Report on Duqu: A collection of computer Malware

Aril Johannes Schultzen

September 10, 2015

Contents

Abstract

This report on Duqu (a collection of computer malware) was an assignment given in the course UNIK4740. It is mainly based on *Duqu: A Stuxnet-like malware found in the wild.*[1] by Boldizsár Bencsáth et al (October 2011) and *W32.Duqu: The precursor to the next Stuxnet*[5] by Symantec (November 2011).

Chapter 1

Introduction

Duqu is a collection of malware discovered by The Laboratory of Cryptography and System Security (CrySyS) of the Budapest University of Technology and Economics in Hungary. They analyzed it and named it Duqu from the prefix " DQ" that the key logger use to name its files. It is an interesting piece of code despite it being anything but technically astonishing. It is however interesting because of its similarities with *Stuxnet* and *Stuxnets* modular design and how these modules combined can be used to create a targeted thread to control systems in nuclear facilities. It is believed that the creator(s) of Duqu also created *Stuxnet* or at least had access to *Stuxnets* source code.

Chapter 2

Prerequisite knowledge

Though not a technical marvel, Duqu exploits numerous mechanisms and features in the Winodws Operating system. Some of these mechanisms will be explained in this section and should be understood before venturing further into this report.

2.1 DLL

A DLL (Dynamic Link Libraries) is Microsoft's implementation of the *shared library* concept used in both Windows and the OS/2 operating system. It can contain both code and data and shares it's file format with the Windows Executable file (EXE). A DLL can be used by multiple programs at the same time. The idea is that it promotes reuse of code while achieving higher memory efficiency.

2.1.1 DLL Injecting and Hooking

A DLL injection is when a DLL is injected into the address space of an already running process. By using an already running process as opposed to making a new process to run the DLL, detection is far easier to avoid [3] DLL Hooking or API Hooking is a range of techniques of intercepting function calls or messages in an operating system, changing the behavior of it [2]. In any case, both DLL injecting and Hooking requires administrative privileges. This means that you need complete control over the victims system before these techniques are employed. In other words, the techniques are used not to gain access, but to avoid detection.

2.1.2 Exports

A DLL contains an *exports table* which contains the name of every function that the DLL exposes (or exports) to other executables. A Java analogy would be the `public` modifier.

2.2 Drivers

A Driver (also known as Device driver) an abstraction layer that provides a software interface to the hardware. A driver can either be written in kernel mode or user mode. When running in kernel mode, the driver has access to every resource and all hardware, this also means that every CPU instruction can be executed and every memory address can accessed. An application written in user-mode can not directly access hardware or memory but has to use APIs instead. This isolation makes a crash in user-mode recoverable instead of catastrophic as in a kernel-mode.

2.2.1 Driver Signing

Digital signatures are used to demonstrate the authenticity of the producer of the driver and the integrity of the code. This can be used to protect an operating system against malicious code, for example, Microsoft Windows 7 is by default set to reject unsigned drivers.

2.3 The Windows registry

The registry is a database used in Microsoft Windows to store settings and options. It can be considered an alternative to the use of INI files. The use of the registry is not compulsory.

2.4 RPC

Remote Procedure Calls (RPC) is a system that allows programmers to write distributed software without worrying about the underlying network code. It is most often used to create a server/client model. [6]

Chapter 3

Targets

According to *W32.Duqu: The precursor to the next Stuxnet*[5] by Symantec (November 2011), Duqu is the precursor to Stuxnet. It is believed that it was made to gather information even tough with a different payload, it could do anything. Judging by infection data, it would seem as if it was made to gather intelligence from industrial infrastructure and system manufactures. This data is believed to be used for a later attack [5]. This makes sense if it was made by the same people as Stuxnet considering how specifically targeted it was.

