

Figure 2: The processes can be categorized into five groups, each containing various technical aspects.

Previously when we had less automated processes, the procedure was much more complex. As we introduce more automation tools and gather experience in this system, the procedure is becoming simpler.

If the procedure is simplified, the processes can be categorized into five groups, each containing various technical aspects, as shown in Figure 2.

The procedure can be described as follows: when a sample is received, sample identification is the first step taken, where previous records are looked up and verified, various anti-virus engines are used to check the sample, and information from other companies is referenced. Next, sample analysis is performed. If the sample is runtime packed, unpacking is carried out prior to sample analysis.

When analysis is over, sample classification is performed based on the analysis record. Finally, auto signature generation generates a diagnostic signature and applies this to the engine.

SAMPLE IDENTIFICATION

When a sample is obtained, it is passed to the AV Engine Team. The sample is checked against previously identified samples and checked using anti-virus products from other companies. We made substantial progress in this step after we published [1], and the improvements are in practical use.

In this process, files are checked for their size and hash value (SHA-2 or MD5) stored in database and they are identified as either normal, malicious or error, which prevents redundancy in processing.

1차 샘플코드	EC0FF9C000000000	접수일당	apn0p0	접수일시	2007-06-20 09:00:00
처리유선	3	출력일여부	N	이수명	
ID	09101000	고객명	0000000000000000	고객번호	10000000
전화번호	0000000000000000	최대본	000-0000000000	고객유형	사이트
주소					
이메일	apn0p0@apn0p0.com				
제목	재부팅하면 계속 감염되는 현상				
문의내용	* HTML Tag? 제거된 내용입니다. 안녕하십니까? 상대방의 컴퓨터에서 전달받은 의심파일입니다. V3에서 진단은 되지만, 재부팅하면 다시 동일하게 감염된다고 합니다. (이벤트 로그 포함)				
HTML로 보기	[HTML로 보기]				
원본	원본 파일을 분석 부탁드립니다. V3에서 진단은 되지만, 재부팅하면 다시 동일하게 감염된다고 합니다. (이벤트 로그 포함) 프로세스: echo.exe 분석되었습니다. 원본 파일 분석 부탁드립니다. 스크립트 실행 시도				
관련사항	*관련된 샘플을 Echo 샘플들 접수단에서 송부될터일정 샘플이 있을 경우 제공됩니다.				

Figure 3: Information provided from sample identification.

When a sample is identified as malicious by competitors' anti-virus scanners, there is a high probability that the sample is a malware variant. This information can be decisive when we have to choose whether to advance through the next steps or skip them and go straight to the automatic signature generation.

Information provided by sample identification is depicted in Figure 3. Information is given regarding the reporter, the reason, and the previous record. It is also possible to track the process status.

The role of AMES is to provide the maximum amount of information by looking up the database, running multiple anti-virus scanners and, when diagnosed as malicious, collecting information from the web.

Diagnoses of competitors' anti-virus scanners are always a very important reference for our diagnosis.

PACKED SAMPLE PROCESSING

Samples nowadays are often runtime-packed or encrypted. Therefore unpacking samples has become essential for analysis and signature extraction. Many analysts are wasting huge amounts of time on first finding out how samples are packed and then unpacking them – and for analysts with less experience this process is even harder and more time-consuming.

The different processes used for detecting runtime-packed samples and unpacking them can be classified into three groups as depicted in Figure 4.

All known approaches are used for unpacking: algorithmic unpacking engine, which is the most precise and effective method using a n anti-virus engine; emulating, which takes more time but can handle complex algorithms; and general unpacking, which runs samples on a controlled virtual machine.

Since there are anti-emulating, anti-monitoring, and virtual machine detect as for emulating, a packer detector has to decide on a suitable emulating algorithm first. General unpacking detects the timing of unpacking by monitoring the unpacking and execution status under a virtual environment.

Packed sample detection

Various methods are used in the detection of a packed or encrypted sample; utilization of algorithms in anti-virus engines, utilization of tools such as PEiD, analysis of specific portions of the sample and so on.

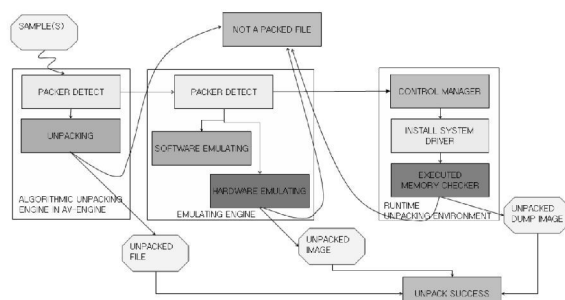


Figure 4: The process of detecting runtime-packed samples and unpacking them can be classified into three groups.

