

Lab 1.5: Reverse Engineering

Analysis of m3.exe

Fredrik Helgesson



DV2582 - Malware Analysis
Blekinge Institute of Technology
371 79 Karlskrona
September 23 2018

1 Introduction

The objective of this lab was to perform a static and dynamic analysis of the malicious executable "m3.exe". During the analysis several analytic tools were used and lastly the dissassembler/debugger IDA Pro were used to analyze the operation of the malware further. Basic Information about the analyzed file can be seen below.

Filename: m3.exe

Filesize: 338.5 KB

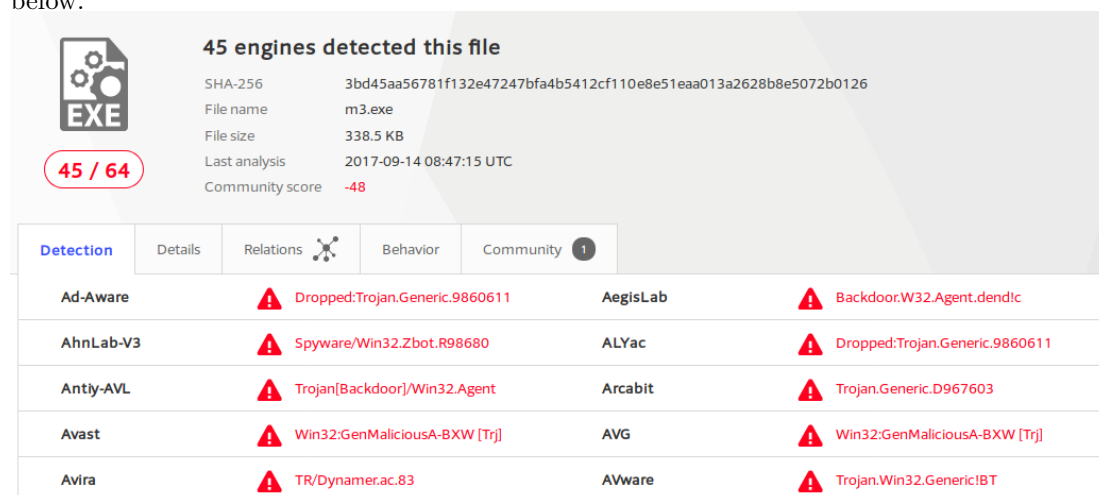
SHA256 Sum: 3bd45aa56781f132e47247bfa4b5412cf110e8e51eaa013a2628b8e5072b0126

Creation time: 2013-10-25 02:52:12

2 Static Analysis

2.1 VirusTotal.com

When uploaded on the malware database VirusTotal 45/64 virus engines identified this file as a malware. Most of the engines has also classified the malware as a type of trojan. A snippet of the VirusTotal output can be seen in the image below.



45 engines detected this file

SHA-256: 3bd45aa56781f132e47247bfa4b5412cf110e8e51eaa013a2628b8e5072b0126
File name: m3.exe
File size: 338.5 KB
Last analysis: 2017-09-14 08:47:15 UTC
Community score: -48

45 / 64

Detection	Details	Relations	Behavior	Community
Ad-Aware	⚠ Dropped:Trojan.Generic.9860611			AegisLab
AhnLab-V3	⚠ Spyware/Win32.Zbot.R98680			ALYac
Antiy-AVL	⚠ Trojan[Backdoor]/Win32.Agent			Arcabit
Avast	⚠ Win32:GenMaliciousA-BXW [Trj]			AVG
Avira	⚠ TR/Dynamer.ac.83			AVware
				Backdoor:W32.Agent.dendic
				Dropped:Trojan.Generic.9860611
				Trojan.Generic.D967603
				Win32:GenMaliciousA-BXW [Trj]
				Trojan.Win32.Generic!BT

2.2 Packing







By comparing the virtual size of the executable with its raw data size a conclusion can be made that the executable is not packed. This hypothesis was strengthened by the fact that neither VirusTotal nor PeID identified any signs of packing. After loading the executable into the dissassembler IDA Pro and

seeing complete code a safe assumption can be made that the malware is not packed.

2.3 Imports









By analyzing the executable in the static analysis tool Dependency Walker several imports were identified. All of the DLLs imported are ordinary and regularly used in all types of programs, but some of the functions called from these libraries tells something about the analyzed executables potentially malicious actions.

The first called function of importance for this analysis is called "IsDebuggerPresent" and is imported from the library "KERNEL32.DLL", the function can be seen in the image below.

	N/A	739 (0x02E3)	InitializeCriticalSectionAndSpinCount	Not Bound
	N/A	747 (0x02EB)	InterlockedDecrement	Not Bound
	N/A	751 (0x02EF)	InterlockedIncrement	Not Bound
	N/A	768 (0x0300)	IsDebuggerPresent	Not Bound
	N/A	772 (0x0304)	IsProcessorFeaturePresent	Not Bound
	N/A	778 (0x030A)	IsValidCodePage	Not Bound








The operation of the function is simply returning a flag based on if the program is ran through a debugger or not. This function can be used by malware to hide the payload of the program when analyzed in a debugger. If the program identifies a debugger it can for example jump over the payload and execute some ordinary code, jump into an eternal sleep or simply jump around in the code to confuse the analyzer. This functionality makes the function call an indicator of compromise. The fact that this function is called will be of importance during the dynamic analysis later in this report.

The second import which could identify some of the malwares actions is "ADVAPI32.DLL". Some of the functions called from this library can be seen in the image below.

	N/A	458 (0x01CA)	RegCloseKey	0x77DD6C17
	N/A	463 (0x01CF)	RegCreateKeyExW	0x77DD775C
	N/A	464 (0x01D0)	RegCreateKeyW	0x77DFBA25
	N/A	466 (0x01D2)	RegDeleteKeyW	0x77DE557B
	N/A	468 (0x01D4)	RegDeleteValueW	0x77DDEDE1
	N/A	473 (0x01D9)	RegEnumKeyExW	0x77DD7BC9
	N/A	474 (0x01DA)	RegEnumKeyW	0x77DDD5D4
	N/A	476 (0x01DC)	RegEnumValueW	0x77DD7EDD



These functions implies that the program is modifying the windows registry. By modifying the registry the program can accomplish many different things depending on which registry keys are modified, one example creating persistence by configuring a malicious program to be ran at start-up or as a Windows Service.

The third type of imported function of importance is also imported from the library "KERNEL32.DLL". These functions are connected to file system modification and can be seen in the image below.

