

Figure 10: OllyDbg displays important event functions and API usage statistics and referenced strings.

OllyDbg is easy to use and very useful with the help of plug-ins, such as FLIRT of IDA, and tools to utilize analysis from IDA.

Figure 10 shows an example of a malware variant, written in P-Code of Visual Basic, where OllyDbg displays important event functions and API usage statistics and referenced strings, which helps the analyst decide the order of further investigation.

The diagram shown in Figure 11 also provides very helpful information and, with the help of the Branch footprint technique we mentioned earlier, the analysis process can be made much easier.

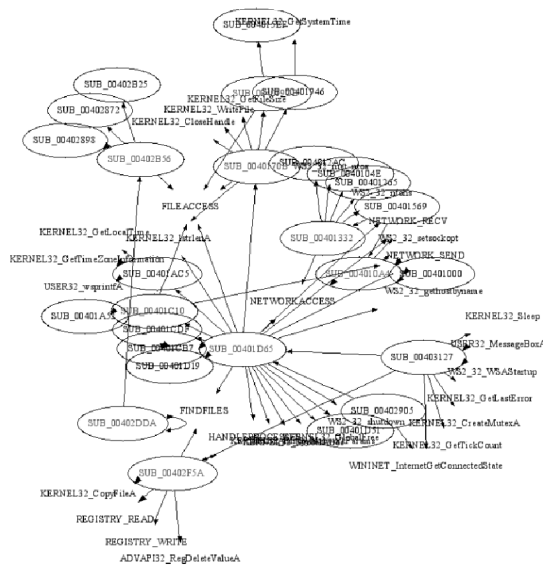


Figure 11.

SAMPLE CLASSIFICATION

Sample classification plays a very important role in generating decision support information. If a mutative malicious code can be traced through its transformation, this will allow us to make a good analysis and to provide good countermeasures as well as to reduce false positive detection by using the normal file information.

Newcomers to the industry often used to experience difficulties and had to go through a lot of trial and error

because the procedure was previously mostly dependent on the knowledge and experience of the analyst. Classification is making a huge contribution in reducing trial-and-error and, therefore, will become the most important part of the expert system.

Many new approaches have been introduced recently. [9] introduced IDA base analysis, which [10] later proved can be used in identifying malicious codes, and [11] showed that using Conditional Flow Information Code can increase the efficiency of 1:n matching. Dynamic base and behaviour base are also being studied.

In [12], we also introduced a classification technique capable of mass processing and we found that it can be used in the detection of variant forms in practice.

Sample classification, which requires the unpacking of packed files, divides codes into sections for 1:n matching, calculation of AOI and Opcode-array, function count, and searches through the database.

AOI is used to reduce the number of samples to put in comparison to find 1:n matching and, in the case of a virus, is also used to find the original file from the database.

Important fields and their descriptions are as follows:

Field	Example	Description
StList	sPSuuuuU	Section's Type List c=Code, d=Data, r=rsrc, l=reloc, u=unknown...
AOIR	90%	AOI rate of File (100%-AOIR is redundancy rate)
IAT	10	The count of Import Address Table
sAOIR	90%	AOI rate of Section
sFNC	10	The number of Function in Section
sCOV	50%	Code Coverages of functions in Section
shAOI	4095	highest AOI of a part in Section (baseline=4096)
slAOI	2048	lowest AOI of a part in Section (baseline=4096)

For malware, it is important to look at the key fields such as StList, AOIR, IAT, sCOV, sFNC, shAOI and slAOI.

Malware

Malware written in high-level languages, including worms and trojans, are often suitable for function-based sample classification. Verifying the Opcode-Array on a per-function basis showed a high accuracy rate.

Figure 12 shows Opcode-Array information extracted from Win-Trojan/LineageHack samples, which has many variants, on a per-function basis. It is noticeable that the same function is found in a large number of files.

Figure 13 depicts the analysed result of a 'NetSky' sample from the AOI point of view, and it shows that even different files often have same or similar amount of information.

Figure 14 shows Opcode-Array information extracted from NetSky samples. It is noticeable that the same function is found in large numbers of files.