We use information theory to quantify the amount of information (AOI), and use it to detect a packed or encrypted sample.

AOI can easily be calculated by using lossless compression, and this can distinguish packed from non packed, and encrypted from non encrypted. Even though the distinction between packed and encrypted is not clear yet, analysis of data characteristics is assumed to be helpful in resolving this uncertainty.

This method is based on the fact that packed files have low redundancy and encrypted files have higher randomness, and allows fast and effective detection of packed or encrypted files. Analysis of non packed files is shown in Figure 5. Non packed codes have an AOIR (Amount of Information Rate) of about 50–60% of the entire code and about 50% of the file entity, in most cases.

Analysis of packed files is shown in Figure 6. Here, AOIR are mostly greater than 80% of the codes, and about 90% of the file entity. shAOI (the highest AOI in section) is noticeably large too.

Runtime-packed files have characteristics such as: AOIR greater than 80%, smaller number of IAT (Import Address Table), smaller number of functions (sFNC) and less than 30% function occupied area (sCOV).

Unpacking

Instead of using API hooking and file system hooking, we can set up an artificial analytic system environment and use Translation Look-aside Buffer (TLB) or a controlled virtual machine.

In [2, 3] we described utilizing TLB in the detection of buffer overruns as well as its potential for use in unpacking, and [4] proved that unpacking can be achieved, by publishing the tool named ‘OllyBonE’.

The Instruction TLB (ITLB) method is to record each memory page to Data TLB (DTLB) only, to unpack a

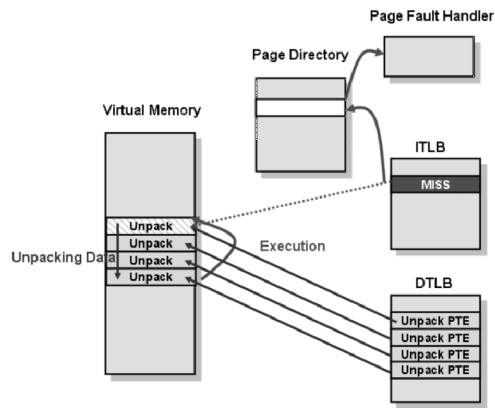


Figure 7: The ITLB method.

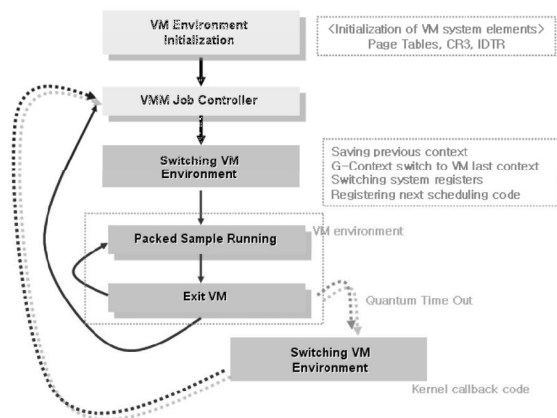


Figure 8: The controlled virtual machine method.

FILENAME	PEID	ETYP	#S	StList	EP	AOIR	SIZE	IAT	sNAME	sFNC	sCOV	sAOIR	sSIZE	shAOI	slAOI
accwiz.exe	Microsoft Vi	EXE	3	msPSCdr	0000BE38	32%	57166	175	+0_text	378	68%	58%	52292	2684	1688
ahui.exe	Microsoft Vi	EXE	3	msPSCdr	6240	56%	55113	147	+0_text	478	90%	63%	72336	2979	1585
alg.exe	Microsoft Vi	EXE	3	msPSCdr	00005B06	46%	20638	104	+0_text	230	28%	58%	33400	2777	1401
arp.exe	Microsoft Vi	EXE	3	msPSCdr	00001EA6	44%	8694	46	+0_text	93	79%	65%	5389	3009	2380
asr_fmt.exe	Microsoft Vi	EXE	3	msPSCdr	000059FF	50%	14705	189	+0_text	296	61%	50%	16677	2627	0
asr_ldm.exe	Microsoft Vi	EXE	3	msPSCdr	2_00E+51	30%	9721	68	+0_text	95	70%	61%	5006	2698	2308
at.exe	Microsoft Vi	EXE	3	msPSCdr	00002D5C	47%	14017	82	+0_text	177	78%	69%	8562	2900	2770
atmadm.exe	Microsoft Vi	EXE	3	msPSCdr	00001C7D	39%	5810	35	+0_text	67	70%	69%	2830	2830	2830
attrib.exe	Microsoft Vi	EXE	3	msPSCdr	000022A8	48%	5515	86	+0_text	108	54%	57%	4747	2544	2203
auditusr.exe	Microsoft Vi	EXE	3	msPSCdr	3573	48%	6937	58	+0_text	96	78%	53%	4347	2207	2140
blastcln.exe	Microsoft Vi	EXE	3	msPSCdr	000053CC	54%	39049	109	+0_text	358	96%	63%	72338	3099	1461
cacls.exe	Microsoft Vi	EXE	3	msPSCdr	3518	45%	10138	62	+0_text	126	78%	66%	5461	2758	2703
calc.exe	Microsoft Vi	EXE	3	msPSCdr	12475	47%	54700	132	+0_text	287	91%	54%	75647	2741	1698
charmap.ex	Microsoft Vi	EXE	3	msPSCdr	0000AB2D	51%	41102	199	+0_text	326	59%	66%	54779	2888	2255
chcp.com	Microsoft Vi	EXE	3	msPSCdr	000016DE	47%	3618	66	+0_text	84	37%	63%	2587	2587	2587
chkdsk.exe	Microsoft Vi	EXE	3	msPSCdr	0000257E	47%	5551	76	+0_text	101	82%	55%	4567	2304	2263
chknfts.exe	Microsoft Vi	EXE	3	msPSCdr	000021F7	48%	5485	87	+0_text	101	53%	57%	4675	2361	2314

