



# 악성코드 완벽 분석 입문 과정



## 최 일 선

### 주요 경력

- 보안프로젝트 기술 이사
- 보안 솔루션 개발
- 파이썬을 활용한 마인크래프트, 파이썬 오픈 소스를 활용한 악성코드 분석, 파이썬을 활용한 데이터 분석 장기과 정 외 다수의 온라인 강의 제작
- 저서 : 비박스를 환경을 활용한 웹 모의해킹 완벽 실습

The collage includes:

- A large blue rectangular graphic with the text "정보보안 전문가 플랫폼" at the top, followed by a large white "보안" character and "프로젝트" below it.
- A business card for "최일선 기술 이사" (Choi Il Sun, Director of Technology). It lists a phone number (010.9891.5357) and email (isc0304@naver.com).
- A screenshot of a presentation slide titled "비박스 환경을 활용한 웹 모의해킹 완벽 실습" (Web Mocking Attack Practical Experiment Using VPS Environment). The slide features a small image of a person and text about the project's team members.



1. 리버싱 기초
2. 윈도우 실행 파일과 패커
3. 악성코드 기초 분석
4. 악성코드 고급 정적 분석: 아이다 활용
5. 악성코드 주요 행위 분석
6. 실전 악성코드 분석 맛보기



# 악성코드 분석 입문기 X-파일

보안프로젝트 최일선



**신한데이터시스템  
디지털의 길을 그리다**

**제8회 신한시큐어 정보보안 토크 콘서트**

일시 : 2017.12.27 (수) 13:00 ~ 18:00

장소 : 중앙대학교 310관 B601

대상 : 보안에 관심 있는 학생  
취업준비생 및 일반인

자세히 보기

주최: 신한데이터시스템, 중앙대학교 산업보안학과  
후원: 사큐리티플러스, 보안대첩, 보안프로젝트, 테크앤파트너사무소



- 1. 범용적으로 적용하기 위해 노력했지만 지극히 개인적인 생각입니다!**
- 2. 같이 성장하여 악성코드 분석에 취업에 성공한 사람의 이야기를 같이  
놓었습니다.**
- 3. 많은 노력이 뒷받침 돼야 합니다!**



- 나는 어떤 사람이었나...

- 수도권 대학 경영학과, 신학과 전공
- 대학교에서는 음악만
- 컴퓨터는 게임만
- 호주에서 4년을 보낸 외국인 노동자
- 컴퓨터 입문 동기 : 게임 자동화





# X 파일 1 – 천리 길은 한 걸음부터

# X 파일 1 – 천리 길은 한 걸음부터



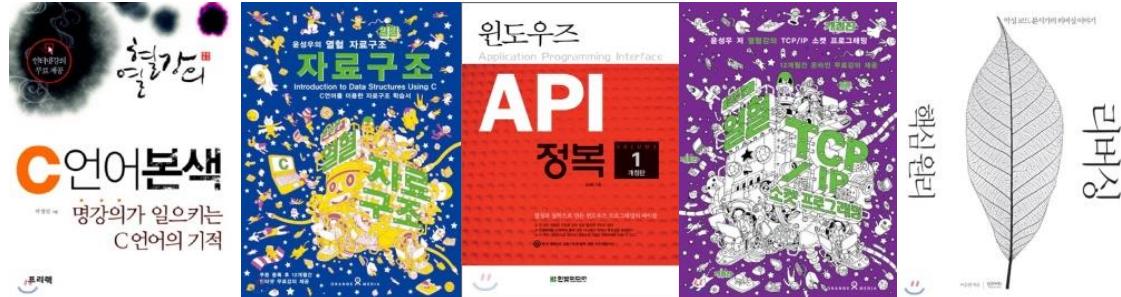
## • 입문은 책부터!

### - 책을 독파하라!

- 가장 쉬운 전문가와의 대화
- 전문가의 의견을 물어보지 않아도 알 수 있다.
- 모든 전공 서적은 해당 관련 내용을 10권정도 읽으면  
나머지는 유사 내용

– 모르는 내용만 빠르게 습득 가능!

### • 책 정독은 책 카피!





- **학원도 괜찮다!**

- 나도 OO 학원 수강생!(검색하면 이 학원만 나옴)
- 학원 또는 커뮤니티 등은 정보의 원천!





## X 파일 2 – 효율적이고 짧은 지름길



- 전문가를 찾아라!

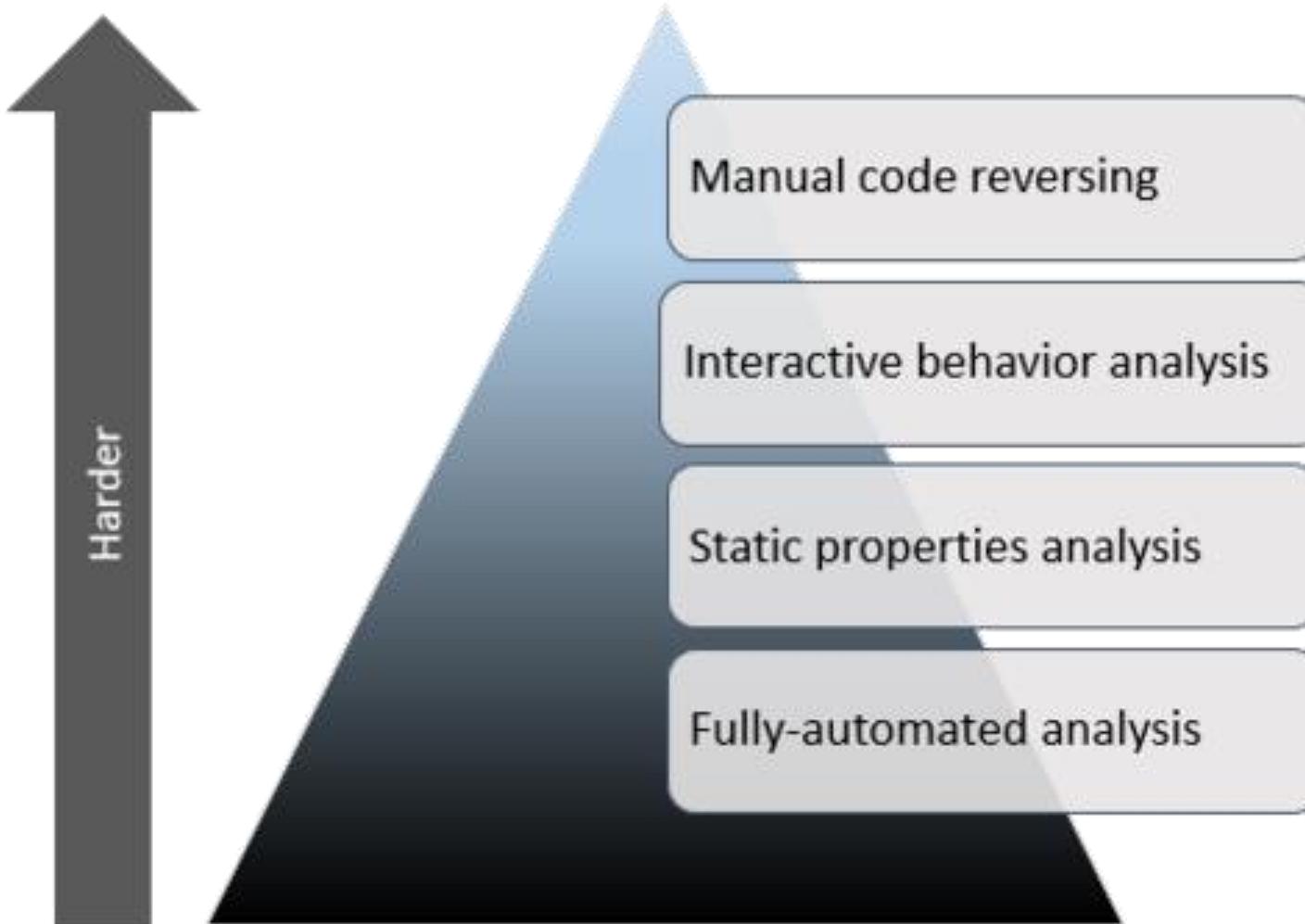
- 정보보안 분야의 수요를 바르게 파악
- 정보보안 분야의 취업 정보를 빠르게 파악
- 취업을 위해 어떤 공부를 해야 할지 알 수 있음
- 불안감을 해소하여 취업준비에 전념할 수 있게 됨
- 공부가 실무에 쓰인다는 점은 동기 부여로 이어짐

EXPERT





- 전문가를 찾아라! - 악성코드 분석의 네 가지 접근 방법





- 전문가를 찾아라! - 악성코드 분석의 네 가지 접근 방법

분석 방법	설명
<b>완전 자동화 분석</b> (Fully-automated analysis)	<ul style="list-style-type: none"> <li>정적 분석 및 동적 분석을 통해 악의적인 행위를 판단</li> <li><b>자동 분석 수행</b> (파일 생성, 수정 과정의 분석, 레지스트리 분석, 네트워크 분석 등)</li> <li>전문가에 의해 분석되는 만큼 <b>상세하거나 정확하지 않을 수 있음</b></li> <li>악성코드 <b>분석 제공 서비스</b></li> </ul>
<b>정적 속성 분석</b> (Static properties analysis)	<ul style="list-style-type: none"> <li>악성코드의 추가 분석을 위해 필요한 단계</li> <li>문자열 헤더 정보, 해시 값, 리소스 정보, 패킹 여부 등 <b>신속하게 정보 획득</b></li> <li>정보들을 활용해 실행 파일 간의 <b>비교 데이터베이스</b>를 구성</li> <li>바이러스 토탈(VirusTotal) 서비스</li> </ul>
<b>대화형 동적 분석</b> (Interactive Behavior Analysis)	<ul style="list-style-type: none"> <li>레지스트리, 파일시스템, 프로세스, 네트워크 활동을 이해하기 위해 분리된 <b>가상 머신 환경에서 실행</b>하며 분석</li> <li><b>메모리 분석</b>을 통해 다른 행위를 추가적으로 분석</li> <li>악의적인 행위의 <b>상세한 과정</b>들을 확인</li> <li>분석가들의 분석 <b>시간이 많이 소요</b></li> </ul>
<b>수동 코드 역공학 분석</b> (Manual Code Reversing)	<ul style="list-style-type: none"> <li>위 과정이 완료된 후에 <b>추가적인 정보를 획득</b>하기 위해 분석하는 행위</li> <li>수동 코드 역공한 분석이 필요한 예           <ul style="list-style-type: none"> <li>- <b>특정 루틴</b>에 난독화가 되어서 복호화가 이루어지는 부분을 더 분석해 추가적인 정보를 획득</li> <li>- 악의적인 도메인 이름 <b>생성 과정의 알고리즘</b> 분석</li> <li>- 행동 분석 과정에서 자신을 숨기고 보여주지 않았던 <b>부분</b>으로 발생되는 <b>다른 기능</b> 이해</li> </ul> </li> </ul>



- 원하는 악성코드 수집하기

- Malware Traffic Analysis

- <http://malware-traffic-analysis.net/2017/>
    - 최신 동향부터 샘플 수집까지 한방에!



### MY BLOG POSTS - [ 2013 ] - [ 2014 ] - [ 2015 ] - [ 2016 ] - [ 2017 ]

- 2017-12-22 - Malspam uses CVE-2017-0199 to distribute Remcos RAT
- 2017-12-21 - Hancitor malspam - Subject: RE: FW: december billing invoice
- 2017-12-21 - Necurs Botnet malspam pushes GlobalImposter ransomware
- 2017-12-20 - Quick post - Hancitor malspam
- 2017-12-19 - Quick post - EITest HoeflerText popups or fake anti-virus pages
- 2017-12-19 - Quick post - Hancitor malspam
- 2017-12-19 - Quick post - Necurs Botnet malspam pushes GlobalImposter ransomware
- 2017-12-18 - Quick post - Hancitor malspam
- 2017-12-18 - A weekend's worth of phishing emails from my inbox
- 2017-12-14 - Ngay campaign Rig EK pushes Quant Loader & Monero CPU miner

# X 파일 2 – 효율적이고 짧은 지름길



- 원하는 악성코드 수집하기
  - Malware Traffic Analysis

MALWARE-TRAFFIC-ANALYSIS.NET



```
5 <script>
6 rHkMhrFCjG="r; }要考虑?BELBEL을fgd?&BS?STXb?BEL?BEL&2?ETXBSa?BSaEOT?1?ETXEOT?whi?;을; STX?EOT?ch]을d?/*f?STXL을EOT을o?EOTi을[(
7 EGEbCxezEP="func을SIkBS을rSIa?,c?*?526?3을189을s*정SOHV을[을at을men?BSENO을tENQBEI?tyEOTENO을[진/?wip?bENO?EOT을[ENOg을men을T요a?이
8 tyVodjAoPk="SOH.STX<ETX>EOT=ENO\"ACK\''BEL) BS(SI DLE\|tDC1\|n";
9 for(nGzAIxJgTe=' ',IFFWhVoCvi=3088,RehfMibmBF=0;IFFWhVoCvi>-1,RehfMibmBF<=3088;IFFWhVoCvi--,RehfMibmBF++)
10 { nGzAIxJgTe+=EGEbCxezEP[RehfMibmBF]; gfgf = 'ned'; /*hg*/if (typeof rHkMhrFCjG[IffWhVoCvi] != 'u'+'ndefi'+gfgf) { nGzAI
11 for (OyuihDPopN=0;OyuihDPopN<=tyVodjAoPk.length-1;OyuihDPopN++)
12 { mGpYFCbXdA = /*gf*/"s\x75bs"+/*gf*/"tr";nGzAIxJgTe=nGzAIxJgTe/*b*/./*b*/replace/*d*/(new RegExp(tyVodjAoPk[mGpYFCbXdA
13 iyRNBFUpTw="l";
14 //KAdPv=this[((14523)??"ev"+"sWQfa"[mGpYFCbXdA](4):"")+iyRNBFUpTw];KAdPv/*sk*/(nGzAIxJgTe);
15 alert(nGzAIxJgTe);
16 </script>
17 <script>
18 xfpLCccpZa="SI50BEL?;Me?Pr로o를u를o?;se원rnBSBEL을gg1?BSBEL을unc을eO?v?o?ETXSI1을ode원mCh원in?SI원ff원SI&SI원rCo원om원tri원et원Str원
19 XbCMaVkjds="va?BSBEL?win원SOH을cri?SI/?67?44h?5ENO원rr?DC1DC1원t원SI1?DC1DLE원SI1원gkg원p원w원Ag원G원m5원Mik원gI원SB원yK원E원Ax원r원9?
20 tTbqtDzBoa="SOH.STX<ETX>EOT=ENO\"ACK\''BEL) BS(SI DLE\|tDC1\|n";
21 for(tiNtiIYTUA=' ',yywwbgAXTF=3190,vcMaHwJcNx=0;yywwbgAXTF>-1,vcMaHwJcNx<=3191;yywwbgAXTF--,vcMaHwJcNx++)
22 { tiNtiIYTUA+=XbCMaVkjds[vcMaHwJcNx]; gfgf = 'ned'; /*hg*/if (typeof xfpLCccpZa[yywwbgAXTF] != 'u'+'ndefi'+gfgf) { tiNti
23 for (VxnQqnaplG=0;VxnQqnaplG<=tTbqtDzBoa.length-1;VxnQqnaplG++)
24 { KMndNlvFfi = /*gf*/"s\x75bs"+/*gf*/"tr";tiNtiIYTUA=tiNtiIYTUA/*b*/./*b*/replace/*d*/(new RegExp(tTbqtDzBoa[KMndNlvFfi
25 IODHOTSmKJ="l";
26 //wBpfH=this[((14523)??"ev"+"sWQfa"[KMndNlvFfi](4):"")+IODHOTSmKJ];wBpfH/*sk*/(tiNtiIYTUA);
27 alert(tiNtiIYTUA);
28 </script>
29 <script>
30 JuEMJmPRkp="SIr; }원tu?BEL;원d원+EOT원x?BEL?BEL?|-BEL?BEL?ETX?ETXBS?EOT?1원ETXEOT?whi?;b원; aBSb?STX?BEL원[c?4e원[/원++?;x?for?]?A[(
31 CKVmajlvbG="fun원n?BEL원rSI?BEL, c?*s294d?h?6?/원meEOT원[원a원le원ENO원rip?b원ENO원ex원a원i?, 원ext원, EOT?get원nt원gNaBS원rip?], a원nt원[(
32 QPbHzFWfls="SOH.STX<ETX>EOT=ENO\"ACK\''BEL) BS(SI DLE\|tDC1\|n";
33 for(JCrsduiRUe=' ',PWmhMAxhzL=575,cYknPmWDWO=0;PWmhMAxhzL>-1,cYknPmWDWO<=575;PWmhMAxhzL--,cYknPmWDWO++)
34 { JCrsduiRUe+=CKVmajlvbG[cYknPmWDWO]; gfgf = 'ned'; /*hg*/if (typeof JuEMJmPRkp[PWmhMAxhzL] != 'u'+'ndefi'+gfgf) { JCrsd
35 for (ZXpCpPGUIu=0;ZXpCpPGUIu<=QPbHzFWfls.length-1;ZXpCpPGUIu++)
36 { NaBIgpaGHA = /*gf*/"s\x75bs"+/*gf*/"tr";JCrsduiRUe=JCrsduiRUe/*b*/./*b*/replace/*d*/(new RegExp(QPbHzFWfls[NaBIgpaGHA
37 JQcFmDcvUp="l";//XhluX=this[((14523)??"ev"+"sWQfa"[NaBIgpaGHA](4):"")+JQcFmDcvUp];XhluX/*sk*/(JCrsduiRUe);
38 //wBpfH=this[((14523)??"ev"+"sWQfa"[KMndNlvFfi](4):"")+IODHOTSmKJ];wBpfH/*sk*/(tiNtiIYTUA);
39 alert(tiNtiIYTUA);
40 </script></body></html>
```

# X 파일 2 – 효율적이고 짧은 지름길



- 원하는 악성코드 수집하기

## - 2017-12-22 - MALSPAM USES CVE-2017-0199 TO DISTRIBUTE REMCOS RAT

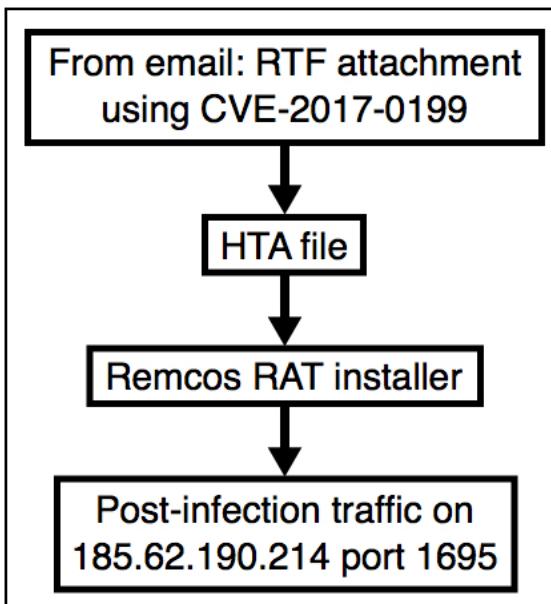
### 2017-12-22 - MALSPAM USES CVE-2017-0199 TO DISTRIBUTE REMCOS RAT

#### ASSOCIATED FILES:

- 2017-12-21-malspam-pushing-Remcos-RAT-1356-UTC.eml.zip 39.9 kB (39,888 bytes)
- 2017-12-22-malspam-pushing-RemcosRAT.pcap.zip 1.0 MB (1,044,955 bytes)
- 2017-12-22-artifacts-from-Remcos-RAT-malspam-infection.zip 1.9 MB (1,875,694 bytes)

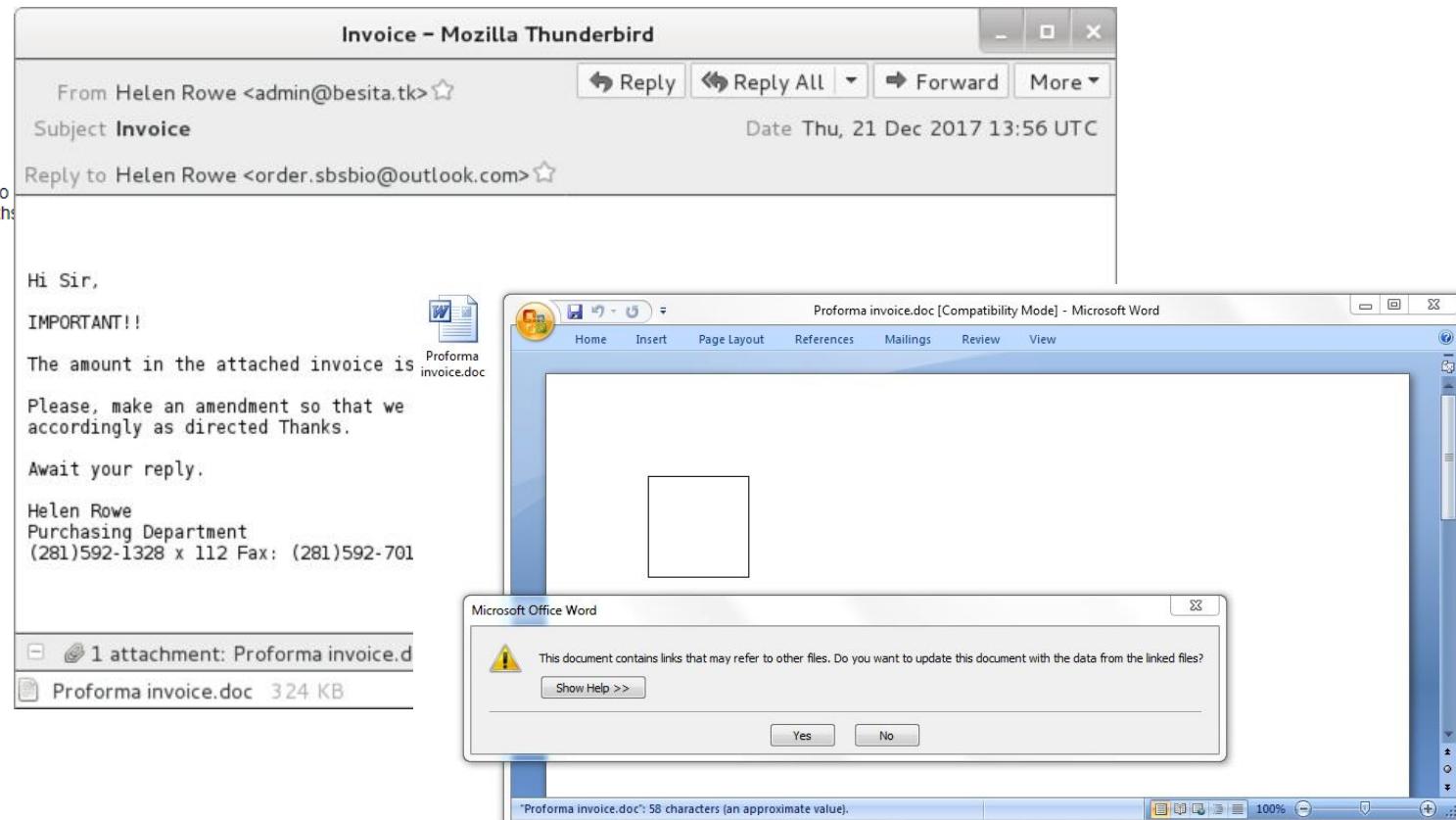
#### NOTES:

- On 2017-12-21, I saw malspam dated 2017-12-21 with an RTF attachment using **CVE-2017-0199** to
- Today's post-infection traffic is similar to Remcos RAT post-infection traffic I reported almost 2 months



Shown above: Flowchart for today's infection.

MALWARE-TRAFFIC-ANALYSIS.NET





- 원하는 악성코드 수집하기
  - Exploit-DB에서 CVE 2017-0199 수집 가능



## Microsoft Office Word - '.RTF' Malicious HTA Execution (Metasploit)

EDB-ID: 41934	Author: Metasploit	Published: 2017-04-25
CVE: CVE-2017-0199	Type: Remote	Platform: Windows
Aliases: N/A	Advisory/Source: Link	Tags: Metasploit Framework (MSF)
E-DB Verified:	Exploit:  Download /  View Raw	Vulnerable App: N/A

```
1 ##  
2 # This module requires Metasploit: http://metasploit.com/download  
3 # Current source: https://github.com/rapid7/metasploit-framework  
4 ##  
5  
6 require 'msf/core'  
7  
8  
9 class MetasploitModule < Msf::Exploit::Remote  
10 Rank = ExcellentRanking  
11  
12 include Msf::Exploit::FILEFORMAT  
13 include Msf::Exploit::Remote::HttpServer::HTML  
14  
15 def initialize(info = {})  
16   super(update_info(info,  
17     'Name'        => "Microsoft Office Word Malicious Hta Execution",  
18     'Description' => %q{  
19       This module creates a malicious RTF file that when opened in  
20       vulnerable versions of Microsoft Word will lead to code execution.  
21       The flaw exists in how a olelink object can make a http(s) request,  
22       and execute hta code in response.  
23     },  
24     'Author'      =>  
25     [  
26       'Haifei Li', # vulnerability analysis  
27       'ryHanson',  
28       'wdormann'  
29     ]  
30   )  
31 end  
32 
```



- 원하는 악성코드 수집하기
  - 해시 값을 알고 있을 때 검색할 수 있는 사이트

This webpage is a free malware analysis service for the community that detects and analyzes unknown threats using a unique Hybrid Analysis technology.

Select file

This free malware analysis service is running Falcon Sandbox v7.20 in the backend. Supporting PE, Office, PDF, APK files and more (e.g. EML). Maximum upload size is 100 MB.  
[Learn more about the standalone version or purchase a private webservice.](#)

By submitting the file, you automatically accept our [Terms of Service](#).

Select file

Share the sample  
 Private

5 + 4 =

# X 파일 2 – 효율적이고 짧은 지름길



- 원하는 악성코드 수집하기

- Malwr (<https://malwr.com>)

**malwr** 

**662138** Total Analyses    **66%** Shared Malware    **267056** Unique Domains

Recent Analyses (see more)	
Nov. 9, 2016,	<a href="#">14ddb457af1d02210d3114d95d010dca</a> 7:47 p.m.
Nov. 9, 2016,	<a href="#">6864f832e36d3e59f0a8abd87d5fa280</a> 7:46 p.m.
Nov. 9, 2016,	<a href="#">6e5feeb23342d91d95adfb672e7e1e3d</a> 7:44 p.m.
Nov. 9, 2016,	<a href="#">4efb38b110ab464f5186ccca9556b5fb</a> 7:39 p.m.
Nov. 9, 2016,	<a href="#">d7db25132b4f5d8f6511c27c5bd2819c</a> 7:32 p.m.
Nov. 9, 2016,	<a href="#">1f1ab5b4af857e2d2e7fb0ecba226d8f</a> 7:26 p.m.
Nov. 9, 2016,	<a href="#">8698536757b9574f60b7292cca9f0122</a> 7:22 p.m.
Nov. 9, 2016,	<a href="#">b7f672751d5346f799ba2a43de41b90d</a> 7:18 p.m.
Nov. 9, 2016,	<a href="#">d3786b1051fb677445d8577334b7e00d</a> 7:17 p.m.
Nov. 9, 2016,	<a href="#">c3403582be266b99ab369b7919d901ea</a> 7:15 p.m.

Analysis			
CATEGORY	STARTED	COMPLETED	DURATION
FILE	2016-11-09 17:45:20	2016-11-09 17:47:39	139 seconds

File Details	
FILE NAME	setup.exe
FILE SIZE	6810648 bytes
FILE TYPE	PE32 executable (GUI) Intel 80386, for MS Windows
MD5	14ddb457af1d02210d3114d95d010dca
SHA1	5ac657c43b5fb67caa2718c10552422e36c3b99e
SHA256	b16c900ef21c0244bdd93e836b29a5e7097ff0b1934dd6d957c9cae15e317fcbb
SHA512	281182d2af031534ee3b0631600f5e0be0b0e3e34f787707a792ad336998796ef2ab05ce4ce3f394a55
CRC32	FF54FF4F
SSDEEP	98304:qxHYvSzqvZ/nhpLTk4v5TBWZFrwF7DeESuXweJPK1eerEbz3JderT7VB4CKsRSE:qxHYvR/TL
YARA	None matched

[Download](#) You need to login

## X 파일 2 – 효율적이고 짧은 지름길



### • 오픈 소스를 활용하라!

- 깃헙에서 수 많은 유용한 오픈 소스를 접할 수 있어요!
- 악성코드 분석 도구가 어떻게 만들어지는 소스로 파악
- 악성코드 분석 도구 직접 사용
- 더 나아가 악성코드 분석 도구 제작

cuckoosandbox / cuckoo

This repository

Search

Pull requests Issues Marketplace Explore

Watch 413 Star 2,992 Fork 1,088

Code Issues 359 Pull requests 60 Projects 0 Wiki Insights

Cuckoo Sandbox is an automated dynamic malware analysis system <http://www.cuckoosandbox.org>

7,527 commits 3 branches 48 releases 134 contributors

Branch: master New pull request Create new file Upload files Find file Clone or download

automatisch committed with jbremer Added filetree XHR error feedback ... Latest commit 0d8cd9e 14 days ago

cuckoo Added filetree XHR error feedback 13 days ago

docs version 2.0.5 19 days ago

stuff make newer signatures and extractors work, update readme links 22 days ago

tests fixup latest monitor symlink issue #2008 (thanks hendl doomedraven) 13 days ago

.codeclimate.yml Add basic .codeclimate.yml 3 years ago

.gitignore move cuckoo/data-private/ to cuckoo/private/ 4 months ago

.travis.yml make newer signatures and extractors work, update readme links 22 days ago

MANIFEST.in do not embed .map files and get rid of trailing slash warning 9 months ago

README.rst make newer signatures and extractors work, update readme links 22 days ago

analyzer symlink to analyzer/ directory a year ago

appveyor.yml make newer signatures and extractors work, update readme links 22 days ago

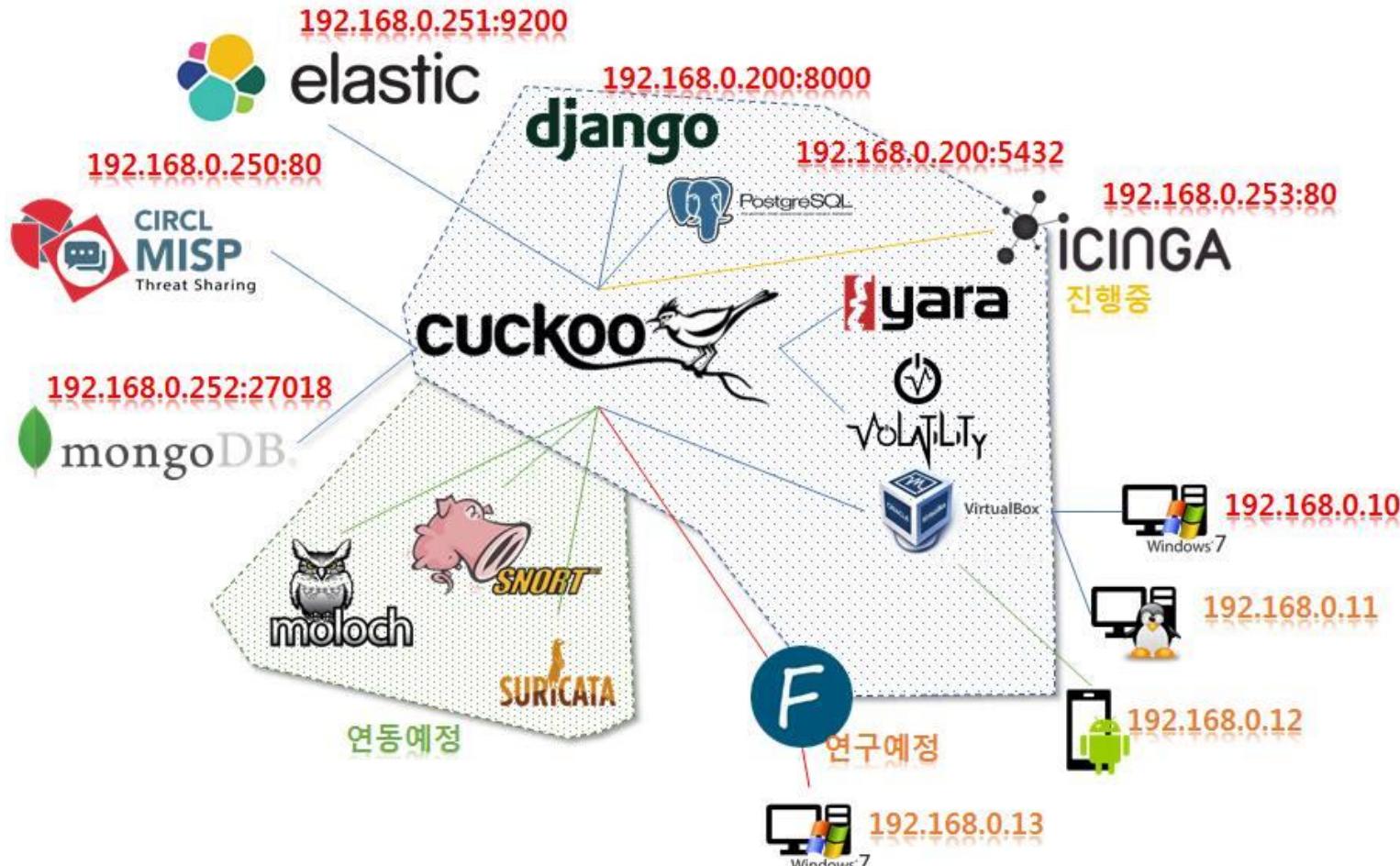
conftest.py adjustments and unit tests for upcoming monitor release 4 months ago

setup.cfg modify unit test for permission denied at system process 22 days ago

setup.py fixup latest monitor symlink issue #2008 (thanks hendl doomedraven) 13 days ago



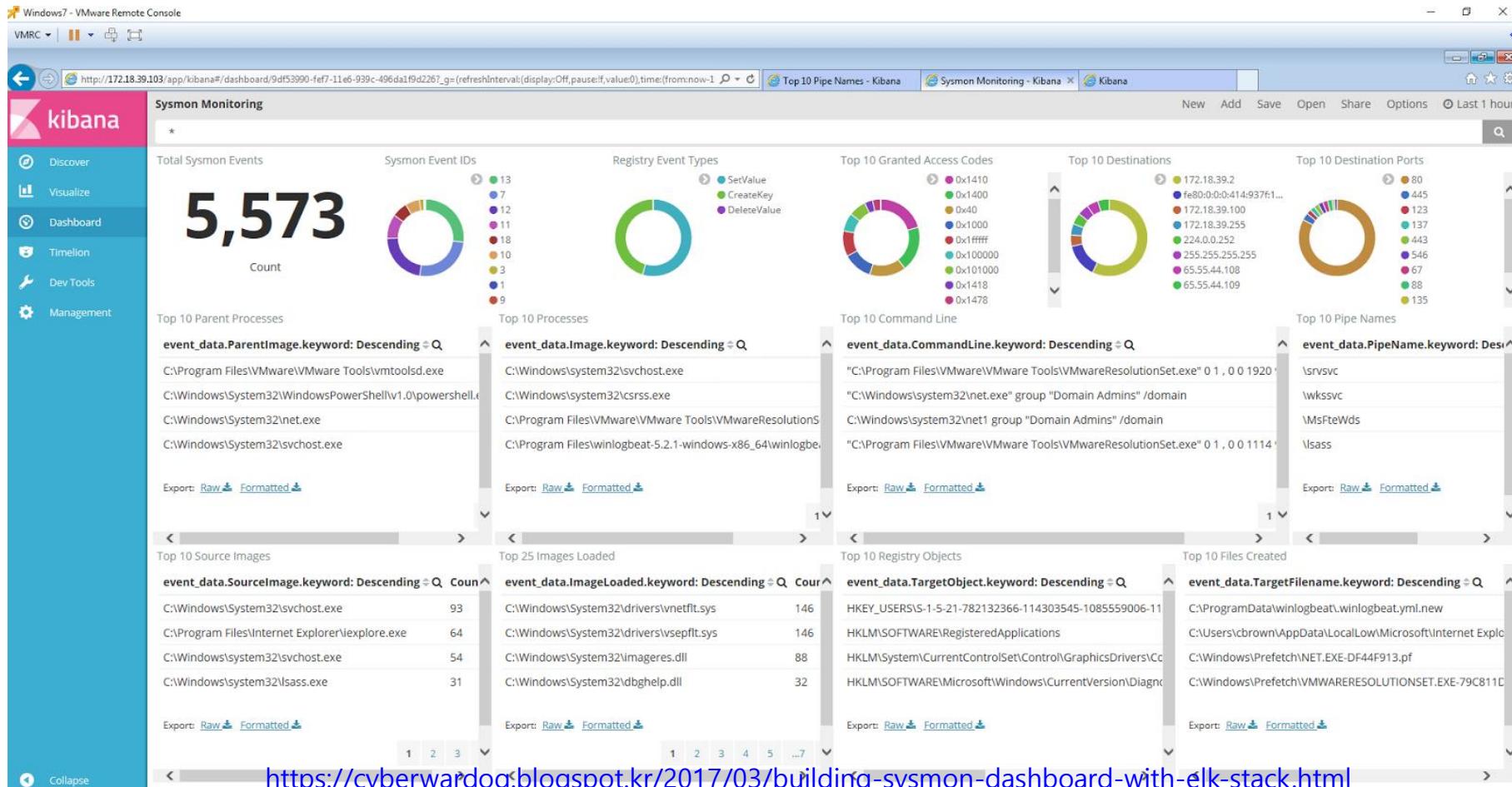
- 오픈 소스를 활용하라!
  - 악성코드 자동 분석 구조



# X 파일 2 – 효율적이고 짧은 지름길



- 오픈 소스를 활용하라!
  - ELK와 Sysmon을 활용한 지능화된 탐지 구조





- **참여하라!**

- 스터디를 구성하라.
- 정기적인 모임과 성과 발표는 매주 공부할 수 있는 계기!
- 사회성을 기르기 인맥을 넓힘
- 원래 어려운 분야이니 어려운 것이 당연하다!
- 커뮤니티에 참여하여 취업의 기회를 높여라!





# X 파일 3 – 삽질! 그리고 또 삽질!

# X 파일 3 – 삽질! 그리고 또 삽질!



- 검색하라!

- 인터넷에 모든 게 다 있다! 하나씩 학습!