	N/A	82 (0x0052)	CloseHandle	Not Bound
	N/A	133 (0x0085)	CreateEventW	Not Bound
	N/A	143 (0x008F)	CreateFileW	Not Bound
	N/A	168 (0x00A8)	CreateProcessW	Not Bound
	N/A	202 (0x00CA)	DecodePointer	Not Bound
	N/A	209 (0x00D1)	DeleteCriticalSection	Not Bound
	N/A	214 (0x00D6)	DeleteFileW	Not Bound

The function calls suggests that the program is creating new files or possibly replacing existing files with new ones. The motive behind the calls are at this stage unknown but could suggest that the program is trying to create persistence in the system or replace benign programs with malicious versions.

The fourth example of functions used by "m3.exe" which could be an indicator of the programs actions were imported from "ADVAPI32.DLL". The functions can be seen in the image below.

	N/A	173 (0x00AD)	DecryptFileW	0x77E1335F
	N/A	203 (0x00CB)	EncryptFileW	0x77E13320

The functions as their names suggests are used to encrypt and decrypt files. Functions with this functionality can be used by malware to obfuscate code and make analysis of the malware harder.

2.4 Strings

When analyzing the strings in "m3.exe" the first thing to notice is that the strings has been modified by some obfuscation algorithm. An example of obfuscated strings can be seen in the image below.

```

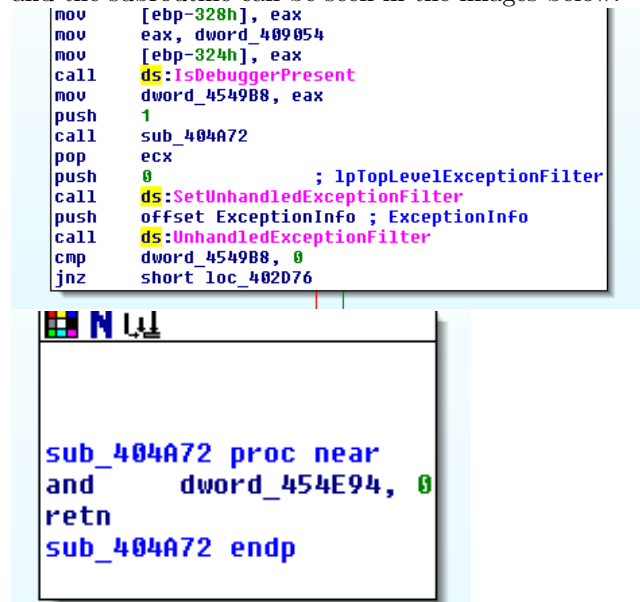
".data:00... 00000016 uni... oGgoell cn
".data:00... 0000001E uni... liDesercpiitno
".data:00... 0000000A uni... oGgoe
".data:00... 00000006 uni... \v6F
".data:00... 00000016 uni... iVereisno
".data:00... 00000016 uni... .1.3121.30
".data:00... 00000016 uni... ttreanNlma
".data:00... 0000001C uni... eoGgoellU dpta
".data:00... 00000008 uni... e dL
".data:00... 0000001A uni... gelaoCypirhg
".data:00... 00000040 uni... toCypirhg t02702-10 0oGgoell cn
".data:00... 0000001E uni... iirganFlinema
".data:00... 00000024 uni... eoGgoelpUadete.ex
".data:00... 00000006 uni... \n4P
".data:00... 00000016 uni... orudtcaNem

```

The algorithm used is simple and the resulting strings are often easy to deobfuscate. An example is the string marked with blue in the image above, a safe assumption can be made that the original string before the obfuscation was "GoogleUpdate.exe". The unobfuscation can be done manually for several strings but it clearly complicates the work for the analyst of the malware. The algorithm is that each chunk of four concurrent ASCII are written in reverse order. For example the string "Fredrik" will therefore be obfuscated as "derFkir" by using this algorithm.

One of the first actions done by this program is creating a new process through Windows CMD. This creation can be seen in the image below as well as the command sent into CMD. The placeholder "%s" stands the name of a file contained in a register. When crosschecking this command against VirusTotal the whole command is revealed: C:\WINDOWS\system32\cmd.exe /c del C:\3BD45A~1 > nul". This command deletes the program itself, "m3.exe". The reason that the name of the file in the command does not match the name of the file is that VirusTotal probably has renamed the file before analyzing it.

The second task the program does is using calling the function "IsDebuggerPresent" mentioned earlier under the "Import"-section. If the return value of this function is true, which means that a debugger is indeed present, the program calls a subroutine which sets the program in a sleeping loop. The function call and the subroutine can be seen in the images below.



After this initial phase the program seems to create the new files under the Windows path: C:\Documents and Settings\<USER>\Templates\.

the folder `|\\Templates\\|` is configured to be hidden in a Windows XP system.

The new files created are:

Noew.SAM

Goopdate.dll

GoogleTool.exe

Windows Message.lnk

The file creation can be seen in the images below where the bottom one contains information from VirusTotal.

```
• .rdata:00408084      unicode 0, <\\Windows Message.lnk>,0
• .rdata:0040808E      align 10h
• .rdata:004080B0      asc_4080B0:      ; DATA XREF: sub_4012B0+6E↑o
• .rdata:004080B0      ; sub_4012B0+9A↑o ...
• .rdata:004080B0      unicode 0, <\\>,0
• .rdata:004080B4      aRundll32_exe:      ; DATA XREF: sub_4012B0+84↑o
• .rdata:004080B4      unicode 0, <rundll32.exe>,0
• .rdata:004080CE      align 10h
• .rdata:004080D0      aNoew_sam:      ; DATA XREF: sub_4012B0+B3↑o
• .rdata:004080D0      unicode 0, <Noew.SAM>,0
• .rdata:004080E2      align 4
• .rdata:004080E4      aGoopdate_dll:      ; DATA XREF: sub_4012B0+DF↑o
• .rdata:004080E4      unicode 0, <Goopdate.dll>,0
• .rdata:004080FE      align 10h
• .rdata:00408100      aGoogletool_exe:      ; DATA XREF: sub_4012B0+10B↑o
• .rdata:00408100      unicode 0, <GoogleTool.exe>,0
• .rdata:0040811E      align 10h
• .rdata:00408120      ; const WCHAR aMyinittest____
• .rdata:00408120      aMyinittest____:      ; DATA XREF: sub_4012B0+2FA↑o
• .rdata:00408120      unicode 0, <MyInitTest....>,0
• .rdata:0040813E      align 10h
```

Files Written

C:\Documents and Settings\<USER>\Templates\GoogleTool.exe

C:\Documents and Settings\<USER>\Templates\Goopdate.dll

C:\Documents and Settings\<USER>\Templates\Noew.SAM

C:\Documents and Settings\<USER>\Start Menu\Windows Message.lnk

The program is according to VirusTotal communicating with the URL "website.baesystems.ca" on the IP "112.175.79.49:8001". The service the URL is directing to is currently down and the domain is no longer in use, which means that no DNS-queries can be made to the address. The IP address can therefore currently not be resolved by a DNS. The IP-address shown above is information gathered by VirusTotal when the server was active. The source of the information can be seen in the image below.

