Analysis and Diversion of Duqu’s Driver

Aurélien Thierry (aurelien.thierry@inria.fr), Fabrice Sabatier, Guillaume Bonfante, Jean-Yves Marion

Malware 2013, October 23rd
Working on...

Analyzing and classifying malwares that:
- Hide their code (obfuscation)
- Detect analysis (anti-debug mechanisms)
- Are self-modifying (through packing)

Our goals:
- (Towards) Better classification
- Automate some time-consuming tasks of reverse-engineers
- Provide reverse-engineering toolkits (IDA plugins and more)
Duqu: first discovered in September 2011 by CrySyS
Detected post mortem
Studies (Symantec\(^1\), Kaspersky\(^2\), Eset, ...) done on the payload and injection mechanisms link it to Stuxnet

We work on a malware detector and got Stuxnet and Duqu samples...
We detect Duqu’s decrypted DLL, knowing Stuxnet’s\(^3\)

---
\(^1\)Symantec. W32.Duqu : The precursor to the Next Stuxnet, October 2011
\(^2\)Kaspersky, The mystery of Duqu
\(^3\)Recognition of binary patterns by Morphological analysis, REcon 2012
Working on... Stuxnet and Duqu

We work on a malware detector and got Stuxnet and Duqu samples...

- We detect Duqu’s decrypted DLL, knowing Stuxnet’s

But the detection was done post-mortem

Could we prevent an infection and how?

\(^4\) Recognition of binary patterns by Morphological analysis, REcon 2012
Duqu: Infection timeline

- Exploitation of a 0-day kernel vulnerability when opening a *word* document
- A DLL is written encrypted on disk
- A driver is installed
- Only the driver is written unencrypted on the hard drive
- At reboot the driver monitors modules launched by the OS
- Once *services.exe* is launched, the DLL is decrypted and injected into it
Our work

For detection we need the decrypted DLL, but before it has been used. We could do that when it is injected.

- We want to monitor launched processes...
- ...which is how Duqu injects itself in services.exe!
- Why not use Duqu’s source code?

So let’s reverse and decompile Duqu’s source code (in C++). And use it to create a defensive driver able to detect Duqu live.
IDA’s decompilation plugin on Duqu’s driver

```
.text:0012F36 ; ------------------ S U B R O U T I N E ------------------------------
.text:0012F36
.text:0012F36
.text:0012F36
.text:0012F36
.text:0012F36 ; signed int __cdecl sub_12F36(int a1, PINAGE_DOS_HEADER a2, int a3)
.text:0012F36 sub_12F36 proc near ; CODE XREF: sub_11670+161p
.text:0012F36            ; sub_11B30+2Efp ...
.text:0012F36 arg_0    - dword ptr 4
.text:0012F36 arg_4    - dword ptr 8
.text:0012F36 arg_8    - dword ptr 0Ch
.text:0012F36
.text:0012F36 push    esi
.text:0012F37 mov     esi, [esp+4+arg_4]
.text:0012F38 mov     eax, '2M'
.text:0012F39 cmp     [esi], ax
.text:0012F3A jz     short loc_12F4A
.text:0012F3B
.text:0012F3B loc_12F45: ; CODE XREF: sub_12F36+27j
.text:0012F3B xor    eax, eax
.text:0012F3C inc    eax
.text:0012F3D pop    esi
.text:0012F3E ret
.text:0012F3F
.text:0012F40 ; ------------------ SUBROUTINE -------------------------------------
.text:0012F40 loc_12F4A: ; CODE XREF: sub_12F36+DTj
.text:0012F40 mov    eax, [esi+8Ch]
.text:0012F41 add    eax, esi
.text:0012F42 mov    ecx, [eax]
.text:0012F43 xor    ecx, 0F750F284h
.text:0012F44 cmp    ecx, 0F750B7D4h
.text:0012F45 jnz    short loc_12F45
.text:0012F46 movzx   ecx, word ptr [eax+8]
.text:0012F47 push   ebx
.text:0012F48 push   edi
.text:0012F49 mov    edi, ecx
.text:0012F4A mov    edx, 5000h
.text:0012F4B xor    edi, 594Fh
.text:0012F4C mov    ebx, edx
.text:0012F4D cmp    bx, dl
.text:0012F4E jnz    short loc_12F9E
.text:0012F4F mov    ecx, 5900h
.text:0012F50 xor    ecx, [eax+10h]
.text:0012F51 cmp    ecx, dx
.text:0012F52 jnz    short loc_13001
.text:0012F53 mov    ecx, 0E0h
```