The screenshot shows the IDA Pro interface with several windows open:

- IDA View-A**: Shows assembly code for the `main` function. Annotations include:
  - 호출 규약** (Calling Convention) pointing to the `Attributes` section.
  - 데이터 타입** (Data Type) pointing to the `array` definition.
  - 메모리 세션** (Memory Session) pointing to the `size` variable.
  - 피연산자** (Operands) pointing to the `array`, `size`, and `ptr` variables.
  - 명령어** (Instruction) pointing to the `push all` instruction.
- 메모리 주소**: A dump window showing memory contents at address `8048434` to `804847b`. Annotations include:
  - 기계어 코드** (Machine Code) pointing to the assembly code.
  - 주소 표기법** (Address Format) pointing to the memory addresses.
  - 시스템 기능 사용** (System Function Usage) pointing to the `call 8048350 <system@plt>` instruction.
- Registers (FPU)**: Registers window showing CPU register values.
- Stack of \_main**: Stack dump window showing the stack frame structure.
- Registers (CPU)**: Registers window showing CPU register values.

# X 파일 3 – 삽질! 그리고 또 삽질!



## • 메모하라!

- 악성코드 분석은 메모다.
- 가능하면 보고서를 작성해라.
  - 악성코드 개당 50페이지 이상



파워쉘(PowerShell)  
을 이용한 문서형 ...

이정현  
★★★★☆ 5명  
구매 4,300원



컨피커 웜 Conficker  
Worm 분석 - 악성...

이정현  
★★★★☆ 5명  
구매 4,100원



PDF 악성코드 분석 -  
악성코드 분석 시...

이정현  
★★★★★ 5명  
구매 3,400원



골든아이(Golden Ey  
e) 랜섬웨어 분석 -...

이정현  
★★★★★ 5명  
구매 6,000원



소니픽처스 해킹의  
악성코드 분석-악...

이정현  
★★★★★ 17명  
구매 4,200원



레비톤(Reveton) 랜  
섬웨어 분석 - 악성...

이정현  
★★★★★ 2명  
구매 8,300원

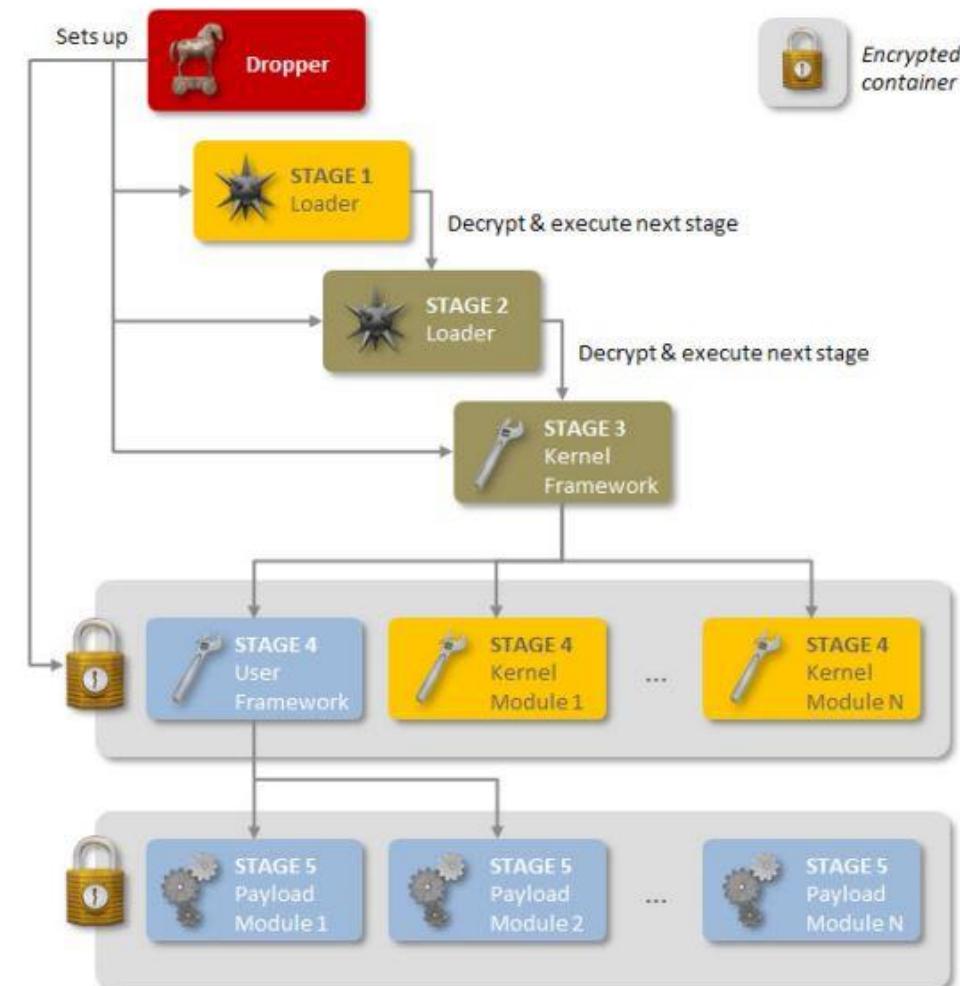


Figure 3. Regin's architecture

<http://www.valuewalk.com/2014/11/symantec-warns-spy-malware-regin/>



## • 메모하라!

- 한 수강생의 2개월 후의 첫 악성코드 보고서 작성 목차

### ▲ 1 개요

- 1.1 프로젝트 주제
- 1.2 프로젝트 추진 배경 및 목표
- 1.3 프로젝트 요약

### 2 ShadowBot 개요

### ▲ 3 기초 정적 분석

- 3.1 VirusTotal 확인
- 3.2 패킹 여부 확인
- 3.3 IAT(Import Address Table) 분석
- ▲ 3.4 호스트 / 네트워크 기반 증거 분석
  - 3.4.1 주요 함수 분석
  - 3.5 주요 문자열 분석

### ▲ 4 기초 동적 분석

- ▲ 4.1 기초 동적 분석 - shadowbot.exe
  - 4.1.1 shadowbot.exe 모니터링 준비
  - 4.1.2 shadowbot.exe 실행
  - 4.1.3 shadowbot.exe 재실행
  - 4.1.4 Regshot 결과
  - 4.1.5 Process Monitor 결과
  - 4.1.6 Process Explorer 확인
  - 4.1.7 Wireshark 결과 확인
  - 4.1.8 DNS 설정 후 shadowbot.exe 실행 결과 확인
- 4.2 기초 정적 분석 - rdshost.dll
- 4.3 기초 정적 분석 - photo album.zip

### ▲ 5 고급 정적 분석

- 5.1 고급 정적 분석 - shadowbot.exe
- 5.2 고급 정적 분석 – rdshost.dll
- 5.3 고급 정적 분석 결과 정리

### ▲ 6 악성코드 기능 확인

- ▲ 6.1 shadowbot 기능 확인을 위한 환경 구성
  - 6.1.1 IRC 서버 설정
  - 6.1.2 Victim의 hosts 파일 설정
  - 6.1.3 Victim의 MSN 메신저 구성
  - 6.1.4 Attacker의 mIRC 설정
- 6.2 shadowbot 테스트

### ▲ 7 고급 동적 분석

- 7.1 explorer.exe의 자식 프로세스를 죽이는 동작 분석
- 7.2 OllyDbg 툴로 rdshost.dll 동적 분석하는 방법

### ▲ 8 shadowbot 악성코드 분석 결과 정리

- 8.1 shadowbot.exe 분석 결과 정리
- 8.2 rdshost.dll 분석 결과 정리
- 9 shadowbot 감염시 치료 방법



- 무조건 부딪쳐라!

- 포기하지 마세요
- 남들도 나와 같은 환경!
- 누구나 첫걸음은 있다!



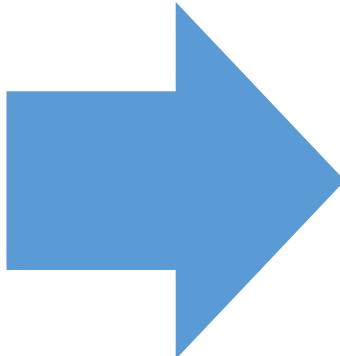


# 1. 리버싱 기초



- 01 어셈블리
  - C/C++ 코드와 어셈블리 코드의 차이
  - 한 가지 동작까지 세세하게 지정

```
1 void 물마심()
2 {
3     BOOL bOpen = 냉장고문오픈();
4     if (bOpen)
5     {
6         물을꺼냄();
7         마심();
8     }
9 }
```

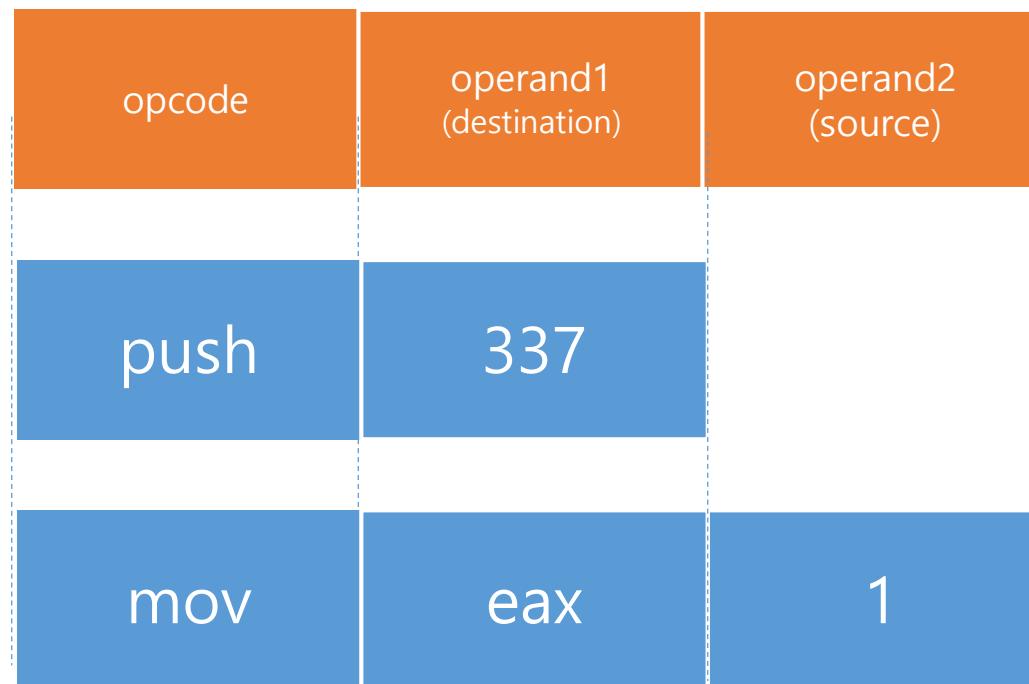


```
1         __asm{
2             냉장고앞으로간다
3             냉장고문을잡는다
4             냉장고문을연다
5             오픈성공:
6             냉장고안을본다
7             손을든다
8             냉장고안에넣는다
9             물병을잡는다
10            물병을꺼낸다
11            뚜껑을연다
12            물을컵에따른다
13            컵을손에든다
14            컵에든것을마신다
15 }
```



- 02 어셈블리의 명령 포맷

- 주로 IA-32를 사용
- 기본형태 : 명령어(옵코드, opcode) + 인자(오퍼랜드, operand 1~2)





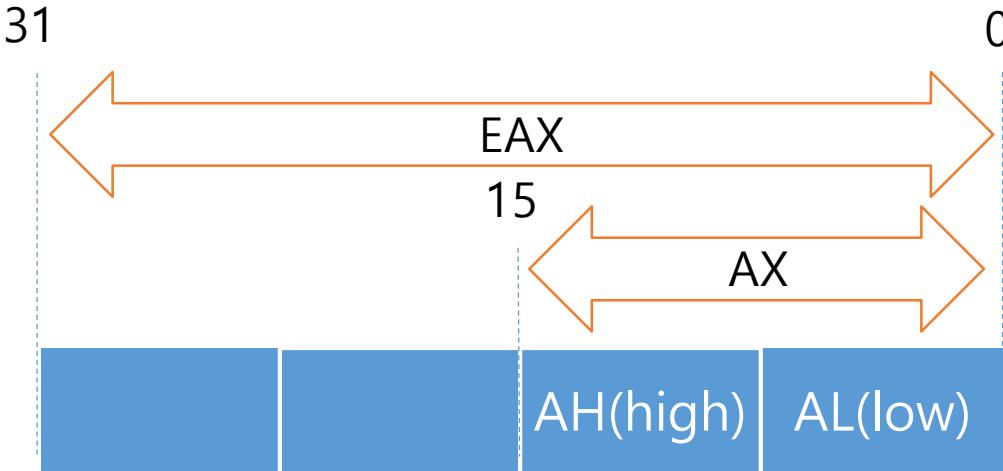
- 03 레지스터 - 레지스터의 종류
  - CPU가 사용하는 변수

레지스터	역할
EAX (Acuumulator)	각종 연산에 쓰임 가장 많이 쓰이는 변수 주로 리턴 값을 저장
EDX (data)	각종 연산에 쓰이는 변수
ECX (Count)	for 문에서 i의 역할 ECX는 미리 값을 정해놓고 0이 될 때까지 진행 변수로 사용해도 무방
EBX	목적이 없는 레지스터 공간이 필요할 때 덤으로 사용

레지스터	역할
ESI, EDI (source Index, de stination Index)	문자열이나 각종 반복데이터를 처리 또는 메모리를 옮기는 데 사용
ESP	스택 포인터
EBP	베이스 포인터
EIP	인스트럭션 포인터



- 03 레지스터 - 레지스트리 단위



32bit	16bit	상위 8bit	하위 8bit
EAX	AX	AH	AL
EDX	DX	DH	DL
ECX	CX	CH	CL
EBX	BX	BH	BL



## • 03 레지스터 - 사용 예

- Plus (c 코드) vs. PlusAsm (Asm 코드)
- 두 함수가 같은 값을 반환

```
1 #include <windows.h>
2 #include <stdio.h>
3 int Plus(int a, int b)
4 {
5     return a + b;
6 }
7
8 __declspec (naked) PlusAsm(int a, int b)
9 {
10     __asm
11     {
12         mov ebx, dword ptr 55:[esp+8] // b를 ebx에 저장
13         mov edx, dword ptr 55:[esp+4] // a를 edx에 저장
14         add edx, ebx // a + b
15         mov eax, edx // 결과값을 eax에 저장
16         retn
17     }
18 }
19
20 void main(int argc, char *argv[])
21 {
22     int value = Plus(3,4);
23     printf("value: %d\n", value);
24     int value2 = PlusAsm(3,4);
25     printf("value2: %d\n", value2);
26
27     return;
28 }
```



- 04 외울 필요가 없는 어셈블리 명령어[1/2]

- PUSH, POP (PUSHAD, POPAD)
- MOV : source를 가져옴
- LEA : source의 주소를 가져옴
- ADD
- SUB
- INT : 인터럽트 명령]



- 04 외울 필요가 없는 어셈블리 명령어[1/2]

- CALL
- INC, DEC (i++, i--)
- AND, OR, XOR
- NOP
- CMP, JMP



- 05 리버스 엔지니어링에 필요한 스택
  - 함수 프롤로그

```
push ebp  
mov  ebp, esp  
sub  esp, 50h
```

ebp 삽입  
esp 값을 ebp에 저장  
esp 50h 위치를 올림(지역 변수 저장 위치)

- 스택 구조

100h	ebp

esp

100h	ebp
...	
50h	

ebp

esp

100h	ebp
...	변수
50h	변수

ebp

esp

진행



## • 6. 함수의 호출 & 리턴 주소

- 메인 함수에서 HelloFunction 함수 호출
- LIFO 방식으로 스택을 삽입

```
1 main()
2 {
3     DWORD dwRet = HelloFunction(0x37, 0x38, 0x39);
4     if (dwRet)
5         // .....
6 }
```



```
push 39h
push 38h
push 37h
call 401300h; HelloFunction
```

함수 호출 시 스택 구성

ebp+0x16	116h	39h
ebp+0x12	112h	38h
ebp+0x8	108h	37h
ebp+0x4	104h	RET 주소
ebp	100h	ebp

ebp

esp



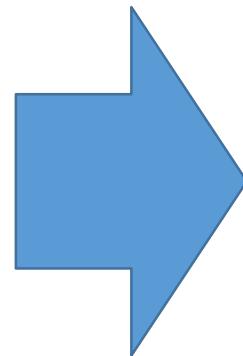
- 참고문헌
  - 리버싱 엔지니어링 바이블, 강병탁 지음



- 01 함수의 기본 구조

- 간단한 sum 함수의 구조

```
1 int sum(int a, int b)
2 {
3     int c = a + b;
4     return c;
5 }
```



```
push ebp
mov ebp, esp
push ecx
mov eax, [ebp+arg_0]
add eax, [ebp+arg_4]
mov [ebp+var_4], eax
mov eax, [ebp+var_4]
mov esp,ebp
pop ebp
ret
```

함수  
프롤로그

함수  
에필로그

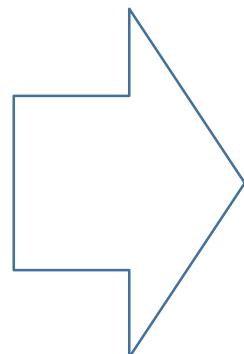


- 02 함수의 호출 규약 - 1) cdecl

- 항상 call 문의 다음 줄을 살펴서 스택을 정리하는 곳을 체크한다.
- cdecl 코드는 call 이후 add esp, 8과 같이 스택을 보정
- add esp, 8 그리고 push 문이 2개
  - 4바이트 파라미터가 두 개임을 추측할 수 있음

```

1 int __cdecl sum(int a, int b)
2 {
3     int c = a + b;
4     return c;
5 }
6
7 int main(int argc, char* argv[])
8 {
9     sum(1,2);
10    return 0;
11 }
```



```

1 sum:
2     push ebp
3     mov ebp, esp
4     push ecx
5     mov eax, [ebp+arg_0]
6     add eax, [ebp+arg_4]
7     mov [ebp+var_4], eax
8     mov eax, [ebp+var_4]
9     mov esp, ebp
10    pop ebp
11    retn
12
13 main:
14    push 2
15    push 1
16    call calling.00401000
17    add esp, 8

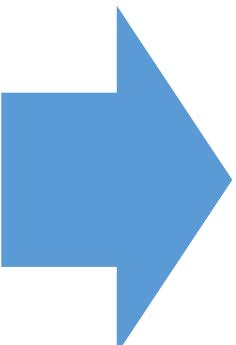
```



- 02 함수의 호출 규약 - 2) stdcall [1/2]

- main 함수 내부에서 스택을 처리하지 않음
- stdcall 코드는 retn 8을 사용하여 스택을 처리
- Win32 API는 stdcall 방식을 사용

```
1 int __stdcall sum(int a, int b)
2 {
3     int c = a + b;
4     return c;
5 }
```



```
1 sum:
2     push ebp
3     mov ebp, esp
4     push ecx
5     mov eax, [ebp+arg_0]
6     add eax, [ebp+arg_4]
7     mov [ebp+var_4], eax
8     mov eax, [ebp+var_4]
9     mov esp, ebp
10    pop ebp
11    retn 8
12
13 main:
14     push 2
15     push 1
16     call calling.00401000
```



- 02 함수의 호출 규약 – 2) stdcall [2/2]

- 2개의 파라미터로 WinExec 함수 호출

```
00E91000 55 PUSH EBP
00E91001 8BEC MOV EBP,ESP
00E91003 6A 05 PUSH 5
00E91005 68 0021E900 PUSH WinExec.00E92100
00E9100A FF15 0020E900 CALL DWORD PTR DS:[<&KERNEL32.WinExec>] WinExec
00E91010 33C0 XOR EAX,EAX
00E91012 5D POP EBP
00E91013 C3 RETN
```

ShowState = SW\_SHOW  
CmdLine = "notepad.exe"

- WinExec 함수 호출 시 프롤로그에서 RETN 8을 발견할 수 있음

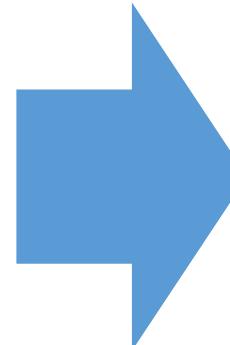
```
770454D8 8BC6 MOV EAX,ESI
770454D9 8B8C24 94000000 MOV ECX,DWORD PTR SS:[ESP+94]
770454E1 5F POP EDI
770454E2 5E POP ESI
770454E3 5B POP EBX
770454E4 33CC XOR ECX,ESP
770454E6 E8 691EFEFF CALL KERNEL32.77027354
770454EB 8BE5 MOV ESP,EBP
770454ED 5D POP EBP
770454EE C2 0800 RETN 8
770454F1 CC INT3
770454F2 CC INT3
770454F3 CC INT3
770454F4 CC INT3
770454F5 CC INT3
```



- 02 함수의 호출 규약 - 3) fastcall [2/2]

- 파라미터가 2개 이하일 경우, 인자를 push로 넣지 않고 edx와 ecx 레지스터를 이용
- 레지스터를 이용하기에 메모리를 이용하는 것보다 훨씬 빠름
- 함수 호출 전, edx와 ecx 레지스터에 값을 넣는 것을 보면 fastcall 규약 함수임을 짐작할 수 있음

```
1 int __fastcall sum(int a, int b)
2 {
3     int c = a + b;
4     return c;
5 }
```



```
1 sum:
2     push ebp
3     mov ebp, esp
4     sub esp, 0Ch
5     mov [ebp+var_C], edx
6     mov [ebp+var_8], ecx
7     mov eax, [ebp+var_8]
8     add eax, [ebp+var_C]
9     mov [ebp+var_4], eax
10    mov eax, [ebp+var_4]
11    mov esp,ebp
12    pop ebp
13    retn
14
15 main:
16     push ebp
17     mov ebp, esp
18     mov edx, 2
19     mov ecx, 1
20     call calling.00401000
21     xor eax, eax
22     pop dbp
23     retn
```



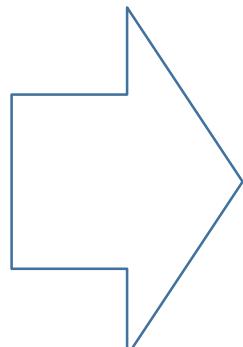
- 2. 함수의 호출 규약 - 4) thiscall [2/2]

- 주로 C++의 클래스에서 this 포인터로 이용
- 현재 객체의 포인터에 ecx에 전달하는 특징을 가짐
- 다음과 같이 ecx 포인터에 오프셋 번지를 더함

파라미터	주소
a	ecx+x
b	ecx+y
c	ecx+z

```

1 Class CTemp
2 {
3     public:
4         int MemberFunc(int a, int b);
5 }
```



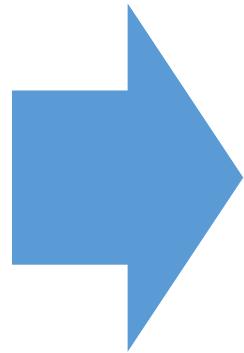
```

1 mov eax, dword ptr [ebp-14h]
2 push eax
3 mov edx, dword ptr [ebp-10h]
4 push edx
5 lea ecx, [ebp-4]
6 call 402000
```



- 03 if 문 [1/5]

```
1 int Temp(int a)
2 {
3     int b = 1;
4
5     if(a==1){
6         a++;
7     }else{
8         b++;
9     }
10
11    return b;
12 }
13
14 int main(int argc, char* argv[])
15 {
16     Temp(1);
17 }
```



```
1 .text:00401000    push    ebp
2 .text:00401001    mov     ebp, esp
3 .text:00401003    push    ecx
4 .text:00401004    mov     dword ptr [ebp-4], 1
5 .text:0040100B    cmp     dword ptr [ebp+8], 1
6 .text:0040100F    jnz    short loc_40101C
7 .text:00401011    mov     eax, [ebp+8]
8 .text:00401014    add     eax, 1
9 .text:00401017    mov     [ebp+8], eax
10 .text:0040101A   jmp    short loc_401025
11 .text:0040101C loc_40101C:
12 .text:0040101C    mov     eax, [ebp-4]
13 .text:0040101F    add     ecx, 1
14 .text:00401022    mov     [ebp-4], ecx
15 .text:00401025
16 .text:00401025 loc_401025:
17 .text:00401025    mov     eax, [ebp-4]
18 .text:00401028    mov     esp, ebp
19 .text:0040102A    pop    ebp
20 .text:0040102B    retn
```



- 03 if 문 [2/5]

```

1 .text:00401000    push    ebp
2 .text:00401001    mov     ebp, esp
3 .text:00401003    push    ecx
4 .text:00401004    mov     dword ptr [ebp-4], 1
5 .text:0040100B    cmp     dword ptr [ebp+8], 1
6 .text:0040100F    jnz    short loc_40101C
7 .text:00401011    mov     eax, [ebp+8]
8 .text:00401014    add     eax, 1
9 .text:00401017    mov     [ebp+8], eax
10 .text:0040101A   jmp    short loc_401025
11 .text:0040101C loc_40101C:
12 .text:0040101C    mov     eax, [ebp-4]
13 .text:0040101F    add     ecx, 1
14 .text:00401022    mov     [ebp-4], ecx
15 .text:00401025 loc_401025:
16 .text:00401025    mov     eax, [ebp-4]
17 .text:00401025    mov     esp, ebp
18 .text:00401028    pop    ebp
19 .text:0040102A    retn
20 .text:0040102B

```

ecx를 사용하기 위해  
기존 ecx 값을 보관

참고

보통 함수의 초반부에 레지스터를 push 문으로 스택에 넣는 코드가 등장한다면 앞으로 이 레지스터를 함수에서 계속 연산 목적으로 사용할 것이기 때 문이라고 생각하면 됨.



- 03 if 문 [3/5]

```
1 .text:00401000      push    ebp  
2 .text:00401001      mov     ebp, esp  
3 .text:00401003      push    ecx  
4 .text:00401004      mov     dword ptr [ebp-4], 1  
5 .text:0040100B      cmp     dword ptr [ebp+8], 1  
6 .text:0040100F      jnz    short loc_40101C  
7 .text:00401011      mov     eax, [ebp+8]  
8 .text:00401014      add     eax, 1  
9 .text:00401017      mov     [ebp+8], eax  
10 .text:0040101A     jmp    short loc_401025  
11 .text:0040101C loc_40101C:  
12 .text:0040101C     mov     eax, [ebp-4]  
13 .text:0040101F     add     ecx, 1  
14 .text:00401022     mov     [ebp-4], ecx  
15 .text:00401025  
16 .text:00401025 loc_401025:  
17 .text:00401025     mov     eax, [ebp-4]  
18 .text:00401028     mov     esp, ebp  
19 .text:0040102A     pop    ebp  
20 .text:0040102B     retn
```

스택 변수 b에 1 값 삽입  
int b=1;



- 03 if 문 [4/5]

```

1 .text:00401000    push    ebp
2 .text:00401001    mov     ebp, esp
3 .text:00401003    push    ecx
4 .text:00401004    mov     dword ptr [ebp-4], 1
5 .text:0040100B    cmp     dword ptr [ebp+8], 1
6 .text:0040100F    jnz    short loc_40101C
7 .text:00401011    mov     eax, [ebp+8]
8 .text:00401014    add     eax, 1
9 .text:00401017    mov     [ebp+8], eax
10 .text:0040101A   jmp    short loc_401025
11 .text:0040101C loc_40101C:
12 .text:0040101C    mov     eax, [ebp-4]
13 .text:0040101F    add     ecx, 1
14 .text:00401022    mov     [ebp-4], ecx
15 .text:00401025
16 .text:00401025 loc_401025:
17 .text:00401025    mov     eax, [ebp-4]
18 .text:00401028    mov     esp, ebp
19 .text:0040102A    pop     ebp
20 .text:0040102B    retn

```

직접적인 if 문  
if (a==1)

{ a++; }

함수 에필로그로 이동



- 03 if 문 [5/5]

```

1 .text:00401000      push    ebp
2 .text:00401001      mov     ebp, esp
3 .text:00401003      push    ecx
4 .text:00401004      mov     dword ptr [ebp-4], 1
5 .text:0040100B      cmp     dword ptr [ebp+8], 1
6 .text:0040100F      jnz    short loc_40101C
7 .text:00401011      mov     eax, [ebp+8]
8 .text:00401014      add     eax, 1
9 .text:00401017      mov     [ebp+8], eax
10 .text:0040101A     jmp    short loc_401025
11 .text:0040101C loc_40101C:
12 .text:0040101C      mov     eax, [ebp-4]
13 .text:0040101F      add     ecx, 1
14 .text:00401022      mov     [ebp-4], ecx
15 .text:00401025
16 .text:00401025 loc_401025:
17 .text:00401025      mov     eax, [ebp-4]
18 .text:00401028      mov     esp, ebp
19 .text:0040102A      pop    ebp
20 .text:0040102B      retn

```

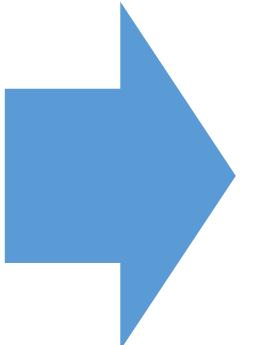
직접적인 if 문  
if (a==1)

if 문이 거짓일 경우  
b++;  
이후 함수 에필로그 진행



- 04 반복문 [1/4]
- 반복문은 for, while, goto 등이 있으나, 컴퓨터 입장에서는 결국 카운터 레지스터를 이용한 반복임

```
1 int loop(int c)
2 {
3     int d;
4
5     for (int i=0; i<=0x100; 1++)
6     {
7         c--;
8         d++;
9     }
10
11    return c+d;
12 }
```



```
1 .text:00401000    push    ebp
2 .text:00401001    mov     ebp, esp
3 .text:00401003    sub     esp, 8
4 .text:00401006    mov     dword ptr [ebp-8], 0
5 .text:0040100D    jmp     short loc_401018
6 .text:0040100F    mov     eax, [ebp-8]
7 .text:00401012    add     eax, 1
8 .text:00401015    mov     [ebp-8], eax
9 .text:00401018    cmp     dword ptr [ebp-8], 100h
10 .text:0040101F   jg      short loc_401035
11 .text:00401021    mov     ecx, [ebp+8]
12 .text:00401024    sub     ecx, 1
13 .text:00401027    mov     [ebp+8], ecx
14 .text:0040102A    mov     edx, [ebp-4]
15 .text:0040102D    add     edx, 1
16 .text:00401030    mov     [ebp-4], edx
17 .text:00401033    jmp     short loc_40100F
18 .text:00401035    mov     eax, [ebp+8]
19 .text:00401038    add     eax, [ebp-4]
20 .text:0040103B    mov     esp, ebp
21 .text:0040103D    pop     ebp
22 .text:0040103E    retn
```



- 04 반복문 [2/4]

```
1 .text:00401000      push    ebp
2 .text:00401001      mov     ebp, esp
3 .text:00401003      sub     esp, 8
4 .text:00401006      mov     dword ptr [ebp-8], 0
5 .text:0040100D      jmp     short loc_401018
6 .text:0040100F      mov     eax, [ebp-8]
7 .text:00401012      add     eax, 1
8 .text:00401015      mov     [ebp-8], eax
9 .text:00401018      cmp     dword ptr [ebp-8], 100h
10 .text:0040101F     jg      short loc_401035
11 .text:00401021      mov     ecx, [ebp+8]
12 .text:00401024      sub     ecx, 1
13 .text:00401027      mov     [ebp+8], ecx
14 .text:0040102A      mov     edx, [ebp-4]
15 .text:0040102D      add     edx, 1
16 .text:00401030      mov     [ebp-4], edx
17 .text:00401033      jmp     short loc_40100F
18 .text:00401035      mov     eax, [ebp+8]
19 .text:00401038      add     eax, [ebp-4]
20 .text:0040103B      mov     esp, ebp
21 .text:0040103D      pop     ebp
22 .text:0040103E      retn
```

```
i++;
```



- 04 반복문 [3/4]

```
1 .text:00401000      push    ebp
2 .text:00401001      mov     ebp, esp
3 .text:00401003      sub     esp, 8
4 .text:00401006      mov     dword ptr [ebp-8], 0
5 .text:0040100D      jmp     short loc_401018
6 .text:0040100F      mov     eax, [ebp-8]
7 .text:00401012      add     eax, 1
8 .text:00401015      mov     [ebp-8], eax
9 .text:00401018      cmp     dword ptr [ebp-8], 100h
10 .text:0040101F     jg     short loc_401035
11 .text:00401021      mov     ecx, [ebp+8]
12 .text:00401024      sub     ecx, 1
13 .text:00401027      mov     [ebp+8], ecx
14 .text:0040102A      mov     edx, [ebp-4]
15 .text:0040102D      add     edx, 1
16 .text:00401030      mov     [ebp-4], edx
17 .text:00401033      jmp     short loc_40100F
18 .text:00401035      mov     eax, [ebp+8]
19 .text:00401038      add     eax, [ebp-4]
20 .text:0040103B      mov     esp, ebp
21 .text:0040103D      pop     ebp
22 .text:0040103E      retn
```

[ebp-8]의 값을 0x100과 비교  
0x100보다 크면 점프



- 04 반복문 [4/4]

```

1 .text:00401000    push    ebp
2 .text:00401001    mov     ebp, esp
3 .text:00401003    sub     esp, 8
4 .text:00401006    mov     dword ptr [ebp-8], 0
5 .text:0040100D    jmp     short loc_401018
6 .text:0040100F    mov     eax, [ebp-8]
7 .text:00401012    add     eax, 1
8 .text:00401015    mov     [ebp-8], eax
9 .text:00401018    cmp     dword ptr [ebp-8], 100h
10 .text:0040101F   jg     short loc_401035
11 .text:00401021    mov     ecx, [ebp+8]
12 .text:00401024    sub     ecx, 1
13 .text:00401027    mov     [ebp+8], ecx
14 .text:0040102A    mov     edx, [ebp-4]
15 .text:0040102D    add     edx, 1
16 .text:00401030    mov     [ebp-4], edx
17 .text:00401033    jmp     short loc_40100F
18 .text:00401035    mov     eax, [ebp+8]
19 .text:00401038    add     eax, [ebp-4]
20 .text:0040103B    mov     esp, ebp
21 .text:0040103D    pop     ebp
22 .text:0040103E    retn

```

크지 않으면  
c--; d+  
+;

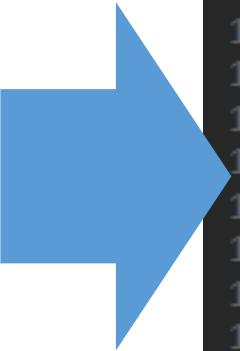
크면  
함수 에필로그를 진행



- 05 구조체와 API Call [1/5]

```

1 void RunProcess()
2 {
3     STARTUPINFO si;
4     PROCESS_INFORMATION pi;
5
6     ZeroMemory(&si, sizeof(si));
7     si.cb = sizeof(si);
8     ZeroMemory(&pi, sizeof(pi));
9
10    // Start the child process.
11    if(!CreateProcess(NULL,
12                      "MyChildProcess",
13                      NULL,
14                      FALSE,
15                      0,
16                      NULL,
17                      NULL,
18                      &si,
19                      &pi))
20    {
21        printf("CreateProcess failed.\n");
22        return;
23    }
24
25    // Wait until child process exits.
26    WaitForSingleObject(pi.hProcess, INFINITE);
27
28    // Close process and thread handles.
29    CloseHandle(pi.hProcess);
30    CloseHandle(pi.hProcess);
31 }
32 }
```



1	0x401000	PUSH	EBP		
2	0x401001	MOV EBP,	ESP		
3	0x401003	SUB	---	0x401034	PUSH
4	0x401006	PUS	21	0x401035	PUSH
5	0x401008	PUS	22	0x401037	PUSH
6	0x40100A	LEA	23	0x401039	PUSH
7	0x40100D	PUS	24	0x40103B	PUSH
8	0x40100E	CAL	25	0x40103D	PUSH
9	0x401013	ADD	26	0x40103F	PUSH
10	0x401016	MOV	27	0x401041	PUSH
11	0x40101D	PUS	28	0x401046	PUSH
12	0x40101F	PUS	29	0x401048	CALL
13	0x401021	LEA	31	0x40104E	DWORD PTR DS:CreatePro
14	0x401024	PUS	32	0x401052	TEST
15	0x401025	CAL	33	0x401057	JNZ SHORT calling.00401061
16	0x40102A	ADD	34	0x40105C	PUSH
17	0x40102D	LEA	36	0x40105F	calling.00407040
18	0x401030	PUS	37	0x401061	CALL
19	0x401031	LEA	38	0x401063	calling.0040116F
20	0x401034	PUS	39	0x401066	ADD ESP, 4
21	0x401035	PUS	40	0x401067	JMP SHORT calling.00401081
22	0x401037	PUS	41	0x40106D	PUSH -1
23	0x401039	PUS	42	0x401070	MOV ECX, DWORD PTR SS:[EBP-10]
24	0x40103B	PUS	43	0x401071	PUSH ECX
25	0x40103D	PUS	44	0x401077	CALL DWORD PTR DS:WaitForSi
46	0x401081			0x40107A	MOV EDX, DWORD PTR SS:[EBP-10]
				0x40107B	PUSH EDX
				0x40107C	CALL DWORD PTR DSS:CloseHan
				0x40107D	MOV EAX, DWORD PTR SS:[EBP-C]
				0x40107E	PUSH EAX
				0x40107F	CALL DWORD PTR DS:CloseHand
				0x401080	MOV ESP, EBP



- 05 구조체와 API Call [2/5]

```

1 0x401000    PUSH   EBP
2 0x401001    MOV    EBP, ESP
3 0x401003    SUB    ESP, 54
4 0x401006    PUSH    44
5 0x401008    PUSH    0
6 0x40100A    LEA    EAX, DWORD PTR SS:[EBP-54]
7 0x40100D    PUSH    EAX
8 0x40100E    CALL   calling.004011A0
9 0x401013    ADD    ESP, 0C
10 0x401016   MOV    DWORD PTR SS:[EBP-54], 44
11 0x40101D   PUSH    10
12 0x40101F   PUSH    0
13 0x401021   LEA    ECX, DWORD PTR SS:[EBP-10]
14 0x401024   PUSH    ECX
15 0x401025   CALL   calling.004011A0
16 0x40102A   ADD    ESP, 0C
17 0x40102D   LEA    EDX, DWORD PTR SS:[EBP-10]
18 0x401030   PUSH    EDX
19 0x401031   LEA    EAX, DWORD PTR SS:[EBP-54]
20 0x401034   PUSH    EAX
21 0x401035   PUSH    0
22 0x401037   PUSH    0
23 0x401039   PUSH    0
24 0x40103B   PUSH    0
25 0x40103D   PUSH    0