Network Communication ⓘ

DNS Resolutions

+ website.baesystems.ca

TCP Communication

112.175.79.49:8001

The probable cause of the network communication is to download the files mentioned above. The creation of these files in combination with the CMD-command mentioned above suggests that "m3.exe" works as a trojan which either downloads further payload or unpacks itself into more files which are hidden in the system. The trojan then deletes itself to clear its tracks. By running the whois service on the IP address we can find information about where the address is registrated, the information suggests that the malware is communicating with a machine registrated to Korea Telecom which is located in South Korea. A snippet of the output from the whois query can be seen in the image below.

```
# ENGLISH

KRNIC is not an ISP but a National Internet Registry similar to APNIC.

[ Network Information ]
IPv4 Address       : 112.160.0.0 - 112.191.255.255 (/11)
Organization Name  : Korea Telecom
Service Name       : KORNET
Address            : Gyeonggi-do Bundang-gu, Seongnam-si Buljeong-ro 90
Zip Code           : 13606
Registration Date   : 20090210

Name               : IP Manager
Phone              : +82-2-500-6630
E-Mail             : kornet_ip@kt.com

- KISA/KRNIC WHOIS Service -
```

The creation of the file "Windows Message.lnk" can be seen in the image below.

```

.text:0040154F      push     offset aWindowsMessage ; "\\Windows Message.lnk"
.text:00401554      lea      ecx, [ebp+szPath]
.text:0040155A      push     104h
.text:0040155F      push     ecx
.text:00401560      call     sub_4017A8
.text:00401565      mov      esi, ds:keybd_event
.text:0040156B      add      esp, 0Ch
.text:0040156E      push     0 ; dwExtraInfo
.text:00401570      push     0 ; dwFlags
.text:00401572      push     0 ; bScan
.text:00401574      push     12h ; bVk
.text:00401576      call     esi ; keybd_event
.text:00401578      push     0 ; dwExtraInfo
.text:0040157A      push     0 ; dwFlags
.text:0040157C      push     0 ; bScan
.text:0040157E      push     11h ; bVk
.text:00401580      call     esi ; keybd_event
.text:00401582      push     0 ; dwExtraInfo
.text:00401584      push     0 ; dwFlags
.text:00401586      push     0 ; bScan
.text:00401588      push     45h ; bVk
.text:0040158A      call     esi ; keybd_event

```

The extension of the file suggests that it is a linking file which works like a shortcut to an object.

3 Dynamic Analysis

The dynamic analysis was done using four different analytic tools. First the analyzing environment was setup by monitoring the system with process monitor and process explorer. The network traffic was also monitored with Wireshark. In the final step of the dynamic analysis the IDA Pro debugger was used to further examine the actions of the program.

3.1 Process Monitor

When ran "m3.exe" creates three new files under C:\Documents and Settings\<USER>\Templates\ as suggested in the static analysis. The file creation process can be seen in the image below.

604	ReadFile	C:\STUDENT_LAB\Lab - Reverse Engineering\m3.exe
604	CreateFile	C:\Documents and Settings\Administrator\Templates\GoogleTool.exe
604	CreateFile	C:\Documents and Settings\Administrator\Templates
604	CloseFile	C:\Documents and Settings\Administrator\Templates
604	ReadFile	C:\\$Directory
604	WriteFile	C:\Documents and Settings\Administrator\Templates\GoogleTool.exe
604	CloseFile	C:\Documents and Settings\Administrator\Templates\GoogleTool.exe
604	CreateFile	C:\Documents and Settings\Administrator\Templates\Goopdate.dll
604	CreateFile	C:\Documents and Settings\Administrator\Templates
604	CloseFile	C:\Documents and Settings\Administrator\Templates
604	WriteFile	C:\Documents and Settings\Administrator\Templates\Goopdate.dll
604	CloseFile	C:\Documents and Settings\Administrator\Templates\Goopdate.dll
604	CreateFile	C:\Documents and Settings\Administrator\Templates\Noew.SAM
604	CreateFile	C:\Documents and Settings\Administrator\Templates
604	CloseFile	C:\Documents and Settings\Administrator\Templates
604	WriteFile	C:\Documents and Settings\Administrator\Templates\Noew.SAM

The executable then start CMD.exe, what command it executed can not be seen in process monitor but a safe assumption can be made that it is going to execute the command stated in the static analysis: "C:\WINDOWS\system32\cmd.exe /c del C:\3BD45A~1 > nul".

The execution of CMD.exe can be seen in the image below.

604	QueryOpen	C:\WINDOWS\system32\cmd.exe	SUCCESS
604	QueryOpen	C:\WINDOWS\system32\cmd.exe	SUCCESS
604	CreateFile	C:\WINDOWS\system32\cmd.exe	SUCCESS
604	CreateFileMapp...	C:\WINDOWS\system32\cmd.exe	SUCCESS
604	QueryStandardl...	C:\WINDOWS\system32\cmd.exe	SUCCESS
604	CreateFileMapp...	C:\WINDOWS\system32\cmd.exe	SUCCESS
604	CloseFile	C:\WINDOWS\system32\cmd.exe	SUCCESS

When running m3.exe a new process, "GoogleTool.exe", is started. The probable way that "GoogleTool.exe" is started the first time is through the file "Windows Message.lnk" but no confirmation of that theory was found.

3.2 Wireshark

In the image below a snippet of the output from Wireshark can be seen.

3	0.000186	10.0.2.15	10.0.2.3	DNS	Standard query A website.baesystems.ca
4	0.021889	10.0.2.15	10.0.2.3	DNS	Standard query PTR 3.2.0.10.in-addr.arpa
5	0.994976	10.0.2.15	10.0.2.3	DNS	Standard query A website.baesystems.ca
6	1.015051	10.0.2.15	10.0.2.3	DNS	Standard query PTR 3.2.0.10.in-addr.arpa
7	1.996430	10.0.2.15	10.0.2.3	DNS	Standard query A website.baesystems.ca
8	2.016524	10.0.2.15	10.0.2.3	DNS	Standard query PTR 3.2.0.10.in-addr.arpa
9	3.999340	10.0.2.15	10.0.2.3	DNS	Standard query A website.baesystems.ca

The image shows that "m3.exe" sends DNS queries to "website.baesystems.ca". The reason behind this could be to check if the website is currently up and running. Other than the queries "m3.exe" does not perform any network communication. This suggests that the files that are created are created from "m3.exe" itself, m3.exe is split up into 3 parts and then removed. The hypothesis of "m3.exe" downloading the files stated in the static analysis is hereby disproved since the executable does not communicate with any website directly.