IDA’s decompilation plugin on Duqu’s driver

```c
signed int __cdecl sub_12F36(int a1, int a2, int a3)
{
    int v4; // eax@3
    unsigned __int16 v5; // cx@4
    int v6; // ecx@7
    int v7; // edx@7
    __int16 v8; // dx@12
    if ( *(WORD *)a2 != 'ZM' ) return 1;
    v4 = a2 + *(DWORD *)(a2 + 60);
    if ( (*(DWORD *)v4 ^ 0xF750F284) != 0xF750B7D4 ) return 1;
    v5 = *(WORD *)(v4 + 4);
    if ( 0x5803 == (v5 ^ 0x594F) )
        if ( (*(WORD *)(v4 + 24) ^ 0x5908) == 0x5803 && *(WORD *)(v4 + 20) == 224 )
            v6 = a1;
            *(DWORD *)a1 = 0;
            v7 = v4 + 120;
    LABEL12:
        *(DWORD *)(v6 + 16) = v7;
        *(DWORD *)(v6 + 26) = *(DWORD *)(v4 + 80);
        *(DWORD *)(v6 + 20) = *(WORD *)(v4 + 20) + v4 + 24;
        v8 = *(WORD *)(v4 + 6);
        *(DWORD *)(v6 + 12) = v4;
        *(DWORD *)(v6 + 4) = a3;
        *(WORD *)(v6 + 24) = v8;
        *(DWORD *)(v6 + 8) = a2;
        return 0;
    }
} // (...) 

Not too friendly...
```
IDA’s decompilation plugin on Duqu’s driver

How to get readable code?
- Recognize and rename constants
- Identify structures and types

How to compile the code? The compiled version should be close to the original binary.
- Find missing calling conventions
- Try many compilation options

Now the source (C++ and ASM) can be analyzed and reused.
if ((pFileHeader→Machine ∧ 0x594F) == (IMAGE_PE_i386_MACHINE ∧ 0x594F)) // 32 bits
    {
        // traitement spécifique
        Continue:
        // traitement commun
        return STATUS_SUCCESS;
    }
else if ((pFileHeader→Machine ^ 0xDE67) == (IMAGE_PE_x86_MACHINE ^ 0xDE67) && (pOptionHeader→Magic ∧ 0x5A08) == IMAGE_PE32_PLUS_MAGIC // PE+: 64 bits && pFileHeader→SizeOfOptionalHeader == 0xF0 )
    {
        pPEData→Status = 1;
        pExportTableRVA = pOptionHeader→DataDirectory [IMAGE_DIRECTORY_ENTRY].VirtualAddress;
        goto Continue;
    }
return STATUS_WAIT_1;

- A Xor 0x5A08 is missing!
- Unable to parse a 64 bits binaries = unable to inject in 64 bits

--

5Other samples of the driver were patched and were signed.
Preparing the injection

The driver’s aim is to inject the decrypted DLL into services.exe

- To avoid detection (services.exe is probably white-listed)
- It needs to allocate memory into any process
- and changes the access rights of the RAM (RW→RWX or RX→RWX)
- It checks that the kernel functions used for injection are not hooked (for defensive purposes)
The driver checks that the functions are in the kernel’s memory addresses (not in user space)

An integrity check (common mask on 11 bytes) is then done on both functions:

### ZwAllocateVirtualMemory

```
.text:00405DDC   B8 11 00 00 00   mov eax 11h
.text:00405DE1   8D 54 24 04  lea edx [esp+ProcessHandle]
.text:00405DE5   9C               pushf
.text:00405DE6   6A 08              push 8
.text:00405DE8   E8 B9 20 00 00  call sub_407EA6
.text:00405DED   C2 14 00            retn 14h
```

### ZwProtectVirtualMemory

```
.text:00406882   B8 89 00 00 00   mov eax 89h
.text:00406887   8D 54 24 04  lea edx [esp+ProcessHandle]
.text:0040688B   9C               pushf
.text:0040688C   6A 08              push 8
.text:0040688E   E8 13 16 00 00  call sub_407EA6
.text:00406893   C2 14 00            retn 14h
```
Injection at boot (simplified)

OS
Injection at boot (simplified)

OS Launches

Duqu’s driver (nfrd965.sys)

Notification: services.exe

Notification: kernel.dll
Injection at boot (simplified)

OS
Launches
Loads

Duqu’s driver (nfrd965.sys)

Notification: services.exe
Notification: kernel.dll

services.exe

data
text

Entrypoint code

01012475 push 70h
01012477 push 10015E0h
...

Launches OS
Loads

Notification: services.exe
Notification: kernel.dll

Duqu’s driver (nfrd965.sys)
Injection at boot (simplified)

OS
  
  Launches
  
  Loads

Duqu’s driver (nfrd965.sys)
  
  Notification: services.exe
  
  Notification: kernel.dll

services.exe
  
  data
    
    PE file
    
    Decrypted DLL

  text
    
    Entry point code
      
      01012475  push   70h
      01012477  push   10015E0h
      
      ...

Injects and restores

Launches OS

Loads OS

Injects and restores Duqu’s driver (nfrd965.sys)

Notices: services.exe, kernel.dll

Injects and restores Decrypted DLL
Injection at boot (simplified)

1. OS Launches
   Loads
   Services

2. Duqu’s driver (nfrd965.sys)

   Notification: services.exe
   Notification: kernel.dll

   Injects and restores
   RX -> RWX

   Decrypted DLL
   PE file

   data
   text

   Entry point code
   01012475 push 70h
   01012477 push 10015E0h

   ...
Injection at boot (simplified)

1. Stores
2. RX -> RWX

OS Launches

Duqu’s driver (nfrd965.sys)

Notification: services.exe
Notification: kernel.dll

services.exe

data

PE file
Decrypted DLL

text

Entrypoint code

01012475 push 70h
01012477 push 10015E0h
...

Injects and restores
Injection at boot (simplified)

OS  Launches  Loads

Duqu’s driver (nfrd965.sys)

Notification: services.exe  Notification: kernel.dll

services.exe

data

PE file

Decrypted DLL

text

Altered entrypoint

01012475  mov  eax,0A18BDh
0101247a  call  eax

1 Stores

2 Modifies

1 RX -> RWX

Infects and restores
Injection at boot (simplified)

1. OS

   Executes

   Duqu’s driver (nfrd965.sys)

      Notification: services.exe
      Notification: kernel.dll

   services.exe

      data

         PE file

         Decrypted DLL

      text

         Altered entrypoint

            mov eax,0A18BDh
            call eax

            ...

Decompilation Functional analysis Injection Defensive version Self-modification
Injection at boot (simplified)

OS

Duqu’s driver (nfrd965.sys)

Notification: services.exe
Notification: kernel.dll

services.exe

data
PE file
Decrypted DLL

text
Altered entrypoint

01012475 mov eax,0A18BDh
0101247a call eax

Payload execution

1
2

13 / 23
Injection at boot (simplified)

1. OS
2. Payload execution
3. Entrypoint restoration
Injection at boot (simplified)

1. OS executes services.exe
2. Payload execution
3. Entrypoint restoration
4. Jump: normal execution

Duqu’s driver (nfrd965.sys)
Notification: services.exe
Notification: kernel.dll

services.exe
- PE file
- Decrypted DLL
- Entrypoint code
  - 01012475 push 70h
  - 01012477 push 10015E0h
  - ...