For malware, the AOI approach can become a practical approach to investigate the relationship between the original

g_funcid	filenames	funcname	cod	ln	inst_crc	ap	lib	pus	flag	fingerprint
0000009AF48C7A7	E00A40035686243:E009400E485C20A:E00A400011	SUB_004010A4	448	154	4102847187	8	0	25	1426067460	614J54515J54J
000000C42051A16	E00A40035686243:E007A0009CFD985:E009400E	SUB_00401332	586	195	542220648	2	0	25	1426067460	44J454454444445J
000000C931C0381	E00A40035686243:E005940E485C20A:E00A40011	SUB_00401708	570	201	634600860	4	0	48	1426067460	4444444J
00000029E9628195	E00A40035686243:E009400E485C20A:E00A40011	SUB_00401946	132	41	3919546969	1	0	2	1426067460	262626
00000058F0F0530F	E00A40035686243:E007A0009CFD985:E009400E	SUB_00401AC5	330	88	4142945039	3	0	15	1426067460	596960
000000467871EC2	E00A40035686243:E007A0009CFD985:E009400E	SUB_00401C10	166	70	207106391	1	0	11	1426067460	4JC4C4D545
0000037AB1436A5	E00A40035686243:E00A400011586A7B	SUB_00401D03	2779	890	2973985367	19	0	266	1426067460	E4EJEE4CJ45544
00000035406C22F1	E00A40035686243:E009400E485C20A:E00A40011	SUB_004028A7	163	53	108032861	1	0	7	1426067460	FE5C44
000000A11B8347A	E00A40035686243:E009400E485C20A:E00A40011	SUB_00402AF8	497	161	464794119	3	0	28	1426067460	44CC4E444444444
0000016E07E26F0	E0730008817D953A	SUB_00414680	1289	366	132280078	12	0	62	1426067460	4454444445444444

Figure 12: Opcode-Array information extracted from Win-Trojan/LineageHack samples.

FILENAME	#	StList	EP	AOIR	SIZE	IAT	sN	sFN	sCC	sAOIR	sSIZE	shAOI	slAOI
Win32.Netsky.worm.19968_[f66]	2	msPSUu	6718	5%	21184	82	+0.	1	0%	13%	44164	2899	147
Win32.Netsky.worm.52736_[fed]	2	msPSUu	17718	5%	21454	82	+0.	1	0%	13%	44164	2899	147
Win32.Netsky.worm.18944_[d7h]	3	msPSUuu	3ABD	3%	21614	93	+0.	2	0%	12%	41046	3009	147
Win32.Netsky.worm.18944_[f1e]	3	msPSUuu	3ABD	3%	22804	93	+0.	2	0%	12%	41046	3009	147
Win32.Netsky.worm.23040_[78i]	3	msPSUuu	47AC	12%	23473	100	+0.	2	0%	15%	50118	3045	147
Win32.Netsky.worm.26624.B_[4]	3	msPSUuu	47AC	12%	23493	100	+0.	218	12%	15%	50118	3045	147
Win32.Netsky.worm.17424_[6t4]	3	msPSUuu	3414	16%	16363	86	+0.	310	91%	20%	34406	3091	147
Win32.Netsky.worm.28672_[f2b]	3	msPSUuu	3414	16%	16362	86	+0.	310	91%	20%	34406	3091	147
Win32.Netsky.worm.31232_[e3f]	1	msPSU	3AE4	18%	18958	0	+0.	193	47%	20%	40668	3093	147
Win32.Netsky.worm.22016_[d4d]	3	msPSUuu	3AE4	19%	19421	89	+0.	271	39%	21%	39080	3093	147
Win32.Netsky.worm.18432.C_[f]	2	msPSUu	3B8F	9%	19065	96	+0.	160	4%	13%	41789	3098	147
Win32.Netsky.worm.16896_[7a8]	2	msPSUu	4B70	12%	18297	84	+0.	257	28%	14%	39867	3098	147
Win32.Netsky.worm.18432.B_[E]	2	msPSUu	3B8F	9%	19297	96	+0.	160	4%	13%	42310	3099	147
Win32.Netsky.worm.16384_[9d]	2	msPSUu	3344	14%	16777	82	+0.	195	14%	15%	34980	3145	147
Win32.Netsky.worm.22016.E_[e]	3	msPSUuu	65FC	25%	39212	82	+0.	280	90%	16%	37794	3145	147
Win32.Netsky.worm.22528.B_[4]	3	msPSUuu	65FC	25%	40386	82	+0.	280	90%	17%	39138	3145	147
Win32.Netsky.worm.30720_[8b6]	3	msPSUuu	3B04	22%	26170	94	+0.	136	7%	23%	47602	3151	147
Win32.Netsky.worm.24064.B_[E]	3	msPSUuu	3DE4	17%	24709	88	+0.	218	97%	19%	50607	3066	149
Win32.Netsky.worm.24840_[3d]	3	msPSUuu	3A44	21%	23642	93	+0.	286	74%	25%	47365	3086	149
Win32.Netsky.worm.22016.D_[2]	2	msPSCr	3E5F	1%	32814	98	+0.	563	###	14%	45305	3090	149
Win32.Netsky.worm.27136_[c4]	2	msPSCr	3E5F	1%	32443	98	+0.	173	0%	14%	45305	3090	149
Win32.Netsky.worm.25352_[3c3]	3	msPSUuu	3B04	22%	24789	94	+0.	322	90%	25%	48297	3106	149
Win32.Netsky.worm.29696_[50]	3	msPSUuu	3B04	24%	27248	94	+0.	322	90%	25%	48297	3106	149