3.3 Conclusion


The conclusion of the analyzing "m3.exe" is that the file is a Trojan which when ran creates 3 new files which contain the payload, "m3.exe" is then removed. The creators of the Trojan has used several tools to complicate the work for the analyst such as obfuscation and anti-debugging.

4 Analysis of "goopdate.dll"

4.1 Static Analysis

4.2 VirusTotal

According to VirusTotal 41/65 anti malware engines identify the DLL file "goopdate.dll" as malicious, most of them have put trojan in their naming of the malware. A snippet of the output from VirusTotal can be seen in the image below.



41 engines detected this file

SHA-256: b5834262c636c64bf0464979b546404554df6619de9d8a5b49cdca53fb834747
File name: Goopdate.dll
File size: 43.5 KB
Last analysis: 2017-09-18 10:15:32 UTC
Community score: -1



41 / 65

Detection
Details
Relations
Community

Ad-Aware	Trojan.Generic.9860611	ALYac	Trojan.Generic.9860611
Antiy-AVL	Trojan/Win32.Generic	Arcabit	Trojan.Generic.D967603
Avast	Win32:GenMaliciousA-BXW [Trj]	AVG	Win32:GenMaliciousA-BXW [Trj]
Avira	TR/Rogue.9860611	AVware	Trojan.Win32.Generic!BT
Baidu	Win32.Trojan.WisdomEyes.16070401....	BitDefender	Trojan.Generic.9860611
CAT-QuickHeal	Trojan.IGENERIC	Comodo	UnclassifiedMalware




4.2.1 Imports

When analyzing the imported functions of the DLL a couple of interesting entries were found. From "ADVAPI32.DLL" the functions in the image below were imported.




PI	Ordinal ^	Hint	Function	Entry Point
	N/A	609 (0x0261)	RegOpenKeyExW	Not Bound
	N/A	638 (0x027E)	RegSetValueExW	Not Bound

This suggests that the DLL is used to modify the Windows Registry. This is often occurring when malware wants to for example create persistence in a system.




All functions in the images below are imported from the library "KERNEL32.DLL" and gives a hint about the operation of the DLL.

	N/A	133 (0x0085)	CreateEventW	Not Bound
	N/A	143 (0x008F)	CreateFileW	Not Bound
	N/A	181 (0x00B5)	CreateThread	Not Bound

The function "CreateFileW" in the image above suggests that "Goopdate.dll" is used to for instance create new files. The reason for this could for example be to create persistence in the system.

	N/A	759 (0x02F7)	IsBadReadPtr	Not Bound
	N/A	768 (0x0300)	IsDebuggerPresent	Not Bound
	N/A	772 (0x0304)	IsProcessorFeaturePresent	Not Bound

The function "IsDebuggerPresesnt" seen in the image above is likely used for the same reason as in "m3.exe", to complicate the analysis process.

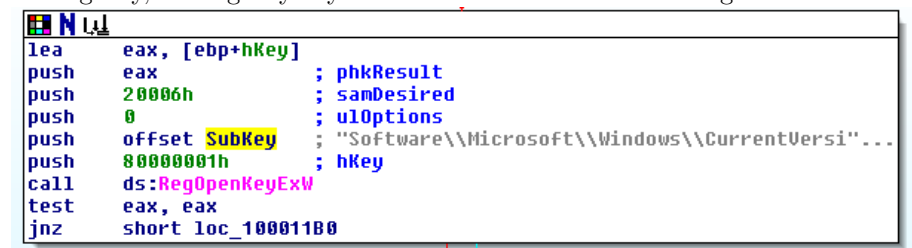
	N/A	1297 (0x0511)	WideCharToMultiByte	Not Bound
	N/A	1298 (0x0512)	WinExec	Not Bound
	N/A	1317 (0x0525)	WriteFile	Not Bound

The last interesting imported function "WinExec" can be seen in the image above. This function is used to start a new process, for example a malicious

executable.

4.2.2 Strings

"goopdate.dll" contained one string which suggests that the DLL is modifying the registry, the registry key modified can be seen in the image below.



```
lea    eax, [ebp+hKey]
push   eax                ; phkResult
push   200006h            ; samDesired
push   0                  ; ulOptions
push   offset SubKey      ; "Software\\Microsoft\\Windows\\CurrentVersi"...
push   80000001h          ; hKey
call   ds:RegOpenKeyExW
test   eax, eax
jnz    short loc_100011B0
```

The key "\Software\Microsoft\Windows\CurrentVersion\Run is used to make executable files run automatically when a user logs into the system, which is a great way for malware to create persistence in a system. The subkey added has the name "Microsoft" and its value will be confirmed during the dynamic analysis

In the image below we can see a part of the function "myExtern" which is the only exported function in the DLL. The operation of the DLL is that it checks if it has been successfully loaded into the "GoogleTool.exe" executable, if so the DLL runs its payload which is contained in the function "myExtern". The payload is executed by running the command "rundll32.exe goopdate.dll, myextern".

```

call    ds:SHGetSpecialFolderPathW
push    offset asc_1000916C ; "\\\"
lea     ecx, [ebp+szPath]
push    104h
push    ecx
call    sub_10001CAC
push    offset aGoogletool_exe ; "GoogleTool.exe"
lea     edx, [ebp+szPath]
push    104h
push    edx
call    sub_10001CAC
add     esp, 18h
push    104h ; nSize
lea     eax, [ebp+Filename]
push    eax ; lpFilename
push    0 ; hModule
call    ds:GetModuleFileNameW
lea     ecx, [ebp+Filename]
push    offset aRundll32_exe ; "rundll32.exe"
push    ecx
call    sub_10001C4C
add     esp, 8
test    eax, eax