Decrypted DLL
01012475
01012477
Towards a defensive version: How Duqu operates

- Uses an exploit to install a driver
- Sets up notifications each time a module is loaded
- Injects its payload into `services.exe`

We want to detect Duqu when it is injected, before the payload is executed.
Detecting the entrypoint’s alteration

- Duqu sets up notifications and waits to inject into services.exe

We could:
- Have a defensive driver with notifications on loaded modules
- Store and monitor the entrypoint of notified processes
Detecting the entrypoint’s alteration

- Duqu sets up notifications and waits to inject into `services.exe`

We could:
- Have a defensive driver with notifications on loaded modules
- Store and monitor the entrypoint of notified processes

Duqu’s driver (nfrd965.sys)

Notification: `services.exe`

Notification: `kernel.dll`

`services.exe`

- Data
  - PE file
  - Decrypted DLL

- Text
  - Altered entrypoint
    - `01012475 mov eax,0A18BDh`
    - `0101247a call eax`
    - ...
Detecting the entrypoint’s alteration

- Duqu sets up notifications and waits to inject into services.exe

We could:
- Have a defensive driver with notifications on loaded modules
- Store and monitor the entrypoint of notified processes

![Diagram](image-url)
Detecting the entrypoint’s alteration

- Duqu sets up notifications and waits to inject into services.exe

We could:
- Have a defensive driver with notifications on loaded modules
- Store and monitor the entrypoint of notified processes

Duqu’s driver (nfrd965.sys)
- Notification: services.exe
- Notification: kernel.dll

Detect the alteration
- Terminates services.exe

Decrypted DLL

Stores
- Modifies

PE file

Decrypted DLL

Altered entrypoint

mov eax,0A18BDh

mov eax,0

call eax

Stores

Modifies
Defense scenario: kill services.exe

OS

Launches

Duqu’s driver (nfrd965.sys)

- Notification: services.exe
- Notification: kernel.dll

Launches OS
Defense scenario: kill services.exe

OS \(\xrightarrow{\text{Launches}}\) Duqu’s driver (nfrd965.sys) \(\xleftarrow{\text{Notification}}\) Defensive driver

Notification: services.exe
Notification: kernel.dll

Notification: services.exe
Notification: kernel.dll
Defense scenario: kill services.exe

OS Launches
Loads

Duqu’s driver (nfrd965.sys)
Notification: services.exe
Notification: kernel.dll

services.exe
data

text
Entrypoint code
01012475 push 70h
01012477 push 10015E0h
...

Defensive driver
Notification: services.exe
Notification: kernel.dll
Defense scenario: kill services.exe

OS
Launches
Loads

Injection

Duqu’s driver (nfrd965.sys)

Notification: services.exe

Notification: kernel.dll

Defensive driver

Notification: services.exe

Notification: kernel.dll

services.exe

Data

PE file

Decrypted DLL

text

Entrypoint code

01012475 push 70h
01012477 push 10015E0h

...
Defense scenario: kill services.exe

- OS launches and loads services.exe.
- Duqu’s driver (nfrd965.sys) injects code and stores the entrypoint.
- Defensive driver detects the injection and sends notifications.
- PE file contains decrypted DLL.
- Entry point code:
  - 01012475 push 70h
  - 01012477 push 10015E0h
  - ...
Defense scenario: kill services.exe

OS

Launches

Loads

Duqu’s driver (nfrd965.sys)

Notification: services.exe

Notification: kernel.dll

Defensive driver

Notification: services.exe

Notification: kernel.dll

Injection

Injection

Stores the entrypoint

Altered entrypoint

01012475 mov eax,0A18BDh
0101247a call eax

...
Defense scenario: kill services.exe

1. OS Launches
2. Notification: services.exe
3. Notification: kernel.dll
4. Notification: services.exe
5. Notification: kernel.dll
6. Injection
7. Altered entrypoint
8. PE file
9. Decrypted DLL
10. Stores the entrypoint
11. Alteration
12. Checks the entrypoint... detects injection!!
Demonstration