```

함수 프롤로그

크기 : 0x44

```
typedef struct _STARTUPINFO {
    DWORD cb;
    LPTSTR lpReserved;
    LPTSTR lpDesktop;
    LPTSTR lpTitle;
    DWORD dwX;
    DWORD dwY;
    DWORD dwXSize;
    DWORD dwYSize;
    DWORD dwXCountChars;
    DWORD dwYCountChars;
    DWORD dwFillAttribute;
    DWORD dwFlags;
    WORD wShowWindow;
    WORD cbReserved2;
    LPBYTE lpReserved2;
    HANDLE hStdInput;
    HANDLE hStdOutput;
    HANDLE hStdError;
} STARTUPINFO, *LPSTARTUPINFO;
```

54h 바이트 스택 확보  
두 개의 구조체

크기 : 0x10

```
typedef struct _PROCESS_INFORMATION {
    HANDLE hProcess;
    HANDLE hThread;
    DWORD dwProcessId;
    DWORD dwThreadId;
} PROCESS_INFORMATION, *LPPROCESS_INFORMATION;
```



- 05 구조체와 API Call [3/5]

```

1  0x401000  PUSH   EBP
2  0x401001  MOV     EBP,  ESP
3  0x401003  SUB    ESP,  54
4  0x401006  PUSH    44
5  0x401008  PUSH    0
6  0x40100A  LEA    EAX,  DWORD PTR SS:[EBP-54]
7  0x40100D  PUSH    EAX
8  0x40100E  CALL    calling.004011A0
9  0x401013  ADD    ESP, 0C
10 0x401016  MOV    DWORD PTR SS:[EBP-54], 44
11 0x40101D  PUSH    10
12 0x40101F  PUSH    0
13 0x401021  LEA    ECX,  DWORD PTR SS:[EBP-10]
14 0x401024  PUSH    ECX
15 0x401025  CALL    calling.004011A0
16 0x40102A  ADD    ESP, 0C
17 0x40102D  LEA    EDX,  DWORD PTR SS:[EBP-10]
18 0x401030  PUSH    EDX
19 0x401031  LEA    EAX,  DWORD PTR SS:[EBP-54]
20 0x401034  PUSH    EAX
21 0x401035  PUSH    0
22 0x401037  PUSH    0
23 0x401039  PUSH    0
24 0x40103B  PUSH    0
25 0x40103D  PUSH    0

```

ZeroMemory()  
\_STARTUPINFO 구조체 초기화

\_STARTUPINFO 구조체의  
첫 번째 멤버 변수에 0x  
44를 삽입

ZeroMemory()  
\_PROCESS\_INFORMATION  
구조체 초기화



- 05 구조체와 API Call [4/5]

```

15 0x401025 CALL calling.004011A0
16 0x40102A ADD ESP, 0C
17 0x40102D LEA EDX, DWORD PTR SS:[EBP-10]
18 0x401030 PUSH EDX
19 0x401031 LEA EAX, DWORD PTR SS:[EBP-54]
20 0x401034 PUSH EAX
21 0x401035 PUSH 0
22 0x401037 PUSH 0
23 0x401039 PUSH 0
24 0x40103B PUSH 0
25 0x40103D PUSH 0
26 0x40103F PUSH 0
27 0x401041 PUSH calling.00407030
28 0x401046 PUSH 0
29 0x401048 CALL DWORD PTR DS>CreateProcessA
30 0x40104E TEST EAX, EAX
31 0x401050 JNZ SHORT calling.00401061
32 0x401052 PUSH calling.00407040
33 0x401057 CALL calling.0040116F
34 0x40105C ADD ESP, 4
35 0x40105F JMP SHORT calling.00401081
36 0x401061 PUSH -1
37 0x401063 MOV ECX, DWORD PTR SS:[EBP-10]
38 0x401066 PUSH ECX
39 0x401067 CALL DWORD PTR DS:WaitForSingleObject
40 0x40106D MOV EDX, DWORD PTR SS:[EBP-10]

```

CreateProcess()를 호출  
인자들 역순으로 전달

리턴 값이 NULL인지 검사  
NULL이 아닐 경우 점프

리턴 값이 NULL인 경우 "Create  
Process failed.%a"를 출력 0040  
116f € printf()



- 05 구조체와 API Call [5/5]

```
20 0x401034 PUSH EAX
21 0x401035 PUSH 0
22 0x401037 PUSH 0
23 0x401039 PUSH 0
24 0x40103B PUSH 0
25 0x40103D PUSH 0
26 0x40103F PUSH 0
27 0x401041 PUSH calling.00407030
28 0x401046 PUSH 0
29 0x401048 CALL DWORD PTR DS:CreateProcessA
30 0x40104E TEST EAX, EAX
31 0x401050 JNZ SHORT calling.00401061
32 0x401052 PUSH calling.00407040
33 0x401057 CALL calling.0040116F
34 0x40105C ADD ESP, 4
35 0x40105F JMP SHORT calling.00401081
36 0x401061 PUSH -1
37 0x401063 MOV ECX, DWORD PTR SS:[EBP-10]
38 0x401066 PUSH ECX
39 0x401067 CALL DWORD PTR DS:WaitForSingleObject
40 0x40106D MOV EDX, DWORD PTR SS:[EBP-10]
41 0x401070 PUSH EDX
42 0x401071 CALL DWORD PTR DSS:CloseHandle
43 0x401077 MOV EAX, DWORD PTR SS:[EBP-C]
44 0x40107A PUSH EAX
45 0x40107B CALL DWORD PTR DS:CloseHandle
46 0x401081 MOV ESP, EBP
47 0x401083 POP EBP
48 0x401084 RETN
```

WaitingForSingleObject &  
CloseHandle을 진행

함수 에필로그



- 참고문헌
  - 리버싱 엔지니어링 바이블, 강병탁 지음



# 리버싱 기초



- **올리디버거**

- 올리 유스척(Olleh Yuschuk)이 개발한 x86 디버거
  - 무료, 사용의 편의성, 기능 확장을 위한 플러그인
- 이뮤니티 시큐리티에서 OllyDbg 1.1 베이스를 구매하여 Immunity 디버거로 재포장
  - 파이썬 인터프리터 추가
  - 익스플로잇 개발자에게 최적화
  - 올리 디버거 자체의 버그 패치
  - 일반적인 올리디버거와 같은 사용방식의 인터페이스를 가짐



- **실행 파일 열기**
  - 파일 열기
  - 드래그
  - 마우스 오른쪽 키를 사용하여 실행
  - 프로그램 가장 앞단을 0xCC로 패치하여 실행
- **실행 중인 프로세스 덧붙이기**
  - File > Attach를 선택



- CPU 인터페이스 – 디스어셈블러/레지스터/스택/덤프 윈도우

**OllyDbg - RegMech.exe - [CPU - main thread, module RegMech]**

File View Debug Plugins Options Window Help

L E M T W H C / K B R ... S ?

0041262C	68 1C364100	PUSH RegMech.0041361C		Registers (FPU)
00412631	E8 EFFFFFFF	CALL RegMech.00412624		EAX 00964370 RegMech.0096
00412636	0000	ADD BYTE PTR DS:[EAX], AL		ECX F4358ACC
00412638	68 00000030	PUSH 30000000		EDX 7C93E514 ntdll.KiFast
0041263D	0000	ADD BYTE PTR DS:[EAX], AL		EBX 00000000
0041263F	0060 00	ADD BYTE PTR DS:[EAX], AH		ESP 0013D58C
00412642	0000	ADD BYTE PTR DS:[EAX], AL		EBP 0013D7B0
00412644	48	DEC EAX		ESI 0096AA30 RegMech.0096
00412645	0000	ADD BYTE PTR DS:[EAX], AL		EDI 0041262C RegMech.0041
00412647	0036	ADD BYTE PTR DS:[ESI], DH		EIP 0041262C RegMech.0041
00412649	BE 4F32F31F	MOV ESI, 1FF3324F	I/O command	C 0 ES 0023 32bit 0{FFFF}
0041264E	EE	OUT DX, AL		P 0 CS 001B 32bit 0{FFFF}
0041264F	4E	DEC ESI		A 1 SS 0023 32bit 0{FFFF}
00412650	94	XCHG EAX, ESP		Z 0 DS 0023 32bit 0{FFFF}
00412651	CF	IRETD		S 0 FS 003B 32bit 7FFDFO
00412652	COAE BB412A75	SHR BYTE PTR DS:[ESI+752A41BB], 0	Shift constant out of range 1..31	T 0 GS 0000 NULL
00412659	0000	ADD BYTE PTR DS:[EAX], AL		D 0
0041265B	0000	ADD BYTE PTR DS:[EAX], AL		O 0 LastErr ERROR_FILE_N
0041265D	0001	ADD BYTE PTR DS:[ECX], AL		EFL 00200212 (NO,NB,NE,A,
0041265F	0000	ADD BYTE PTR DS:[EAX], AL		ST0 empty -UNORM D1D8 010
00412661	0040 00	ADD BYTE PTR DS:[EAX], AL		ST1 empty 0.0
00412664	0000	ADD BYTE PTR DS:[EAX], AL		ST2 empty 0.0
00412666	0100	ADD DWORD PTR DS:[EAX], EAX		ST3 empty 0.0

0041361C=RegMech.0041361C

Address	Hex dump	ASCII	0013D58C	0115BF64	RETURN to 0115BF64
00964000	5F 6E E2 77 5B EA E2 77 32 86 E2 77 D2 B7 E2 77	_n? [侑	0013D590	00400000	ASCII "MZ"
00964010	EA B5 E2 77 FA 6B E2 77 79 6F E2 77 70 5B E2 77	援???yc	0013D594	00000000	
00964020	E0 5F E2 77 23 AD E2 77 00 00 00 00 52 FF 7D 7C ??#?w.	??#?w.	0013D598	00151F86	
00964030	E9 FF 7D 7C FD FD 7D 7C 4A 93 7D 7C 74 A1 7D 7C ?}][義	?}][義	0013D59C	0000000A	
00964040	A1 9E 7D 7C 49 61 80 7C 3C 01 7E 7C 41 B7 7D 7C 舊}[Iat	舊}[Iat	0013D5A0	0013FF2C	
00964050	FF FC 7D 7C FB CD 82 7C 33 0C 80 7C 26 B4 7F 7C ? 灘?	? 灘?	0013D5A4	00000000	
00964060	35 2F 84 7C 12 41 7E 7C D4 1A 7D 7C F1 9A 7D 7C 5/?~	5/?~	0013D5A8	7FFDD000	
00964070	40 AE 7D 7C 21 FE 93 7C 7B 1D 7D 7C 30 FE 93 7C @?!?!	@?!?!	0013D5AC	7C94A1BA	RETURN to ntdll.7C94A1BA from ntdll.7C94A1BA
00964080	A8 C1 7D 7C 8B 99 7D 7C 6B 23 7D 7C 6D 0C 7E 7C 𩫇}[竈	𩫇}[竈	0013D5B0	003F0000	
00964090	F2 1E 7D 7C 18 35 80 7C B7 24 7D 7C 30 25 7D 7C ?}][士	?}][士	0013D5B4	40000060	
009640A0	DF E9 7D 7C BB EA 7D 7C D0 97 7D 7C E7 9B 7D 7C (生}][社	(生}][社	0013D5B8	7C94005D	RETURN to ntdll.7C94005D from ntdll.7C93E5
009640B0	12 18 7D 7C EF OF 7E 7C 28 1A 7D 7C 9C EE 7D 7C 𩫇}[?~	𩫇}[?~	0013D5BC	011678C5	
009640C0	F9 F9 7F 7C 7D EE 7D 7C 40 BA 7D 7C DF 9F 7E 7C 桢D}]?~	楗D}]?~	0013D5C0	00000080	
009640D0	D0 21 7D 7C D5 C0 82 7C 31 49 83 7C AD 97 80 7C ?}][藍?	?}][藍?	0013D5C4	003F2748	
009640E0	F0 BF 82 7C D2 97 80 7C 07 07 7E 7C 97 29 80 7C 嘴??o	嘴??o	0013D5C8	7C98F8B4	RETURN to ntdll.7C98F8B4 from ntdll.RtlCo
009640F0	36 23 7D 7C E3 5F 7E 7C 54 1E 7D 7C A5 B9 7D 7C 6#}?~	6#}?~	0013D5CC	7C940435	RETURN to ntdll.7C940435

Command :

Hardware breakpoint 2 at RegMech 0041262C

Paused

# 리버싱 기초



- 메모리 맵 - 메모리에 프로그램이 배열 상태 확인 가능
- View > Threads를 통해 현재 실행 스레드 확인 가능
  - 각 스레드는 개별 스택을 가지고 있음

\* OllyDbg - RegMech.exe - [Threads]

Ident	Entry	Data block	Last error	Status	Priority	User time	System time
00000FE8	0093F4C3	7FFDF000	ERROR_SUCCESS (00)	Paused	32 + 0	0.2968 s	0.3281 s

Context menu for thread 00000FE8:

- Actualize
- Suspend
- Set priority
- Open in CPU
- Dump thread data block
- Kill thread
- Copy to clipboard
- Sort by
- Appearance

올리디버거의 Threads 창

\* OllyDbg - RegMech.exe - [Memory map]

Address	Size	Owner	Section	Contains	Type	Access	Initial	Mapped as
00010000	00001000				Priv	RW	RW	
00020000	00001000				Priv	RW	RW	
00030000	00008000				Priv	RW	RW	
00120000	00001000				Priv	RW	Guar	RW
00121000	0001F000				Priv	RW	Guar	RW
00140000	00003000				Map	R	R	
00150000	00013000				Priv	RW	RW	
00250000	00006000				Priv	RW	RW	
00260000	00003000				Map	RW	RW	
00270000	00016000				Map	R	R	
00290000	00004000				Map	R	R	
002E0000	00004000				Map	R	R	
00330000	00006000				Map	R	R	
00340000	00004000				Map	R	R	
00390000	00001000				Priv	RW	RW	
003A0000	00001000				Priv	RW	RW	
003B0000	00001000				Priv	RW	RW	
003C0000	00003000				Map	R	R	
003D0000	00005000				Priv	RW	RW	
003E0000	00002000				Map	R	R	
003F0000	00000000				Priv	RW	RW	
00400000	00001000	RegMech			Imag	R	RWE	
00401000	00505000	RegMech	.text		Imag	R	RWE	
00906000	00000000	RegMech	.data		Imag	R	RWE	
00914000	00040000	RegMech	.text1		Imag	R	RWE	
00954000	00010000	RegMech	.adata		Imag	R	RWE	
00964000	00020000	RegMech	.data1		data,import	Imag	R	RWE
00984000	00130000	RegMech	.pdata		Imag	R	RWE	
00AB4000	00093000	RegMech	.rsrc		resources	Imag	R	RWE
00B50000	00006000				Map	R E	R E	
00C10000	00002000				Map	R E	R E	
00C20000	00103000				Map	R	R	
00D30000	000CC000				Map	R E	R E	
01030000	00001000				Priv	RW	RW	
01130000	0004B000				Priv	RW	RW	
01180000	0000C000				Priv	RW	RW	
01190000	00002000				Map	R	R	

올리디버거의 Memory Map 창



- OllyDbg 코드 실행 옵션

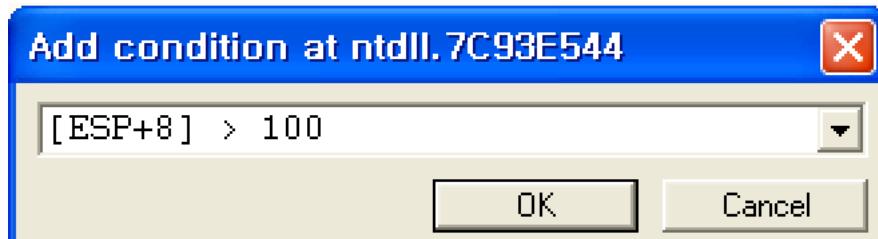
기능	메뉴	핫키
실행(Run/Play)	Debug > Run	F9
정지(Pause)	Debug > Pause	F12
선택까지 실행	Breakpoint > Run to Selection	F4
리턴까지 실행	Debug > Execute till Return	CTRL + F9
사용자 코드 전까지 실행	Debug > Execute till User Code	ALT + F9
싱글 스텝/스텝 인트	Debug > Step Into	F7
스텝 오버	Debug > Step Over	F8
함수 따라가기		Enter
뒤로 가기		-



- OllyDbg 브레이크포인트 옵션

기능	메뉴	핫키
소프트웨어 브레이크 포인트	Breakpoint > Toggle	F2
조건 브레이크포인트	Breakpoint > Conditional	SHIFT + F2
하드웨어 브레이크 포인트	Breakpoint > Hardware, on Execution	
접근(읽기, 쓰기 또는 실행)에 대한 메모리 브레이크 포인트	Breakpoint > Memory, on Access	F2 (메모리 선택)
쓰기에 대한 메모리 브레이크 포인트	Breakpoint > Memory, on Write	

- 조건 브레이크포인트 활용 예





- DLL 파일 역시 디버깅 가능
  - 디버거에서 직접 실행하지 않고 loaddll.exe라는 더미 프로그램 사용



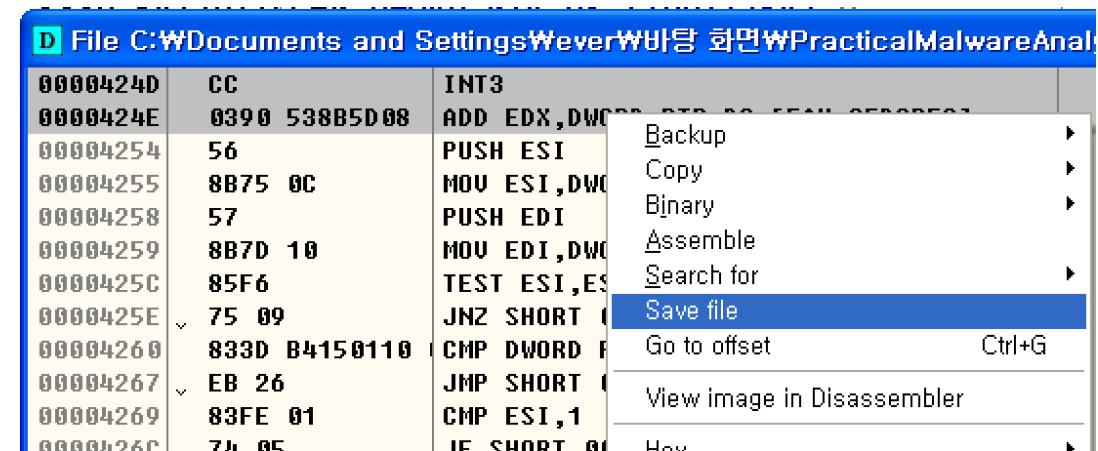
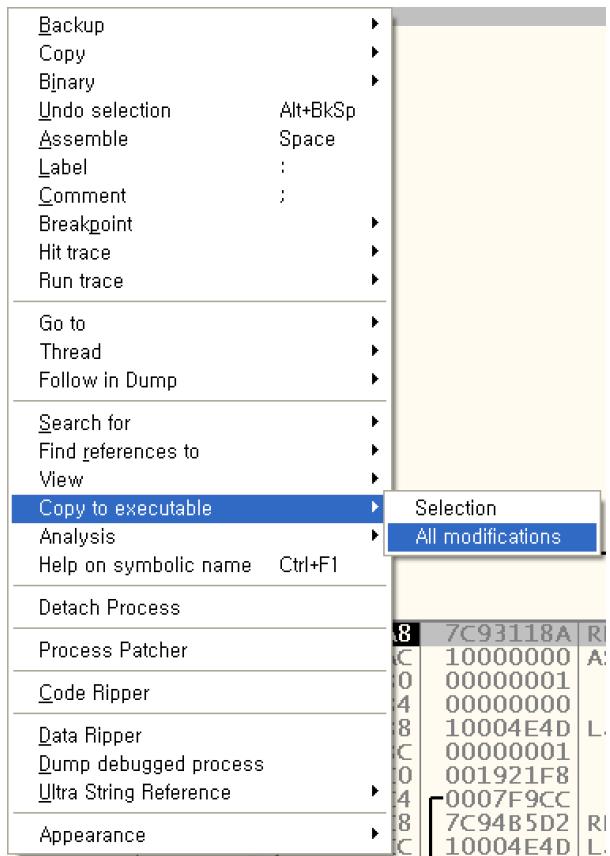
- DLL을 로드하면 DllMain에서 정지

Address	OpCode	Mnemonic	Comments
10004E4D	\$ 55	PUSH EBP	
10004E4E	. 8BEC	MOV EBP,ESP	
10004E50	. 53	PUSH EBX	
10004E51	. 8B5D 08	MOV EBX, DWORD PTR SS:[EBP+8]	
10004E54	. 56	PUSH ESI	
10004E55	. 8B75 0C	MOV ESI, DWORD PTR SS:[EBP+C]	
10004E58	. 57	PUSH EDI	
10004E59	. 8B7D 10	MOV EDI, DWORD PTR SS:[EBP+10]	
10004E5C	. 85F6	TEST ESI,ESI	
10004E5F	75 00	JMP SHORT Lab03-02 10004E60	



- 예외 처리
  - SHIFT + F7/F8/F9
  - Options > Debugging Options > Exceptions

## 파치





# 리버싱 기초



- lena tutorial?

# TUTS4YOU

Latest   Downloads   Forums   Blogs   Submissions   Search Engine   Site Rip  

## Lenas Reversing for Newbies [ Nice collection of tutorials aimed particularly for newbie reverse enginners... ]

Name	Date	Author	Size	DL's	Rating	Get
Reversing for Newbies - Complete	02 September 2006 - 09:14	Lena	139.59 MB	305134	Not rated	
Reversing for Newbies - Index	02 September 2006 - 08:26	Lena	0	7735	Not rated	
Reversing for Newbies 01	02 September 2006 - 09:09	Lena	1.33 MB	135865	9.5/34	
Reversing for Newbies 02	02 September 2006 - 09:19	Lena	1.2 MB	55722	9.1/21	
Reversing for Newbies 03	02 September 2006 - 09:21	Lena	1.71 MB	46215	8.5/15	
Reversing for Newbies 04	02 September	.	0.67 MB	37500	8.0/10	

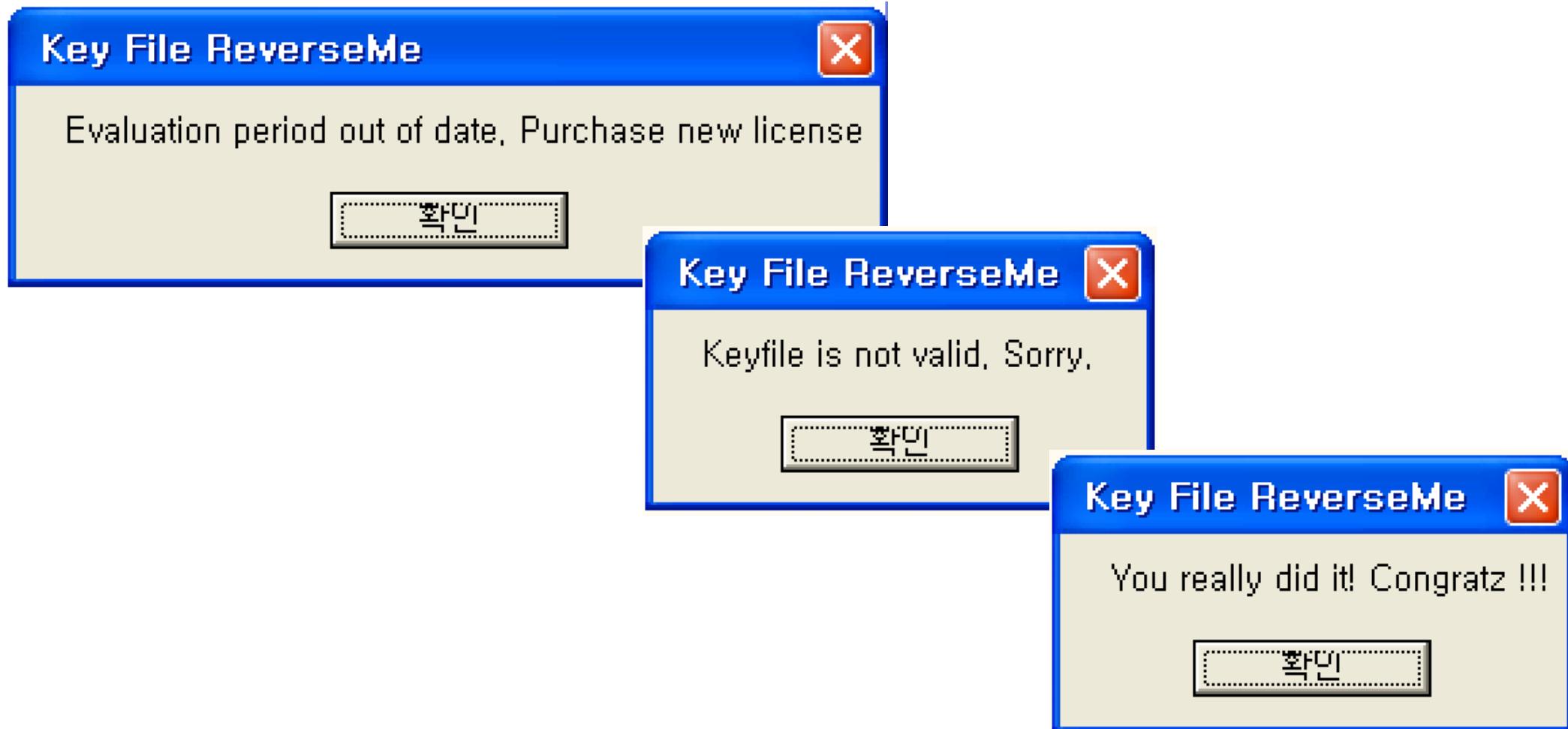
## 리버싱 기초



- 올리 디버거 CPU창 분석



- ReverseMe Crack!





- Lena 1 분석 실습



## 2. 윈도우 실행 파일과 패커

## 2. 윈도우 실행 파일과 패커



- 01 PE 파일 개요

- Portable Executable File Format

파일(File)이

- + 이식 가능한 다른 곳에 옮겨져도(Portable)
- + 실행 가능하도록 (Executable)
- + 만든 포맷(Format)

- 위키피디아([https://ko.wikipedia.org/wiki/PE\\_포맷](https://ko.wikipedia.org/wiki/PE_포맷))

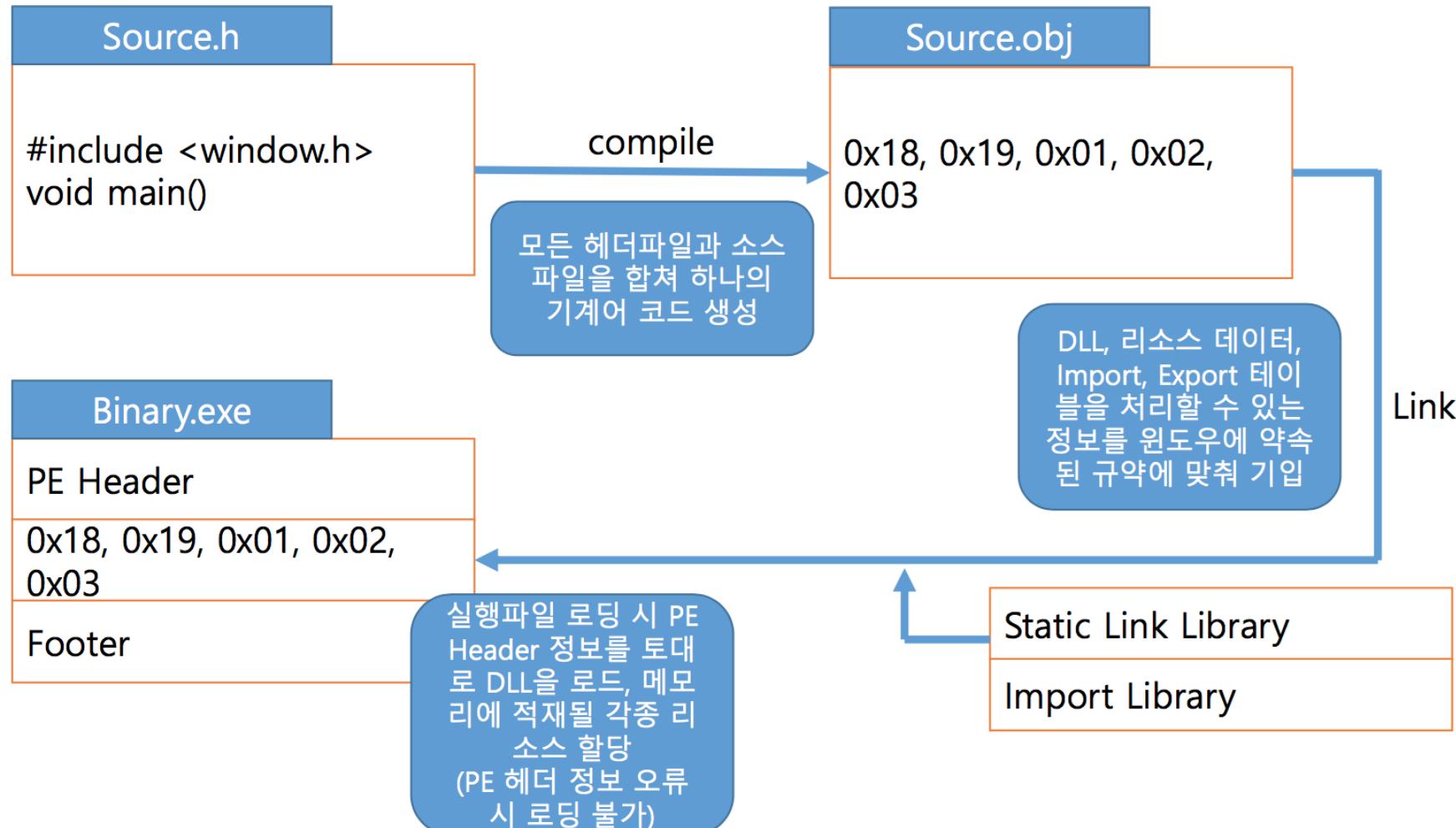
- 윈도우에서 사용하는 실행 파일, DLL 파일 등을 위한 파일 형식
    - 윈도우 로더가 실행 가능한 코드를 관리하는데 필요한 정보를 캡슐화한 데이터 구조체
    - 링킹을 위한 동적 라이브러리 참조, API 익스포트와 임포트 테이블, 자원 관리 데이터 그리고 TLS 데이터를 포함

## 2. 윈도우 실행 파일과 패커



### • 01 PE 파일 개요

- 실행 파일 컴파일 과정





## 2. 윈도우 실행 파일과 패커

- 01 PE 파일 개요

- PE 파일 분석에 사용되는 도구

- 1. PEview

- <https://www.aldeid.com/wiki/PEView>

- 2. Stud\_PE

- <http://www.cgsoftlabs.ro/studpe.html>

- 3. PEiD

- <https://www.aldeid.com/wiki/PEiD>

- 4. exeinfo

- <http://www.nirsoft.net/utils/exeinfo.html>

- 5. pestudio

- <https://www.winitor.com/>

- 6. preframe.py

- <https://github.com/guelfoweb/peframe>

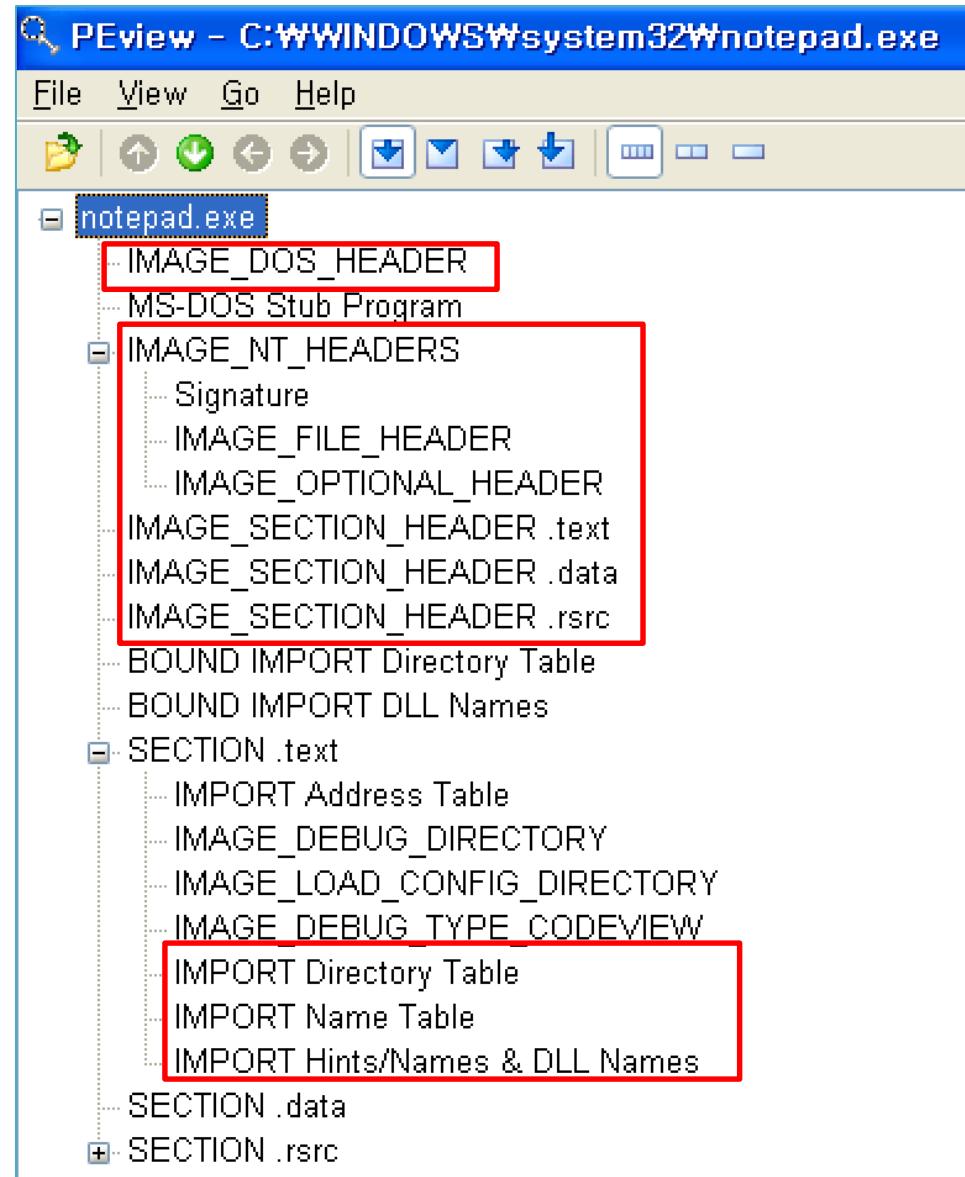
## 2. 윈도우 실행 파일과 패커



### • 02 PE 파일 구조

#### - 주요 구조

- IMAGE\_DOS\_HEADER
- IMAGE\_NT\_HEADER
- IMAGE\_FILE\_HEADER
- IMAGE\_OPTIONAL\_HEADER
- IMAGE\_SECTION\_HEADER
- IMAGE\_IMPORT\_DESCRIPTOR
- IMAGE\_EXPORT\_DIRECTORY
- IMAGE\_IMPORT\_BY\_NAME
- IMAGE\_THUNK\_DATA32



peview.exe로 바라본 PE 구조



## 2. 윈도우 실행 파일과 패커

- 02 PE 파일 구조

- IMAGE\_DOS\_HEADER

```
1 typedef struct _IMAGE_DOS_HEADER
2 {
3     WORD e_magic;
4     WORD e_cblp;
5     WORD e_cp;
6     WORD e_crlc;
7     WORD e_cparhdr;
8     WORD e_minalloc;
9     WORD e_maxalloc;
10    WORD e_ss;
11    WORD e_sp;
12    WORD e_csum;
13    WORD e_ip;
14    WORD e_cs;
15    WORD e_lfanrc;
16    WORD e_ovno;
17    WORD e_res[4];
18    WORD e_oemid;
19    WORD e_oeminfo;
20    WORD e_res2[10];
21    LONG e_lfanew;
22 } IMAGE_DOS_HEADER, *PIMAGE;
```

IMAGE\_NT\_HEADER의  
구조체 위치를 알림

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00000000	4D	5A	90	00	03	00	00	00	04	00	00	00	FF	FF	00	00
00000010	B8	00	00	00	00	00	00	00	40	00	00	00	00	00	00	00
00000020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000030	00	00	00	00	00	00	00	00	00	00	00	00	E0	00	00	00
00000040	0E	1F	BA	OE	00	B4	09	CD	21	B8	01	4C	CD	21	54	68

## 2. 윈도우 실행 파일과 패커



#### • 02 PE 파일 구조

- IMAGE\_NT\_HEADER 1
    - Signature
      - “PEWOW0”
      - 4바이트 바이러스에 자신의 시그니쳐를 심기도 함
      - 바이러스나 악성코드 감염 표식용으로 사용 (지금은 안됨)

```
1 typedef struct _IMAGE_NT_HEADERS {  
2     DWORD             Signature;  
3     IMAGE_FILE_HEADER FileHeader;  
4     IMAGE_OPTIONAL_HEADER OptionalHeader;  
5 } IMAGE_NT_HEADERS, *PIMAGE_NT_HEADERS;
```

## 2. 윈도우 실행 파일과 패커



### • 02 PE 파일 구조

#### - IMAGE\_NT\_HEADER 2

- 나머지는 두 개의 큰 구조체로 이루어짐
- FileHeader, OptionalHeader

```
1 typedef struct _IMAGE_NT_HEADERS {
2     DWORD             Signature;
3     IMAGE_FILE_HEADER FileHeader;
4     IMAGE_OPTIONAL_HEADER OptionalHeader;
5 } IMAGE_NT_HEADERS, *PIMAGE_NT_HEADERS;
```

```
1 typedef struct _IMAGE_OPTIONAL_HEADER {
2     WORD           Magic;
3     BYTE           MajorLinker;
4     BYTE           MinorLinker;
5     DWORD          SizeOfCode;
6     DWORD          SizeOfInitialCommit;
7     DWORD          SizeOfUninitializedData;
8     DWORD          AddressOfEntryPoint;
9     DWORD          BaseOfCode;
10    DWORD         BaseOfData;
11    DWORD         ImageBase;
12    DWORD         SectionAlignment;
13    DWORD         FileAlignment;
14    WORD          MajorOperatingSystemVersion;
15    WORD          MinorOperatingSystemVersion;
```

```
1 typedef struct _IMAGE_FILE_HEADER {
2     WORD   Machine;
3     WORD   NumberOfSections;
4     DWORD  TimeDateStamp;
5     DWORD  PointerToSymbolTable;
6     DWORD  NumberOfSymbols;
7     WORD   SizeOfOptionalHeader;
8     WORD   Characteristics;
9 } IMAGE_FILE_HEADER, *PIMAGE_FILE_HEADER;
```