Chapter 4

Installation of Duqu

The following sections covers one of the most discussed method of delivering and installing Duqu. The process is divided into two, the *Preparation* and the *Installation.*

4.1   Preparation

Duqu was delivered to the target by using a specially crafted Microsoft Word document. According to Microsoft the Duqu malware exploits a problem in T2EMBED.DLL which is called by the TrueType font parsing engine [4]. This exploit in the Win32k TrueType font parsing engine allows arbitrary code to run in kernel mode. This arbitrary code will from now on in this report be reference to as the *prep-code.*

Once the Word document is opened and the exploit triggered, the prep-code will do a check to determine whether or not the target has been infected. This is done by checking the registry for the following value:

```
HKEY_LOCAL_MACHINE\ SOFTWARE\Microsoft\Windows\
    CurrentVersion\Internet Settings\Zones\4\"CF1D"
```

Unless the value is found, indicating that the computer is compromised, the prep-code decrypts two files from the Word document:

1. `jminet7.sys` or `cmi4432.sys`, both drivers.

2. `netp191.pnf` or `cmi4432.pnf`, both DLLs.

On of the differences between `jminet7.sys` and `cmi4432.sys` is that latter actually was signed (2.2.1) by C-Media Electronics Inc. For the sake of simplicity, it is assumed that the `jminet7.sys` and `netp191.pnf` is the files used in this attack. According to the CrySys report [1], the files are very similar and it is theorized that one of them provides the functionality to utilize the key logger. The prep-code executes `jminet7.sys` (driver) which injects the `netp191.pnf` into `services.exe`. This behavior is

defined by the configuration file `netp192.pnf` which is loaded and decrypted. Before the prep-code exits, it executes `netp191.pnf` and overwrite itself with zeros.

4.2   Installation

At this point, the prep-code no longer exists in RAM, and the driver (`jminet7.sys`) passes the execution to `netp192.pnf`. This is where things get a little tricky. During the preparation process, `netp192.pnf` was injected into `services.exe`. Now, the same code but launched by the driver, decrypts three files from the injected `netp192.pnf` code. These files are:

1. `netp191.pnf` (extracted from itself)

2. A .sys driver file used as a load point for Duqu after reboot

3. The installer configuration file

A time frame is defined in the installer configuration file. The installer will terminate if the it executed outside this time frame. If it does not, the installer will pass the execution to Duqus main DLL (`netp192.pnf`) by hooking `ntdll.dll` (see 2.1.1). The main DLL has a number of exports (see A for all). Installation is handles by number 4 and 5. Export 4 is used to inject the main DLL into a suitable process and to pass a pointer to the three decrypted files. Export 5 drops the earlier mentioned driver used as a load point into the following directory:

`%System%\Drivers\`

The drivers name is defined by the installation configuration file (decrypted earlier). A service is created to make sure that the load point driver is loaded every time the computer is booted. Finally, the main DLL and the installation configuration file is encrypted and named according to the definition in the installation configuration file and placed in the following directory:

`%Windir%\inf\`

Appendix A

NETP191.PNF Exports

Table A.1: Table of NETP191.PNF exports [5]

1	Data initializer
2	Run export 6
3	Get version info from configuration data
4	Inject self into process, run export 5 (32-bit only)
5	Setup, depends on install status. Before install: Drop provided load point driver and create service. After install: Load resource 302
6	Routine for cleanup
7	Start RPC
8	Identical to export 1, but uses a delay timer.

Chapter 5

Consequences

Chapter 6

Mitigation

Chapter 7

Conclusion

Complete Bibliography

[1] Boldizsár Bencsáth et al. *Duqu: A Stuxnet-like malware found in the wild*. Report. Accessed: 7-9-2015. Budapest University of Technology and Economics: Department of Telecommunications.

[2] Jurriaan Bremer. *x86 API Hooking Demystified*. `http://jbremer.org/x86-api-hooking-demystified/`. Accessed: 10-09-2015.

[3] Dejan Lukan. *API Hooking and DLL Injection on Windows*. `http://resources.infosecinstitute.com/api-hooking-and-dll-injection-on-windows/`. Accessed: 9-09-2015.

[4] Ryan Naraine. *Microsoft issues temporary 'fix-it' for Duqu zero-day*. `http://www.zdnet.com/article/microsoft-issues-temporary-fix-it-for-duqu-zero-day/`. Accessed: 8-09-2015.

[5] Symantec. *W32.Duqu: The precursor to the next Stuxnet*. Report. Accessed: 8-9-2015. Symantec Security Response.

[6] Microsoft TechNet. *What Is RPC*. `https://technet.microsoft.com/en-us/library/cc787851.aspx`. Accessed: 8-09-2015.