Figure 13: The analysed results of a 'NetSky' sample from the AOI point of view.

g_funcid	filenames	funcname	cod	ln	inst_crc	ap	lib	pus	flag	fingerprint
0000009AF48C7A7	E00A40035686243:E009400E485C20A:E00A400011	SUB_004010A4	448	154	4102847187	8	0	25	1426067460	614J54515J54J
000000C42051A16	E00A40035686243:E007A0009CFD985:E009400E	SUB_00401332	586	195	542220648	2	0	25	1426067460	44J454454444445J
000000C931C0381	E00A40035686243:E005940E485C20A:E00A40011	SUB_00401708	570	201	634600860	4	0	48	1426067460	4444444J
00000029E9628195	E00A40035686243:E009400E485C20A:E00A40011	SUB_00401946	132	41	3919546969	1	0	2	1426067460	262626
00000058F0F0530F	E00A40035686243:E007A0009CFD985:E009400E	SUB_00401AC5	330	88	4142945039	3	0	15	1426067460	596960
000000467871EC2	E00A40035686243:E007A0009CFD985:E009400E	SUB_00401C10	166	70	207106391	1	0	11	1426067460	4JC4C4D545
0000037AB1436A5	E00A40035686243:E00A400011586A7B	SUB_00401D03	2779	890	2973985367	19	0	266	1426067460	E4EJEE4CJ45544
00000035406C22F1	E00A40035686243:E009400E485C20A:E00A40011	SUB_004028A7	163	53	108032861	1	0	7	1426067460	FE5C44
000000A11B8347A	E00A40035686243:E009400E485C20A:E00A40011	SUB_00402AF8	497	161	464794119	3	0	28	1426067460	44CC4E444444444
0000016E07E26F0	E0730008817D953A	SUB_00414680	1289	366	132280078	12	0	62	1426067460	4454444445444444

Figure 14: Opcode-Array information extracted from NetSky samples.

and the variants. Opcode-Array, however, turned out to be a far more precise and effective method in the detection of malware.

As for worms and Trojan horses, the AOI approach can be utilized in the detection of runtime-packed codes and, in the case of unpacked samples, can be used in search of similar samples.

Virus

A virus file means that a normal file is infected by a virus and, from the AOIR point of view, classification can be used to find the original file which is the closest match and locate the infection.

This means, since the rest of the uninfected parts are identical to that of original, that the original can be found by using the AOIR of each part of the section.

Therefore, by dividing the PE section into chunks of about 4KB in size and calculating the AOI of each, the original can be found by comparing a certain number of arrays to the AOIs.

When infected by a virus, AOIR becomes higher because of the inserted code and, in cases where it is inserted into the last

section, i.e. .reloc or .rsrc, the last section shows either an increased number of functions or increased function-occupied portion.

Figure 15 shows the key information of the original and the infected file where the similarity is not seen.

In many cases, sAOIR, shAOI, and slAOI are useful information in search of the original (Figure 16).

If too many matches or no matches are found from the above, the section can be divided into 4KB pieces and the original can be found by comparing the #_AOI of the divided parts (Figure 17).