Figure 5: Analysis of non packed files.

FILENAME	PEID	ETYP	#S	StList	EP	AOIR	SIZE	IAT	sNAME	sFNC	sCOV	sAOIR	sSIZE	shAOI	slAOI
AB07051E	Nothing foun	EXE	3	msPSuUu	000619C	98%	198195	3	+1_xwxr1	6	0%	101%	324162	4210	2862
AB07051E	Nothing foun	EXE	3	msPSuDr	7860	91%	12607	6	+1_data	10	0%	100%	12304	4187	4031
AB07051E	Nothing foun	DLL	3	msPSuDr	00006D5	56%	5832	6	+1_data	7	1%	102%	4204	4204	4204
AC070315	Nothing foun	DLL	3	msPSuUu	000139F	87%	30349	3	+1_uufu1	5	13%	93%	53604	4208	3252
AC070321	Nothing foun	DLL	3	msPSuUu	00013FA	86%	32259	3	+1_uufu1	5	16%	91%	52727	4208	3073
AC070321	Nothing foun	EXE	3	msPSuUu	000642D	98%	211709	3	+1_uufu1	6	0%	101%	324747	4210	3231
AC07033C	Xtreme-Prote	EXE	3	msPSuUu	0000A5D	94%	16801	11	+1_?u1	23	20%	98%	24330	4208	3290
AC070403	Nothing foun	EXE	3	msPSuUu	000263A	94%	64210	3	+1_e3dx8yc	6	0%	98%	121113	4210	3117
AC070403	Nothing foun	DLL	3	msPSuUu	000140C	87%	30426	3	+1_!onk1	4	10%	93%	53379	4208	3049
AC070403	Nothing foun	EXE	3	msPSCur	0000BFA	88%	54266	55	+0_text	106	21%	94%	100991	4210	2567
AC070403	UPX 0.89.6	EXE	3	msPSuUu	9800	93%	19083	5	+1_LJPX1	1	1%	101%	25039	4208	4155
AC070403	nSpack V2.x	EXE	3	msPSuUu	0000CB3	87%	15292	8	+1_nsp1	12	13%	95%	23352	4208	2312
AC070404	Nothing foun	EXE	3	msPSuUu	0	87%	28261	3	+1_28khk1	25	13%	92%	45611	4208	3019

Figure 6: Analysis of packed files.

runtime-packed file, and to catch the exception timing when OEP execution fails due to ITLB missing Page Table Entry (PTE). See Figure 7.

Another method is using a controlled virtual machine. This method, illustrated in Figure 8, is to execute a packed sample under a virtual environment that is separated from the host operating system and managed by a separate VMM, and to watch the memory at specific times to get the unpacked image.

A controlled virtual machine, at initialization, sets up separate memory spaces and page tables, and individual IDTs to handle exceptions.

The virtual machine’s loader executes a packed code and it is mapped to the prepared memory space. The running code is switched to the host OS thread by the scheduler,

which is shared by the host OS and the virtual machine. Then, when the task is switched to VMM, TrapFrame information is used to obtain the register information of the last execution on the virtual machine and, for VMM to run normally, that last register information of VMM is updated to the TrapFrame information. By repeating this procedure, the virtual environment and the host OS can cooperate with each other.