```

4.3 Dynamic Analysis

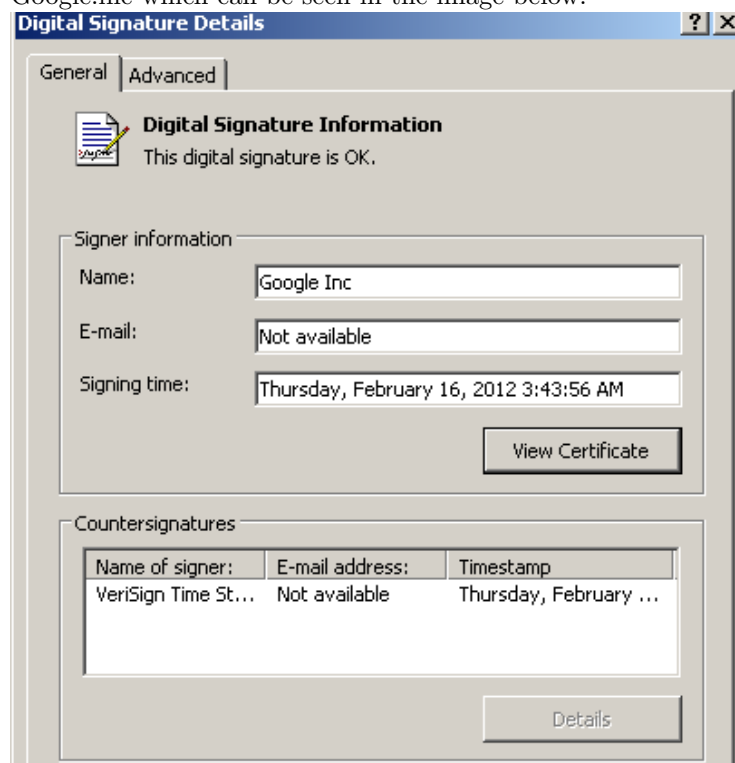
The dynamic analysis of goodupdate is done by loading the dll into the Windows utility rundll32.exe. By inserting a breakpoint in the start of the exported function "MyExtern" it is possible to step by step execute the program and see what each executionstep does. The debugging did not result in any discoveries that has not yet been covered in other sections of the analysis.

4.3.1 Windows Registry


As the string containing a registry key discovered above suggested "goopdate.dll" does creates a new subkey under
 "\\Software\\Microsoft\\Windows\\CurrentVersion\\Run with the name "Microsoft" and the datavalue
 "C:\\Documents and Settings\\Administrator\\Templates\\GoogleTool.exe" which is the path to "GoogleTool.exe". The reason behind this registry modification is as discussed when the registry key was first discovered in "goopdate.dll", to create persistence in the system. By making "GoogleTool.exe" execute and thereby loading the malicious dll "goopdate.dll" the malware has a permanent backdoor into the system given that the registry entry is not removed.

5 Analysis of "GoogleTool.exe"

The "GoogleTool.exe" differs from the other files. The executable is verified by Google.inc which can be seen in the image below.



The verification means that this file has not been modified since creation by Google. By extracting the MD5 sum from the file (506708142bc63daba64f2d3ad1dcd5bf) and crosschecking it against known files it turns out that the file is the same as "GoogleUpdate.exe" which is an executable file that runs the Google Updater which is used for downloading, installing and automatically update Google applications. This file is therefore not analyzed further since it is not modified. One way of verifying that "GoogleTool.exe" is benign is by uploading it on VirusTotal. The result confirms that the files original name is "GoogleUpdate.exe", that the file is signed by several valid signers and that it only considered malicious by 1/64 malware engines. This 1 engine is probably a false-positive. These snippets of information found on VirusTotal can be seen in the images below.



One engine detected this file

SHA-256 9c36a08d9e7932ff4da7b5f24e6b42c92f28685b8abe964c870e8d7670fd531a

File name GoogleTool.exe

File size 113.91 KB

Last analysis 2018-09-23 17:55:50 UTC

Community score +150

1 / 69

Detection

Details

Relations

Behavior

Community 4

Cylance	Unsafe	Ad-Aware	Clean
AegisLab	Clean	AhnLab-V3	Clean
Alibaba	Clean	ALYac	Clean
Antiy-AVL	Clean	Arcabit	Clean
Avast	Clean	Avast Mobile Security	Clean

Signature Verification

Signed file, valid signature

File Version Information

Copyright	Copyright 2007-2010 Google Inc.
Product	Google Update
Description	Google Installer
Original Name	GoogleUpdate.exe
Internal Name	Google Update
File Version	1.3.21.103
Date Signed	3:43 AM 2/16/2012

Signers

- Google Inc
- Thawte Code Signing CA - G2
- thawte

The vulnerability in "GoogleUpdate.exe" and "GoogleTool.exe" is that it dynamically links with a library with the exact name of "goopdate.dll" to be able execute its operations. By swapping the original "gooupdate.dll" with a malicious DLL with the exact same name, "GoogleTool.exe" can be used as a backdoor. Since "GoogleTool.exe" is verified and therefore considered trusted

by both system and user infecting it creates a persistent malware in the system.

6 Analysis of "Noew.SAM"

By using the Linux program "file" on Noew.SAM we get the output seen in the image below.

```
Fredrik@Fredrik-HP-ElliteBook-820-G2:~/Downloads/Lab5 - Reverse Engineering$ file Noew.SAM
Noew.SAM: data
```

This suggests that the file is not containing any known headers, it only contains a chunk of data. When taking a closer look at the data with the Linux program "cat" we notice that it is obfuscated. A snippet of the output from "cat" can be seen in the image below.

```
AyiF
eNdniftx
WeliF
iFdNfTsrWeli
LteGEtsarorr
CteGerrurPtnseco
sevoMeliF
WaerChTetdaer
VteGisrexEno
WtrivAlaucoll
mreTtaniorPessec
trivFlau
eereG
cliFtziSe
eaerCiFet
WeluM
uBitlTetydiWoahCe
rrtslAnel
ediWrahCuMoTBitl
etylG
laboomeMtSyrSuta
```

There are big similarities with the obfuscated strings found in "m3.exe" which is not surprising since "Noew.SAM" originated from "m3.exe". It is probable that this file contains the payload of the malware but to execute it the file needs to be unobfuscated. The code for unobfuscating "Noew.SAM" is located in the malicious dll "goupdate.dll" and can be seen in the image below.

```

lea     ecx, [ebp+FileName]
push    104h
push    ecx
call    sub_10001CAC
push    offset aNoew_sam ; "Noew.SAM"
lea     edx, [ebp+FileName]
push    104h
push    edx
call    sub_10001CAC
add     esp, 18h

```

To be able to analyze "Noew.SAM" it needs to be unobfuscated, this can be done either during dynamic analysis when the deobfuscation is done as intended, by the malware or by cracking the obfuscation algorithm.