The process above uses the characteristic of a virus that it does not change all parts of the original code and it is assumed that the original file can be found in the database.

Also, it can find the infected location of the virus in the sample and virus size.

DEFINITION-BASED DETECTION OF MALICIOUS CODES

An expert system will mainly depend on the classification technique, and this is based on the assumption that viruses and

worms duplicate themselves by their definitions.

This approach uses the basic definition of virus and analyses actions on the infectee. Parasitic viruses and overwriting viruses are good examples. Parasitic viruses append malicious codes at the end of the original file and change the EntryPoint or alter the control flow like EPO. Since most of the viruses have to modify the original code area to activate the virus code, detection of this kind of behaviour can be used to identify a virus.

If a sample code has been altered from the original file, it has to be suspected as a malicious code even if it has passed the classification process as a normal code.

EPO viruses can be identified by watching them making changes to the infected files. Even though malicious codes have anti-monitoring features, changes can always be monitored by comparing the original at file open and the altered at file close.

A virus-detection technique using monitoring methods is implemented and is in use. It is based on the scanner example of IFSKit, but implemented as system drivers using SDT hooking, and also is capable of monitoring registry accesses as well as files.

Worms have similar characteristics too, except in the case of

FILENAME	PEID	E #	SList	EP	SIZE	IAT	sNAME	sFNC	sCOV	sAOIR	sSIZE	shAOI	slAOI
notepad.exe	Microsoft	1E	3 msPSCdr	739D	36992	201	+0_.text	303	70%	64%	35494	2756	1166
notepad.exe	Borland	DtE	3 msPSCdR	248A4	103741	201	+2_.rsrc	8	5%	85%	173305	4210	1379
regedit.exe	Microsoft	1E	3 msPSCdR	168EC	62647	315	+0_.text	718	78%	58%	110057	2786	1343
regedit.exe	Microsoft	1E	3 msPSCdR	168EC	137007	315	+0_.text	718	78%	58%	110077	2786	1343
regedit.exe	Borland	DtE	3 msPSCdR	74AA4	130253	315	+2_.rsrc	9	3%	77%	163769	4210	1119

Figure 15: The key information of the original and the infected.

FILENAME	SECTION_1	SECTION_1	SECTION_1															
FILENAME	sNAME	sFNC	sCOV	sSIZE	shAOI	slAOI	sNAME	sFNC	sCOV	sSIZE	shAOI	slAOI	sNAME	sFNC	sCOV	sSIZE	shAOI	slAOI
notepad.exe	.text	303	64%	19708	2756	1166	.data	0	19%	407	407	407	.rsrc	0	55%	18887	2752	1379
notepad.exe	.text	281	64%	19708	2756	1166	.data	0	19%	407	407	407	.rsrc	8	84%	89447	4210	1379
regedit.exe	.text	718	58%	57093	2786	1343	.data	0	30%	312	312	312	.rsrc	0	36%	13975	1657	1119
regedit.exe	.text	718	59%	57103	2786	1343	.data	0	30%	312	312	312	.rsrc	0	70%	95977	3709	1119
regedit.exe	.text	690	58%	57093	2786	1343	.data	0	30%	312	312	312	.rsrc	9	76%	84549	4210	1119

Figure 16: In many cases, sAOIR, shAOI, and slAOI are useful information in search of the original.

FILENAME	sNAME	#PT	0_AOI	0_FNs	1_AOI	1_FNs	2_AOI	2_FNs	3_AOI	3_FNs	4_AOI	4_FNs	5_AOI	5_FNs	6_AOI	6_FNs	7_AOI	7_FNs
notepad.exe	.text	8	2727	214	2588	12	2590	15	#	21	2576	12	2622	12	+2683	17	1166	0
a.ex_	.text	8	2727	213	2588	11	2590	13	#	21	2576	9	2622	6	2683	8	1166	0

Figure 17: If too many matches or no matches are found, the section can be divided into 4KB and the original can be found by comparing the #_AOI of the divided parts.