## 2. 윈도우 실행 파일과 패커



- 02 PE 파일 구조

- IMAGE\_NT\_HEADER 2

- IMAGE\_FILE\_HEADER [1/2]

```
1 typedef struct _IMAGE_FILE_HEADER {  
2     WORD Machine;           → 어떤 CPU에서 실행 가능한지 알림  
3     WORD NumberOfSections;  → (일반 Desktop, Laptop에서 사용할 경우, 별로  
4     DWORD TimeStamp;       필요 없음)  
5     DWORD PointerToSymbolTable;  
6     DWORD NumberOfSymbols;  
7     WORD SizeOfOptionalHeader;  
8     WORD Characteristics;  
9 } IMAGE_FILE_HEADER, *PIMAGE_FILE_HEADER;
```

이 파일이 가진 세션의 개수를 알림  
(일반적으로 .text, .rdata, .data, .rsrc  
4개 섹션이 존재)

obj → EXE 파일을 만든 시간을 알림  
(델파이로 만들어진 파일은 항상  
1992년으로 표시됨)

## 2. 윈도우 실행 파일과 패커



- 02 PE 파일 구조
  - IMAGE\_NT\_HEADER 2
    - IMAGE\_FILE\_HEADER [2/2]

IMAGE\_OPTIONAL\_HEADER32의 구조체 크기를 알림  
(PE를 로딩하기 위한 굉장히 중요한 구조체를 담고 있음, 운영체제마다 크기가 다를 수 있어 PE 로더에서는 이 값을 먼저 확인)

```
1 | typedef struct _IMAGE_FILE_HEADER {
2 |     WORD Machine;
3 |     WORD NumberOfSections;
4 |     DWORD TimeDateStamp;
5 |     DWORD PointerToSymbolTable;
6 |     DWORD NumberOfSymbols;
7 |     WORD SizeOfOptionalHeader; ←
8 |     WORD Characteristics; ↓
9 | } IMAGE_FILE_HEADER, *PIMAGE_FILE_HEADER;
```

이 파일이 어떤 형식인지 알림

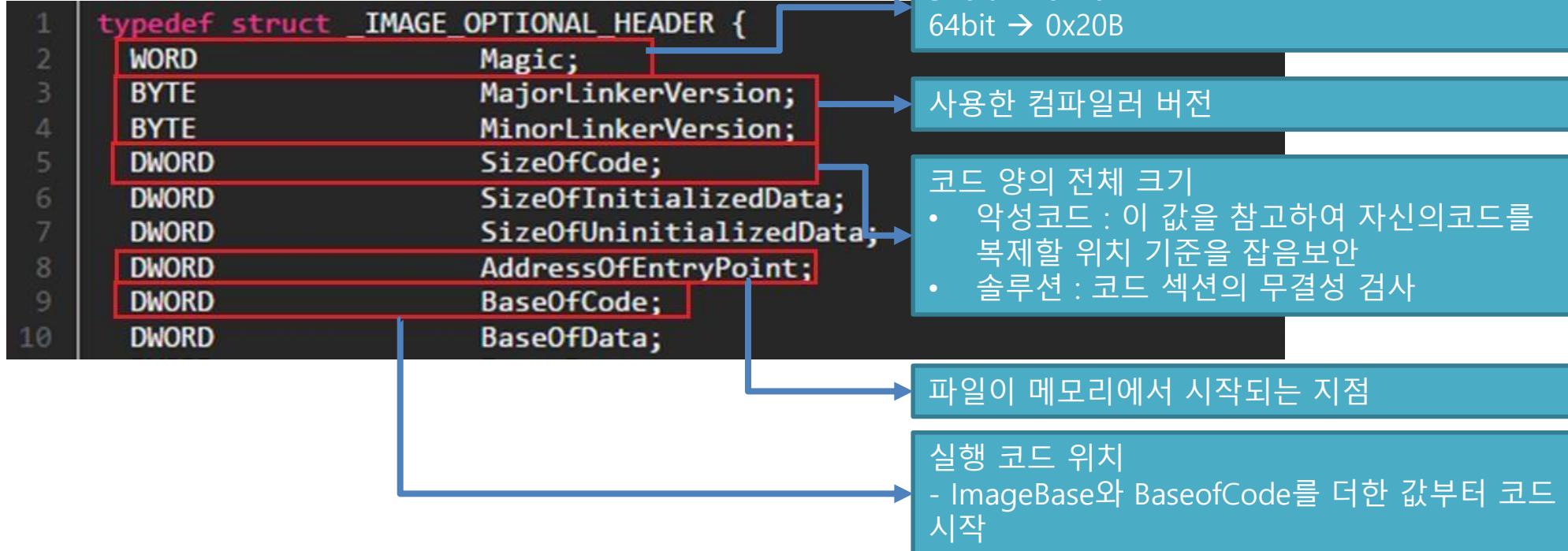
## 2. 윈도우 실행 파일과 패커



- 02 PE 파일 구조

- IMAGE\_NT\_HEADER 2

- IMAGE\_OPTIONAL\_HEADER [1/3] : Standard Fields



## 2. 윈도우 실행 파일과 패커



### • 02 PE 파일 구조

#### - IMAGE\_NT\_HEADER 2

- IMAGE\_OPTIONAL\_HEADER [2/3] : NT additional fields

```
11    DWORD      ImageBase;
12    DWORD      SectionAlignment;
13    DWORD      FileAlignment;
14    WORD       MajorOperatingSystemVersion;
15    WORD       MinorOperatingSystemVersion;
16    WORD       MajorImageVersion;
17    WORD       MinorImageVersion;
18    WORD       MajorSubsystemVersion;
19    WORD       MinorSubsystemVersion;
20    DWORD      Win32VersionValue;
21    DWORD      SizeOfImage;
22    DWORD      SizeOfHeaders;
23    DWORD      CheckSum;
24    WORD       Subsystem;
25    WORD       DllCharacteristics;
26    DWORD      SizeOfStackReserve;
27    DWORD      SizeOfStackCommit;
28    DWORD      SizeOfHeapReserve;
29    DWORD      SizeOfHeapCommit;
30    DWORD      LoaderFlags;
31    DWORD      NumberOfRvaAndSizes;
32    IMAGE_DATA_DIRECTORY DataDirectory[IMAGE_NUMBEROF_DIRECTORY];
33 } IMAGE_OPTIONAL_HEADER, *PIMAGE_OPTIONAL_HEADER;
```

로드할 가상 메모리 주소  
각 세션을 정렬하기 위한 정렬 단위  
(기본값 0x1000)  
EXE/DLL이 메모리에 로딩됐을 때  
전체크기  
PE 헤더의 크기를 알림  
(기본값 0x1000)  
IMAGE\_DATA\_DIRECTORY 구조체  
- VirtualAddress와 Size 필드  
- Export, Import, Rsrc 디렉터리와  
IAT 등의 가상 주소와 크기 정보

## 2. 윈도우 실행 파일과 패커



- 02 PE 파일 구조

- IMAGE\_NT\_HEADER 2
  - IMAGE\_DATA\_DIRECTORY 구조체

[https://msdn.microsoft.com/ko-kr/library/windows/desktop/ms680305\(v=vs.85\).aspx](https://msdn.microsoft.com/ko-kr/library/windows/desktop/ms680305(v=vs.85).aspx)

Represents the data directory.

### Syntax

C++

```
typedef struct _IMAGE_DATA_DIRECTORY {  
    DWORD VirtualAddress;  
    DWORD Size;  
} IMAGE_DATA_DIRECTORY, *PIMAGE_DATA_DIRECTORY;
```

### Members

#### VirtualAddress

The relative virtual address of the table.

#### Size

The size of the table, in bytes.

## 2. 윈도우 실행 파일과 패커



### • 02 PE 파일 구조

#### - IMAGE\_NT\_HEADER 2

##### • IMAGE\_DATA\_DIRECTORY 구조체

#### Remarks

The following is a list of the data directories. Offsets are relative to the beginning of the optional header.

Offset (PE/PE32+)	Description
96/112	Export table address and size
104/120	Import table address and size
112/128	Resource table address and size
120/136	Exception table address and size
128/144	Certificate table address and size
136/152	Base relocation table address and size
144/160	Debugging information starting address and size
152/168	Architecture-specific data address and size
160/176	Global pointer register relative virtual address
168/184	Thread local storage (TLS) table address and size
176/192	Load configuration table address and size
184/200	Bound import table address and size
192/208	Import address table address and size
200/216	Delay import descriptor address and size
208/224	The CLR header address and size
216/232	Reserved

#### IMAGE\_OPTIONAL\_HEADER64

```
typedef struct _IMAGE_OPTIONAL_HEADER {  
    WORD Magic;  
    BYTE MajorLinkerVersion;  
    BYTE MinorLinkerVersion;  
    DWORD SizeOfCode;  
    DWORD SizeOfInitializedData;  
    DWORD SizeOfUninitializedData;  
    DWORD AddressOfEntryPoint;  
    DWORD BaseOfCode;  
    ULONGLONG ImageBase  
    DWORD SectionAlignment;  
    DWORD FileAlignment;  
    WORD MajorOperatingSystemVersion;  
    WORD MinorOperatingSystemVersion;  
    WORD MajorImageVersion;  
    WORD MinorImageVersion;  
    WORD MajorSubsystemVersion;  
    WORD MinorSubsystemVersion;  
    DWORD Win32VersionValue;  
    DWORD SizeOfImage;  
    DWORD SizeOfHeaders;  
    DWORD CheckSum;  
    WORD Subsystem;  
    WORD DllCharacteristics;  
    ULONGLONG SizeOfStackReserve;  
    ULONGLONG SizeOfStackCommit;  
    ULONGLONG SizeOfHeapReserve;  
    ULONGLONG SizeOfHeapCommit;  
    DWORD LoaderFlags;  
    DWORD NumberOfRvaAndSizes;  
    IMAGE_DATA_DIRECTORY DataDirectory[IMAGE_NUMBEROF_DIRECTORY];  
} IMAGE_OPTIONAL_HEADER64, *PIMAGE_OPTIONAL_HEADER64;
```

PE : 32비트 포맷

PE32+ 또는 PE+ : 64비트 버전

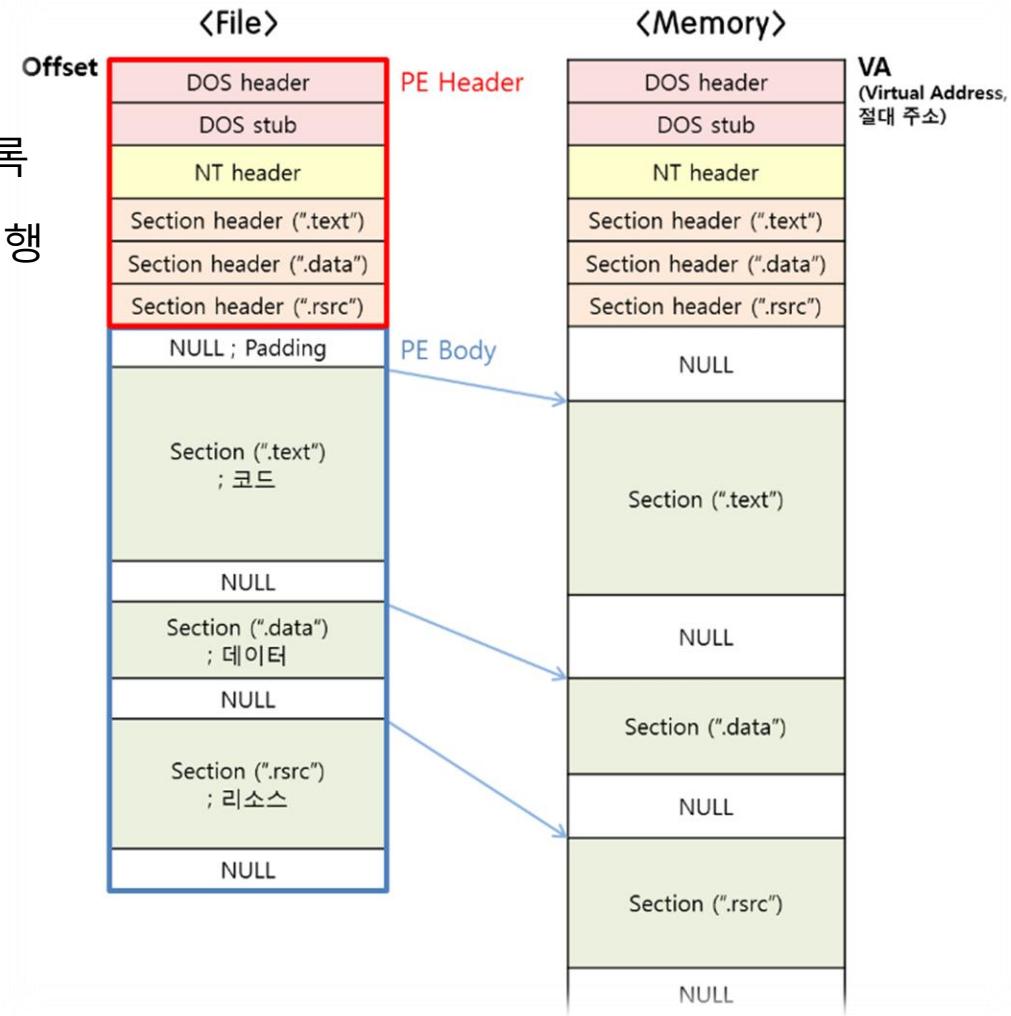
## 2. 윈도우 실행 파일과 패커



### • 02 PE 파일 구조

#### - 섹션(Sections)

- PE 파일에서 섹션은 프로그램의 실제 내용을 담고 있는 블록
- PE가 가상 주소 공간에 로드된 후 섹션 내용이 참고되고 실행



파일 로드 전후 용량 묘사  
(<http://blue-shadow.tistory.com/38>)

## 2. 윈도우 실행 파일과 패커



### • 02 PE 파일 구조 - 섹션의 종류

섹션	이름	설명
.txt	코드 섹션	<ul style="list-style-type: none"><li>프로그램을 실행하기 위한 코드를 담는 섹션</li><li>CPU의 명령 포인터가 되는 IP 레지스터는 이 섹션 내에 존재하는 번지를 가짐</li><li>32비트의 경우 VC++ 7.0부터 실행 기능을 가진 동시에 초기화되지 않은 데이터를 담은 .textss 섹션이 존재(증분 링크 옵션)</li></ul>
.data	데이터 섹션	<ul style="list-style-type: none"><li>초기화된 전역 변수들을 담고 있는 읽고 쓰기가 가능한 섹션 (이전에는 초기화되지 않은 데이터를 위한 데이터 섹션으로 .bss 섹션을 제공 가상 메모리에 매핑될 때 보통 .data 섹션에 병합)</li><li>64비트에서는 PE 파일에서부터 .bss 섹션과 .data 섹션에 병합</li></ul>
.rdata	읽기 전용 데이터 섹션	<ul style="list-style-type: none"><li>문자열 상수나 C++ 가상 함수 테이블 등을 배치</li><li>코드 상에 참조하는 읽기 전용 데이터(.edata, .debug 등)도 이 섹션에 병합</li></ul>
.reloc	기준 재배치 섹션	<ul style="list-style-type: none"><li>실행 파일에 대한 기준 재배치 정보를 담고 있는 섹션</li></ul>
.edata	내보내기(export) 섹션	<ul style="list-style-type: none"><li>내보낼 함수에 대한 정보를 담고 있는 섹션</li><li>.rdata에 병합되기 때문에 DLL에서 별도의 세션이 존재하지 않음</li></ul>
.idata	가져오기(import) 섹션	<ul style="list-style-type: none"><li>가져올 dll과 그 함수 및 변수에 대한 정보를 담고 있는 섹션</li><li>IAT(Import Address Table)이 존재</li><li>.rdata에 병합</li></ul>
.didat	지연 로드 섹션	<ul style="list-style-type: none"><li>지연 로딩(Delay-Loading)을 위한 세션</li></ul>
.tls	TLS 섹션	<ul style="list-style-type: none"><li>__declspec(thread) 지시어와 함께 선언되는 스레드 지역 저장소(Thread Local Storage)를 위한 섹션</li></ul>
.rsrc	리소스 섹션	<ul style="list-style-type: none"><li>대화상자, 아이콘, 커서, 버전 정보 등의 윈도우 PE 파일이 담고 있는 리소스 관련 데이터들이 배치</li></ul>
.debug	디버깅 섹션	<ul style="list-style-type: none"><li>디버깅 정보를 포함</li><li>MS는 오래전부터 이 섹션에 디버깅 관련 기초 정보만을 담고, 실제 정보는 PDB 파일에 별도로 보관</li></ul>

## 2. 윈도우 실행 파일과 패커



### • 02 PE 파일 구조

#### - IMAGE\_Section\_Header

- 주로 각 섹션에 대한 이름, 시작 주소, 사이즈 등의 정보를 관리하는 구조체

```
1 typedef struct _IMAGE_SECTION_HEADER {
2     BYTE Name[IMAGE_SIZEOF_SHORT_NAME];
3     union {
4         DWORD PhysicalAddress;
5         DWORD VirtualSize;
6     } Misc;
7     DWORD VirtualAddress;
8     DWORD SizeOfRawData;
9     DWORD PointerToRawData;
10    DWORD PointerToRelocations;
11    DWORD PointerToLinenumbers;
12    WORD NumberOfRelocations;
13    WORD NumberOfLinenumbers;
14    DWORD Characteristics;
15 } IMAGE_SECTION_HEADER, *PIMAGE_SECTION_HEADER;
```

IMAGE\_Section\_Header에 대한 정보 출력

```
== Section Table : 7 ==

01 .textbss
    VirtSize      : 0x10000      VirtAddr      : 0x1000
    raw data offs : 0x0          raw data size : 0x0
    relocation offs : 0x0        relocations   : 0x0
    line # offs   : 0x0          line #'s     : 0x0
    characteristics : 0xE00000A0  CODE
    UNINITIALIZED_DATA
    MEM_EXECUTE
    MEM_READ
    MEM_WRITE

02 .text
    VirtSize      : 0x4526       VirtAddr      : 0x11000
    raw data offs : 0x400        raw data size : 0x4600
    relocation offs : 0x0        relocations   : 0x0
    line # offs   : 0x0          line #'s     : 0x0
    characteristics : 0x60000020  CODE
    MEM_EXECUTE
    MEM_READ

03 .rdata
    VirtSize      : 0x2561       VirtAddr      : 0x16000
    raw data offs : 0x4A00        raw data size : 0x2600
    relocation offs : 0x0        relocations   : 0x0
    line # offs   : 0x0          line #'s     : 0x0
    characteristics : 0x40000040  INITIALIZED_DATA
    MEM_READ

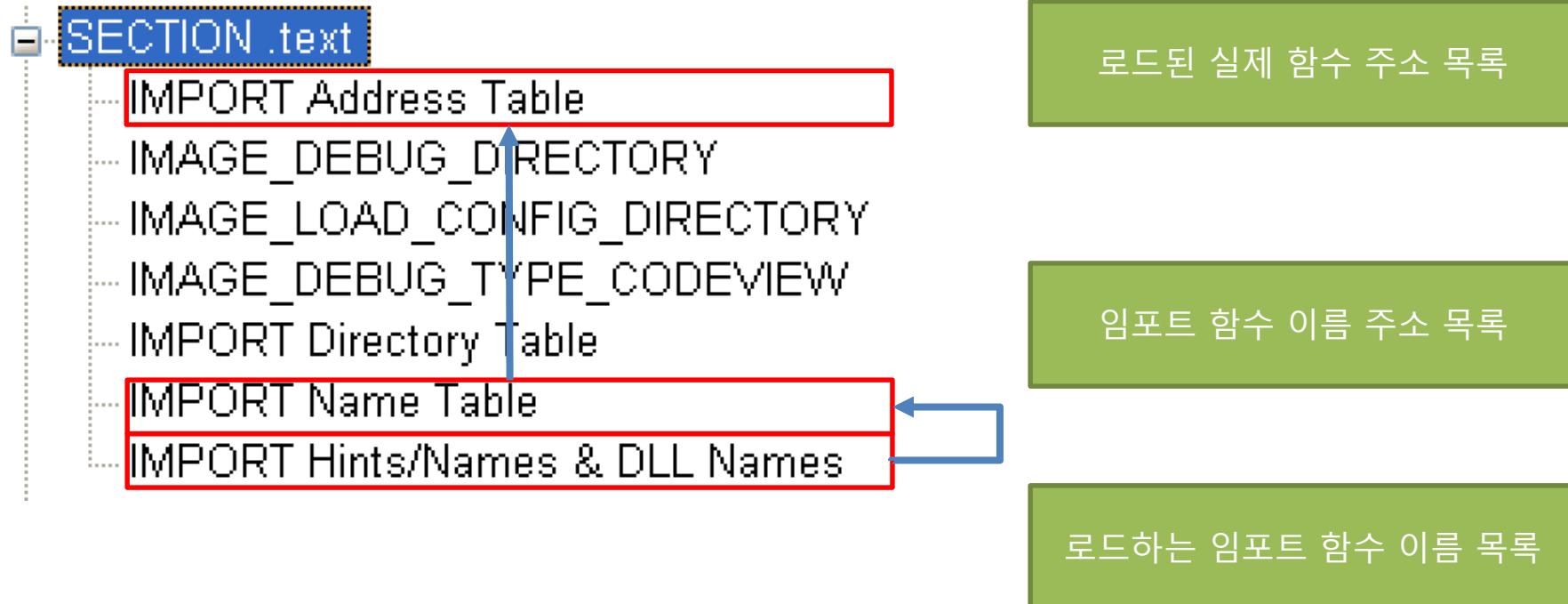
04 .data
    VirtSize      : 0x579        VirtAddr      : 0x19000
    raw data offs : 0x7000        raw data size : 0x200
    relocation offs : 0x0        relocations   : 0x0
```

## 2. 윈도우 실행 파일과 패커



- 03 IMAGE\_Section\_Header

- IAT 호출 구조





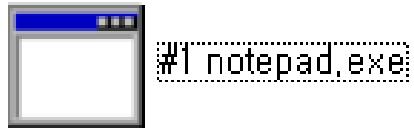
## 2. 윈도우 실행 파일과 패커

- 레퍼런스
  - 리버스 엔지니어링 바이블, 강병탁 저
  - IMAGE\_DATA\_DIRECTORY structure (MSDN)  
[https://msdn.microsoft.com/ko-kr/library/windows/desktop/ms680305\(v=vs.85\).aspx](https://msdn.microsoft.com/ko-kr/library/windows/desktop/ms680305(v=vs.85).aspx)

## 2. 윈도우 실행 파일과 패커



- 실습 문제!
  - PE 파일 복원하기



#1 notepad.exe



#3 notepad.exe  
Notepad  
Microsoft Corporation



#5 notepad.exe  
Notepad  
Microsoft Corporation



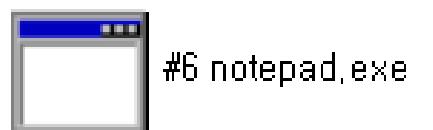
notepad.exe  
Notepad  
Microsoft Corporation



#2 notepad.exe



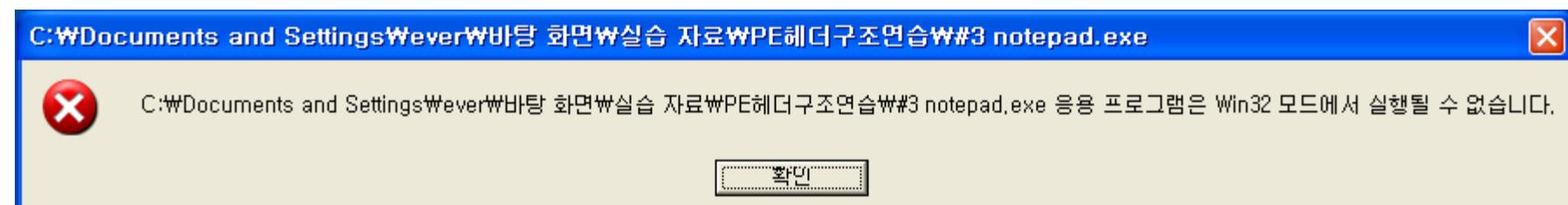
#4 notepad.exe  
Notepad  
Microsoft Corporation



#6 notepad.exe



PEview.exe  
PE/COFF File Viewer  
Wayne J. Radburn



## 2. 윈도우 실행 파일과 패커



### • 01 패커 개요

- 패커(Packer)이란?
  - 실행 파일 압축기
- 사용 목적
  - PE 파일의 크기를 줄이고자 하는 목적
  - PE 파일 내부 코드와 리소스(string, API 등)를 감추기 위한 목적

※ 프로텍터?

- 패킹 기술
- 리버싱을 막기 위한 다양한 기법 추가
- 원본 파일보다 크기가 커질 수 있음

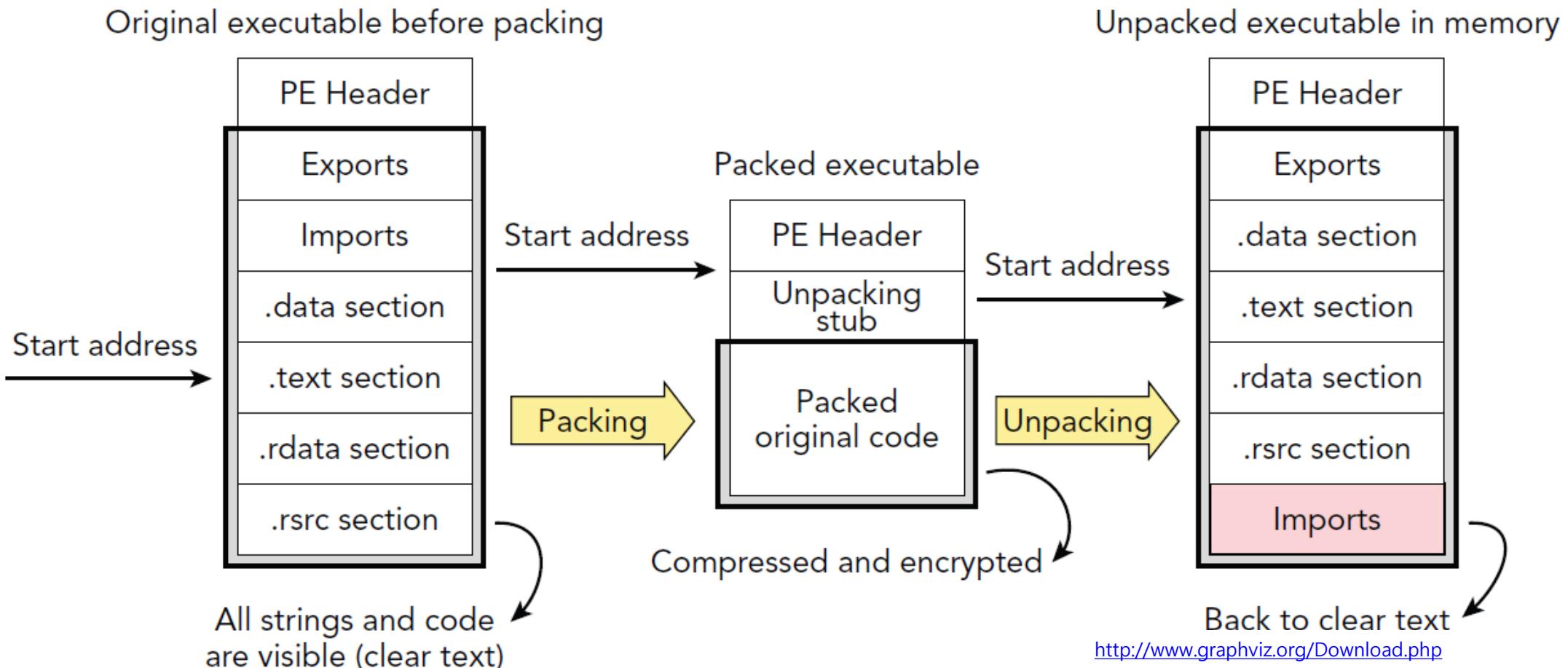


## 2. 윈도우 실행 파일과 패커



### • 01 패커 개요

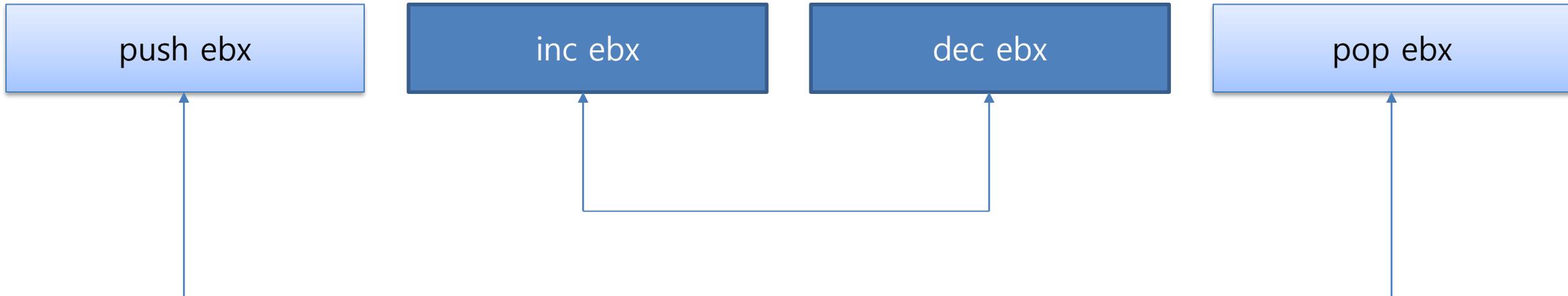
- 메모리 내 파일 패킹 구조



## 2. 윈도우 실행 파일과 패커



- 02 각종 백신 우회 방법
  - 시그니처 변경
    - 단순히 변수 몇 개를 추가 및 함수 위치를 변경하여 리빌드
  - 쓰레기 코드를 통한 우회 방법
    - 네 줄이 수행된 결과를 보면 아무것도 바뀌는 것이 없음 ( 대칭 구조 )



## 2. 윈도우 실행 파일과 패커

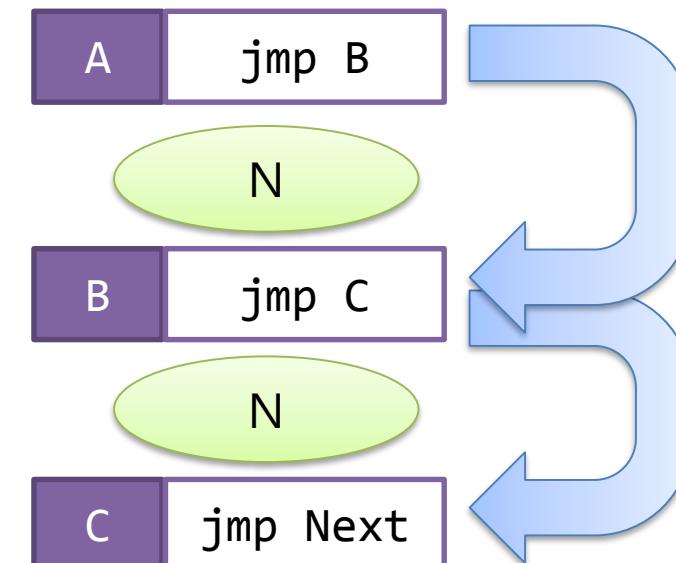


- 02 각종 백신 우회 방법

- 헤더 변조
  - TimeDateStamp, Optional Header의 CheckSum, 메모리 속성 변조
- 대체 가능한 어셈블리어

코드		설명
sub ebp, 7	add ebp, -7	동일하다.
add ebx, eax	adb ebx, eax	물론 CF에 따라 변화는 있다.
call 401184	push 401184 retn	call

- 실행조차 되지 않는 코드



## 2. 윈도우 실행 파일과 패커



- 03 Themida

- 지상 최고의 패커라 불림
- 안티 디버깅과 안티 분석, 언패킹 분석을 매우 어렵게 하는 안전한 패커
- VMware, 디버거, 프로세스 모니터(ProcMon, Process Monitor) 분석을 방지하는 기능
- 패킹된 실행 파일은 이례적으로 크기가 큼
- 언패커가 존재하나 Themida 버전과 프로그램을 패킹할 때 사용한 설정에 따라 성공률이 다름





## 2. 윈도우 실행 파일과 패커

### • 04 UPX 실습

```
C:\>"C:\Documents and Settings\Administrator\바탕 화면\upx391w\upx391w\upx.exe"
          Ultimate Packer for executables
          Copyright (C) 1996 - 2013
UPX 3.91w      Markus Oberhumer, Laszlo Molnar & John Reiser   Sep 30th 2013

Usage: upx [-123456789dlthVL] [-qvfk] [-o file] file..

Commands:
  -i      compress faster           -9      compress better
  -d      decompress               -l      list compressed file
  -t      test compressed file     -U      display version number
  -h      give more help           -L      display software license

Options:
  -q      be quiet                 -v      be verbose
  -oFILE write output to 'FILE'
  -f      force compression of suspicious files
  -k      keep backup files
file...  executables to (de)compress

Type 'upx --help' for more detailed help.

UPX comes with ABSOLUTELY NO WARRANTY; for details visit http://upx.sf.net
```

## 2. 윈도우 실행 파일과 패커



- Lena 34

- 01 The Goals of Registry Machine
  - 안티디버깅 기법 우회
  - 인증 우회
  - 로더를 이용한 패치

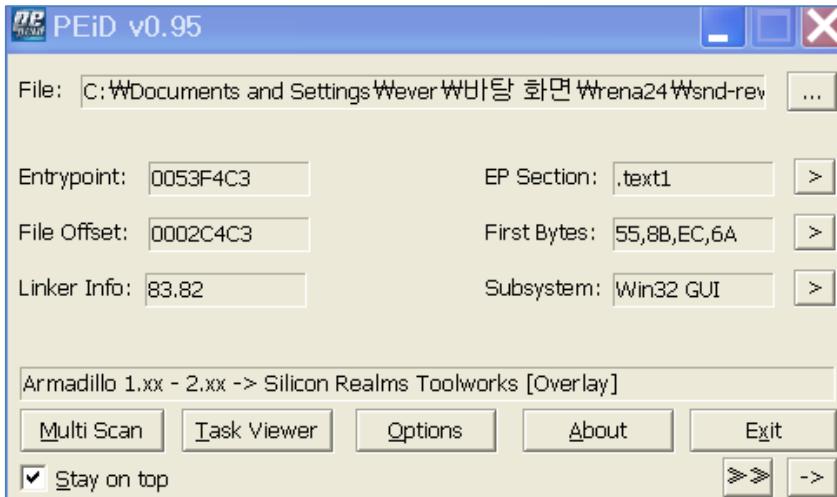


## 2. 윈도우 실행 파일과 패커

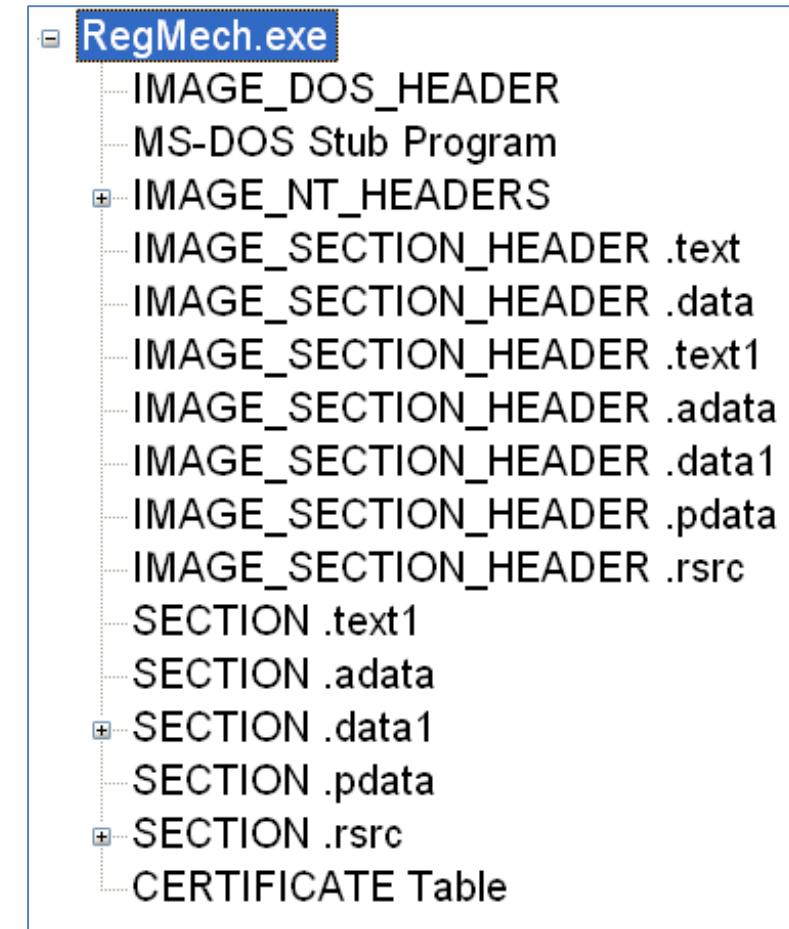


- 02 프로그램 패킹 여부 확인

- PEID



- PEView



## 2. 윈도우 실행 파일과 패커

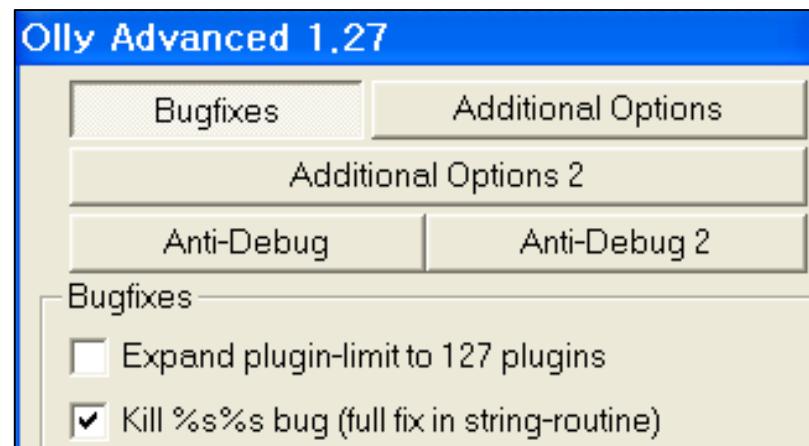


### • 03 안티디버깅 기법

- 올리디버거 버그를 사용한 OutputDebugStringA

01151ED4	50	PUSH EAX	
01151ED5	FF15 78221601	CALL DWORD PTR DS:[1162278]	kernel32.OutputDebugStringA
01151EDB	A1 D81E1701	MOU EAX,DWORD PTR DS:[1171ED8]	
01151EE0	8B48 68	MOU ECX,DWORD PTR DS:[EAX+68]	
01151EE3	3348 64	XOR ECX,DWORD PTR DS:[EAX+64]	
01151EE6	3348 5C	XOR ECX,DWORD PTR DS:[EAX+5C]	
01151EE9	F7C1 00008000	TEST ECX,800000	
01151EEF	v 74 05	JE SHORT 01151EF6	
01151EF1	E8 FCD20000	CALL 0115F1F2	
01151EF6	8B45 08	MOU EAX,DWORD PTR SS:[EBP+8]	
01151EF9	8B40 0C	MOU EAX,DWORD PTR DS:[EAX+C]	
01151EFC	8945 CC	MOU DWORD PTR SS:[EBP-34],EAX	
01151EFF	8D45 CC	LEA EAX,DWORD PTR SS:[EBP-34]	
EAX=0013D60C, (ASCII "%s")			