Because of the simplicity of the obfuscation algorithm shown in the analysis of "m3.exe", the file could be deobfuscated using the simple python-script seen in the image below.

```

f = open("Noew.SAM", "r+")
f2 = open("Unobfuscated", "w+")

while True :
    buf = f.read(4)
    buf = buf[::-1]
    f2.write(buf)
    if (buf == ""):
        break

```

The result of the script can be seen in the image below by printing a list of strings that were formerly obfuscated.

```

LookupPrivilegeValueW
RegQueryValueExW
CloseServiceHandle
StartServiceW
ChangeServiceConfigW
OpenServiceW
OpenSCManagerW
ControlService
DeleteService
GetUserNameW
QueryServiceConfigW
EnumServicesStatusW
UnlockServiceDatabase
LockServiceDatabase
ADVAPI32.dll

```

The unobfuscated file will be analyzed during this section under the name "Un-

obfuscated.exe”.

6.1 Static Analysis

6.1.1 VirusTotal

When uploading the now unobfuscated file on VirusTotal 38/57 anti-malware engines detected the file in their database. The result can be seen in the image below.

38 engines detected this file

SHA-256: 04f868264e113f85f960b65a57b671beba48d3c90f6ee0ecfdf18db8f9819682
File name: Noew2.SAM
File size: 140 KB
Last analysis: 2016-09-29 11:19:23 UTC

38 / 57











Detection	Details	Community
Ad-Aware	DeepScan.Generic.Malware.LV/g.2F8A...	AhnLab-V3 Trojan/Win32.Zbot.N1032582684
ALYac	DeepScan.Generic.Malware.LV/g.2F8A...	Antiy-AVL Trojan/Win32.SGeneric
Arcabit	DeepScan.Generic.Malware.LV/g.2F8A...	Avast Win32:Agent-AQGZ [Trj]
AVG	Agent4.BIQH	Avira TR/Detplock.dqgj

6.1.2 Imports

The first thing to notice about the imports is that the number of imports suggests that the analyzed file is complex. When loading the file into IDA Pro it is also clear that the file is a portable executable of the application type DLL. This information can be found in the image below.

```
;
; Input MD5 : F573CEDB34EF3C92D686F9B03C078589
; File Name : Z:\Lab5 - Reverse Engineering\Unobfuscated
; Format : Portable executable for 80386 (PE)
; Imagebase : 10000000
; Section 1. (virtual address 00001000)
; Virtual size : 00015028 ( 86056.)
; Section size in file : 00016000 ( 90112.)
; Offset to raw data for section: 00001000
; Flags 60000020: Text Executable Readable
; Alignment : default
; OS type : MS Windows
; Application type: DLL 32bit
```

Similar to "m3.exe" "Unobfuscated.exe" is using imports used to create new files and modify the registry seen in the images below.

	N/A	70 (0x0046)	CopyFileW	Not Bound
	N/A	78 (0x004E)	CreateDirectoryW	Not Bound
	N/A	80 (0x0050)	CreateEventW	Not Bound
	N/A	86 (0x0056)	CreateFileW	Not Bound
	N/A	459 (0x01CB)	RegCloseKey	
	N/A	467 (0x01D3)	RegCreateKeyW	
	N/A	493 (0x01ED)	RegOpenKeyExW	
	N/A	504 (0x01F8)	RegQueryValueExW	
	N/A	517 (0x0205)	RegSetValueExW	
	N/A	588 (0x024C)	StartServiceW	


There are also imports suggesting the actual payload of the malicious DLL. The import shown in the image below suggests that the DLL has the functionality to take screenshots.

```

push    0                ; X
push    80000000h         ; dwStyle
push    offset WindowName ; lpWindowName
push    offset ClassName  ; "#32770"
push    0                ; dwExStyle
call    ds:CreateWindowExW
push    0                ; nID
push    eax              ; hwndParent
push    0                ; nHeight
push    0                ; nWidth
push    0                ; y
push    0                ; x
push    50000000h         ; dwStyle
push    offset szWindowName ; "CVideoCap"
mov     [esi+10h], eax
call    capCreateCaptureWindowW
mov     [esi+14h], eax

```

There are also several imports that suggests that the DLL has functionality to gather information about the infected system. Some of the imported functions with this functionality can be seen in the images below.

 100170B4
 GetComputerNameW
 KERNEL32

```

call    ds:GetVersion    ; Get current version number of Windows
                                ; and information about the operating system platform
push    1
mov     dword_100218A0, eax
call    sub_1000CB17

```

I GlobalMemoryStatus	100170B0
I GlobalMemoryStatusEx	100170D4

6.1.3 Strings

There are plenty of strings in the executable which could be considered to be indications of compromise. One string that stands out is shown in the image below.

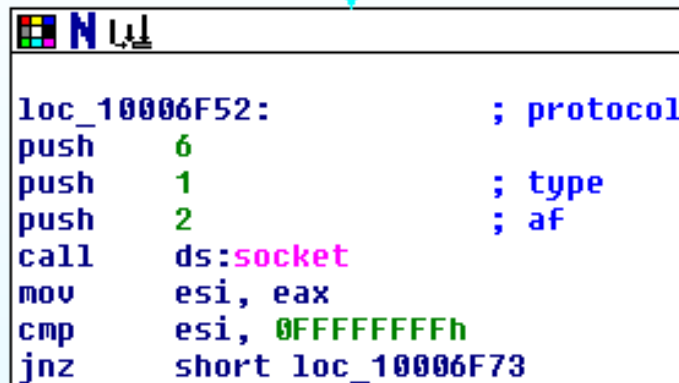


```

lea     esi, [esp+10490h+buf]
mov     ecx, offset aDarkshell ; "DarkShell"

```

This suggests that the DLL is the DarkShell backdoor. The backdoor connects to a Command & Control server to receive instructions and send information or data from the infected machine [1]. The socket connection to the remote server "website.baesystems.ca" can be seen in the images below.



```

loc_10006F52:                ; protocol
push     6
push     1                    ; type
push     2                    ; af
call     ds:socket
mov     esi, eax
cmp     esi, 0FFFFFFFFh
jnz     short loc_10006F73

```



```

stosw
stosb
mov     eax, [esp+18h]
push     eax                    ; in
call     ds:inet_ntoa
and     ebp, 0FFFFFFh
push     offset aHttp1_1UserAge ; " HTTP/1.1 User-Agent: MyApp/0.1 "...
push     ebp
push     eax
push     offset aConnect ; "CONNECT "
lea     ecx, [esp+0FCh+buf]
push     offset aSSDS ; "%S%S:%d%S "
push     ecx                    ; char *
call     _sprintf
mov     edi, ds:send
add     esp, 18h

```

The DLL has several different functions. A list of names of Anti-Virus programs found in the strings of the program suggests that the DLL has the functionality to identify which antivirus is running in the system. A part of the strings can be seen in the image below.