For demonstration and visualization purposes:

- We launch the drivers on demand (first Duqu’s, then the defensive)
- We renamed calc.exe into services.exe
- We patched Duqu’s driver so it launches in debug mode:

  The defensive driver detects the alteration of services.exe’s entrypoint and terminates the process
  The detection occurs when kernel32.dll is loaded
Demonstration

<table>
<thead>
<tr>
<th>ProcessImageInformation: PEB=0x7ffdf000 ImageBaseAddress=0x01000000 ProcessId=0xaa4</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParsePEModule: PE</td>
</tr>
<tr>
<td>Entry point bytes at 0x01012475: 0x6a 0x70 0x68 0xe0 0x15 0x00 0x01 0xe8 (stored)</td>
</tr>
<tr>
<td>ProcessImageInformation: Entry point=0x01012475 DataDirectory=0x01000168</td>
</tr>
<tr>
<td>ProcessImageName: \Device\HDV1\Documents and Settings\fabric\Bureau\services.exe</td>
</tr>
<tr>
<td>ProcessImageName: save processID=0xaa4</td>
</tr>
<tr>
<td>CreateProcessNotify: Create=1 0x748=&gt;0xaa4</td>
</tr>
<tr>
<td>CreateProcessNotify: Entry point checksum=0x49af1bf2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LoadImageNotifyRoutine: ImageBaseAddress=0x7c8d0000 ProcessId=0xb88</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParsePEModule: PE</td>
</tr>
<tr>
<td>Entry point bytes at 0x7c80b64e: 0x8b 0xff 0x55 0x8b 0xec 0x83 0x7d 0x0c</td>
</tr>
<tr>
<td>LoadImageNotifyRoutine: Entry point=0x7c80b64e DataDirectory=0x7c800168</td>
</tr>
<tr>
<td>-&gt; Verify services.exe : Entry point at 0x01012475:</td>
</tr>
<tr>
<td>0xb8 0xbd 0x18 0x0a 0x00 0xff 0xd0 0xe8</td>
</tr>
</tbody>
</table>

!!!>

Checksum error ! ! ! ! ! ! ! ! ! !

Terminating services.exe
drvTerminateProcess( 2952 )
Perspectives

We detect the alteration of the entrypoint and prevent the execution of the payload.

What’s next? We’d like to:
- Scan the memory looking for EXE
- Scan them with the detector

This POC was about Duqu’s infection into services.exe.
- It works because Duqu does its injection over the course of two notifications
- It could be adapted to detect alterations that result from other attacks (and monitor every process)
Duqu’s injection has a self-modifying behavior...
How to analyze a self-modifying binary?

- Change of “code wave” once a previously written address is executed

Simple static analysis usually fails to find modified code

- Use of dynamic analysis (Pin tools)
- Tainting all written and executed memory addresses
- During dynamic analysis, do a static analysis on the visible parts
For now: packers

- Packers use self-modification to obfuscate binaries
- Original binary is only readable after some execution

We packed hostname.exe and tried to find its code in packed binaries.

TELock: The code of hostname.exe is entirely found in the 17th (last) wave

ACprotect: 256 waves

ASpack: 3 waves

Detection on (dynamic + static) traces allows to detect the packed variant of a binary.
Binviz: Visualize waves

For analysis we want to browse waves of a trace (hostname.exe packed by ASPack)

- Standalone tool using dot and the trace (generated by Pin)

On hostname.exe packed by ASPack
For analysis we want to browse waves of a trace (hostname.exe packed by ASPack)

- Standalone tool using dot and the trace (generated by Pin)

On hostname.exe
Conclusion

- Decompiling and analyzing Duqu’s driver: its injection into services.exe
- Building a defensive driver from Duqu’s, able to detect and prevent an infection
- Working on detection and analysis of self-modifying binaries

Thank you

Any question? (aurelien.thierry@inria.fr)