- Olly Advanced 플러그인을 사용하여 우회



## 2. 윈도우 실행 파일과 패커

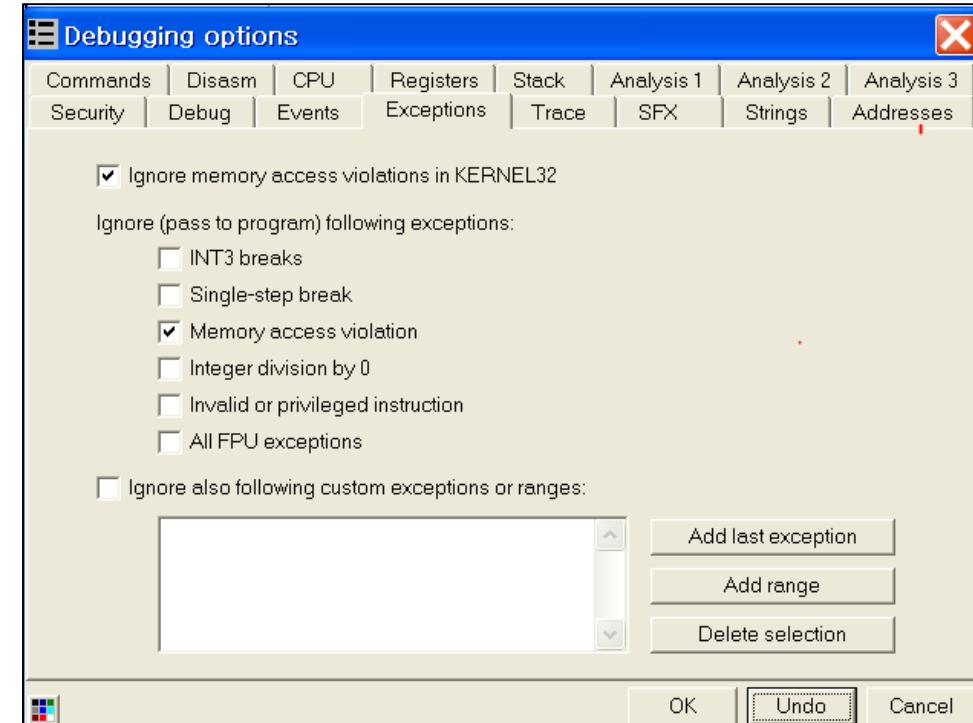


### • 03 안티디버깅 기법

#### - 예외 처리를 이용한 안티디버깅 기법

0115C148 33C0 XOR EAX,EAX  
0115C14A 8900 MOU DWORD PTR DS:[EAX],EAX  
0115C14C 90 NOP  
0115C14D v E9 57010000 JMP 0115C2A9  
0115C152 FF75 EC PUSH DWORD PTR SS:[EBP-14]  
0115C155 E8 36FEFFFF CALL 0115BF90  
0115C15A 59 POP ECX  
0115C15B C3 RETN  
0115C15C 8B65 E8 MOU ESP,DWORD PTR SS:[EBP-18]  
0115C15F v 70 07 JO SHORT 0115C168  
0115C161 v 7C 03 JL SHORT 0115C166  
0115C163 v EB 05 JMP SHORT 0115C16A  
0115C165 E8 74FBEBF9 CALL FB01BCDE  
0115C16A A1 A4231701 MOU EAX,DWORD PTR DS:[11723A4]  
0115C16F 85C0 TEST EAX,EAX  
0115C171 v 0F84 0C010000 JE 0115C283  
0115C177 8B50 04 MOU EDX,DWORD PTR DS:[EAX+4]  
0115C17A 8B00 50241701 MOU ECX,DWORD PTR DS:[1172450]  
0115C180 3BD1 CMP EDX,ECX  
0115C182 8B1D 54241701 MOU EBX,DWORD PTR DS:[1172454]  
0115C188 v 72 04 JB SHORT 0115C18E  
0115C18A 3BD3 CMP EDX,EBX  
0115C18C v 72 55 JB SHORT 0115C1E3  
0115C18E 8B3D 58241701 MOU EDI,DWORD PTR DS:[1172458]  
0115C194 3BD7 CMP EDX,EDI  
0115C196 8B35 5C241701 MOU ESI,DWORD PTR DS:[117245C1]  
  
EAX=00000000  
DS: [00000000]-???:  
Address Hex dump ASCII  
00964000 5F 6E E2 77 4B EA E2 77 32 86 E2 77 C2 B7 E2 77 \_n?K侑w2?n?W?z?  
00964010 DA B5 E2 77 FA 6B E2 77 79 6F E2 77 70 5B E2 77 濶???yo?p[?  
00964020 E0 5F E2 77 13 AD E2 77 00 00 00 00 12 FF 80 7C ??■?w....■ |  
00964030 A9 FF 80 7C BD FD 80 7C 2E 93 80 7C 64 A1 80 7C ? |?| .|.d|?  
00964040 91 9E 80 7C C1 60 83 7C FC 00 81 7C 31 B7 80 7C 韻 |?||?|?  
00964050 BF FC 80 7C 93 C1 85 7C AB 0B 83 7C B6 C3 82 7C 韵 |?|?  
00964060 AD 20 87 7C 82 4B 81 7C D4 1A 80 7C E1 9A 80 7C ???.? |?|  
00964070 30 AE 80 7C 01 FE 93 7C 7B 1D 80 7C 10 FE 93 7C 0?|?|(|?|?  
00964080 98 C1 80 7C 7B 99 80 7C 6B 23 80 7C AD 2F 81 7C ស |?|k# |?  
00964090 F2 1E 80 7C 90 34 83 7C B7 24 80 7C 30 25 80 7C ? |?|? |0% |  
  
Command :  
Access violation when writing to [00000000] - use Shift+F7/F8/F9 to pass exception to program

#### - 디버깅 옵션에서 Memory Access Violation 체크

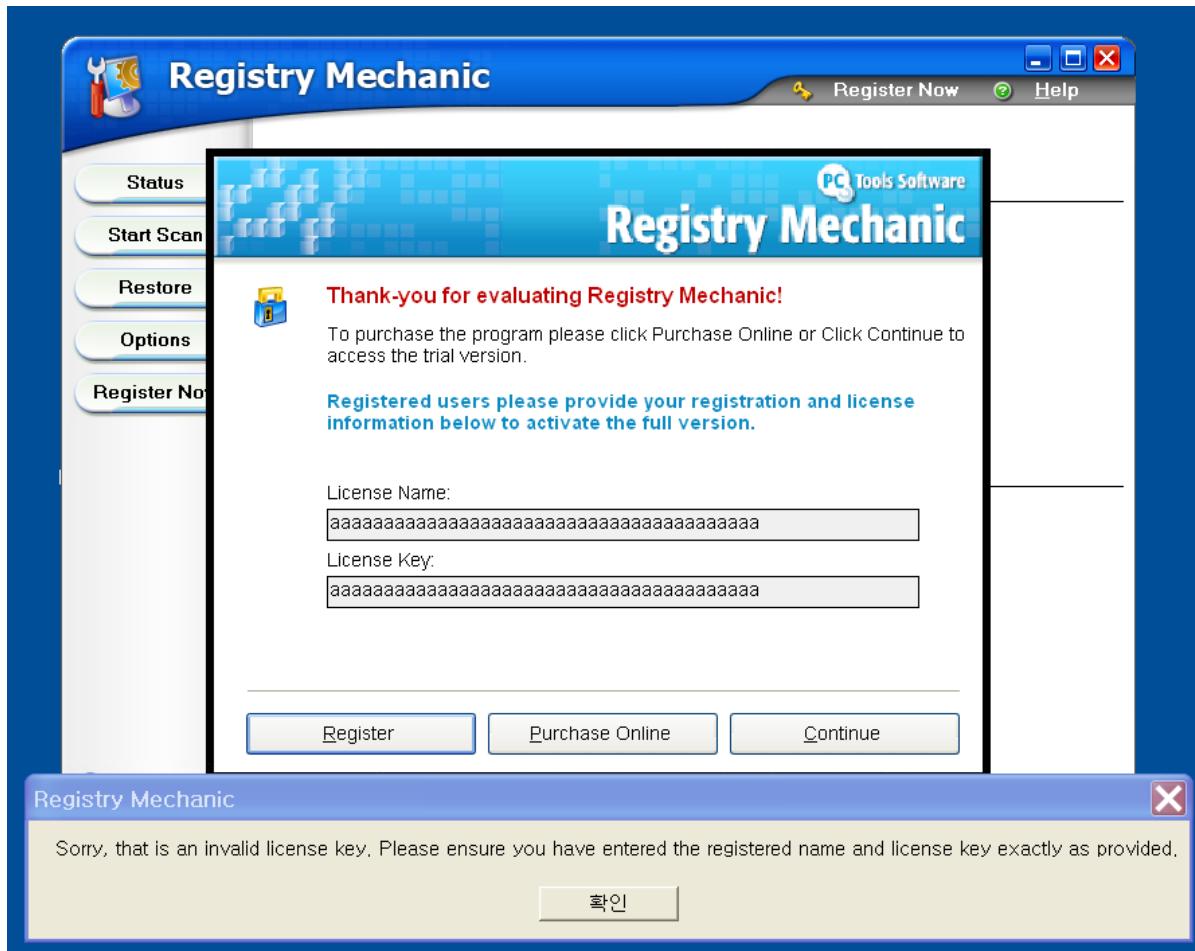


## 2. 윈도우 실행 파일과 패커



### • 04 레지스트리 등록 우회하기

- Alert Window (디버거에서 F12를 눌러 정지)



- Call Stack 확인하기 후 비교문에 HWBP

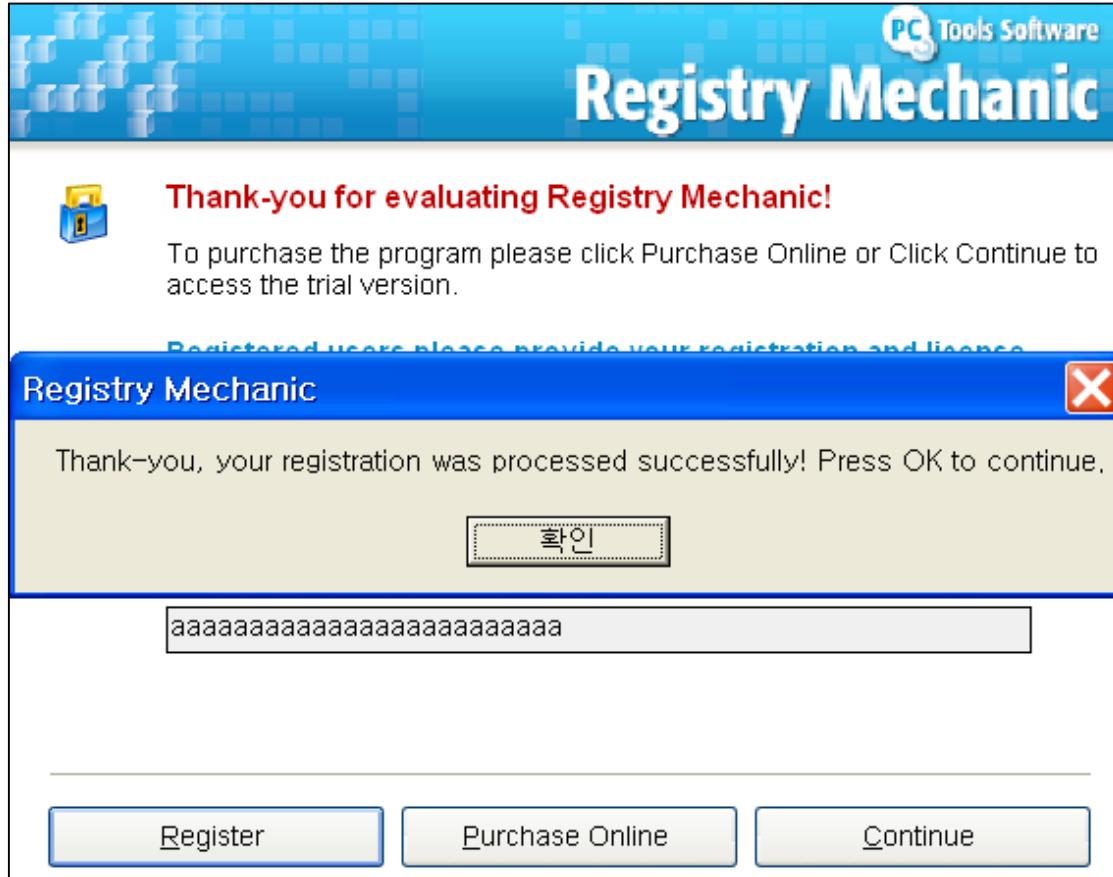
Address	Stack	Procedure	Called from	Frame
0012B400	77CF9418	Includes ntdll.KiFastSystemCallRet	USER32.77CF9416	0012B434
0012B404	77D0770A	USER32.WaitMessage	USER32.77D07705	0012B434
0012B438	77D049C4	USER32.77D0757B	USER32.77D049BF	0012B434
0012B460	77D1A956	USER32.77D0490E	USER32.77D1A951	0012B45C
0012B720	77D1A2BC	USER32.SoftModalMessageBox	USER32.77D1A2B7	0012B71C
0012B870	77D1A10B	USER32.77D1A147	USER32.77D1A106	0012B86C
0012B8DC	733DF6B2	Includes USER32.77D1A10B	MSUBUM60.733DF6B0	0012B8D8
0012B91C	733DF52E	Includes MSUBUM60.733DF6B2	MSUBUM60.733DF52B	0012B918
0012B944	733DF829	MSUBUM60.733DF414	MSUBUM60.733DF824	0012B940
0012B974	733D3BF0	MSUBUM60.733DF798	MSUBUM60.733D3BEB	0012B970
0012B9D8	7344D07A	MSUBUM60.733D3963	MSUBUM60.7344D075	0012B9D4
0012BA50	0088BD5B	? MSUBUM60.rtcMsgBox	RegMech.0088BD55	0012BA4C

```
0088BCEC MOVSX EDX,WORD PTR SS:[EBP-F4]
0088BCF3 TEST EDX,EDX
0088BCF5 JE RegMech.0088BD7C
0088BCFB MOU DWORD PTR SS:[EBP-4],9
0088BD02 MOU DWORD PTR SS:[EBP-70],80020004
0088BD09 MOU DWORD PTR SS:[EBP-78],0A
0088BD10 MOU DWORD PTR SS:[EBP-60],80020004
0088BD17 MOU DWORD PTR SS:[EBP-68],0A
0088BD1E MOU DWORD PTR SS:[EBP-50],80020004
0088BD25 MOU DWORD PTR SS:[EBP-58],0A
0088BD2C MOU DWORD PTR SS:[EBP-A0],RegMech.00906
0088BD36 MOU DWORD PTR SS:[EBP-A8],4008
0088BD40 LEA EAX,DWORD PTR SS:[EBP-78]
0088BD43 PUSH EAX
0088BD44 LEA ECX,DWORD PTR SS:[EBP-68]
0088BD47 PUSH ECX
0088BD48 LEA EDX,DWORD PTR SS:[EBP-58]
0088BD4B PUSH EDX
0088BD4C PUSH 0
0088BD4E LEA EAX,DWORD PTR SS:[EBP-A8]
0088BD54 PUSH EAX
0088BD55 CALL DWORD PTR DS:[401118] MSUBUM60.rtcMsgBox
```



## 2. 윈도우 실행 파일과 패커

- 04 레지스트리 등록 우회하기
  - 재시도 하면 인증 성공



## 2. 윈도우 실행 파일과 패커



- 04 레지스트리 등록 우회하기
  - 언패킹 이후 패칭 자동화 - RISC's Process Patcher 사용

The screenshot shows the RISC's Process Patcher interface. On the left, a Notepad window titled "RegMeca\_loader.rpp - 메모장" displays a registry patch script:

```
T=50000:  
F=RegMech.exe:  
O=loader.exe:  
R:  
P=88BCF5/0F,84,81,00,00,00/E9,82,00,00,00,90:  
$
```

A large blue arrow points from the Notepad to a folder containing two files: "RegMech.exe" and "loader.exe".

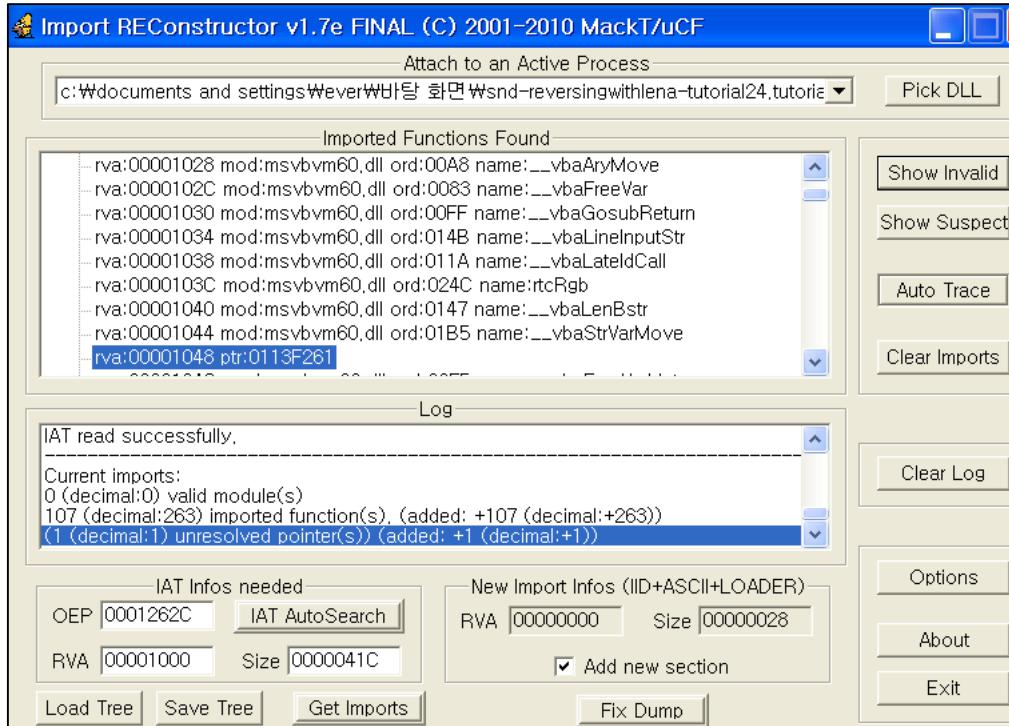
옵션	설명
T	패치할 타이밍(단위 : ms)
F	패치 대상 바이너리
O	생성될 로더 바이너리 이름
R	쓰레드 재시작
P	패치 진행 주소 / 원본 명령어 / 변경하고자 하는 명령어

## 2. 윈도우 실행 파일과 패커



### • 04 레지스트리 등록 우회하기

- 언패킹 이후 패칭 자동화 - ImportREC



## 2. 윈도우 실행 파일과 패커



### • 04 레지스트리 등록 우회하기

- 해당 프로그램이 **MSVBVM60.DLL**을 사용하는 것으로 보아 VB로 제작한 프로그램임을 추측!
- VB로 제작된 프로그램의 EP는 “VB5!”로 시작하는 문자열이 존재

**M Memory map**

Address	Size	Owner	Section	Contains	Type	Access	Initial	Mapped as
00010000	00001000				Priv	RW	RW	
00020000	00001000				Priv	RW	RW	
00030000	00008000				Priv	RW	RW	
00120000	00001000				Priv	RW	Guar	RW
00121000	0001F000			stack of ma	Priv	RW	Guar	RW
00140000	00003000				Map	R	R	

**D Dump – RegMech:.text 00401000..00905FFF**

Enter binary string to search for: VB5!

ASCII: VB5!

UNICODE:

HEX+04: 56 42 35 21

Entire block     Case sensitive

OK Cancel

The screenshot shows the Immunity Debugger interface. On the left, there's a memory dump window titled 'Dump – RegMech:.text 00401000..00905FFF' containing a search dialog for 'VB5!'. The search results show the string 'VB5!' at address 0041361C. On the right, the CPU register pane shows assembly code with several JMP instructions highlighted in yellow.

**\* - [CPU - main thread, module RegMech]**

C File View Debug Plugins Options Window Help

Registers: L E M T W H C / K B R ... S

004125B8	- FF25 58114000	JMP DWORD PTR DS:[401158]
004125BE	- FF25 74124000	JMP DWORD PTR DS:[401274]
004125C4	- FF25 84124000	JMP DWORD PTR DS:[401284]
004125CA	- FF25 A4124000	JMP DWORD PTR DS:[4012A4]
004125D0	- FF25 08114000	JMP DWORD PTR DS:[401108]
004125D6	- FF25 40114000	JMP DWORD PTR DS:[401140]
004125DC	- FF25 10114000	JMP DWORD PTR DS:[401110]
004125E2	- FF25 50114000	JMP DWORD PTR DS:[401150]
004125E8	- FF25 C0124000	JMP DWORD PTR DS:[4012C0]
004125EE	- FF25 78134000	JMP DWORD PTR DS:[401378]
004125F4	- FF25 88134000	JMP DWORD PTR DS:[401388]
004125FA	- FF25 1C124000	JMP DWORD PTR DS:[40121C]
00412600	- FF25 D8104000	JMP DWORD PTR DS:[4010D8]
00412606	- FF25 00104000	JMP DWORD PTR DS:[401000]
0041260C	- FF25 5C104000	JMP DWORD PTR DS:[40105C]
00412612	- FF25 38114000	JMP DWORD PTR DS:[401138]
00412618	- FF25 4C124000	JMP DWORD PTR DS:[40124C]
0041261E	- FF25 64124000	JMP DWORD PTR DS:[401264]
00412624	- FF25 58134000	JMP DWORD PTR DS:[401358]
0041262A	0000	ADD BYTE PTR DS:[EAX], AL
0041262C	68 1C364100	PUSH RegMech.0041361C
00412631	E8 EFFFFFFF	CALL RegMech.00412624
0041263C	0000	ADD BYTE PTR DS:[EAX], AL

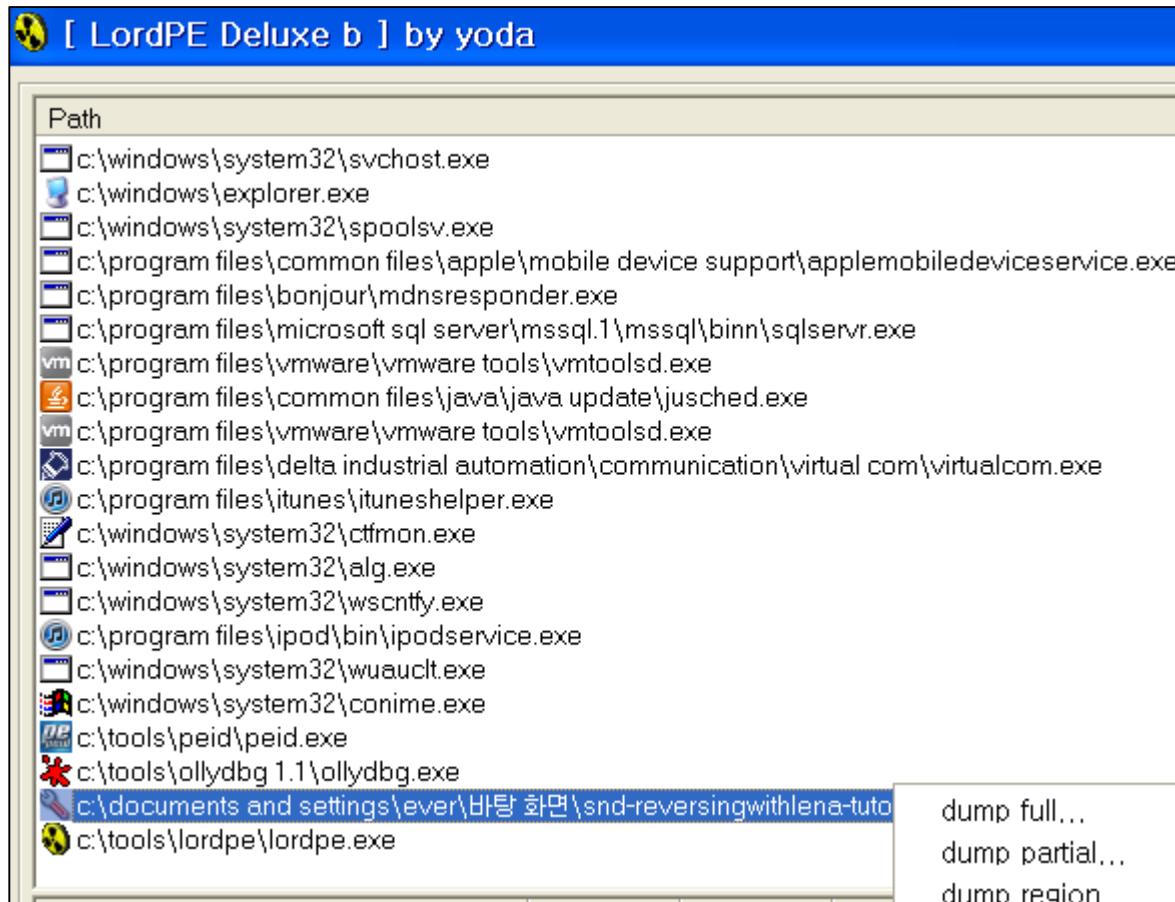
## 2. 윈도우 실행 파일과 패커



- 04 레지스트리 등록 우회하기

- 언패킹 이후 패칭 자동화 – ImportREC

- LordPE를 사용하여 덤프 – 마우스 오른쪽키 > Dump full...

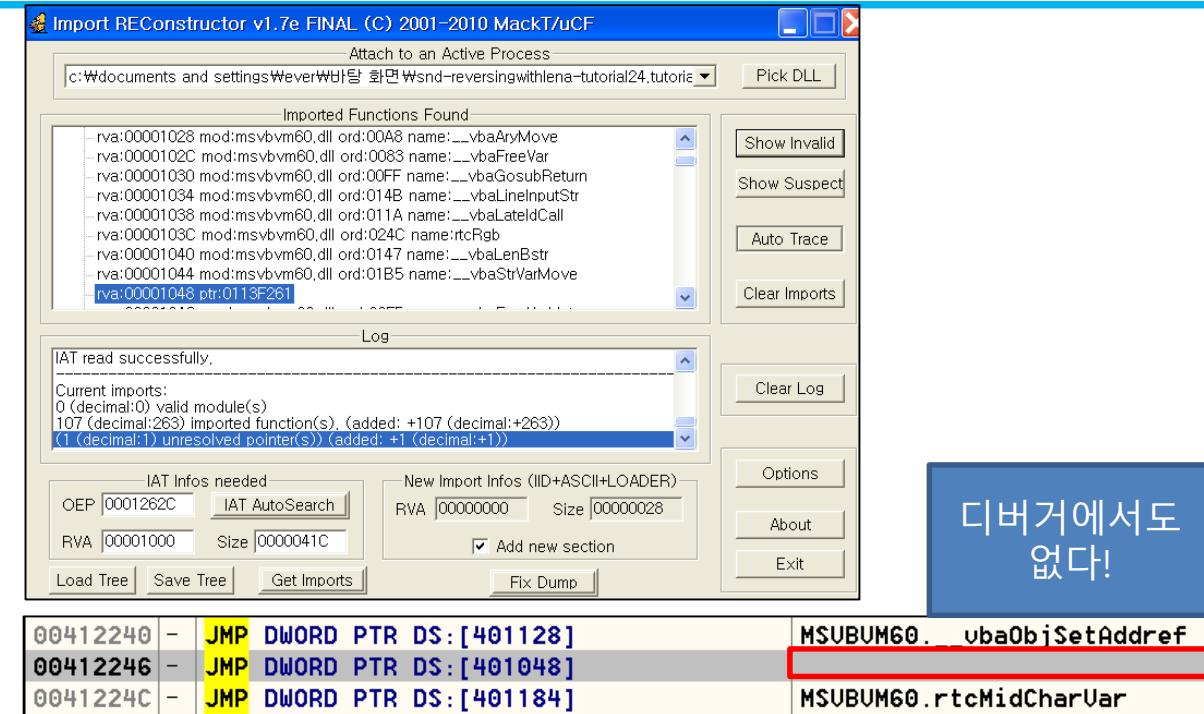
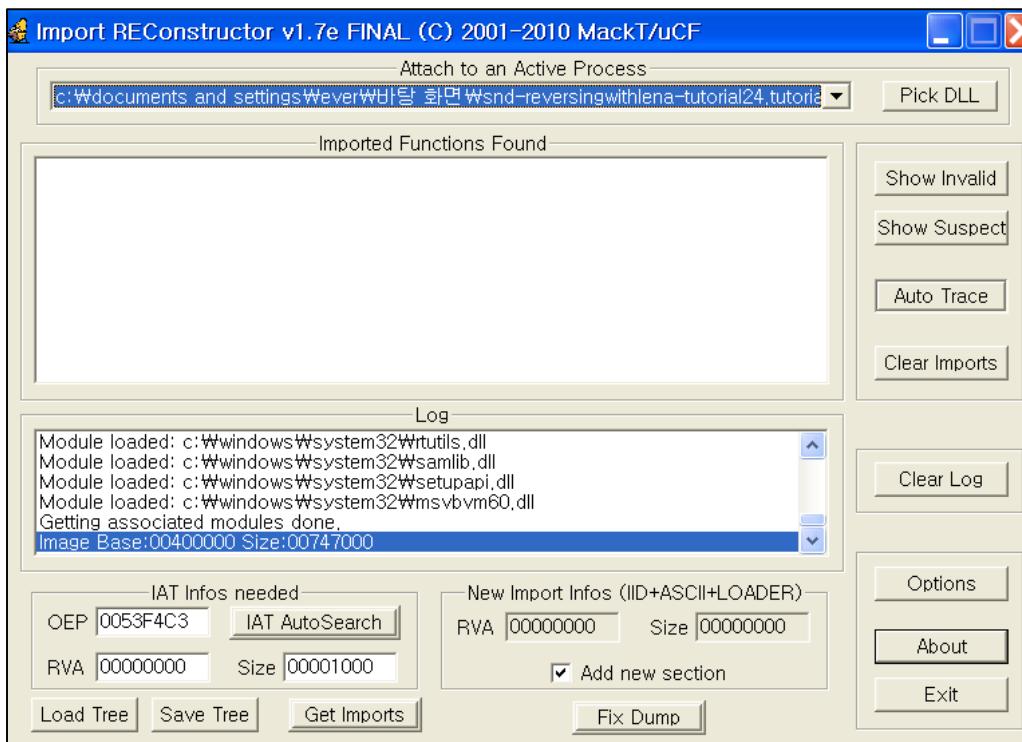


## 2. 윈도우 실행 파일과 패커



### • 04 레지스트리 등록 우회하기

- 언패킹 이후 패칭 자동화 – ImportREC
  - ImportREC – IAT 자동 찾기



## 2. 윈도우 실행 파일과 패커



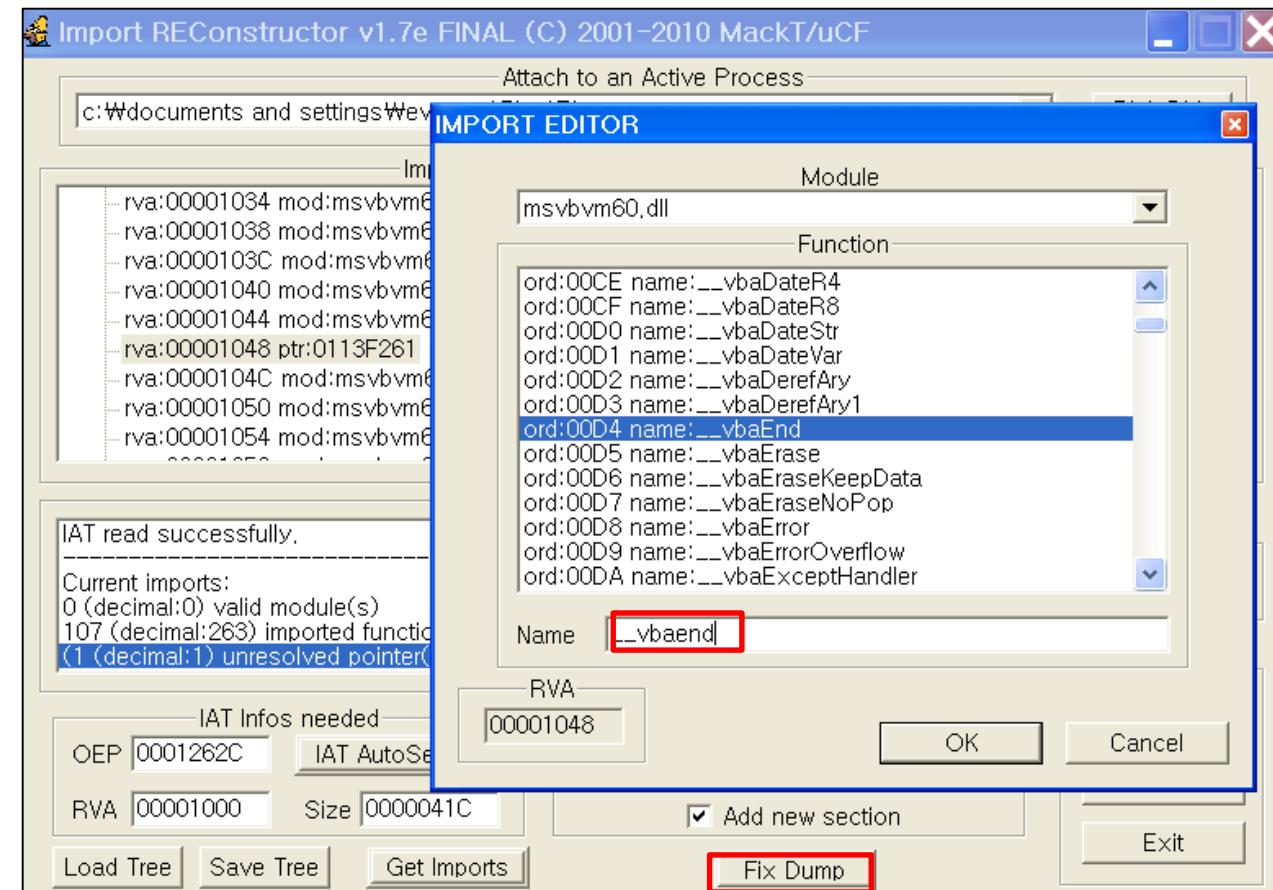
### • 04 레지스트리 등록 우회하기

- 언패킹 이후 패칭 자동화 – ImportREC
  - "\_vbaend"를 입력하고 Fix Dump 클릭

```
0113F261 PUSH EBP
0113F262 MOU EBP,ESP
0113F264 PUSH -1
0113F266 PUSH 11626D8
0113F26B PUSH 1160D50
0113F270 MOU EAX,DWORD PTR FS:[0]
0113F276 PUSH EAX
0113F277 MOU DWORD PTR FS:[0],ESP
0113F27E SUB ESP,10
0113F281 PUSH EBX
0113F282 PUSH ESI
0113F283 PUSH EDI
0113F284 MOU DWORD PTR SS:[EBP-18],ESP
0113F287 XOR ESI,ESI
0113F289 PUSH 1167464
0113F28E CALL DWORD PTR DS:[11620E8]
0113F294 TEST EAX,EAX
0113F29C IF SHORT 0113F2A0
```