"..."	.data:10...	00000020	uni...	Avast Antivirus
"..."	.data:10...	0000001C	uni...	AVG Antivirus
"..."	.data:10...	00000018	uni...	BitDefender
"..."	.data:10...	0000000E	uni...	Dr.Web
"..."	.data:10...	00000028	uni...	Kaspersky Antivirus
"..."	.data:10...	00000028	uni...	Nod32 Antivirus 2.x
"..."	.data:10...	0000002A	uni...	Ewido Security Suite
"..."	.data:10...	00000022	uni...	McAfee VirusScan
"..."	.data:10...	00000032	uni...	Panda Antivirus/Firewall
"..."	.data:10...	00000020	uni...	Symantec/Norton
"..."	.data:10...	00000028	uni...	PC-cillin Antivirus
"..."	.data:10...	00000012	uni...	F-Secure
"..."	.data:10...	00000040	uni...	Kingsoft Internet Security 2008

There are also several keyboard buttons listed in the strings. This suggests that the program is either using them for keylogging purposes which is a functionality that DarkShell is known for, or to simulate key presses which could for example be used to start programs. One theory is that key-press simulation is used to start the new process "GoogleTool.exe" which is ran when "m3.exe" is ran.

```

• .data:1001CB34 dd offset aPageUp ; "[PageUp]"
• .data:1001CB38 dd offset aDel ; "[Del]"
• .data:1001CB3C dd offset aEnd ; "[End]"
• .data:1001CB40 dd offset aPagedown ; "[PageDown]"
• .data:1001CB44 dd offset aLeft ; "[Left]"
• .data:1001CB48 dd offset aUp_0 ; "[Up]"
• .data:1001CB4C dd offset aRight ; "[Right]"
• .data:1001CB50 dd offset aDown ; "[Down]"
• .data:1001CB54 dd offset aNumLock ; "[Num Lock]"
• .data:1001CB58 dd offset asc_1001CE6C ; "/"
• .data:1001CB5C dd offset asc_1001CDC8 ; "*"
• .data:1001CB60 dd offset asc_1001CEF4 ; "-"
• .data:1001CB64 dd offset asc_1001CDC4 ; "+"
• .data:1001CB68 dd offset a0 ; "0"
• .data:1001CB6C dd offset a1 ; "1"
• .data:1001CB70 dd offset a2 ; "2"
• .data:1001CB74 dd offset a3 ; "3"
• .data:1001CB78 dd offset a4 ; "4"
• .data:1001CB7C dd offset a5 ; "5"
• .data:1001CB80 dd offset a6 ; "6"
• .data:1001CB84 dd offset a7 ; "7"
• .data:1001CB88 dd offset a8 ; "8"
• .data:1001CB8C dd offset a9 ; "9"

```

The theory of key-press simulation is strengthened by the fact that "Deobfuscated.dll" calls functions which are suited for this task. The function calls can be seen in the image below.

```

• .rdata:1001B73C aSetcursorpos db 'SetCursorPos',0
• .rdata:1001B749 db 0
• .rdata:1001B74A db 0D6h ; +
• .rdata:1001B74B db 2
• .rdata:1001B74C aMouse_event db 'mouse_event',0
• .rdata:1001B758 db 0Eh
• .rdata:1001B759 db 1
• .rdata:1001B75A aGetdesktopwind db 'GetDesktopWindow',0
• .rdata:1001B76B db 0

```

Another functionality of "Unobfuscated.dll" is that it is trying to execute the file "kmd.exe" with the command `kmd.exe /c \"%s\`". Executables with this name are known to have been used in several different malware. The command calling this executable can be seen in the image below.

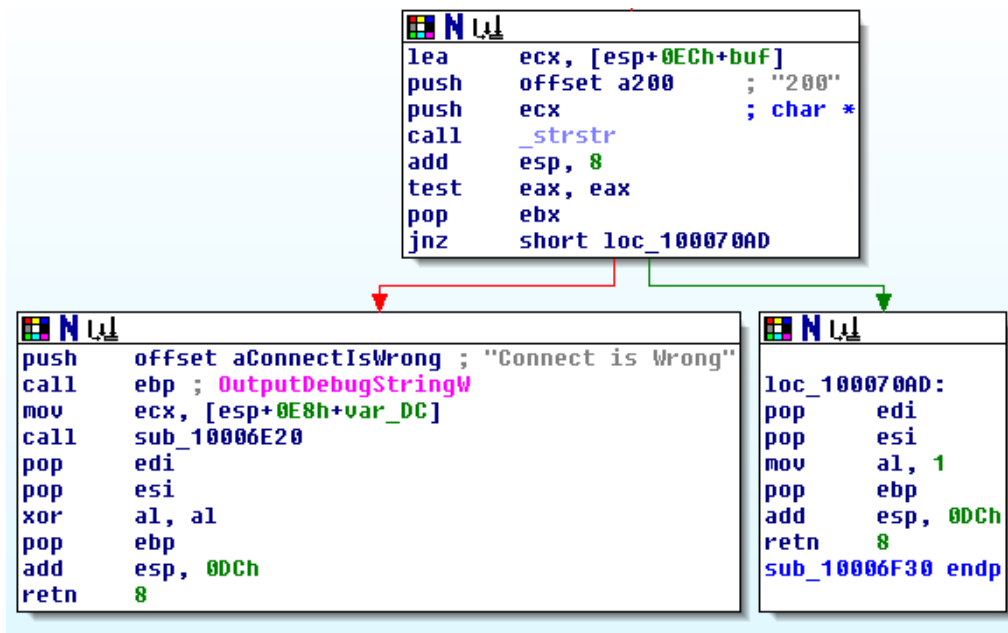
```

mov     esi, 05.1301000W
lea     edx, [esp+0A74h+ExistingFileName]
push    offset String2 ; "\\cmd.exe"
push    edx ; lpString1
call    esi ; lstrcatW
lea     eax, [esp+0A74h+NewFileName]
push    offset aKmd_exe ; "\\kmd.exe"
push    eax ; lpString1
call    esi ; lstrcatW
lea     ecx, [esp+0A74h+NewFileName]
push    ecx ; lpFileName
call    ds:DeleteFileW
lea     edx, [esp+0A74h+NewFileName]
push    0 ; bFailIfExists
lea     eax, [esp+0A78h+ExistingFileName]
push    edx ; lpNewFileName
push    eax ; lpExistingFileName
call    ds:CopyFileW
mov     esi, [esp+0A74h+arg_4]
mov     ecx, [esp+0A74h+arg_0]
push    esi
push    ecx
lea     edx, [esp+0A7Ch+CommandLine]
push    offset aKmd_exeCSS ; "kmd.exe /c \"%s\" \"%s\""
push    edx ; LPWSTR

```

6.2 Dynamic Analysis

One problem with dynamically analyzing "Unobfuscated.dll" is that it is not a standalone file and it has to be linked into an executable to perform its operations. In this case the dll is created to be linked into "goopdate.dll" and therefore linked into the executable "GoogleTools.exe". When pingging "website.baesystems.ca" another problem is discovered. Since the server is down the malicious DLL "Unobfuscated.dll" has no C&C server to gain instructions from, this makes analyzing the operations of the DLL harder since it does not perform the same malicious activity as intended. In the image below we can see how the DLL checks if the connection status is 200 (successful connection) by trying to receive instructions from the server, if connection problems occur it changes instruction path into error handling.