FILENAME	PEID	E #	SList	EP	SIZE	IAT	sNAME	sFNC	sCOV	sAOIR	sSIZE	shAOI	slAOI
notepad.exe	Microsoft	1E	3 msPSCdr	739D	36992	201	+0_.text	303	70%	64%	35494	2756	1166
a.ex_	Borland	DtE	3 msPSCdR	248A4	103741	201	+2_.rsrc	8	5%	85%	173305	4210	1379

Figure 18: A comparison of the original and virus infected.

FILENAME	SECTION_1	SECTION_1	SECTION_1															
FILENAME	sNAME	sFNC	sCOV	sSIZE	shAOI	slAOI	sNAME	sFNC	sCOV	sSIZE	shAOI	slAOI	sNAME	sFNC	sCOV	sSIZE	shAOI	slAOI
notepad.exe	.text	303	64%	19708	2756	1166	.data	0	19%	407	407	407	.rsrc	0	55%	18887	2752	1379
a.ex_	.text	281	64%	19708	2756	1166	.data	0	19%	407	407	407	.rsrc	8	84%	89447	4210	1379

Figure 19: The AOI in the entire section of the divided parts can be used in a database search.

FILENAME	sNAME	#PT	0_AOI	0_FNs	1_AOI	1_FNs	2_AOI	2_FNs	3_AOI	3_FNs	4_AOI	4_FNs	5_AOI	5_FNs	6_AOI	6_FNs	7_AOI	7_FNs
notepad.exe	.text	8	2727	214	2588	12	2590	15	2756	21	2576	12	2622	12	+2683	17	1166	0
a.ex_	.text	8	2727	213	2588	11	2590	13	2756	21	2576	9	2622	6	2683	8	1166	0

Figure 20: The .text section is composed of seven parts.

FILENAME	sNAME	#PT	0_AOI	0_FNs	1_AOI	1_FNs	2_AOI	2_FNs	7_AOI	7_FNs	8_AOI	8_FNs	24_AOI	24_FNs	25_AOI	25_FNs
notepad.exe	.rsrc	9	1379	0	2752	0	2708	0	2062	0	544	0				
a.ex_	.rsrc	26	1379	0	2752	1	2708	1	2062	1	3301	0	4159	0	+178E	0

Figure 21: The first eight parts are identical while 17 parts are newly added, and EntryPoint is located in 26th section.

file-less worms. Once a worm invades a computer, it is copied to a limited location for execution. After the copy process is done, it registers itself in the registry for execution. Therefore by handling this behaviour, it is not difficult to identify the infection processes of many worms.

However, when worms spread through SMTP and network transfer, it is difficult to verify them with current tools. Even though it may be a feasible solution to set up a VirtualPC (or VMware) with virtual network environment, identifying the behaviour under that environment is still difficult.

From the behavioural point of view, identifying a Trojan horse is not easy. From the classification point of view, identification is much easier but when there is a small number of variants or they were created with a specific purpose, it is more difficult to identify them and we have to rely on the classification technique – and, with no information available, manual analysis has to be performed by a human.

If behavioural characteristics can be collected and stored in a database, it may be worth a try, however to define harmful intention from a behaviour itself is still very difficult.

With the case of a virus as an example, the technique of finding the original and diagnosing a virus by comparing it with the infected file is described below. Diagnosis becomes precise when it is supported by the result of a dynamic analysis.

Parasitic virus

Sample 'a.ex_' is a case where 'notepad.exe' is infected by a virus. Both dynamic analysis and static analysis provide enough information to support that this is a virus.

A comparison of the original file and the virus infected file is shown in Figure 18.

'a.ex_' is the suspected virus sample, and a database search shows that the amount of information in a section is the match of 'notepad.exe'.

A database search also shows that 'notepad.exe' matches the sample by the amount of information (sAOIR, shAOI, slAOI) in the .text section. The amount of information in either the entire section or the properly divided parts can be used in a database search to find similar files with higher accuracy (Figure 19).

Based on the analysis, it is assumed that the virus code is inserted in the third section, which is the .rsrc section, and it is a packed code with sCOV being about 85%. The fact that the .rsrc section did not have functions

(sFNC) in the original also supports this.

From a closer look, the .text section is composed of seven parts and AOI values are identical in both, which tells us that their originals are identical (Figure 20).

On the other hand, a closer look at the .rsrc section shows that first eight parts are identical while 17 parts are newly added, and the EntryPoint is located in 26th section. This increases the probability of a virus infection (Figure 21).

