000:1003499E 0000:1003499F 0000:100349A0 0000:100349A1 0000:100349A2 0000:100349A3 0000:100349A4 0000:100349A5 0000:100349A6 0000:100349A7 0000:100349A8 0000:100349A9 0000:100349AA 0000:100349AB 0000:100349AC 0000:100349AD 0000:100349AE 0000:100349AF 0000:100349B0 0000:100349B1 0000:100349B2 0000:100349B3 0000:100349B4 0000:100349B5 0000:100349B6 0000:100349B7 0000:100349B8 0000:100349B9 0000:100349BA 0000:100349BB 0000:100349BC 0000:100349BD 0000:100349BE 0000:100349BF 0000:100349C0 0000:100349C1 0000:100349C2 0000:100349C3 0000:100349C4 0000:100349C5 0000:100349C6 0000:100349C7 0000:100349C8 0000:100349C9 0000:100349CA 0000:100349CB 0000:100349CC 0000:100349CD 0000:100349CE 0000:100349CF 0000:100349D0 0000:100349D1 0000:100349D2 0000:100349D3 0000:100349D4 0000:100349D5 0000:100349D6 0000:100349D7 0000:100349D8 0000:100349D9 0000:100349DA 0000:100349DB 0000:100349DC 0000:100349DD 0000:100349DE 0000:100349DF 0000:100349E0 0000:100349E1 0000:100349E2 0000:100349E3 0000:100349E4 0000:100349E5 0000:100349E6 0000:100349E7 0000:100349E8 0000:100349E9 0000:100349EA 0000:100349EB 0000:100349EC 0000:100349ED 0000:100349EE 0000:100349EF 0000:100349F0 0000:100349F1 0000:100349F2 0000:100349F3 0000:100349F4 0000:100349F5 0000:100349F6 0000:100349F7 0000:100349F8 0000:100349F9 0000:100349FA 0000:100349FB 0000:100349FC 0000:100349FD 0000:100349FE 0000:100349FF 0000:10034A00 0000:10034A01 0000:10034A02 0000:10034A03 0000:10034A04 0000:10034A05 0000:10034A06 0000:10034A07 0000:10034A08 0000:10034A09 0000:10034A0A 0000:10034A0B 0000:10034A0C 0000:10034A0D 0000:10034A0E 0000:10034A0F 0000:10034A10 0000:10034A11 0000:10034A12 0000:10034A13 0000:10034A14 0000:10034A15 0000:10034A16 0000:10034A17 0000:10034A18 0000:10034A19 0000:10034A1A 0000:10034A1B 0000:10034A1C 0000:10034A1D 0000:10034A1E 0000:10034A1F 0000:10034A20 0000:10034A21 0000:10034A22 0000:10034A23 0000:10034A24 0000:10034A25 0000:10034A26 0000:10034A27 0000:10034A28 0000:10034A29 0000:10034A2A 0000:10034A2B 0000:10034A2C 0000:10034A2D 0000:10034A2E 0000:10034A2F 0000:10034A30 0000:10034A31 0000:10034A32 0000:10034A33 0000:10034A34 0000:10034A35 0000:10034A36 0000:10034A37 0000:10034A38 0000:10034A39 0000:10034A3A 0000:10034A3B 0000:10034A3C 0000:10034A3D 0000:10034A3E 0000:10034A3F 0000:10034A40 0000:10034A41 0000:10034A42 0000:10034A43 0000:10034A44 0000:10034A45 0000:10034A46 0000:10034A47 0000:10034A48 0000:10034A49 0000:10034A4A 0000:10034A4B 0000:10034A4C 0000:10034A4D 0000:10034A4E 0000

The Duqu C&C code makes use of small image files to hide its communications over unencrypted channels, i.e. HTTP. The original Duqu used a JPEG file, and known versions of Duqu 2.0 use a similar JPEG file as well as a new, larger GIF file. Also, the layout of the data section did not change much: the image data is preceded by short AES encryption keys (string "sh123456" in Duqu, two binary DWORDs in Duqu 2.0) followed by the LZO version string "2.03".

[illegible]

Image data used for hiding C&C communication in them: JPEG in Duqu, similar JPEG in Duqu Bet and GIF in a different version of Duqu Bet. Note the preceding LZQ version string "2.03" and encryption keys.

The large number of similarities between the Duqu 2011 code and the new Duqu 2.0 samples indicates that the new code represents a new iteration of the malware platform. The new version could not have been built without access to the 2011 Duqu source code. Hence, we conclude that the authors are the same or working together.

VICTIMS OF DUQU 2.0

Victims of Duqu 2.0 were found in several places, including western countries, the Middle East and Asia. The actor appears to compromise both final and utilitarian targets, which allow them to improve their cyber capabilities.