ASCII "MSUBUM60.DLL"

kernel32.GetModuleHandleA





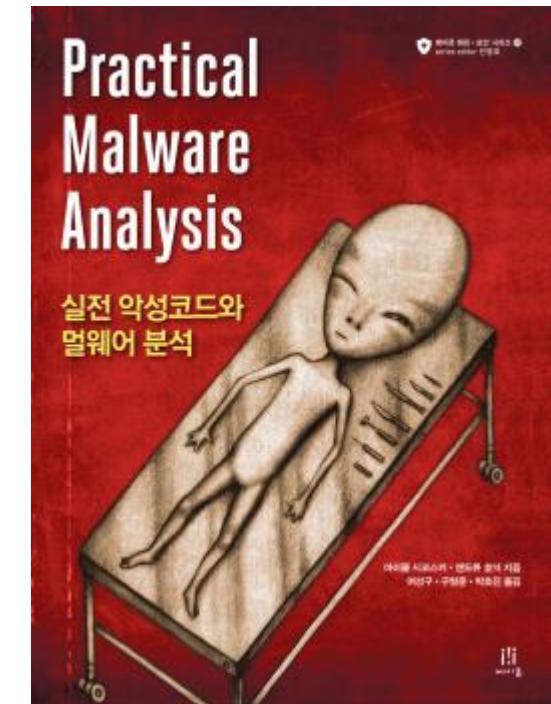
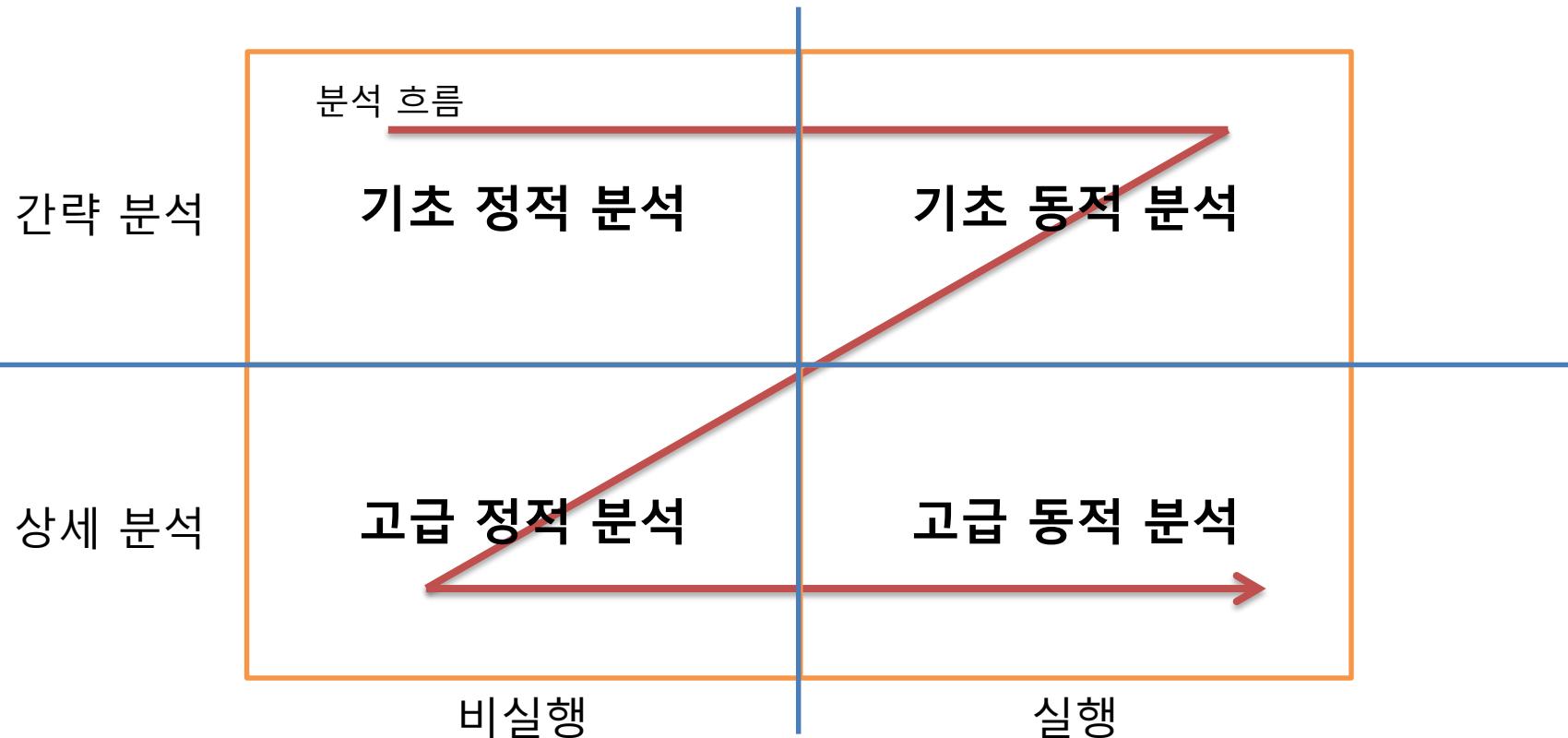
### 3. 악성코드 기초 분석

### 3. 악성코드 기초 분석



#### • 01 기초 정적 분석

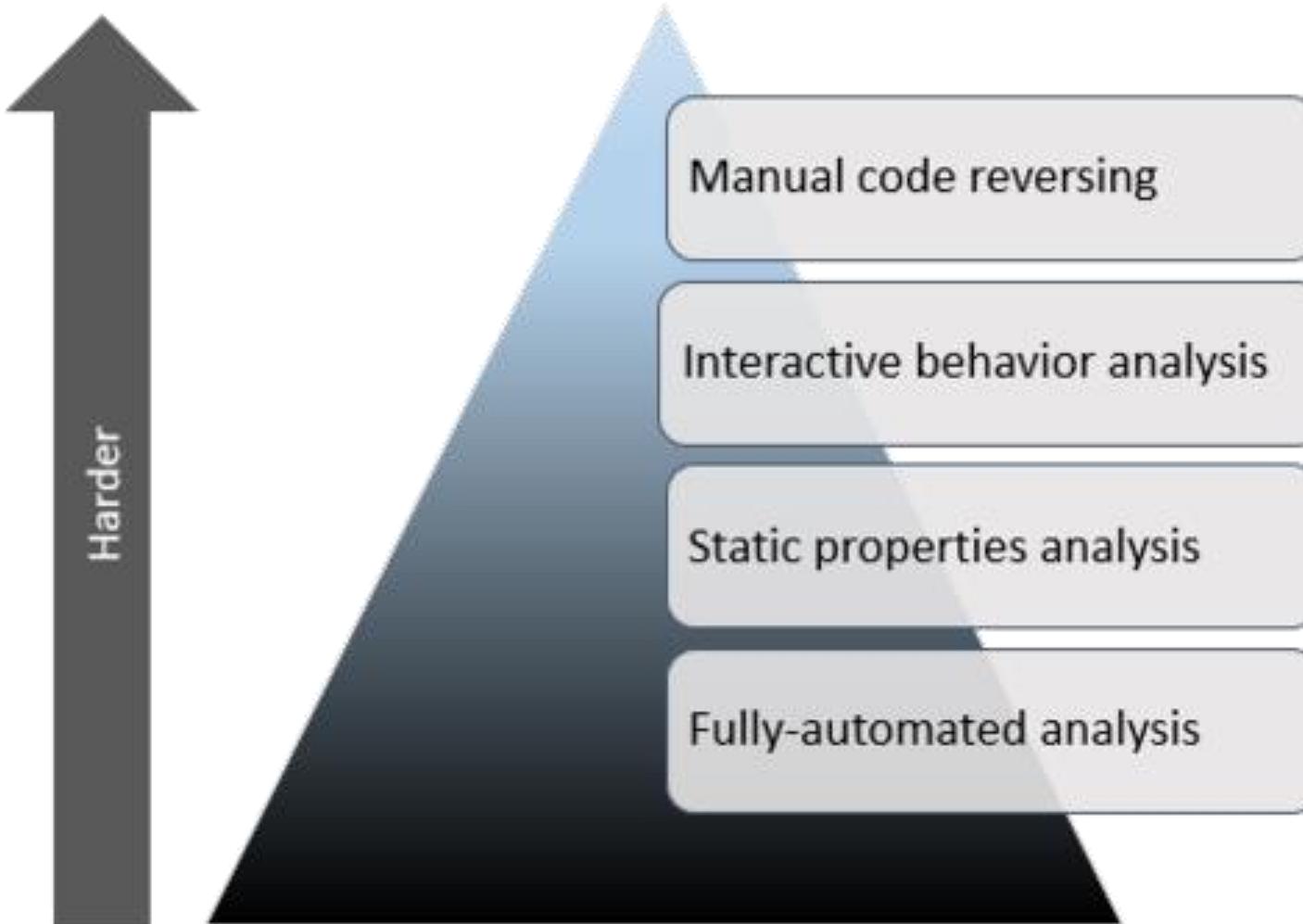
- 실전 악성코드와 멀웨어 분석에서 소개하는 악성코드 방법 4가지





### 3. 악성코드 기초 분석

- 악성코드 분석의 네 가지 접근 방법





### 3. 악성코드 기초 분석

- 악성코드 분석의 네 가지 접근 방법

분석 방법	설명
완전 자동화 분석 (Fully-automated analysis)	<ul style="list-style-type: none"><li>• 정적 분석 및 동적 분석을 통해 악의적인 행위를 판단</li><li>• 자동 분석 수행 (파일 생성, 수정 과정의 분석, 레지스트리 분석, 네트워크 분석 등)</li><li>• 전문가에 의해 분석되는 만큼 상세하거나 정확하지 않을 수 있음</li><li>• 악성코드 분석 제공 서비스</li></ul>
정적 속성 분석 (Static properties analysis)	<ul style="list-style-type: none"><li>• 악성코드의 추가 분석을 위해 필요한 단계</li><li>• 문자열 헤더 정보, 해시 값, 리소스 정보, 패킹 여부 등 신속하게 정보 획득</li><li>• 정보들을 활용해 실행 파일 간의 비교 데이터베이스를 구성</li><li>• 바이러스 토탈(VirusTotal) 서비스</li></ul>
대화형 동적 분석 (Interactive Behavior Analysis)	<ul style="list-style-type: none"><li>• 레지스트리, 파일시스템, 프로세스, 네트워크 활동을 이해하기 위해 분리된 가상 머신 환경에서 실행하며 분석</li><li>• 메모리 분석을 통해 다른 행위를 추가적으로 분석</li><li>• 악의적인 행위의 상세한 과정들을 확인</li><li>• 분석가들의 분석 시간이 많이 소요</li></ul>
수동 코드 역공학 분석 (Manual Code Reversing)	<ul style="list-style-type: none"><li>• 위 과정이 완료된 후에 추가적인 정보를 획득하기 위해 분석하는 행위</li><li>• 수동 코드 역공한 분석이 필요한 예<ul style="list-style-type: none"><li>- 특정 루틴에 난독화가 되어서 복호화가 이루어지는 부분을 더 분석해 추가적인 정보를 획득</li><li>- 악의적인 도메인 이름 생성 과정의 알고리즘 분석</li><li>- 행동 분석 과정에서 자신을 숨기고 보여주지 않았던 부분으로 발생되는 다른 기능 이해</li></ul></li></ul>



### 3. 악성코드 기초 분석

### 3. 악성코드 기초 분석



#### • 01 기초 정적 분석

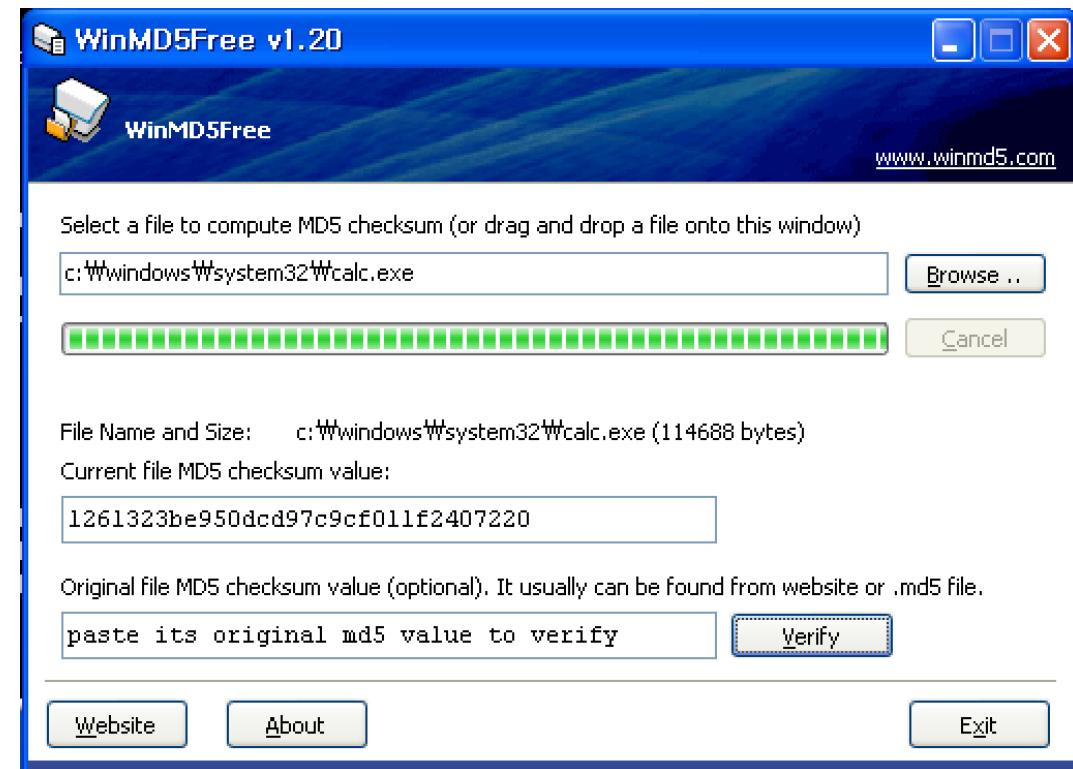
- 기초 정적 분석이란?
  - 악성코드 분석 시 가장 먼저 진행  
(악성코드 연구의 시작 단계)
  - 프로그램의 기능을 파악하기 위해 코드나 프로그램의 구조를 분석  
(프로그램을 실행시키지 않음)
- 분석방법
  - 악성 여부를 판단하는 안티바이러스 도구 사용
  - 악성코드 판별 해시 사용
  - 파일의 문자열, 함수, 헤더에서 개략적인

### 3. 악성코드 기초 분석



#### • 02 기초 정적 분석 도구

- 악성코드의 지문(Finger Print) 확인
- 해시를 이름으로 사용
- 악성코드 식별을 위해 해시 공유
- 식별 여부를 위해 해시 검색



### 3. 악성코드 기초 분석



#### • 02 기초 정적 분석 도구

- 문자열 검색
- 프로그램 메시지 출력
- URL 접속
- 특정 위치로 파일 복사

```
C:\WINDOWS\system32\cmd.exe
GPADDINGXXPADDINGPADDINGXXXPADDINGPADDINGX
C:\Documents and Settings\ever>"C:\Documents and Settings\ever\바탕 화면\기초 정
적 분석 도구\strings.exe" c:\Windows\system32\calc.exe | more

Strings v2.51
Copyright (C) 1999-2013 Mark Russinovich
Sysinternals - www.sysinternals.com

!This program cannot be run in DOS mode.

$xF
$xF
$xF9
$xF
$xF
$xF
$xF
$y?D$x?9
$x?T
$x?9
$x?9
$x?Rich
$x?
.text
` .data
```

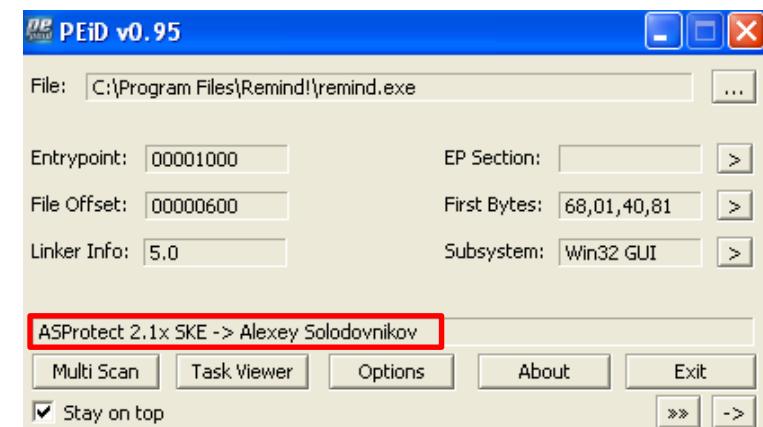
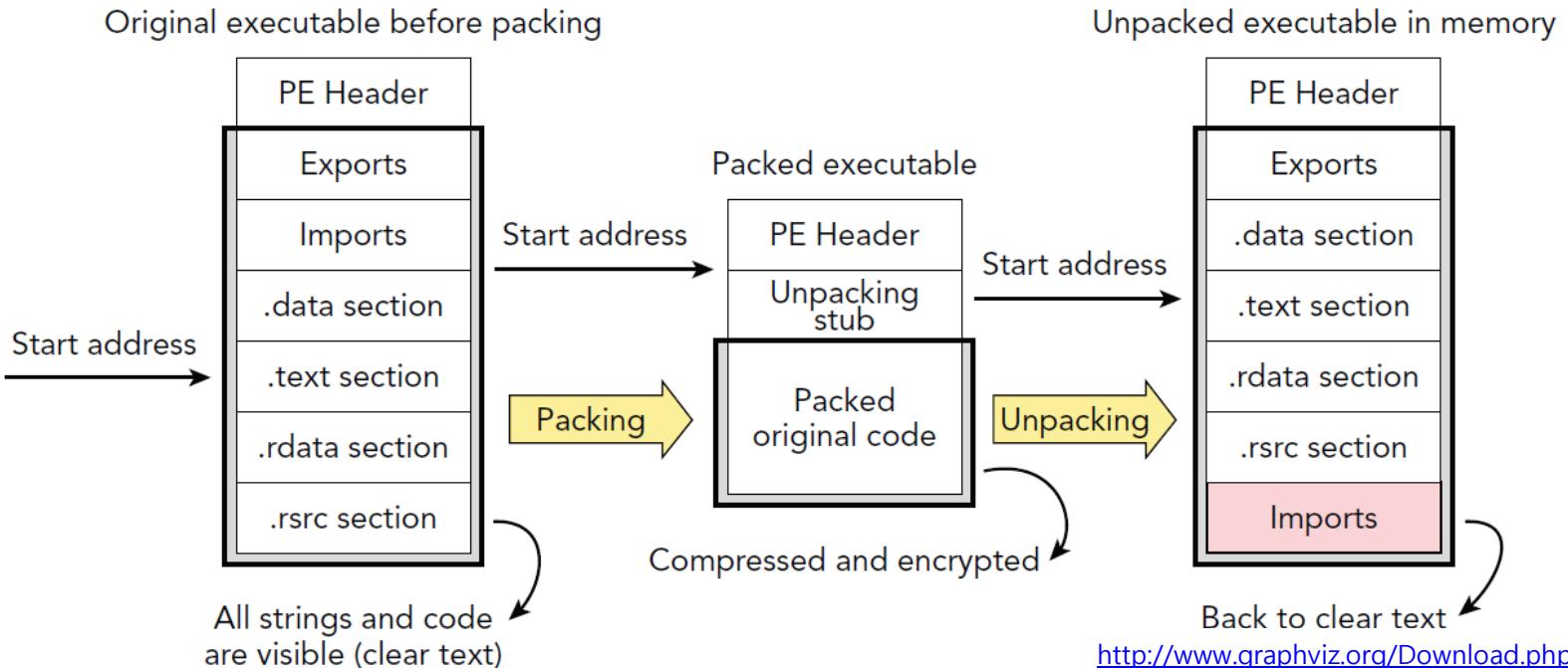
### 3. 악성코드 기초 분석



#### • 02 기초 정적 분석 도구

##### - 패킹(Packing) 확인

- 패킹의 여부를 확인하고 언패킹하여 리버싱을 보다 쉽게 하기 위한 단계

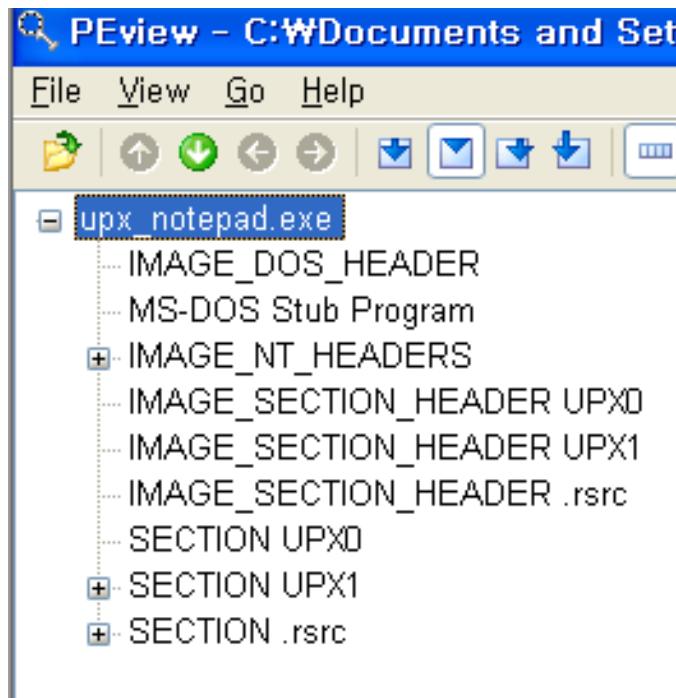




### 3. 악성코드 기초 분석

#### • 02 기초 정적 분석 도구

- PE 세부 구조를 확인
- 패킹 징후?
  - Size of Raw Data(원래 데이터 크기)보다 Virtual Size(가상 크기)가 월등히 크다면 다른 파일을 올리기 위함으로 의심해 볼 수 있다.
  - 섹션의 이름이 변경됨



pView	Data	Description	Value
00000000	55 50 58 30	Name	UPX0
00000004	00 00 00 00		
00000008	00010000	Virtual Size	
0000000C	00001000	RVA	
00000010	00000000	Size of Raw Data	
00000014	00000400	Pointer to Raw Data	
00000018	00000000	Pointer to Relocations	
0000001C	00000000	Pointer to Line Numbers	
00000020	0000	Number of Relocations	
00000022	0000	Number of Line Numbers	
00000024	E0000080	Characteristics	
	00000080		IMAGE_SCN_CNT_UNINITIALIZED_DATA
	20000000		IMAGE_SCN_MEM_EXECUTE
	40000000		IMAGE_SCN_MEM_READ
	80000000		IMAGE_SCN_MEM_WRITE

### 3. 악성코드 기초 분석



#### • 02 기초 정적 분석 도구

##### - DLL 의존성 조사

- Dependency Walker
  - Kernel32.dll : 메모리, 파일, 하드웨어 접근과 조작
  - Advapi32.dll : 서비스 관리자, 레지스트리 같은 추가 윈도우 핵심 컴포넌트
  - User32.dll : 유저 인터페이스 (버튼, 스크롤바, 사용자 행위 제어, 반응 컴포넌트)
  - Gdi32.dll : 그래픽 보기 및 조작
  - Ntdll.dll : 윈도우 커널 인터페이스
  - WS2\_32.dll : 윈도우 소켓 네트워크
  - Wininet.dll : FTP, HTTP, NTP와 같은 상위 수준 프로토콜 구현

Module	Time Stamp	Size	Attributes	Machine	Subsystem	Debug	Base	File Ver	Product Ver	Image Ver
ADVAPI32.DLL	02/09/09 7:52p	671,744	A	Intel x86	Win32 console	Yes	0x71F50000	5.1.2600.5755	5.1.2600.5755	5.1
CALC.EXE	04/14/08 9:00p	114,688	A	Intel x86	Win32 GUI	Yes	0x01000000	5.1.2600.0	5.1.2600.0	5.1
GDI32.DLL	10/09/13 10:12p	297,744	A	Intel x86	Win32 console	Yes	0x77E20000	5.1.2600.6460	5.1.2600.6460	5.1
KERNEL32.DLL	03/12/14 7:48p	1,230,848	A	Intel x86	Win32 console	Yes	0x7C7D0000	5.1.2600.6532	5.1.2600.6532	5.1
MSVCRT.DLL	04/14/08 9:00p	343,040	A	Intel x86	Win32 GUI	Yes	0x77BC0000	7.0.2600.5512	6.1.8638.5512	5.1
NTDLL.DLL	12/10/10 12:15a	636,416	A	Intel x86	Win32 console	Yes	0x7C930000	5.1.2600.6055	5.1.2600.6055	5.1
RPCRT4.DLL	11/07/13 2:38p	591,360	A	Intel x86	Win32 console	Yes	0x77D80000	5.1.2600.6477	5.1.2600.6477	5.1

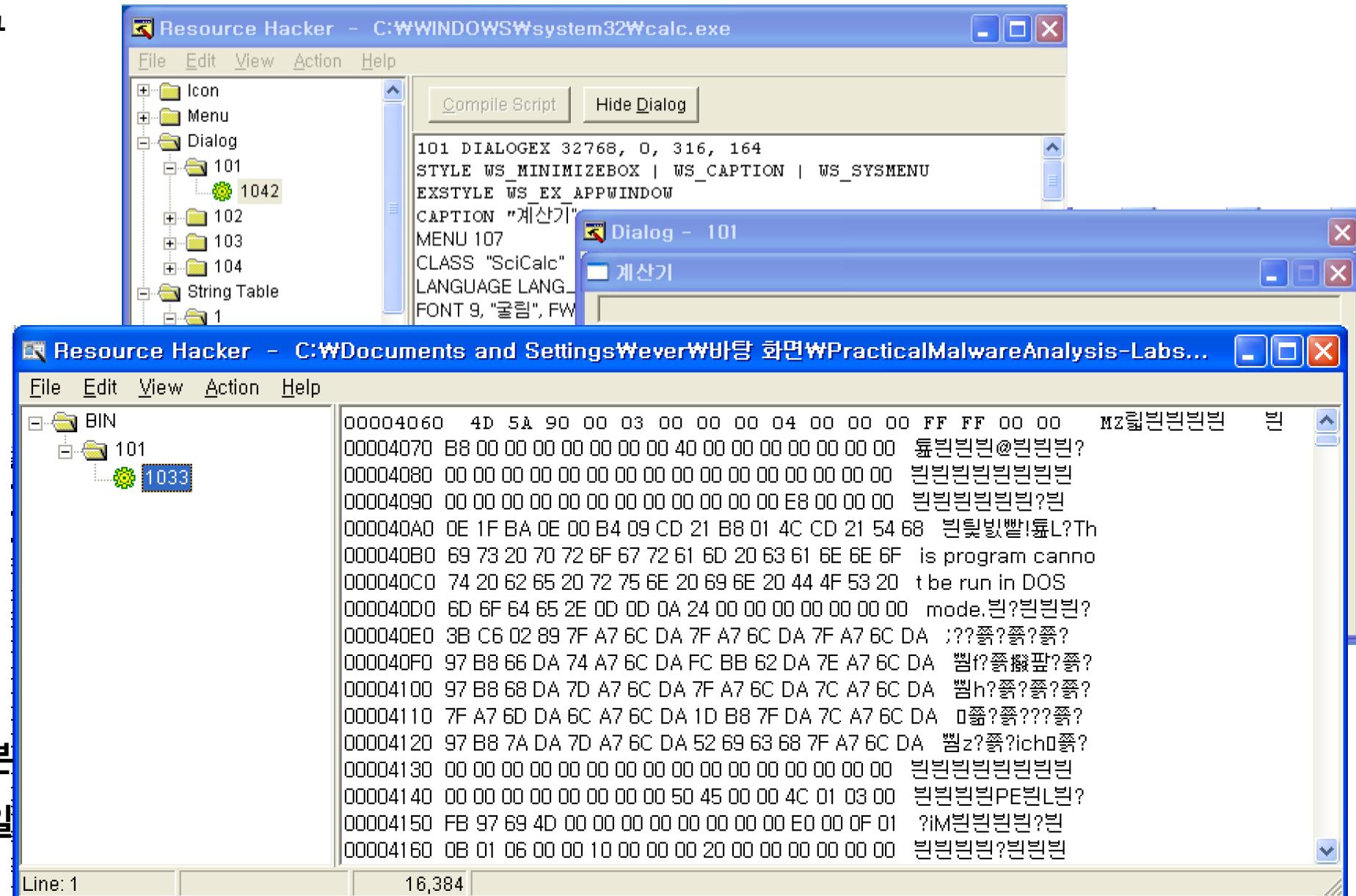


### 3. 악성코드 기초 분석

- 02 기초 정적 분석 도구

- 리소스 확인
- 리소스 해커 사용
  - 아이콘 섹션
  - 메뉴 섹션
  - 대화상자 섹션
  - 문자열 테이블
  - 버전정보 세션

리소스 해커로 본  
바이너리 파일



### 3. 악성코드 기초 분석



- **실습 1-1**
  - 이 실습은 Lab01-01.exe와 Lab01-01.dll 파일을 사용한다. 파일에 관한 정보를 얻으려면 1장에서 사용한 기법과 도구를 사용하고 다음 질문에 답해보자.
- **질문**
  - 1. <http://www.VirusTotal.com/> 에 파일을 업로드한 후 보고서를 보자. 기존 안티바이러스 시그니처에 일치하는 파일이 존재하는가?
  - 2. 이 파일은 언제 컴파일 되었는가?
  - 3. 이 파일이 패킹되거나 난독화 징후가 있는가? 그렇다면 무엇으로 판단했는가?
  - 4. 임포트를 보고 악성코드 행위를 알아낼 수 있는가? 그렇다면 어떤 임포트인가?
  - 5. 감염된 시스템에서 검색할 수 있는 다른 파일이나 호스트 기반의 증거가 존재하는가?
  - 6. 감염된 장비에서 이 악성코드를 발견하기 위해 사용한 네트워크 기반의 증거는 무엇인가?
  - 7. 이 파일의 목적은 무엇이라고 판단했는가?

### 3. 악성코드 기초 분석



- 실습 1-1

- Time Date Stamp

The screenshot shows two instances of the PEview application side-by-side, comparing the headers of two files: Lab01-01.dll and Lab01-01.exe.

**Lab01-01.dll Header Data:**

pFile	Data	Description	Value
000000E4	014C	Machine	IMAGE_FILE_MACHINE_I386
000000E6	0004	Number of Sections	
000000E8	4D0E2FE6	Time Date Stamp	2010/12/19 16:16:38 UTC
000000EC	00000000	Pointer to Symbol Table	
000000F0	00000000	Number of Symbols	
000000F4	00E0	Size of Optional Header	

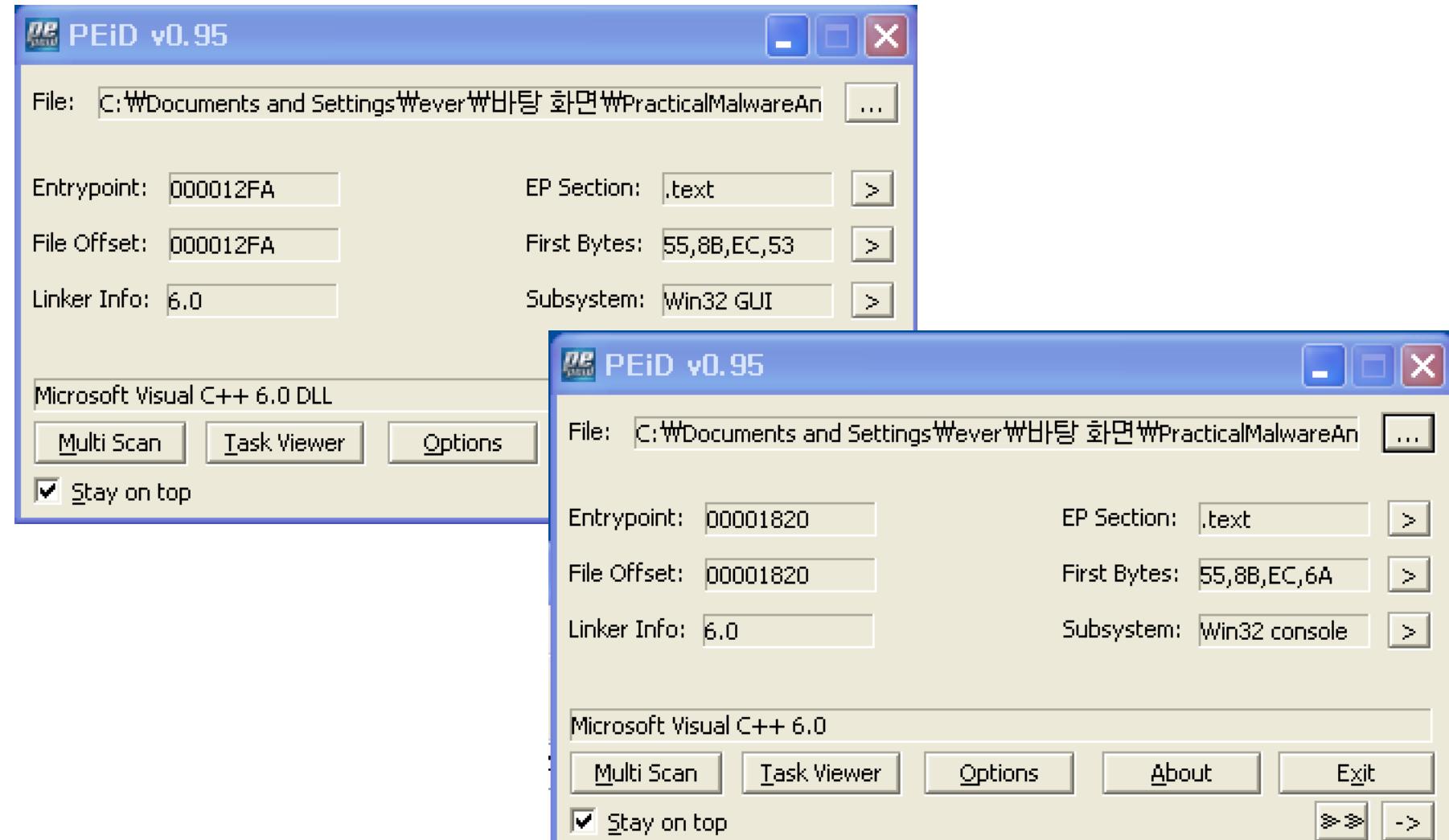
**Lab01-01.exe Header Data:**

pFile	Data	Description	Value
000000EC	014C	Machine	IMAGE_FILE_MACHINE_I386
000000EE	0003	Number of Sections	
000000F0	4D0E2FD3	Time Date Stamp	2010/12/19 16:16:19 UTC
000000F4	00000000	Pointer to Symbol Table	
000000F8	00000000	Number of Symbols	
000000FC	00E0	Size of Optional Header	
000000FE	010F	Characteristics	
	0001		IMAGE_FILE_RELOCS_STRNG
	0002		IMAGE_FILE_EXECUTABLE
	0004		IMAGE_FILE_LINE_NUMS_S
	0008		IMAGE_FILE_LOCAL_SYMS
	0100		IMAGE_FILE_32BIT_MACHIN

### 3. 악성코드 기초 분석



- 실습 1-1
  - 패킹 여부 확인



### 3. 악성코드 기초 분석



- 실습 1-1

- 문자열 확인

#### Lab01-01.exe strings

kerne132.dll

kernel32.dll

.exe

C:\\*\

C:\Windows\System32\kerne132.dll

Kernel32.

Lab01-01.dll

C:\Windows\System32\Kernel32.dll

WARNING\_THIS\_WILL\_DESTROY\_YOUR\_MACHINE

#### Lab01-01.dll Strings

exec

sleep

hello

127.26.152.13

### 3. 악성코드 기초 분석



#### • 실습 1-1

- Lab01-01.exe 사용 API 확인

pFile	Data	Description	Value
00002000	00002124	Hint/Name RVA	001B CloseHandle
00002004	00002132	Hint/Name RVA	02B0 UnmapViewOfFile
00002008	00002144	Hint/Name RVA	01B5 IsBadReadPtr
0000200C	00002154	Hint/Name RVA	01D6 MapViewOfFile
00002010	00002164	Hint/Name RVA	0035 CreateFileMappingA
00002014	0000217A	Hint/Name RVA	0034 CreateFileA
00002018	00002188	Hint/Name RVA	0090 FindClose
0000201C	00002194	Hint/Name RVA	009D FindNextFileA
00002020	000021A4	Hint/Name RVA	0094 FindFirstFileA
00002024	000021B6	Hint/Name RVA	0028 CopyFileA
00002028	00000000	End of Imports	KERNEL32.dll
0000202C	000021D0	Hint/Name RVA	0291 malloc
00002030	000021DA	Hint/Name RVA	0249 exit
00002034	000021EE	Hint/Name RVA	00D3 __exit
00002038	000021F6	Hint/Name RVA	0048 __XcptFilter
0000203C	00002204	Hint/Name RVA	0064 __p__initenv
00002040	00002214	Hint/Name RVA	0058 __getmainargs
00002044	00002224	Hint/Name RVA	010F __initterm

### 3. 악성코드 기초 분석



- 실습 1-1

- Lab01-01.dll 사용 API 확인

PEview - C:\Documents and Settings\ever\바탕 화면\PracticalMalwareAnalysis-Lab\Lab01-01.dll

pFile	Data	Description	Value
00002000	00002116	Hint/Name RVA	0296 Sleep
00002004	0000211E	Hint/Name RVA	0044 CreateProcessA
00002008	00002130	Hint/Name RVA	003F CreateMutexA
0000200C	00002140	Hint/Name RVA	01ED OpenMutexA
00002010	00002108	Hint/Name RVA	001B CloseHandle
00002014	00000000	End of Imports	KERNEL32.dll
00002018	0000219C	Hint/Name RVA	009D _adjust_fdiv
0000201C	00002192	Hint/Name RVA	0291 malloc
00002020	00002186	Hint/Name RVA	010F _initterm
00002024	0000217E	Hint/Name RVA	025E free
00002028	00002168	Hint/Name RVA	02C0 strncmp
0000202C	00000000	End of Imports	MSVCRT.dll
00002030	80000017	Ordinal	0017
00002034	80000073	Ordinal	0073
00002038	8000000B	Ordinal	000B
0000203C	80000004	Ordinal	0004
00002040	80000013	Ordinal	0013
00002044	80000016	Ordinal	0016
00002048	80000010	Ordinal	0010
0000204C	80000003	Ordinal	0003
00002050	80000074	Ordinal	0074
00002054	80000009	Ordinal	0009
00002058	00000000	End of Imports	WS2_32.dll

### 3. 악성코드 기초 분석



- **실습 1-2**
  - Lab01-02.exe 파일을 분석하라
- **질문**
  - 1. <http://www.VirusTotal.com/> 에 Lab01-02.exe 파일을 업로드하자. 기존 안티바이러스 정의된 것과 일치하는가?
  - 2. 이 파일이 패킹되거나 난독화된 징후가 있는가? 그렇다면 무엇으로 판단하였는가? 파일이 패킹되어 있다면 언패킹해보자.
  - 3. 임포트를 보고 악성코드의 기능을 알아낼 수 있는가? 그렇다면 어떤 임포트를 보고 알 수 있었는가?
  - 4. 감염된 시스템에서 악성코드를 인식하는 데 어떤 호스트 기반이나 네트워크 기반의 증거를 사용했는가?

### 3. 악성코드 기초 분석



#### • 실습 1-2

- <http://www.VirusTotal.com/> 에 Lab01-02.exe 파일을 업로드하자. 기존 안티바이러스 정의된 것과 일치하는가?

**virustotal**

SHA256: c876a332d7dd8da331cb8eee7ab7bf32752834d4b2b54eaa362674a2a48f64a6  
파일 이름: Lab01-02.exe  
탐지 비율: 36 / 56  
분석 날짜: 2016-06-04 20:22:04 UTC ( 18시간, 54분 전 )

56 16

분석 File detail Relationships 추가 정보 댓글 4 투표

안티바이러스	결과	업데이트
ALYac	Trojan.Startpage.3072	20160604
AVG	Downloader.Generic12.CMEY	20160604
AVware	Trojan.Win32.Generic!BT	20160604
AegisLab	Troj.Downloader.Gen!c	20160604
AhnLab-V3	Trojan/Win32.StartPage	20160604

### 3. 악성코드 기초 분석



#### • 실습 1-2

- 이 파일이 패킹되거나 난독화된 징후가 있는가? 그렇다면 무엇으로 판단하였는가? 파일이 패킹되어 있다면 언패킹해보자.