Based on the information, a brief description as shown in Figure 22 can be provided.

A virus detector under a dynamic analysis environment also identifies this as a virus after watching the suspicious behaviour of the sample. The fact that the sample opens, modifies a file and the size of added data is over 70KB becomes the basis of this decision. See Figure 23.

EPO virus

The sample below turned out to be an EPO after the final analysis. In the case of EPO, jumps or calls are first looked for and then the body of a virus is found as a result.

A database search first reports that the sample is similar to 'regedit.exe' (see Figure 24). In this case, both match by the number of functions and the code coverage area, and sAOIR, shAOI, slAOI in the .text section.

```

FILENAME: A.EX_ ( VIRUS )
ORG : (SAFE) NOTEPAD.EXE(97%,DIFF=6), 37KB, msPScdr, EP=0x739D

Traditional Virus ( Parasitic Append )
Virus at part 95t~26st in .rsrc
Virus may be packed

A.EX_, EXE, Borland Delphi 3.0, EXE, 3, msPScdr, EP=0x248A4
SIZE 104K (add 67KB from ORG )
EntryPoint=0x248A4, .rsrc <changed from 0x739D, .text>

SECTIONS
.text 7 parts, 281 Funcs (EQU from ORG)
.data MEM reserve
.rsrc 26 parts, 8 Funcs <changed, add 17 parts>

Analysis Summary
2 files created in C:\WINDOWS\Downloaded Program Files
- tnumb.db, munju.exe
2 files created in C:\
  tnumb.db, munju.exe
tnumb.dll loaded on 12 files
Modified 4 files
- notepad.exe regedit.exe project.exe getfile.exe
    
```

Figure 22: A brief description based on the information provided.

```

6211 Explorer EXE ZwCreateFile.W 00100080 HANDLE=00000714 : W?7WC:Wnotepad.exe(36), OrgSize=67584
6212 Explorer EXE ZwReadFile 00000714 64
6214 Explorer EXE ZwReadFile 00000714 248
6216 Explorer EXE ZwReadFile 00000714 40
6218 Explorer EXE ZwWriteFile 00000714 uPos=552 Size=40 ** MODIFY ORIGINAL AREA
6219 Explorer EXE ZwWriteFile 00000714 uPos=67072 Size=71320 ** APPEND VIRUS BODY
6220 Explorer EXE ZwWriteFile 00000714 uPos=138952 Size=516 ** APPEND VIRUS BODY
6221 Explorer EXE ZwWriteFile 00000714 uPos=138908 Size=1 ** APPEND VIRUS BODY
6222 Explorer EXE ZwWriteFile 00000714 uPos=138909 Size=1 ** APPEND VIRUS BODY
6223 Explorer EXE ZwWriteFile 00000714 uPos=138910 Size=1 ** APPEND VIRUS BODY
6224 Explorer EXE ZwWriteFile 00000714 uPos=138911 Size=1 ** APPEND VIRUS BODY
6225 Explorer EXE ZwWriteFile 00000714 uPos=138912 Size=1 ** APPEND VIRUS BODY
6226 Explorer EXE ZwWriteFile 00000714 uPos=138913 Size=1 ** APPEND VIRUS BODY
6227 Explorer EXE ZwWriteFile 00000714 uPos=138914 Size=1 ** APPEND VIRUS BODY
6228 Explorer EXE ZwWriteFile 00000714 uPos=138915 Size=1 ** APPEND VIRUS BODY
6229 Explorer EXE ZwWriteFile 00000714 uPos=138916 Size=1 ** APPEND VIRUS BODY
6230 Explorer EXE ZwWriteFile 00000714 uPos=138917 Size=1 ** APPEND VIRUS BODY
6577 Explorer EXE ZwWriteFile 00000714 uPos=224 Size=248 ** MODIFY ORIGINAL AREA
6578 Explorer EXE ZwClose : F6865D60 ** MAY BE PARASITIC VIRUS, APPENDED TYPE **
6579 Explorer EXE DeleteHandle 00000714 H 00000000 L 00022000
    
```

Figure 23: The sample is identified as a virus after watching suspicious behaviour.

The structure of codes and EP are same, but EP has been moved from the position of about 65% of the file size to 37% of that, and the file size became larger than this.