Most of the final targets appear to be similar to their 2011 goals – which is to spy on Iran’s nuclear program. Some of the new 2014-2015 infections are linked to the P5+1 events and venues related to the negotiations with Iran about a nuclear deal. The threat actor behind Duqu appears to have launched attacks at the venues for some of these high level talks. In addition to the P5+1 events, the Duqu 2.0 group has launched a similar attack in relation to the 70th anniversary event of the liberation of Auschwitz-Birkenau.

The other type of targets for the new attacks are what we call “utilitarian” targets. These are companies that the attackers compromise to improve their cyber capabilities. For instance, in 2011, the attackers compromised a certificate authority in Hungary; obviously, this would allow them to generate digital certificates, which can be further used to sign malware samples. The same pattern can be seen with the Duqu 2.0 infections. Some of the companies infected with Duqu 2.0 operate in the sector of Industrial Control Systems as well as industrial computers.

ATTRIBUTION

As usual, attribution of cyberattacks over the Internet is a difficult task. In the case of Duqu, the attackers use multiple proxies and jumping points to mask their connections. This makes tracking an extremely complex problem.

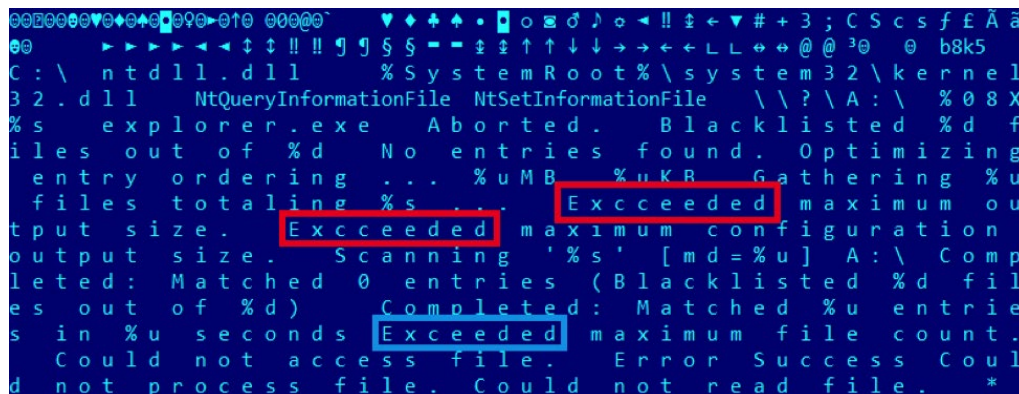
Additionally, the attackers have tried to include several false flags throughout the code, designed to send researchers in the wrong direction. For instance, one of the drivers contains the string “ugly.gorilla”, which obviously refers to ⁹Wang Dong, a Chinese hacker believed to be associated with the APT1/Comment Crew. The usage of the Camellia cypher in the MSI VFSes, previously seen in APT1-associated Poison Ivy samples is another false flag planted by the attackers to make researchers believe they are dealing with APT1 related malware. The “romanian.antihacker” string used in the “portserv.sys” driver is probably designed to mimic “w00tw00t.at.blackhats.romanian.anti-sec” requests that are often seen in server logs or simply point to an alleged Romanian origin of the attack. The usage of rare compression algorithms can also be deceptive. For instance, the LZJB algorithm used in some of the samples is rarely seen in malware samples; it has been used by MiniDuke which we reported in early 2013.

Nevertheless, such false flags are relatively easy to spot, especially when the attacker is extremely careful not to make any other mistakes.

During our 2011 analysis, we noticed that the logs collected from some of the proxies indicated the attackers appear to work less on Fridays and didn’t appear to work at all on Saturdays, with their regular work week starting on Sunday. They also compiled binaries on January 1st, indicating it was probably a normal work day for them. The compilation timestamps in the binaries seemed to suggest a time zone of GMT+2 or GMT+3. Finally, their attacks would normally occur on Wednesdays, which is why we originally called them the “Wednesday Gang”. While the 2014 attack against Kaspersky Lab also took place on a Wednesday, the gang made huge OPSEC improvements compared to their older 2011 operations, including faking all the timestamps in PE files, removing the debug paths and internal module names for all plugins.

The 2014 Duqu 2.0 binaries contain several strings in almost perfect English but one of them has a minor mistake indicating the involvement of non-native speakers. The usage of “Exccceeded” instead of “Exceeded” in the file-harvesting module of Duqu 2.0 is the only language mistake we observed.

9 <http://www.fbi.gov/wanted/cyber/wang-dong/view>



Misspelling of the word "Exceeded" in Duqu 2.0.

Most interesting, one of the victims appear to have been infected both by the Equation Group and by the Duqu group at the same time; this suggests the two entities are different and competing with each other to obtain information from this victim.