The screenshot displays two windows side-by-side. The left window is 'PEview' showing the file structure of 'Lab01-02.exe'. The right window is 'PEiD v0.95' showing file metadata and analysis results.

**PEview - C:\Documents and Settings\ever\바탕 화면\Practical**

File View Go Help

File Explorer View Options

Lab01-02.exe

- IMAGE\_DOS\_HEADER
- MS-DOS Stub Program
- + IMAGE\_NT\_HEADERS
  - IMAGE\_SECTION\_HEADER UPX0
  - IMAGE\_SECTION\_HEADER UPX1
  - IMAGE\_SECTION\_HEADER UPX2
- SECTION UPX0
- SECTION UPX1
- + SECTION UPX2

pFile	Data	Description
000001D8	55 50 58 30	Name
000001DC	00 00 00 00	
000001E0	00004000	Virtual Size
000001E4	00001000	RVA
000001E8	00000000	Size of Raw Data
000001EC	00000400	Pointer to Raw Data
000001F0	00000000	Pointer to Relocations
000001F4	00000000	Pointer to Line Numbers
000001F8	0000	Number of Relocations

**PEiD v0.95**

File: C:\Documents and Settings\ever\바탕 화면\PracticalMalwareAnalysis\Lab01-02.exe

Entry point: 00005410 EP Section: UPX1

File Offset: 00000810 First Bytes: 60,BE,00,50

Linker Info: 6.0 Subsystem: Win32 console

Nothing found \*

Multi Scan Task Viewer Options About Exit

Stay on top

### 3. 악성코드 기초 분석



#### • 실습 1-2

- 임포트를 보고 악성코드의 기능을 알아낼 수 있는가? 그렇다면 어떤 임포트를 보고 알 수 있었는가?

```
C:\Documents and Settings\ever\바탕 화면\PracticalMalwareAnalysis-Labs\Practical Malware Analysis Labs\BinaryCollection\Chapter_1L>upx -o Unpacked-Lab01-02.exe -d Lab01-02.exe

          Ultimate Packer for eXecutables
          Copyright (C) 1996 - 2008
UPX 3.03v      Markus Oberhumer, Laszlo Molnar & John Reiser   Apr 27th 2008

  File size       Ratio     Format      Name
-----  -----  -----
  16384 <-    3072  18.75%  win32/pe  Unpacked-Lab01-02.exe

Unpacked 1 file.
```

Unpacking을 해야 내용이 보인다!

Unpacked-Lab01-02.exe의 Import Table

pFile	Data	Description	Value
00002000	0000223C	Hint/Name RVA	0000 CreateServiceA
00002004	0000224C	Hint/Name RVA	0000 StartServiceCtrlDispatcherA
00002008	0000226A	Hint/Name RVA	0000 OpenSCManagerA
0000200C	00000000	End of Imports	ADVAPI32.dll
00002010	0000219E	Hint/Name RVA	0000 SystemTimeToFileTime
00002014	000021B4	Hint/Name RVA	0000 GetModuleFileNameA
00002018	000021C8	Hint/Name RVA	0000 CreateWaitableTimerA
0000201C	000021DE	Hint/Name RVA	0000 ExitProcess
00002020	000021EC	Hint/Name RVA	0000 OpenMutexA
00002024	000021F8	Hint/Name RVA	0000 SetWaitableTimer
00002028	0000220A	Hint/Name RVA	0000 WaitForSingleObject
0000202C	00002220	Hint/Name RVA	0000 CreateMutexA
00002030	0000222E	Hint/Name RVA	0000 CreateThread
00002034	00000000	End of Imports	KERNEL32.DLL
00002038	0000227A	Hint/Name RVA	0000 __exit
0000203C	00002282	Hint/Name RVA	0000 __XcptFilter
00002040	00002290	Hint/Name RVA	0000 exit
00002044	00002296	Hint/Name RVA	0000 __p__initenv
00002048	000022A6	Hint/Name RVA	0000 __getmainargs
0000204C	000022B6	Hint/Name RVA	0000 __initterm
00002050	000022C2	Hint/Name RVA	0000 __setusermatherr
00002054	000022D4	Hint/Name RVA	0000 __adjust_fdiv
00002058	000022E2	Hint/Name RVA	0000 __p__commode
0000205C	000022F0	Hint/Name RVA	0000 __p__fmode
00002060	000022FC	Hint/Name RVA	0000 __set_app_type
00002064	0000230C	Hint/Name RVA	0000 __except_handler3
00002068	0000231E	Hint/Name RVA	0000 __controlfp
0000206C	00000000	End of Imports	MSVCR7.dll
00002070	0000232A	Hint/Name RVA	0000 InternetOpenUrlA
00002074	0000233C	Hint/Name RVA	0000 InternetOpenA
00002078	00000000	End of Imports	WININET.dll

### 3. 악성코드 기초 분석



- **실습 1-2**

- 감염된 시스템에서 악성코드를 인식하는 데 어떤 호스트 기반이나 네트워크 기반의 증거를 사용했는가?

#### UnpackedLab01-02.exe Strings

MalService

Malservice

HGL345

<http://www.malwareanalysisbook.com>

Internet Explorer 8.0

### 3. 악성코드 기초 분석



- **실습 1-3**
  - Lab01-03.exe 파일을 분석하라.
- **질문**
  - 1. http://www.VirusTotal.com/ 에 Lab01-03.exe 파일을 업로드하자. 기존 안티바이러스 정의된 것과 일치하는가?
  - 2. 이 파일이 패킹되거나 난독화된 징후가 있는가? 그렇다면 무엇으로 판단했는가? 파일이 패킹돼 있고 가능하다면 언패킹해보자.
  - 3. 임포트를 보고 악성코드의 기능을 알아낼 수 있는가? 그렇다면 어떤 임포트를 보고 알 수 있었는가?
  - 4. 감염된 시스템에서 악성코드를 인식하는데 어떤 호스트 기반이나 네트워크 기반의 증거를 사용했는가?

### 3. 악성코드 기초 분석



- **실습 1-4**
  - Lab01-04.exe 파일을 분석하라.
- **질문**
  - 1. http://www.VirusTotal.com/ 에 Lab01-04.exe 파일을 업로드하자. 기존 안티바이러스 정의된 것과 일치하는가?
  - 2. 이 파일이 패킹되거나 난독화된 징후가 있는가? 그렇다면 무엇으로 판단했는가? 파일이 패킹되어 있고 가능하다면 언패킹해보자.
  - 3. 이 프로그램은 언제 컴파일 되었는가?
  - 4. 임포트를 보고 악성코드의 기능을 알아낼 수 있는가? 그렇다면 어떤 임포트를 보고 알 수 있었는가?
  - 5. 감염된 시스템에서 악성코드를 인식하는데 어떤 호스트 기반이나 네트워크 기반의 증거를 사용했는가?
  - 6. 이 파일은 리소스 섹션에 하나의 리소스가 있다. Resource Hacker를 이용해 리소스파일을 점검하고 리소스를 추출해보자. 리소스로부터 무엇을 알 수 있는가?

### 3. 악성코드 기초 분석



#### • 01 기초 동적 분석 개요

- 기초 동적 분석이란?
  - 프로그램을 직접 실행하며 분석
  - 악성코드 분석에서 가장 프로그램의 영향을 쉽게 파악
  - 프로그램의 기능을 파악하기 위해 악성코드 실행 전후 상태를 조사 및 분석
- 분석방법
  - 악성코드 실습 시 발생하는 호스트/네트워크 환경 구성
  - 파일, 프로그램 실행, 레지스트리, 서비스 등 관련 항목 변경 사항 확인
  - 실행 시 발생하는 네트워크 트래픽 분석

### 3. 악성코드 기초 분석



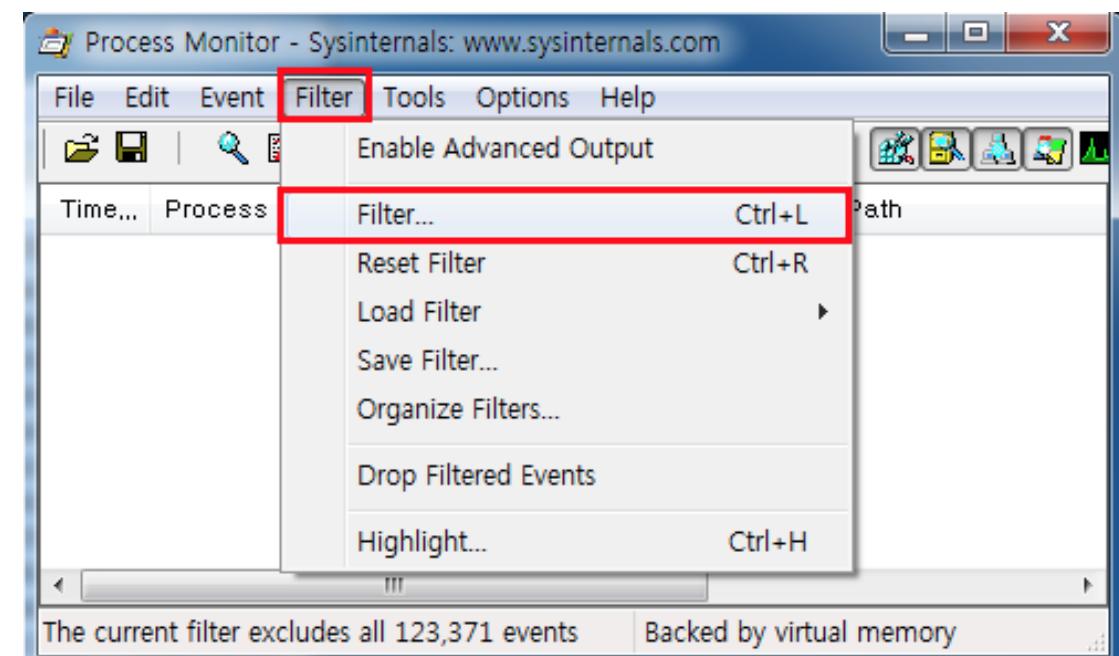
#### • 02 기초 동적 분석 도구

- 프로세스 모니터(procmon.exe)

- 특정 레지스트리, 파일 시스템, 네트워크, 프로세스, 스레드 행위를 모니터링하는 고급 도구

- ProcMon의 한계

- 특정 GUI와 장치 I/O 제어를 통한 루트킷 탐지 불가
    - 네트워크 행위에 대해 일관성 있는 탐지 불가



### 3. 악성코드 기초 분석



#### • 02 기초 동적 분석 도구

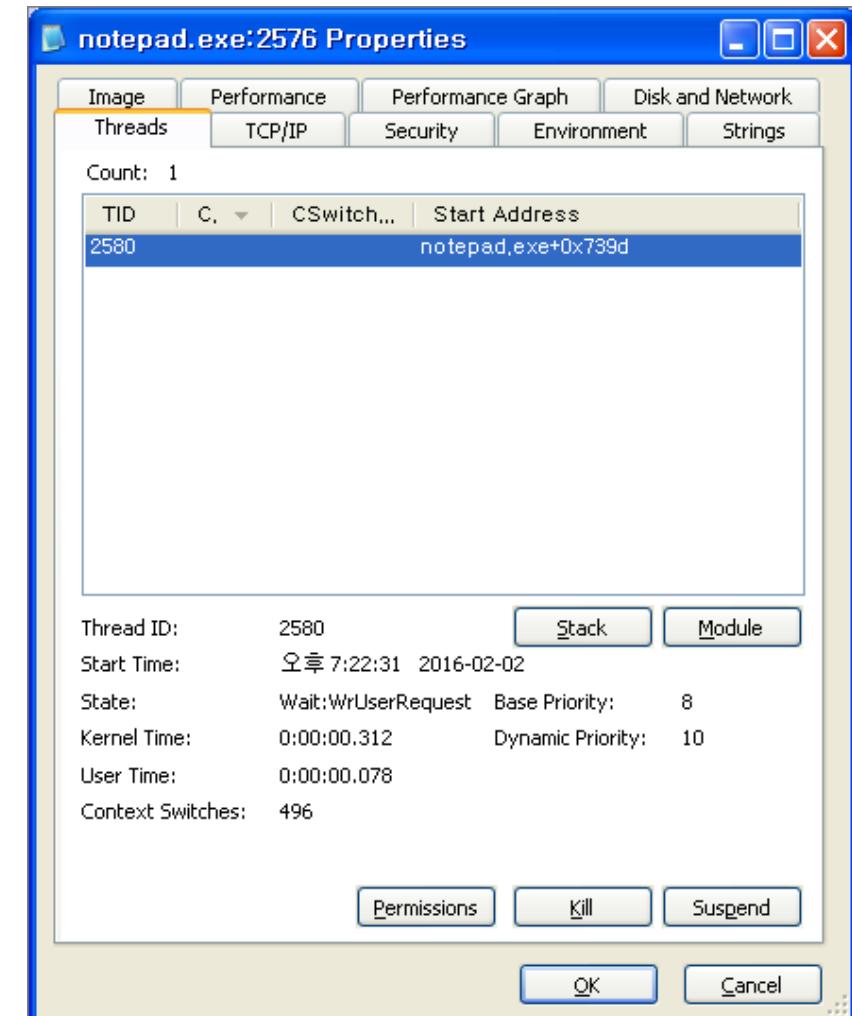
- 프로세스 익스플로러(procexp.exe)

- 프로세스(EPROCESS 구조체)에 관련된 많은 내용을 확인 가능

#### 프로세스 내의 다양한 객체 확인

Type	Name
Desktop	\Default
Directory	\KnownDLLs
Directory	\Windows
Directory	\BaseNamedObjects
Event	\BaseNamedObjects\userenv: User Profile setup event
File	C:\WINDOWS\system32
File	C:\WINDOWS\WinSxS\x86_Microsoft,Windows,Common-Controls_6595b641...
File	C:\WINDOWS\WinSxS\x86_Microsoft,Windows,Common-Controls_6595b641...
File	\Device\KsecDD
Key	HKLM
Key	HKLM\Software\Microsoft\Windows NT\CurrentVersion\Drivers32
Key	HKLM\Software\Microsoft\Windows NT\CurrentVersion\Drivers32
Key	HKCU
Key	HKLM\SYSTEM\ControlSet001\Control\Nls\Locale\Alternate Sorts
Key	HKLM\SYSTEM\ControlSet001\Control\Nls\Locale
Key	HKLM\SYSTEM\ControlSet001\Control\Nls\Language Groups
Key	HKCU\Software\Microsoft\Windows\ShellNoRoam
Key	HKCU\Software\Microsoft\Windows\ShellNoRoam\MUICache
Key	HKEY_CURRENT_USER
Key	HKCU\Software\Classes
Key	HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer
Key	HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\FileExts
KeyedEvent	\KernelObjects\CritSecOutOfMemoryEvent
Mutant	\BaseNamedObjects\SHIMLIB_LOG_MUTEX
Mutant	\BaseNamedObjects\CTF_LBES MutexDefault\1-5-21-1844237615-132657467...
Mutant	\BaseNamedObjects\CTF_Compact MutexDefault\1-5-21-1844237615-132657467...

프로세스 스레드/권한/네트워크/메모리  
등 다양한 정보 확인 가능



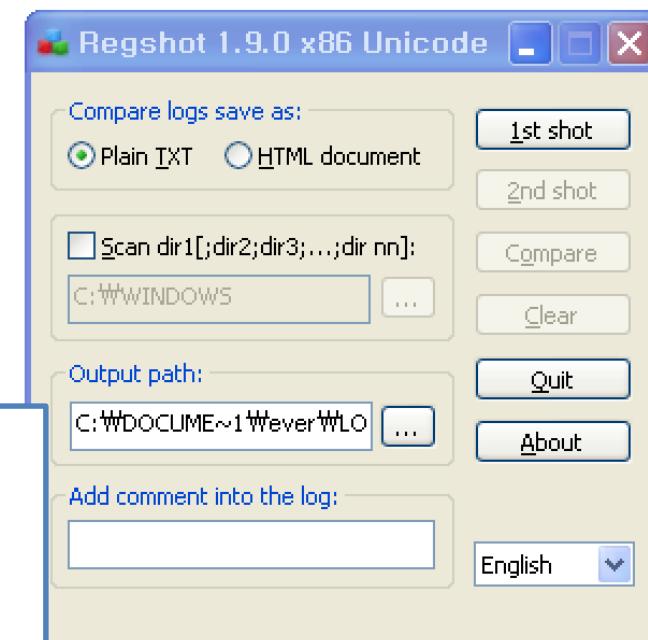
### 3. 악성코드 기초 분석



#### • 02 기초 동적 분석 도구

##### - RegShot

- 두 레지스트리의 스냅샷을 찍고 비교하는 툴
- 악성코드는 자동실행을 위해 레지스트리를 자주 건드림



1. 악성코드 실행 전에 1st shot을 찍음
2. 악성코드 실행 후에 2nd shot을 찍음
3. Compare하여 전후 비교

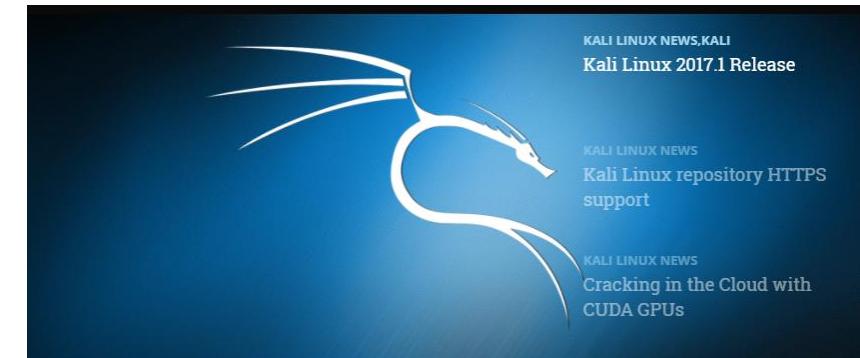
### 3. 악성코드 기초 분석



#### • 02 기초 동적 분석 도구

- INetSim

- 가짜 서비스를 제공하기에 최고의 도구!
- 일반적인 로깅 및 중앙 집중식 제어 기능을 사용하여 다양한 인터넷 서비스를 시뮬레이션 할 수 있는 편안한 단일 제품군
- 칼리리눅스에서 별도의 설치 없이 INetSim 실행 가능
- 알려지지 않은 맬웨어 샘플의 네트워크 동작에 대한 런타임 런타임 분석을 수행하기 위한 도구
- 실험실 환경에서 맬웨어가 일반적으로 사용하는 인터넷 서비스를 시뮬레이트
- HTTP, HTTPS, FTP, IRC, DNS, SMTP 등 서비스를 실행
- **칼리 리눅스 VM 이미지 다운로드 :** <https://www.offensive-security.com/kali-linux-vmware-virtualbox-image-download/>



### 3. 악성코드 기초 분석



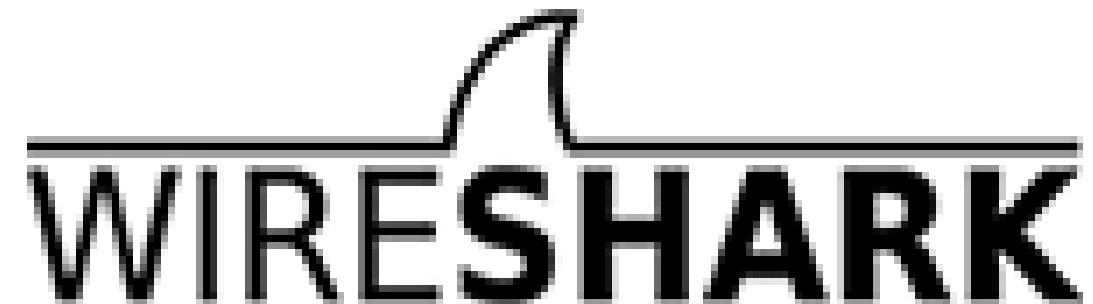
- 02 기초 동적 분석 도구

- WireShark 개요

- 세계에서 가장 널리 쓰이는 네트워크 분석 프로그램
    - 네트워크상에서 캡처한 데이터에 대한 네트워크/상위 레이어 프로토콜의 정보를 제공
    - 패킷을 캡처하기 위해 pcap 네트워크 라이브러리를 사용

- Wireshark의 강점 :

- 쉬운 설치.
    - GUI 인터페이스를 이용한 간단한 사용법.
    - 매우 다양한 기능들



### 3. 악성코드 기초 분석



#### • 02 기초 동적 분석 도구

- Wireshark
  - 인터페이스

Menus

Shortcuts

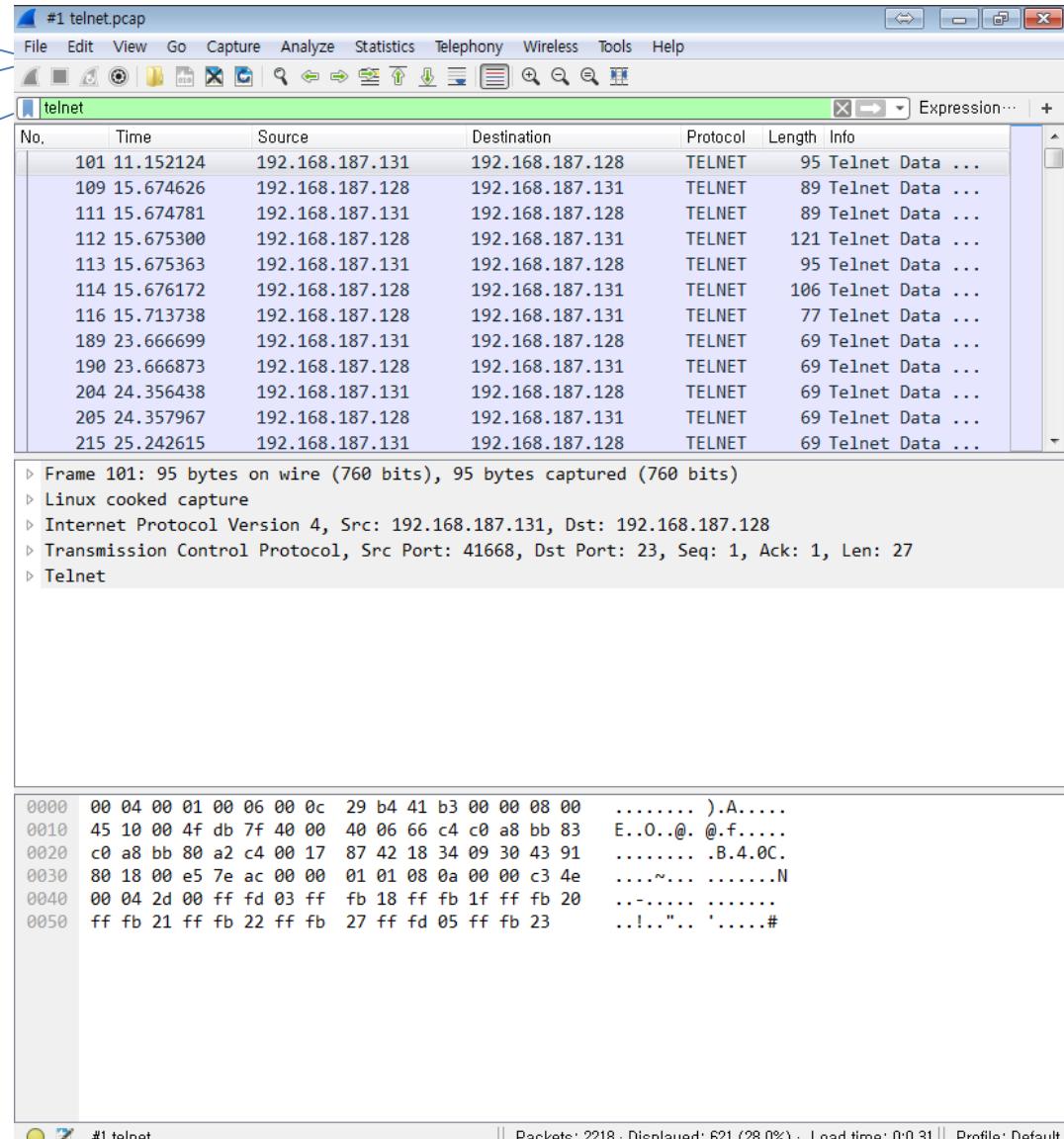
CaptureFilter

PacketListPanel

Packet Details Panel

Dissector Panel

Misc



### 3. 악성코드 기초 분석



#### • 02 기초 동적 분석 도구

- Wireshark 인터페이스 Menus
  - 플랫폼 상단의 8개 메뉴는 Wireshark을 설정하는데 사용



메뉴	설명	메뉴	설명
File	캡처 데이터를 열거나 저장	Statistics	Wireshark의 통계 데이터를 사용
Edit	패킷을 찾거나 표시. 프로그램 전역적인 속성들을 설정	Telephony	
View	Wireshark 플랫폼의 보이는 모양을 설정	Wireless	
Go	캡처된 데이터의 특정 위치로 이동	Tools	
Capture	캡처 필터 옵션을 설정하고 캡처를 시작	Help	오프라인 혹은 온라인 도움말 보기
Analyze	분석 옵션을 설정		

### 3. 악성코드 기초 분석



- 02 기초 동적 분석 도구

- Wireshark 인터페이스 Shortcuts



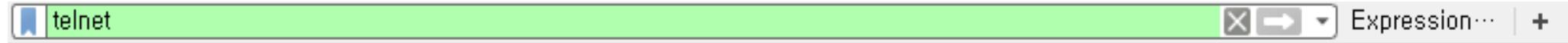
- 유용한 단축키들이 메뉴 바로 아래 존재
    - 마우스를 아이콘 위에 올려 놓으면 자세한 정보 팝업

### 3. 악성코드 기초 분석



- 02 기초 동적 분석 도구

- Wireshark 인터페이스 Capture Filter



- 캡처된 로그 정보를 필터링
    - 원하는 특정 데이터를 찾을 때 사용
    - 후에 와이어샤크 필터리 문법에 대해 따로 배움

### 3. 악성코드 기초 분석



- 02 기초 동적 분석 도구

- Wireshark 인터페이스 Packet List Panel

No.	Time	Source	Destination	Protocol	Length	Info
101	11.152124	192.168.187.131	192.168.187.128	TELNET	95	Telnet Data ...
109	15.674626	192.168.187.128	192.168.187.131	TELNET	89	Telnet Data ...
111	15.674781	192.168.187.131	192.168.187.128	TELNET	89	Telnet Data ...
112	15.675300	192.168.187.128	192.168.187.131	TELNET	121	Telnet Data ...
113	15.675363	192.168.187.131	192.168.187.128	TELNET	95	Telnet Data ...
114	15.676172	192.168.187.128	192.168.187.131	TELNET	106	Telnet Data ...
116	15.713738	192.168.187.128	192.168.187.131	TELNET	77	Telnet Data ...

- 캡처된 모든 패킷의 Source/destination MAC/IP 주소, TCP /UDP 포트 번호, 프로토콜, 패킷 내용 등의 정보 디스플레이
- 열은 추가/삭제 가능
- Edit menu -> Preferences를 통해 패널의 색상 변경

### 3. 악성코드 기초 분석



- 02 기초 동적 분석 도구

- Wireshark 인터페이스 Packet Details Panel [1/2]

- 선택된 패킷에 대한 상세 정보 제공

선택된 패킷

No.	Time	Source	Destination	Protocol	Length	Info
101	11.152124	192.168.187.131	192.168.187.128	TELNET	95	Telnet Data ...
109	15.674626	192.168.187.128	192.168.187.131	TELNET	89	Telnet Data ...
111	15.674781	192.168.187.131	192.168.187.128	TELNET	89	Telnet Data ...
112	15.675300	192.168.187.128	192.168.187.131	TELNET	121	Telnet Data ...
113	15.675363	192.168.187.131	192.168.187.128	TELNET	95	Telnet Data ...
114	15.676172	192.168.187.128	192.168.187.131	TELNET	106	Telnet Data ...
116	15.713738	192.168.187.128	192.168.187.131	TELNET	77	Telnet Data ...
189	23.666699	192.168.187.131	192.168.187.128	TELNET	69	Telnet Data ...
190	23.666873	192.168.187.128	192.168.187.131	TELNET	69	Telnet Data ...
204	24.356438	192.168.187.131	192.168.187.128	TELNET	69	Telnet Data ...
205	24.357967	192.168.187.128	192.168.187.131	TELNET	69	Telnet Data ...
215	25.242615	192.168.187.131	192.168.187.128	TELNET	69	Telnet Data ...

상세정보

```
▶ Frame 101: 95 bytes on wire (760 bits), 95 bytes captured (760 bits)
▶ Linux cooked capture
▶ Internet Protocol Version 4, Src: 192.168.187.131, Dst: 192.168.187.128
▶ Transmission Control Protocol, Src Port: 41668, Dst Port: 23, Seq: 1, Ack: 1, Len: 27
▶ Telnet
```

### 3. 악성코드 기초 분석



#### • 02 기초 동적 분석 도구

- Wireshark 인터페이스 Packet Details Panel [2/2]

- OSI layer별로 표시
- 자세한 정보들을 확장하거나 축소

TCP 헤더 상세정보

```
Transmission Control Protocol, Src Port: 41668, Dst Port: 23, Seq: 1, Ack: 1, Len: 27
  Source Port: 41668
  Destination Port: 23
  [Stream index: 15]
  [TCP Segment Len: 27]
  Sequence number: 1      (relative sequence number)
  [Next sequence number: 28      (relative sequence number)]
  Acknowledgment number: 1      (relative ack number)
  Header Length: 32 bytes
  ▶ Flags: 0x018 (PSH, ACK)
  Window size value: 229
  [Calculated window size: 29312]
  [Window size scaling factor: 128]
  Checksum: 0x7eac [unverified]
  [Checksum Status: Unverified]
  Urgent pointer: 0
  ▶ Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
  ▶ [SEQ/ACK analysis]
  ▶ Telnet
  ▶ Do Suppress Go Ahead
```

### 3. 악성코드 기초 분석



#### • 02 기초 동적 분석 도구

- Wireshark 인터페이스 Dissector Panel(packet bytes Panel)
  - packet details 패널과 내용은 같은 데이터를 16진수로 디스플레이

The screenshot shows the Wireshark interface with a captured file named '#1 telnet.pcap'. The packet list pane displays three Telnet data packets. The details pane shows the selected packet's header information, including Source (192.168.187.131), Destination (192.168.187.128), Protocol (TELNET), Length (95), and Info (Telnet Data). The bytes pane shows the raw hex and ASCII data for the selected packet.

선택된 패킷

선택된 헤더

선택된 헤더의 로우 데이터

No.	Time	Source	Destination	Protocol	Length	Info
101	11.152124	192.168.187.131	192.168.187.128	TELNET	95	Telnet Data ...
109	15.674626	192.168.187.128	192.168.187.131	TELNET	89	Telnet Data ...
111	15.674781	192.168.187.131	192.168.187.128	TELNET	89	Telnet Data ...

Frame 101: 95 bytes on wire (760 bits), 95 bytes captured (760 bits)  
Linux cooked capture  
Internet Protocol Version 4, Src: 192.168.187.131, Dst: 192.168.187.128  
Transmission Control Protocol, Src Port: 41668, Dst Port: 23, Seq: 1, Ack: 1, Len: 27  
Telnet

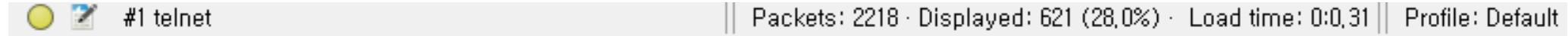
Offset	Hex	ASCII
0000	00 04 00 01 00 06 00 0c 29 b4 41 b3 00 00 08 00	..... ).A.....
0010	45 10 00 4f db 7f 40 00 40 06 66 c4 c0 a8 bb 83	E..0..@. @.f.....
0020	c0 a8 bb 80 a2 c4 00 17 87 42 18 34 09 30 43 91	.....B.4.0C.
0030	80 18 00 e5 7e ac 00 00 01 01 08 0a 00 00 c3 4e	....~....N
0040	00 04 2d 00 ff fd 03 ff fb 18 ff fb 1f ff fb 20	.....
0050	ff fb 21 ff fb 22 ff fb 27 ff fd 05 ff fb 23	...!... '....#

### 3. 악성코드 기초 분석



#### • 02 기초 동적 분석 도구

- Wireshark 인터페이스 Misc (화면 최하단)



- 캡처 동작이 진행 혹은 정지 상태 여부.
- 파일 프로파일
- 저장된 파일 이름
- 캡처된 패킷 수(Packets)
- 화면에 표시된 패킷 수(Displayed)

### 3. 악성코드 기초 분석



#### • 02 기초 동적 분석 도구

- Wireshark 컬럼 설정 (<http://malware-traffic-analysis.net/tutorials/index.html>)
  - Wireshark는 훌륭한 도구! 그러나 기본 열 표시는 일반적으로 수행하는 분석 유형에 대해 효과적으로 작동하지 않는다....
  - 대부분의 사람들은 기본 구성에서 열 변경
  - Wireshark의 기본 열 : 패킷 번호, 시간, 소스, 대상, 프로토콜, 길이 및 정보

No.	Time	Source	Destination	Protocol	Length	Info
101	11.152124	192.168.187.131	192.168.187.128	TELNET	95	Telnet Data ...
109	15.674626	192.168.187.128	192.168.187.131	TELNET	89	Telnet Data ...
111	15.674781	192.168.187.131	192.168.187.128	TELNET	89	Telnet Data ...
112	15.675300	192.168.187.128	192.168.187.131	TELNET	121	Telnet Data ...
113	15.675363	192.168.187.131	192.168.187.128	TELNET	95	Telnet Data ...
114	15.676172	192.168.187.128	192.168.187.131	TELNET	106	Telnet Data ...
116	15.713738	192.168.187.128	192.168.187.131	TELNET	77	Telnet Data ...

### 3. 악성코드 기초 분석



#### • 02 기초 동적 분석 도구

- Wireshark 컬럼 설정 (<http://malware-traffic-analysis.net/tutorials/index.html>)
  - 우리의 환경 설정을 편집하여 이것을 바꾸자 (Edit -> Preferences -> Appearance -> Columns).

The screenshot shows the Wireshark interface with a packet capture titled '#1 telnet.pcap'. The main window displays three Telnet data packets. The preferences dialog is open, specifically the 'Columns' tab under the 'Appearance' section. This dialog lists various fields like No., Time, Source, Destination, Protocol, Length, and Info, each with a checkbox indicating whether it should be displayed in the main pane. The 'Length' field is currently selected. The bottom right of the preferences dialog has buttons for '확인' (Confirm), '취소' (Cancel), and '도움말' (Help).

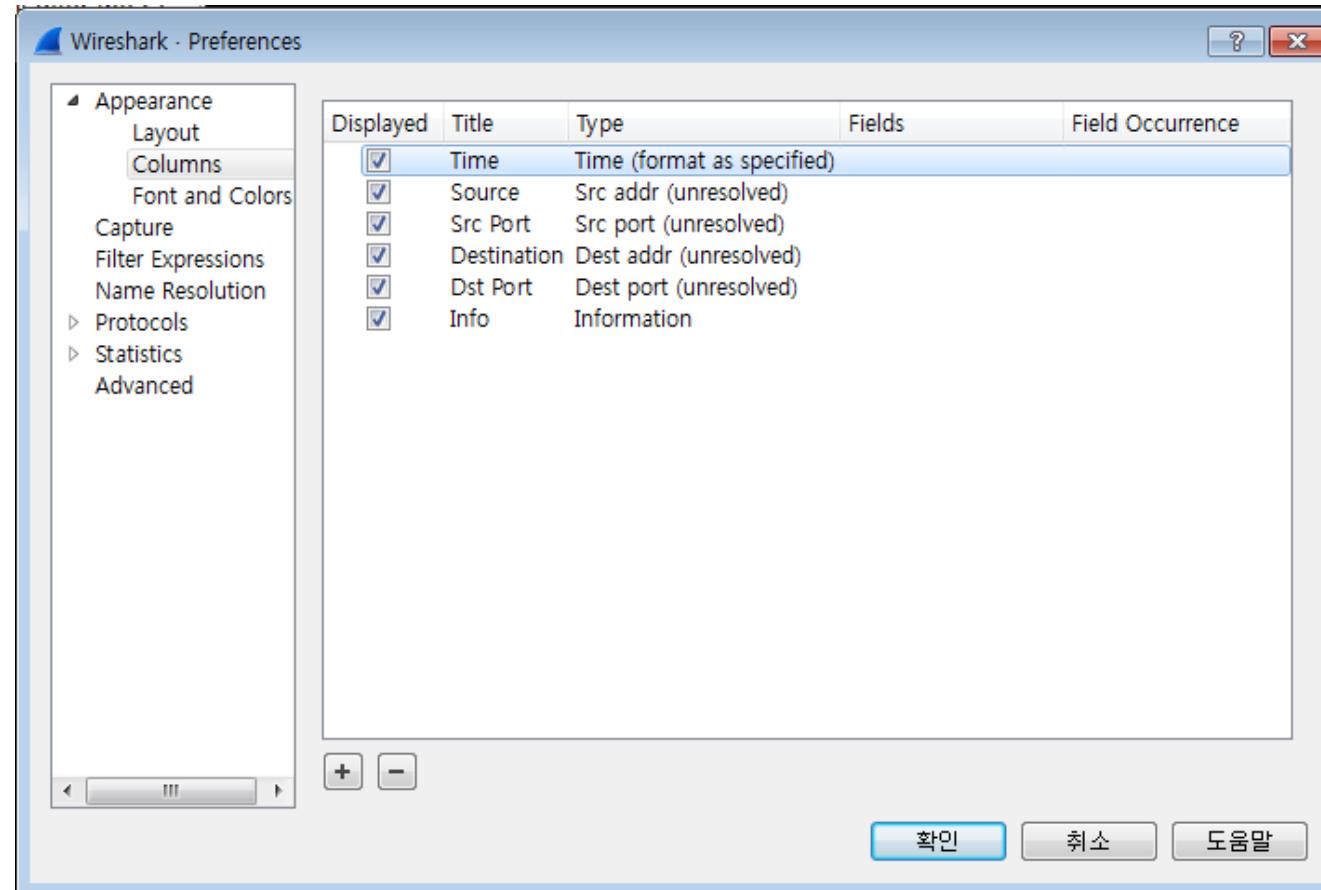
### 3. 악성코드 기초 분석



#### • 02 기초 동적 분석 도구

- Wireshark 컬럼 설정 (<http://malware-traffic-analysis.net/tutorials/index.html>)

- 다음과 같이 +, - 버튼을 활용하여 편집(unresolved : 결심이 서지 않은!)