As we look at the sections, the .text section has been slightly altered and the .rsrc section is now significantly larger (see Figure 25).

Taking a closer look at the parts, it can be seen that the 9th part in the .text section has been altered, which is suspected to be the location of the code to call the EPO virus (see Figure 26).

The data used to identify the location of the virus is shown in Figure 27, and the virus is located at the 10th–34th part. The first 1st–9th parts have the same amount of information.

FILENAME	PEID	E #S	StList	EP	SIZE	IAT	sNAME	sFNC	sCOV	sAOIF	sSIZE	shAOI	slAOI
regedit.exe	Microsoft 1E	3	msPScdr	168EC	62647	315	+0_.text	718	78%	58%	110057	2786	1343
b.ex_	Microsoft 1E	3	msPScdr	168EC	137007	315	+0_.text	718	78%	58%	110077	2786	1343

Figure 24: A database search first reports that the sample is similar to 'regedit.exe'.

FILENAME	SECTION_1				SECTION_1				SECTION_1									
	sNAME	sFNC	sCOV	sSIZE	shAOI	slAOI	sNAME	sFNC	sCOV	sSIZE	shAOI	slAOI	sNAME	sFNC	sCOV	sSIZE	shAOI	slAOI
notepad.exe	.text	718	58%	57093	2786	1343	.data	0	30%	312	312	312	.rsrc	0	36%	13975	1657	1119
b.ex_	.text	718	59%	57103	2786	1343	.data	0	30%	312	312	312	.rsrc	0	70%	95977	3709	1119

Figure 25: The .text section has been slightly altered and the .rsrc section is larger.

FILENAME	sNAME	#T	0_AOI	0_FNs	1_AOI	1_FNs	2_AOI	2_FNs	7_AOI	7_FNs	8_AOI	8_FNs	9_AOI	9_FNs	21_AC	21_FN	22_AC	22_FN	23_AC	23_FN	24_SZ
regedit.exe	.text	24	2362	314	2431	12	2764	14	2400	37	2597	23	2725	24	+245	35	2301	0	1343	0	
b.ex_	.text	24	2362	314	2439	12	2764	14	2400	37	2599	23	2725	24	+245	35	2301	0	1343	0	

Figure 26: It is the 9th part of the .text section that has been altered. This is suspected to be the location of the code to call the EPO virus.

FILENAME	sNAME	#T	0_AOI	0_FNs	1_AOI	1_FNs	2_AOI	2_FNs	8_AOI	8_FNs	9_AOI	9_FNs	31_AC	31_FN	32_AC	32_FN	33_AC	33_FN
regedit.exe	.rsrc	10	1119	0	1591	0	1304	0	1567	0	740	0						
regedit.exe.El	.rsrc	34	1119	0	1591	0	1304	0	1567	0	1690	0	3709	0	2269	0	132	0

Figure 27: Data used to identify the location of the virus.

If the information is put together, we can make an assumption that the EPO-calling code is in the 9th part of the .text section and the virus code is at the end of the .rsrc section.

This can be applied to the samples that correspond to definition of viruses, and we can conclude that they are viruses since both dynamic analysis and static analysis give the same results (see Figure 28).

```

FILENAME: B.EX_
ORG : (SAFE) REGEDIT.EXE(100%), 62KB, msPScdr, EP=0x168EC

May be EPO virus.
Virus at part 10st~34st

B.EX_, EXE, Microsoft Visual C++ 7.0, EXE, 3, msPScdr, EP=0x168EC
SIZE 137K (add 75KB from ORG )
EntryPoint=0x168EC, .text <unchanged>
virus branch at 9 st part of .text
virus virus at part 11~34 of .rsrc
virus may be encrypted

SECTIONS
.text 24 parts, 718 Funcs **EP <unchanged>
.data 1 parts, MEM reserve
.rsrc 34 parts <changed, add 24 parts>

Analysis Summary
Modified 3 files
- notepad.exe regedit.exe viewfunc.exe
    
```

Figure 28.

AUTO SIGNATURE GENERATION

One other major factor of an expert system is the safe signature extraction technique. No matter what criterion is applied to signature extraction, it is hardly possible for an analyst to follow guidelines strictly since tens of hundreds of samples have to be processed each day.