One thing to note when running the malware and monitoring its actions is that the process "cmd.exe" discovered in the static analysis is not present, neither the file in the system. The reason behind this could be that this file is downloaded from the Command & Control server after malware deployment and then use it when instructed to. Since the server is down this communication could not occur.

7 Analysis Conclusion

The executable "m3.exe" is a trojan containing 4 files which together forms a complete malware. When "m3.exe" is executed it creates the files "goopdate.dll", "GoogleTool.exe", "Noew.SAM" and "Windows Message.lnk". The file "m3.exe" is at the end of this process removed and "GoogleTools.exe" is started as a new process. "GoogleTools.exe" in itself is not a malicious program but it contains a hardcoded link with the dynamic library "goopdate.dll" which the trojan has created a modified malicious version of. The most important task of "goopdate.dll" is to unobfuscate the payload of the malware which is contained in "Noew.SAM". When unobfuscated it turns out that the file contains a Windows backdoor known as "DarkShell" which connects to a remote server to receive instructions and send output. The functionality of the payload is wide but some of the features discovered in the analyzed program are information gathering, screencapturing and anti-virus identification.

8 System Cleanup

8.1 Indicators of compromise

The indicators of compromise found during the analyze can be categorized in two different categories, paths, Registry entries and processes.

Paths:

C:\Documents and Settings\<USER>\Templates\Noew.SAM
C:\Documents and Settings\<USER>\Templates\GoogleTool.exe
C:\Documents and Settings\<USER>\Templates\goopdate.dll

Registry Entries:

"\Software\Microsoft\Windows\CurrentVersion\Run\
Subkey: Microsoft
Value: C:\Documents and Settings\Administrator\Templates\GoogleTool.exe

Processes:

GoogleTool.exe

8.2 Cleaning instructions

To clean up a system infected by this malware the network should first be disconnected. If the malware is a full version of "DarkShell" it is known to spread over the network. Disconnecting is also an insurance that the malware does not receive any last malicious instructions before getting removed. Secondly the process GoogleTool.exe should be terminated. The next step is to remove the malicious files and the registry entry seen under "Indicators of Commpromise" in the section above. Lastly the system should be restarted and process explorer should be used to make sure that the "GoogleTool.exe"-process is not running again, if that is the case make sure the registry entry is removed correctly. One way to determine how widely spread the malware is in a network is by tapping the network and determining which IP-addresses are communicating with the Command & Control server located on the URL "website.baesystems.ca".

9 Questions & Answers

1. *What are the characteristics by which you can define the programming language the program is written? List them.*
 1. Identifying the programming language by loading the PE file into PEiD.
 2. Uploading the file on VirusTotal and looking at the TRiD information.
 3. Depending on what standard libraries are linked into the program you can determine which language it is written in. For example if "stdio" is included it suggests that the program is written in C/C++.
 4. It is also possible to identify the programming language by identifying

which Runtime libraries are loaded, an example is found in the image below which suggests that the programming language is C++.

```

• .text:0040100C      jnz     snort_10C_401C9F
• .text:0040100E      push    12010h
• .text:00401013      push    offset aMicrosoftVisua ; "Microsoft Visual C++ Runtime Library"
• .text:00401018      push    edi
• .text:0040101A      call    sub_401700

```

2. What types of call conventions do you know? Give a short description of them. What is the default convention for calling functions of the system libraries Windows?

Name	CDECL	STDCALL	FASTCALL
Parameters	The parameters are pushed on the stack from left-to-right. The caller of a function must clean up the stack after the call.	The parameters are pushed on the stack from left-to-right. The difference between STDCALL and CDECL is that in STDCALL the callee has to clean up the stack when its function is completed.	The first 2 parameters for a function are passed in ECX and EDX. If more parameters are required they are pushed onto the stack.
Return Value	The return value of a function is stored in EAX	The return value of a function is stored in EAX	The return value of a function is stored in EAX
Non-Volatile Registers	EBP, ESP, EBX, ESI, EDI	EBP, ESP, EBX, ESI, EDI	EBP, ESP, EBX, ESI, EDI

The default calling convention for functions Windows system libraries is STDCALL.

3. Give the examples of standard C constructions translation. Determine the high-level construction of a given section of the assembler code it corresponds to.

One example of standard C construction is the IF-statement below:

```
if( ( x < y ) {Statement1;} )
```

In assembly translation of this is:

```

mov  eax, x; //move x into eax register
cmp  y,  eax; //Compare eax register with y
j1   Statement1; //Jump if x < y

```

4. List the main types of malicious programs. Which groups of system functions correspond to those types?

1. **Adware:** Malware used to deliver advertisements. The ads can be delivered in the form of pop-up ads on websites.

2. **Ransomware:** Malware typically used to encrypt a system and state a ransom for the system owner which has to pay for the system to be unencrypted.

3. **Rootkit:** Malware that is used to remotely control a computer with-

out the user noticing. A typical way of planting a rootkit is by swapping a default system utility such as the program "cat" in Linux and change it to a similar program but with the additional functionality of creating a backdoor into the system.

4. Trojan: Malware which camouflage itself as a wanted program. A legit program can also come bundled with an unwanted trojan. A typical trojan is a "Trojan Downloader" which is used to download additional unwanted software.

5. Spyware: Malware used to gather information about the system such as hardware/software information and web surfing patterns.

6. Worm: Malware which can propagate itself over the internet by exploiting operating system vulnerabilities.

7. Virus: Malware that can spread over the internet but the major difference between a Virus and a Worm is that the Virus requires human activity to spread, a classic example is a user opening an attached infected document contained in an email.

10 References

1. <https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Backdoor:Win32/Darkshell.A&ThreatID=-2147324436>