In this virtual environment, the execution of a sample is terminated when a call is made to the API that is unrelated to the packer's characteristic, and the unpacked image can be obtained by looking at the memory at this point [5].

SAMPLE ANALYSIS

Three approaches exist in sample analysis: static analysis, dynamic analysis, and human interactive analysis. While static analysis involves the use of *IDA* and the ADVFUNC module, dynamic analysis utilizes monitoring methods in the detection of viruses and worms by their defined behaviours and is fully automated.

Human interactive analysis is carried out by a human expert, and takes place when dynamic analysis fails.

Static analysis

The benefit of static analysis is that entire codes can be analysed. Since it does not involve execution, some codes can be executed virtually but in a limited way, and, since calls to API are not evaluated, it allows only for very simple operations.

Moreover, static analysis cannot cope with dynamic decryption applied codes, and analysis of runtime-packed codes also has no meaning unless they are unpacked. Performing analysis on the memory under execution, of course, can remove these problems.

Since static analysis, unlike dynamic analysis, is performed throughout the entire code, it can determine the EPO codes or the codes that are not executed because one of the execution conditions is not satisfied.

Utilizing powerful tools such as *IDA* is a good approach in static analysis. However, there are occasions when *IDA* fails to find functions and to recognize libraries.

For uncommonly structured codes, such as Visual Basic codes, *IDA* offers very limited capability and therefore we recommend the use of customized tools with *IDA*.

The ADVFUNC module, which was introduced in [3], can be used for finding functions and recognizing the libraries of Visual Basic codes. *IDA* can perform further analysis once the information is passed to it by plug-ins. See Figure 9.

In the case of HLL-written codes, combining ADVFUNC and *IDA*, can lead to improved analysis and therefore provide more in-depth information.

ADVFUNC is capable of interpreting the P-Code of Visual Basic codes, of analysing that information, and of performing analysis on a running process as well. Its strengths can effectively be utilized by passing the ADVFUNC module's analytic information to *IDA*, and by using *IDA* to perform static analysis.

Dynamic analysis

Dynamic analysis consists of execution, behaviour monitoring, and analysis. The *VirtualPC* and *VMware* are used as environment systems, and *Process Monitor* or *Ethereal* are used as monitoring tools.

The most important key aspect of automated dynamic analysis is verifying the sample's execution status. While the behaviours of viruses or worms are already defined and can be analysed according to their definitions, behaviours of other kinds of malware are hard to define and therefore difficult to analyse.

A program terminates on conditions as follows [6, 7]:

1. Broken codes terminate the program with an exception.
2. A malicious code terminates on detection of VMM.
3. Execution conditions are not all satisfied.
4. Execution continues.
5. A program terminates normally.

Dynamic analysis requires a technique to verify that a code is fully executed. We do not have statistical data to identify this normal execution (full execution) yet, and what we are doing currently is repeating executions and watching them for more than five minutes.

The Branch footprinting technique is under development in order to monitor and record every function call of a sample. A single step or DR register can be used for this technique, but Branch Trap is assumed to be a more effective method [8].

In dynamic analysis, execution of a malicious code can be watched by monitoring API calls. A code page check can also be performed every time a call to an API is made.

Hooks using SDT and filter drivers to watch the file I/O status using Installable File System (IFS) are currently in use.

Human interactive analysis

Human interactive analysis is when an analyst performs an analysis and makes a decision. Unlike the automation, we provide an adequate environment and sufficient monitoring tools to an analyst and rely on his abilities. With the help of powerful debuggers and monitoring tools, analysis can be done more quickly. *OlyDbg* and *WinDbg* are the most commonly used tools nowadays.

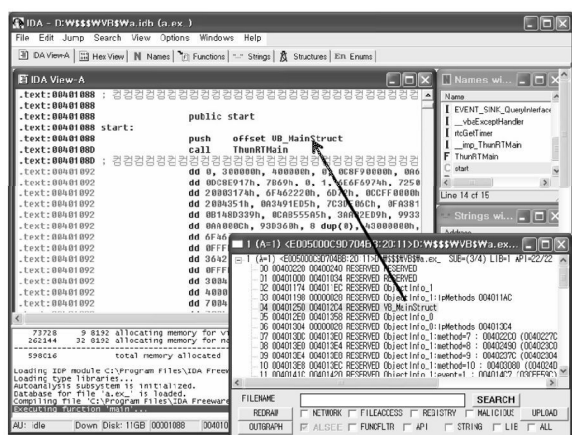


Figure 9: The ADVFUNC module can be used for finding functions and recognizing libraries of Visual Basic codes. *IDA* can perform further analysis.