Therefore, as long as the signature is extracted from parts that represent the characteristics of functions or their roles, extraction can become safe and, as a result, false-positives can be reduced, preventing human errors from inexperience by consistently keeping the same criterion throughout signature extraction.

To ensure safety in signature extraction, it is very important to know the language the code was written in and understand the way each language compiles source codes and make them into binaries. The better the characteristics of languages are understood, the safer the signature extraction becomes.

Using signatures is currently ineffective for viruses but very effective for malware written in high-level languages where the effectiveness is over 80%.

The diagram in Figure 29 depicts the signature extraction process briefly, and the procedure is dependent on whether a file is packed or not.

Signature generation requires more in-depth consideration because if signatures are generated for not only the malicious codes but also normal codes, these signatures can be compared to the signature of a new malicious code to check potential errors in both signatures. This could contribute to reducing

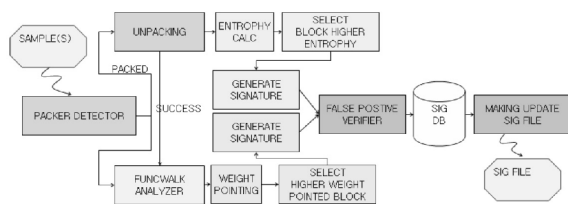


Figure 29: The signature extraction process.

the similarity test in the classification technique and reducing false positives too.

The automated signature extraction system contains the classification technique and is highly effective, capable of processing 200 cases per hour, and is performing a huge role in reducing false positive detection of viruses as worms or Trojan horses with the help of normal signatures.

CONCLUSION

An expert system is a system that stores and provides accumulated knowledge and experience of many experts.

Working as an anti-virus researcher for many years, I have seen many people coming into this area, but giving up and leaving after a while. It is the desperate requirement for accumulated knowledge and experience that often frustrates new researchers. I believe that as the expert system takes steps toward to the solution, this can make the job much easier and free people from treadmill jobs.

The core of the expert system is to store the valuable information and, based on the information, to make decisions or to provide sufficient information to help in the decision process.

Viruses and many worms are easily identifiable because they have typical behaviours such as infection and self-replication. However, identifying tools with special purposes, such as hacking tools for games, is a very difficult job.

This system has a sample classification technique at its core, and also includes signature extraction where it is possible for an analyst to make errors.

For the remainder of our study, we will be focusing on branch footprinting in dynamic analysis that is yet not complete and detection of Trojan horses.

The results of these studies are expected to further improve the efficiency of the system. And if we could identify and trace the special purpose tools such as game hacking tools that many variants are found with, it will help with reducing the number of analyses to be performed significantly.

The expert system, in the end, will merely be the system that is a collection of our current automation environment. If a sample can be handled automatically to signature extraction, and if human error can be reduced from it, that will have a huge impact on us.

REFERENCES

- [1] Hwang, K-b. V3Genie System: an automated multi-scanner system. Proceedings of the Virus Bulletin International Conference. 2003.
- [2] Hwang, K-b.; Jung, D. SmartAegis Software Requirement Specification. 2005.
- [3] Hwang, K-b.; Jung, D. SmartAegis Final Report. 2006.
- [4] Stewart, J. (Joe Stewart) OllyBone: Semi-Automatic Unpacking on IA-32. Recon 2006.
- [5] Jung, D. Generic Unpacking based on Controlled Virtual Machine. Internal Paper, AhnLab, 2007.
- [6] Schipka, M. Tracing Execution Paths. Proceedings of the Virus Bulletin International Conference. 2005.
- [7] Ferrie, P. Attack on Virtual Machine Emulators. Proceedings of AVAR 2006.
- [8] Ko, H. Branch Foot-printing Tech. based on Branch Trap. Internal Paper, AhnLab. 2007.
- [9] Flake, H. Graph-Based Binary Analysis. BlackHat Briefings, 2002.
- [10] Carrea, E.; Erdelyi, G. Digital Genome Mapping. Proceedings of the Virus Bulletin International Conference. 2004.
- [11] Perriot, F. Towards Agile Reverse Engineering. Proceedings of the Virus Bulletin International Conference. 2005.
- [12] Hwang, K-b.; Jung, D. Collaborative Malicious Codes Analysis System. Proceedings of the Virus Bulletin International Conference. 2006.