CONCLUSIONS

During the 2011 Duqu attacks, we concluded that its main purpose could have been to spy on Iran's nuclear program. Some of the victims appear to have been "utility", such as one certificate authority in Hungary, which was compromised by Duqu and ultimately that led to its discovery. The group behind Duqu hacks these "utility" victims in order to gain certain technical abilities such as signing their malware with trusted certificates or to serve as platforms for further attacks.

The 2014/2015 Duqu 2.0 appears to be a massive improvement over the older “Tilded” platform, although the main orchestrator and C&C core remains largely unchanged. Back in 2011 we pointed out to the usage of ¹⁰Object Oriented C as an unusual programming technique. The 2014 version maintains the same core, although some new objects in C++ have been added. The compiler used in the 2014 is newer and it results in different code optimizations. Nevertheless, the core remains the same in functionality and it is our belief it could not have been created by anyone without access to the original Duqu source code. Since these have never been made public and considering the main interest appears to have remained the same, we conclude the attackers behind Duqu and Duqu 2.0 are the same.

The targeting of Kaspersky Lab represents a huge step for the attackers and an indicator of how quick the cyber-arms race is escalating. Back in 2011 and 2013 respectively, ¹¹RSA and ¹²Bit9, were hacked by Chinese-language APT groups, however, such incidents were considered rare. In general, an attacker risks a lot targeting a security company – because they can get caught and exposed. The exact reason why Kaspersky Lab was targeted is still not clear – although the attackers did seem to focus on obtaining information about Kaspersky’s future technologies, Secure OS, anti-APT solutions, KSN and APT research.

10 <https://securelist.com/blog/research/32354/the-mystery-of-duqu-framework-solved-7/>

11 <https://blogs.rsa.com/anatomy-of-an-attack/>

12 <https://blog.bit9.com/2013/02/08/bit9-and-our-customers-security/>

From a threat actor point of view, the decision to target a world-class security company must be quite difficult. On one hand, it almost surely means the attack will be exposed – it's very unlikely that the attack will go unnoticed. So the targeting of security companies indicates that either they are very confident they won't get caught, or perhaps they don't care much if they are discovered and exposed. By targeting Kaspersky Lab, the Duqu attackers have probably taken a huge bet hoping they'd remain undiscovered; and lost.

For a security company, one of the most difficult things is to admit falling victim to a malware attack. At Kaspersky Lab, we strongly believe in transparency, which is why we are publishing the information herein. For us, the security of our users remains the most important thing – and we will continue to work hard to maintain your trust and confidence.

REFERENCES

1. Duqu: A Stuxnet-like malware found in the wild <https://www.crysys.hu/publications/files/bencsathPBF11duqu.pdf>
2. Duqu: The Precursor to the next Stuxnet http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/w32_duqu_the_precursor_to_the_next_stuxnet.pdf
3. The Mystery of Duqu: Part One <https://securelist.com/blog/incidents/31177/the-mystery-of-duqu-part-one-5/>
4. The Mystery of Duqu: Part Two <https://securelist.com/blog/incidents/31445/the-mystery-of-duqu-part-two-23/>
5. The Mystery of Duqu: Part Three <https://securelist.com/blog/incidents/31486/the-mystery-of-duqu-part-three-9/>
6. The Mystery of Duqu: Part Five <https://securelist.com/blog/incidents/31208/the-mystery-of-duqu-part-five-6/>
7. The Mystery of Duqu: Part Six (The Command and Control Servers) <https://securelist.com/blog/incidents/31863/the-mystery-of-duqu-part-six-the-command-and-control-servers-36/>
8. The Mystery of Duqu: Part Ten <https://securelist.com/blog/incidents/32668/the-mystery-of-duqu-part-ten-18/>
9. The Mystery of Duqu Framework Solved <https://securelist.com/blog/research/32354/the-mystery-of-duqu-framework-solved-7/>
10. The Duqu Saga Continues <https://securelist.com/blog/incidents/31442/the-duqu-saga-continues-enter-mr-b-jason-and-tvs-dexter-22/>



[Securelist](#), the resource for Kaspersky Lab experts' technical research, analysis and thoughts



[Kaspersky Lab B2C Blog](#)



[Kaspersky Lab security news service](#)



[Eugene Kaspersky Blog](#)



[Kaspersky Lab B2B Blog](#)



[Kaspersky Lab Academy](#)

Kaspersky Lab, Moscow, Russia
www.kaspersky.com

All about Internet security:
www.securelist.com

Find a partner near you:
www.kaspersky.com/buyoffline

Kaspersky Lab HQ

39A/3 Leningradskoe Shosse
Moscow, 125212
Russian Federation

[More contact details](#)

Tel: +7-495-797-8700
Fax: +7-495-7978709

Follow us



[Twitter.com/Kaspersky](https://twitter.com/Kaspersky)



[Facebook.com/Kaspersky](https://facebook.com/Kaspersky)



[Youtube.com/Kaspersky](https://youtube.com/Kaspersky)