### 3. 악성코드 기초 분석



#### • 02 기초 동적 분석 도구

- Wireshark 시간 설정 (<http://malware-traffic-analysis.net/tutorials/index.html>)

- View -> Time Display Format -> Date and Time of Day

The screenshot shows the Wireshark interface with the 'View' menu open. A large black arrow points downwards from the 'Date and Time of Day' option in the 'Time Display Format' submenu. The main window displays a list of network packets, with the first few entries being Telnet sessions.

Wireshark

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Time

2017-05-03 18:02:00.226741

Main Toolbar  
Filter Toolbar  
Wireless Toolbar  
Status Bar  
Packet List  
Packet Details  
Packet Bytes

Time Display Format

- Date and Time of Day (1970-01-01 01:02:03.123456) Ctrl+Alt+1
- Year, Day of Year, and Time of Day (1970/001 01:02:03.123456)
- Time of Day (01:02:03.123456) Ctrl+Alt+2
- Seconds Since 1970-01-01 Ctrl+Alt+3
- Seconds Since Beginning of Capture Ctrl+Alt+4
- Seconds Since Previous Captured Packet Ctrl+Alt+5
- Seconds Since Previous Displayed Packet Ctrl+Alt+6
- UTC Date and Time of Day (1970-01-01 01:02:03.123456) Ctrl+Alt+7
- UTC Year, Day of Year, and Time of Day (1970/001 01:02:03.123456)
- UTC Time of Day (01:02:03.123456) Ctrl+Alt+8
- Automatic (from capture file)
- Seconds
- Tenths of a second
- Hundredths of a second
- Milliseconds
- Microseconds

Src Port Destination Dst Port Info

41668 192.168.187.128 23 Telnet Data ...  
23 192.168.187.131 41668 Telnet Data ...  
41668 192.168.187.128 23 Telnet Data ...

Copyright [www.boanproject.com](http://www.boanproject.com) All rights reserved

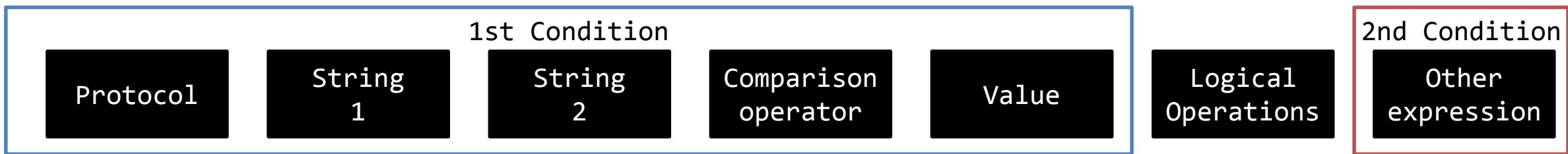
### 3. 악성코드 기초 분석



#### • 02 기초 동적 분석 도구

- Wireshark 디스플레이 필터 – [https://openmaniak.com/kr/wireshark\\_use.php](https://openmaniak.com/kr/wireshark_use.php)

- 캡처된 데이터에서 원하는 정보를 찾을 때 사용
- 다음과 같은 구문(Syntax)을 사용



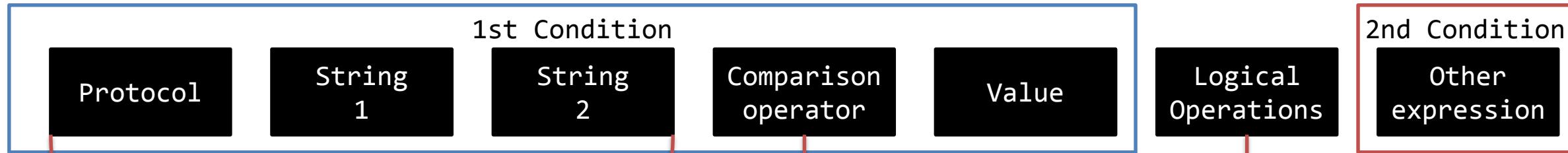
ftp              passive              ip              ==              10.2.3.4              and              icmp.type

### 3. 악성코드 기초 분석



#### • 02 기초 동적 분석 도구

- Wireshark 디스플레이 필터 – [https://openmaniak.com/kr/wireshark\\_use.php](https://openmaniak.com/kr/wireshark_use.php)



비교 연산자		
영문 표기	C언어 표기	의미
eq	==	같다
ne	!=	틀리다
gt	>	크다
lt	<	작다
ge	>=	크거나 같다
le	<=	작거나 같다

논리 표현식	영문 표기	C언어 표기	의미
and		&&	논리곱
or			논리합
xor		^^	배타적 논리합
not		!	부정

#### 검색 연산자

구문	의미
contains	프로토콜 내에 field 또는 slice에 해당 값을 포함되는 경우
matches	프로토콜 또는 텍스트 문자열 내에 주어진 Perl 정규표현식이 일치하는 경우

### 3. 악성코드 기초 분석



#### • 02 기초 동적 분석 도구

- Wireshark 디스플레이 필터 – [https://openmaniak.com/kr/wireshark\\_use.php](https://openmaniak.com/kr/wireshark_use.php)
- Expression 기능

The screenshot shows the Wireshark interface with a packet list and an open 'Display Filter Expression' dialog box.

**Wireshark Main Window:** The main window displays a list of network packets from a file named '#2 ftp.pcap'. The columns include No., Time, Source, Destination, Protocol, Length, and Info. The 'Info' column shows detailed protocol analysis for each packet.

**Display Filter Expression Dialog:** This dialog box allows users to define custom display filters. It includes fields for 'Field Name' (a dropdown menu listing various network protocols and formats), 'Relation' (operators like 'is present', '=', '!=', '>'), 'Value' (a text input field), 'Predefined Values' (a dropdown menu), and 'Range (offset:length)' (input fields for offset and length). The 'Search:' field at the bottom is empty, and there is a note about 'No display filter'.



### 3. 악성코드 기초 분석

- 02 기초 동적 분석 도구

- Wireshark 자주 사용하는 디스플레이 필터 Top 10

- 참고문헌 : <http://www.lovemytool.com/blog/2010/04/top-10-wireshark-filters-by-chris-greer.html>

디스플레이 필터	설명
ip.addr == 10.0.0.1	출발지나 목적지의 ip가 10.0.0.1인 경우 출력
ip.addr == 10.0.0.1 && ip.addr == 10.0.0.2	두 개의 정의된 IP 주소 모두 출력
http or arp	모든 http와 dns 프로토콜 출력
tcp.port == 4000	출발지나 목적지의 포트가 4000인 TCP 패킷 출력
tcp.flags.reset == 1	모든 TCP reset 플래그가 활성화된 패킷 출력
http.request	모든 HTTP GET 요청 패킷 출력
tcp contains traffic	'traffic'라는 단어를 포함하는 TCP 패킷 출력 (특정 문자열이나 유저 이름을 출력할 때 효과적)
!(arp or icmp or dns)	괄호 내용을 모두 제외한 패킷을 출력
contains 33:27:58	헥스 값(0x33 0x27 0x58) 필터
tcp.analysis.retransmission	추적에서 모든 재전송을 표시 (느린 응용 프로그램 성능 및 패킷 손실을 추적 할 때 효과)

### 3. 악성코드 기초 분석



#### • 02 기초 동적 분석 도구

##### - Wireshark 통계 : Capture File Properties

- 상단 메뉴에 있는 "statistics"을 클릭하면 다양한 통계자료 사용 가능
- 여기서는 몇 가지 통계 정보만 확인

Capture File Properties

Protocol Hierarchy

Conversations

Endpoints

Packet Length

IO Graphs

HTTP

Statistics    Telephony    Wireless    Tools    Help

Capture File Properties    Ctrl+Alt+Shift+C

Resolved Addresses

Protocol Hierarchy

Conversations

Endpoints

Packet Lengths

I/O Graph

Service Response Time

DHCP (BOOTP) Statistics

ONC-RPC Programs

29West

ANCP

BACnet

Collectd

DNS

Flow Graph

HART-IP

HPFEEDS

HTTP

HTTP2

Sametime

TCP Stream Graphs

UDP Multicast Streams

IPv4 Statistics

IPv6 Statistics

### 3. 악성코드 기초 분석



#### • 02 기초 동적 분석 도구

- Wireshark 통계 : Capture File Properties
  - 기본적인 통계 정보
    - » capture 파일 속성
    - » capture 시간
    - » capture 필터 정보
    - » display 필터 정보

Wireshark - Capture File Properties - #2 ftp

Details

File	
Name:	C:\Users\gasbugs\OneDrive\[진행]프로젝트\[온라인]악성코드 네트워크 트래픽 분석\샘플\protocols\#2 ftp.pcap
Length:	472 kB
Format:	Wireshark/tcpdump/... - pcap
Encapsulation:	Linux cooked-mode capture
Snapshot length:	262144

Time	
First packet:	2017-05-03 18:33:41
Last packet:	2017-05-03 18:34:34
Elapsed:	00:00:52

Capture	
Hardware:	Unknown
OS:	Unknown
Application:	Unknown

Interfaces				
Interface	Dropped packets	Capture filter	Link type	Packet size limit
Unknown	Unknown	Unknown	Linux cooked-mode capture	262144 bytes

Statistics			
Measurement	Captured	Displayed	Marked
Packets	1036	1036 (100,0%)	N/A
Time span, s	52,472	52,472	N/A
Average pps	19,7	19,7	N/A
Average packet size, B	440,5	440,5	N/A
Bytes	456072	456072 (100,0%)	0
Average bytes/s	8691	8691	N/A
Average bits/s	69 k	69 k	N/A

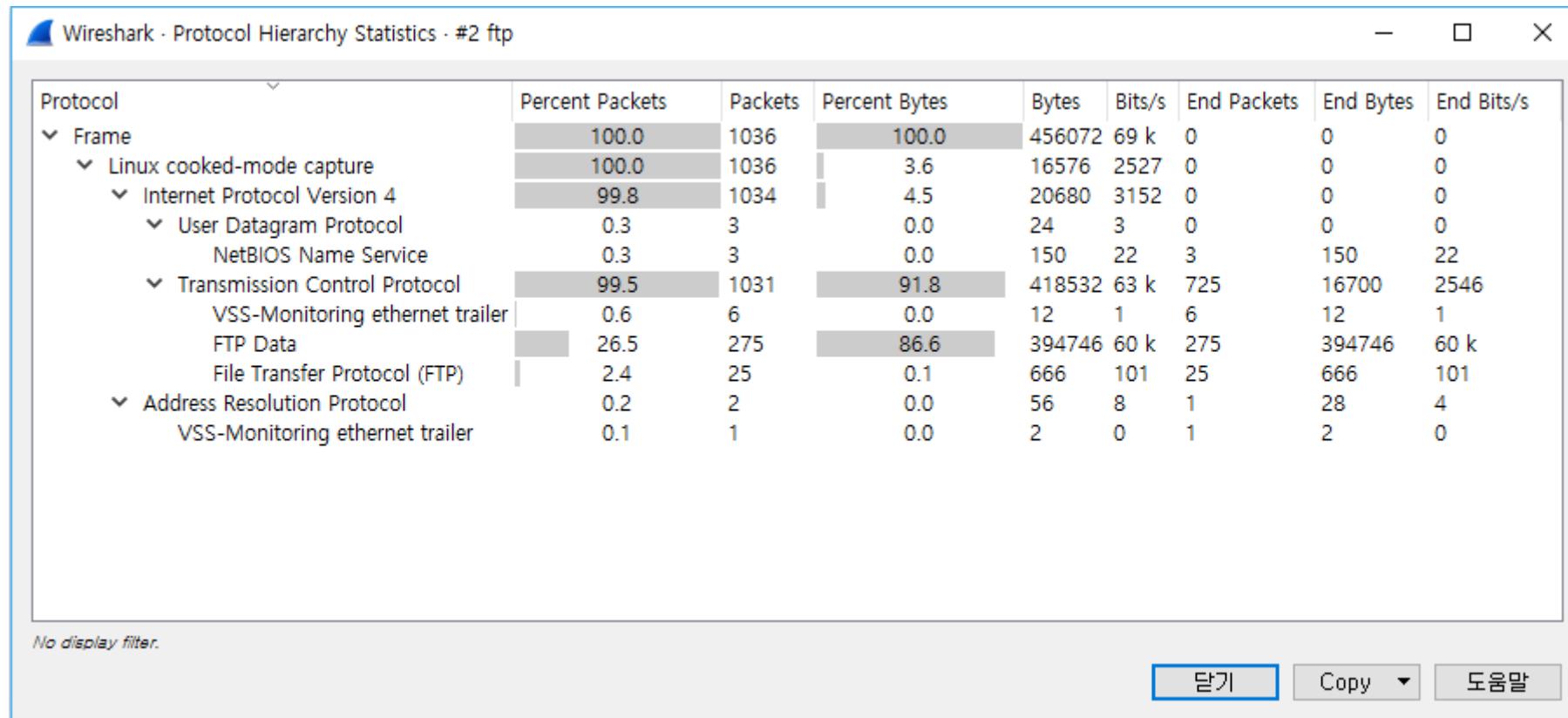
### 3. 악성코드 기초 분석



- 02 기초 동적 분석 도구

- Wireshark 통계 : Protocol Hierarchy

- protocol hierarchy창에서는 각 OSI layer별로 세부적인 데이터를 확인

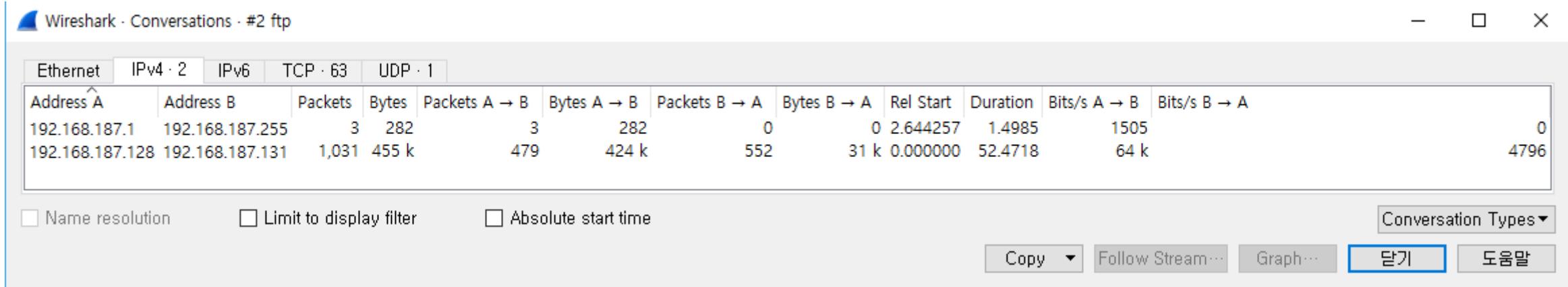


### 3. 악성코드 기초 분석



#### • 02 기초 동적 분석 도구

- Wireshark 통계 : Conversations



- Ethernet, IP, TCP, UDP의 4개의 탭 활성화
- 어떤 두 호스트 사이의 트래픽 통계를 출력
- 각 탭의 프로토콜 명 옆에 있는 숫자는 패킷을 주고 받은 개체의 수를 나타냄



### 3. 악성코드 기초 분석

#### • 02 기초 동적 분석 도구

- Wireshark 통계 : Endpoints

The screenshot shows the Wireshark interface with the title bar "Wireshark · Endpoints · #2 ftp". Below the title bar is a menu bar with tabs: Ethernet, IPv4 · 4, IPv6, TCP · 18, and UDP · 2. The "Ethernet" tab is selected. The main window displays a table titled "Endpoints" with the following columns: Address, Packets, Bytes, Packets A → B, Bytes A → B, Packets B → A, Bytes B → A, Latitude, and Longitude. The table contains four rows of data:

Address	Packets	Bytes	Packets A → B	Bytes A → B	Packets B → A	Bytes B → A	Latitude	Longitude
192.168.187.1	3	282	3	282	0	0	—	—
192.168.187.128	1,031	455 k	479	424 k	552	31 k	—	—
192.168.187.131	1,031	455 k	552	31 k	479	424 k	—	—
192.168.187.255	3	282	0	0	3	282	—	—

At the bottom of the window, there are several buttons: "Name resolution" (unchecked), "Limit to display filter" (unchecked), "Endpoint Types" (dropdown menu), "Copy" (button), "Map" (button), "닫기" (button, highlighted in blue), and "도움말" (button).

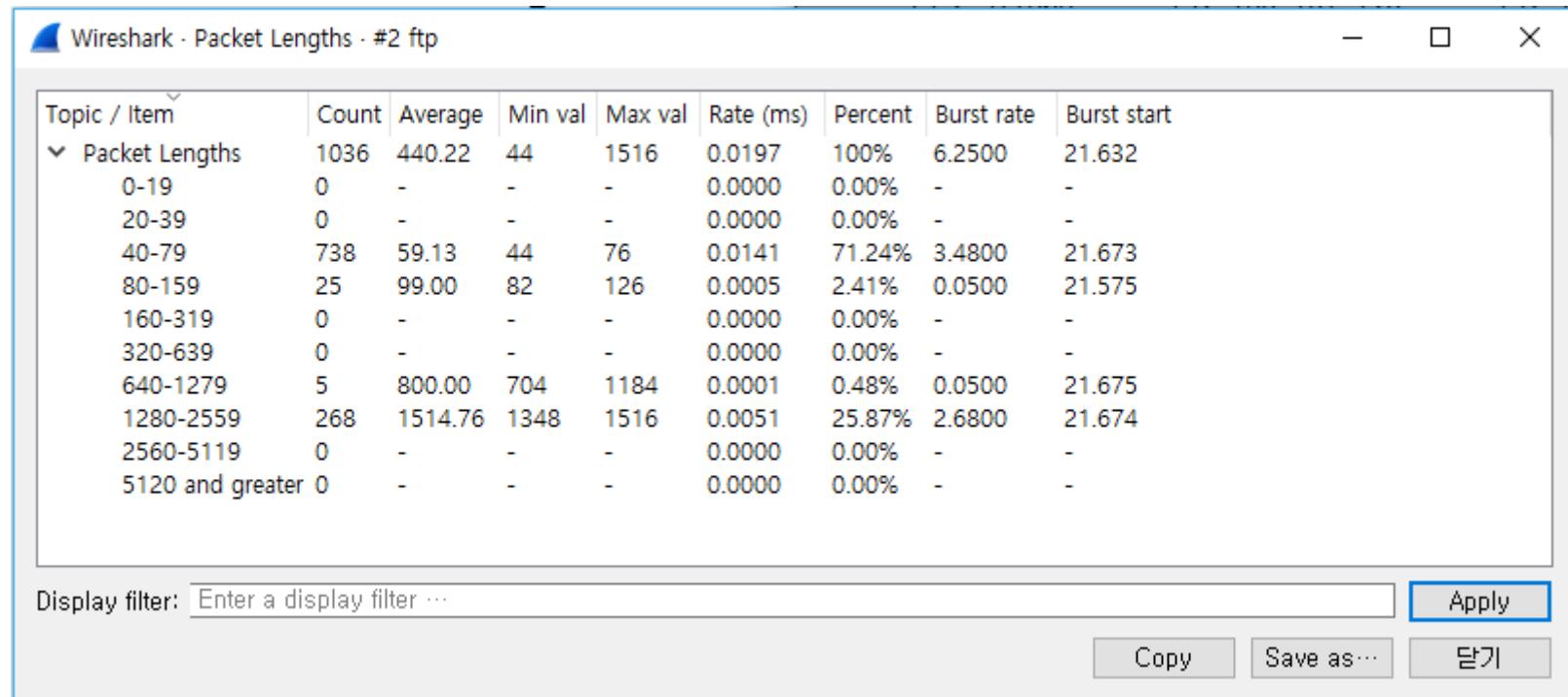
- 각 장치 별로 주고 받은 데이터에 대한 통계 정보
- 탭에서 프로토콜 이름 옆에 있는 숫자는 개체의 수를 나타냄

### 3. 악성코드 기초 분석



- 02 기초 동적 분석 도구

- Wireshark 통계 : Packet Length
  - 패킷 길이에 대한 정보를 출력

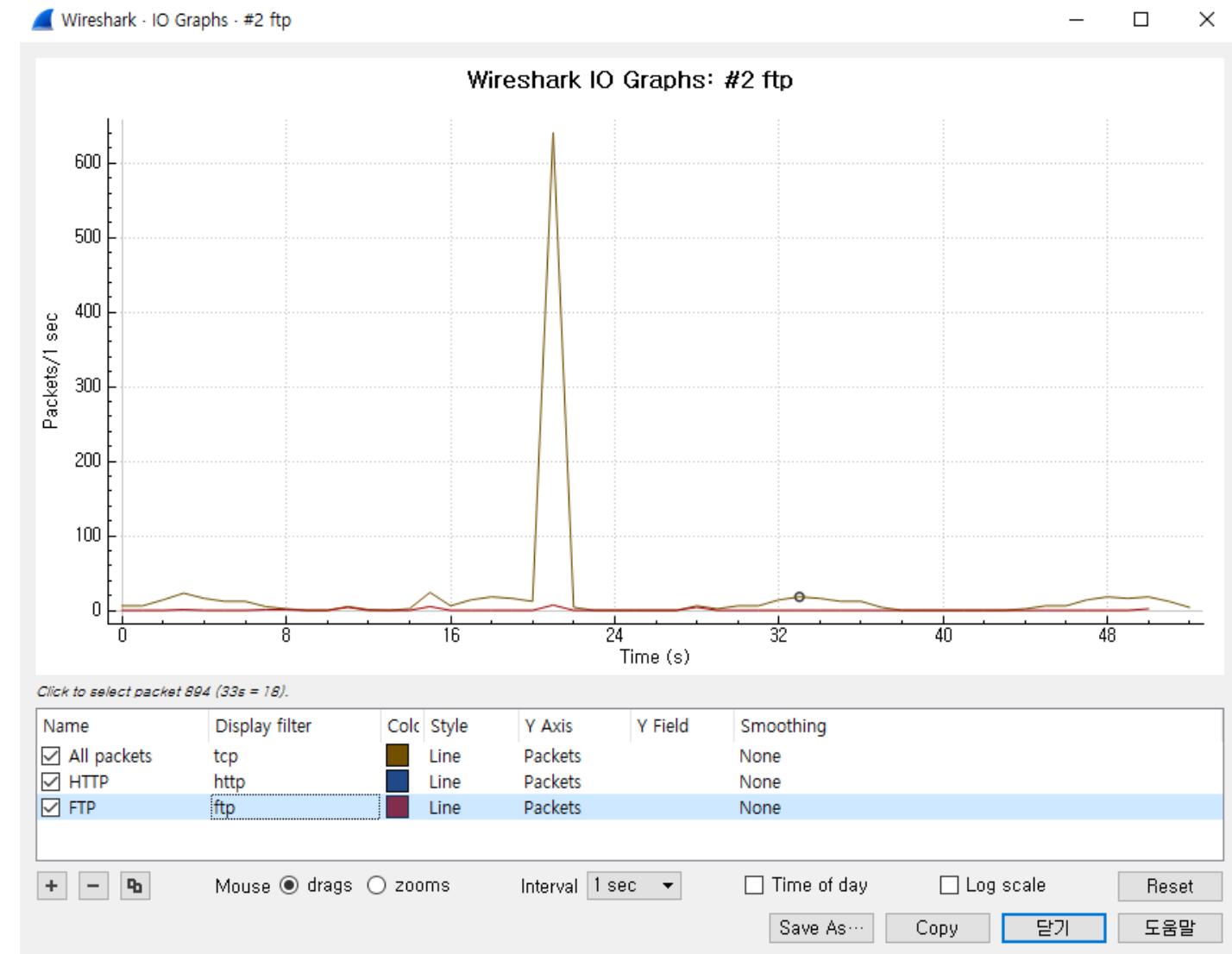


### 3. 악성코드 기초 분석



#### • 02 기초 동적 분석 도구

- Wireshark 통계 : IO Graphs
  - 기본적인 그래프
  - 각 디스플레이 필터 별로 같은 그래프 창 안에 다른 그래프 추가
  - 오른쪽 예제에서는 "tcp", "http", "ftp"를 각각 출력



### 3. 악성코드 기초 분석



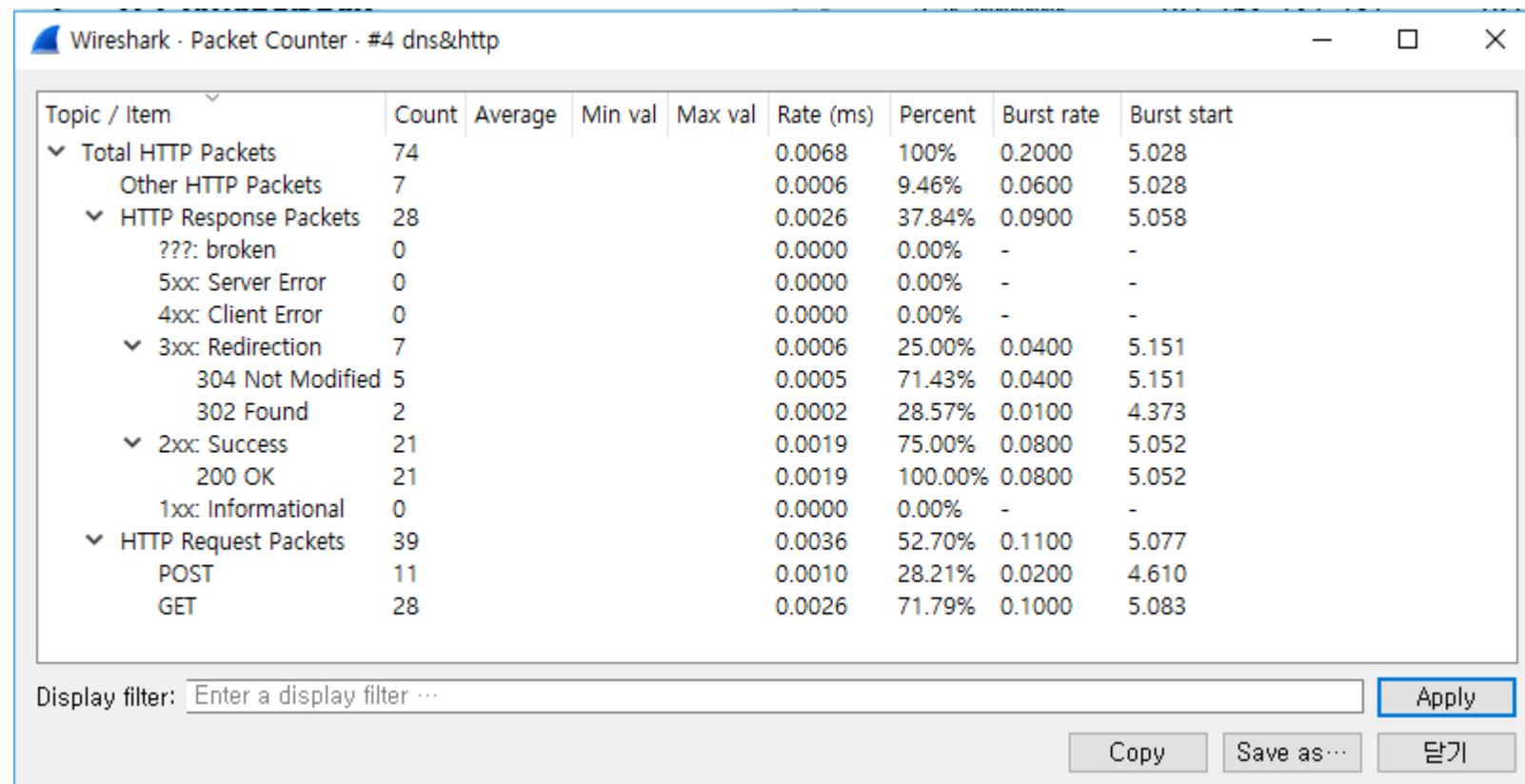
- 02 기초 동적 분석 도구

- Wireshark 통계 : HTTP

- HTTP 분석을 위한 다양한 정보 제공
  - 아래에 세 가지 하위 섹션

- Load Distribution
    - Packet Counter
    - Requests

Packet Counter : HTTP 응답/요청 패킷 통계



### 3. 악성코드 기초 분석



#### • 02 기초 동적 분석 도구

- Wireshark 통계 : HTTP

Wireshark - Requests · #4 dns&http

Topic / Item

- HTTP Requests by HTTP Host
  - www.naver.com
  - /
  - www.daum.net
    - /pub/media.html?dummy=1494086427703
    - /pub/channel.html?dummy=1494260539715
    - /pub/blog.html?dummy=1495246254165
    - /
  - tj.symcd.com
    - /
  - t1.daumcdn.net
    - /daumtop\_chanel/op/20170502024637308.jpg
  - ss.symcd.com
    - /
  - sr.symcd.com
    - /
  - shop3.daumcdn.net
    - /thumb/C220x108/?fname=http%3A%2F%2Fshop2.daumcdn.net%2Fshophow%2Fsib%2F0\_170502180048\_NBcq
  - shop2.daumcdn.net
    - /thumb/C220x108/?fname=http%3A%2F%2Fshop3.daumcdn.net%2Fshophow%2Fsib%2F0\_170502175941\_LFWW

Display filter: Enter a display filter ...

Copy Save as...

Requests : HTTP 요청 내역

Load Distribution : 호스트별 HTTP 통신 분산 통계

Wireshark - Load Distribution · #4 dns&http

Topic / Item	Count	Average	Min val	Max val	Rate (ms)	Percent	Burst rate	Burst start
HTTP Responses by Server Address	28				0.0026	100%	0.0900	5.058
27.0.237.56	5	0.0005	17.86%	0.0400	5.052			
OK	5	0.0005	100.00%	0.0400	5.052			
27.0.237.26	2	0.0002	7.14%	0.0200	4.506			
OK	2	0.0002	100.00%	0.0200	4.506			
23.43.11.27	7	0.0006	25.00%	0.0200	4.738			
OK	7	0.0006	100.00%	0.0200	4.738			
203.217.238.25	1	0.0001	3.57%	0.0100	11.758			
OK	1	0.0001	100.00%	0.0100	11.758			
203.133.172.63	5	0.0005	17.86%	0.0500	5.113			
OK	5	0.0005	100.00%	0.0500	5.113			
203.133.166.55	2	0.0002	7.14%	0.0100	5.136			
OK	2	0.0002	100.00%	0.0100	5.136			
202.179.177.21	1	0.0001	3.57%	0.0100	14.841			
OK	1	0.0001	100.00%	0.0100	14.841			
117.18.237.29	3	0.0003	10.71%	0.0200	5.594			
OK	3	0.0003	100.00%	0.0200	5.594			
114.108.157.50	1	0.0001	3.57%	0.0100	4.373			
OK	1	0.0001	100.00%	0.0100	4.373			
113.29.189.167	1	0.0001	3.57%	0.0100	5.137			
OK	1	0.0001	100.00%	0.0100	5.137			

Display filter: Enter a display filter ...

Copy Save as... Apply 닫기

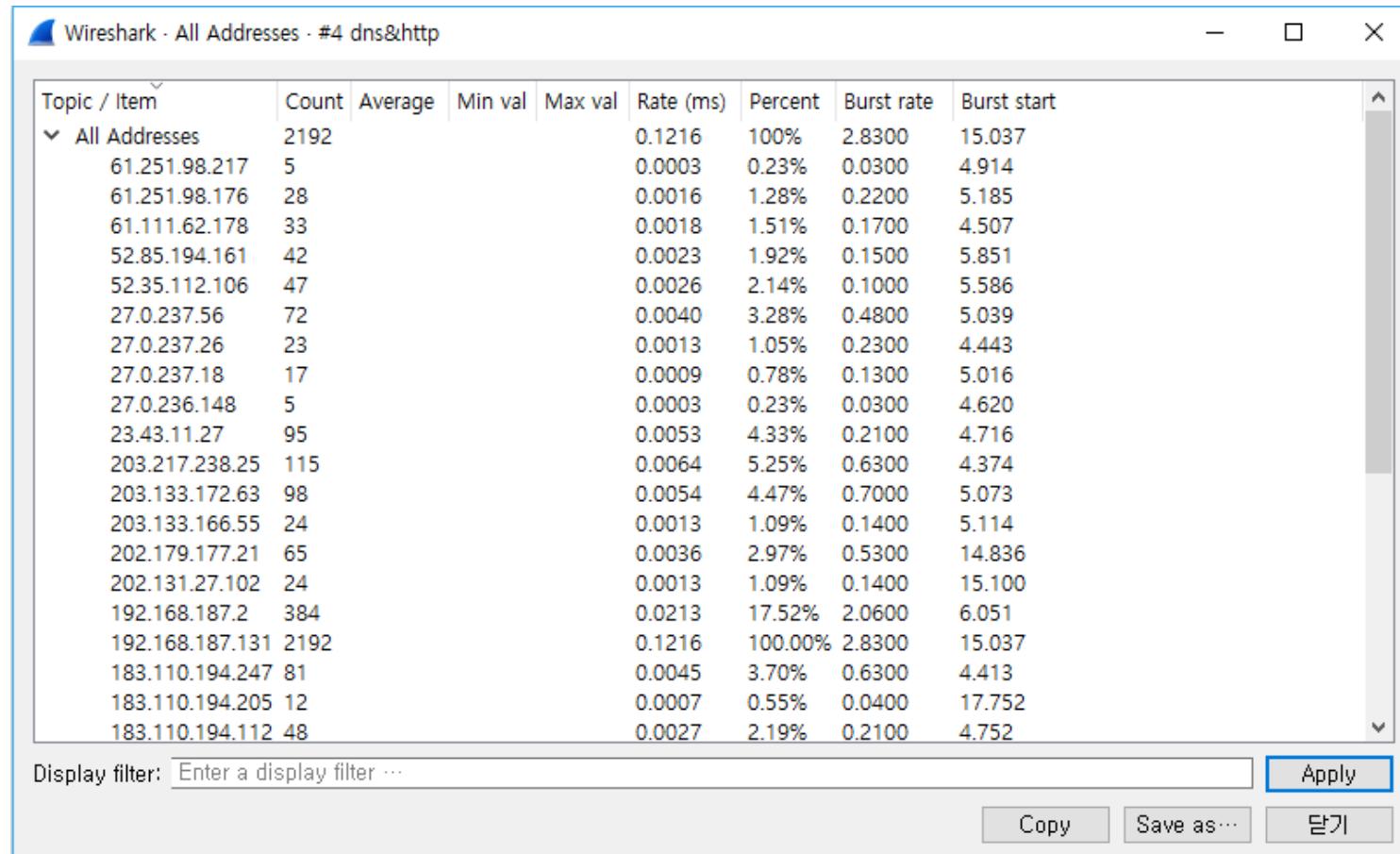
### 3. 악성코드 기초 분석



- 02 기초 동적 분석 도구

- Wireshark 통계 : IPv4 Statistics

- 네트워크 패킷의 출발지, 목적지 IP 주소 통계

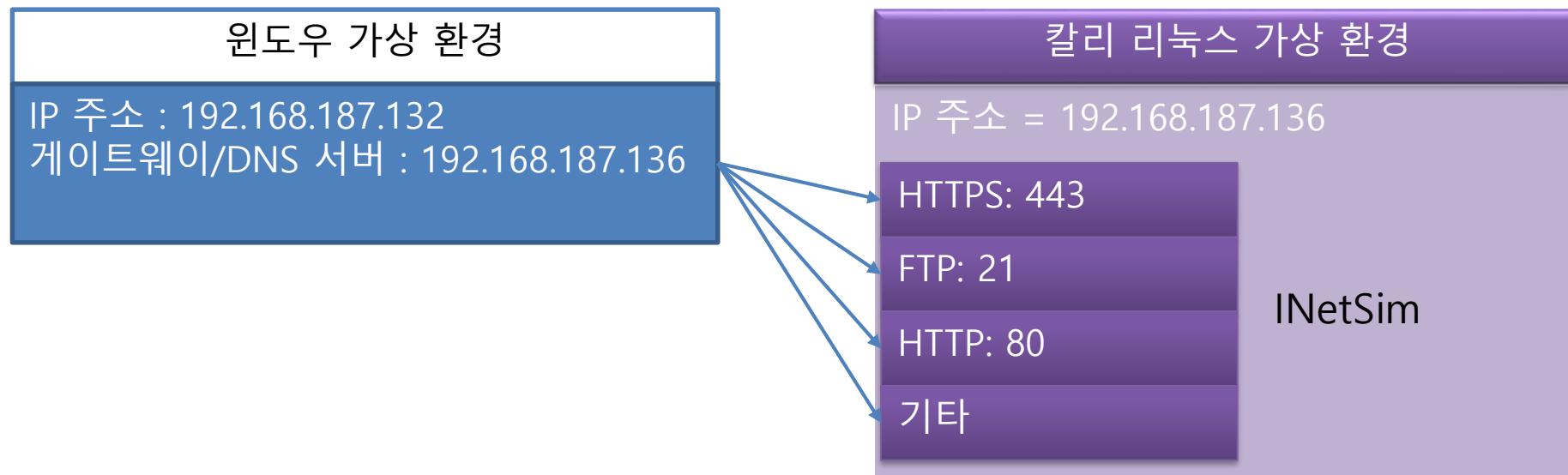


### 3. 악성코드 기초 분석



#### • 02 기초 동적 분석 실습 환경 구성하기

- 네트워크 드라이브 및 INetSim 가상 네트워크 설정 및 실행
- ProcMon을 실행하고 악성코드명으로 필터링 설정
- ProcExp 실행
- Regshot으로 첫 번째 스냅샷 수집
- Wireshark 실행, 패킷 스니핑



### 3. 악성코드 기초 분석



- **실습 3-1**
  - 기초 동적 분석 도구를 이용해 Lab03-01.exe 파일에서 발견된 악성코드를 분석하라.
- **질문**
  - 1. 악성코드의 임포트 함수의 문자열은 무엇인가?
  - 2. 악성코드임을 의미하는 호스트 기반 표시자는 무엇인가?
  - 3. 악성코드를 인식할 수 있는 유용한 네트워크 기반의 시그니처가 존재하는가? 있다면 무엇인가?

### 3. 악성코드 기초 분석



- **실습 3-1**

- 1. 악성코드의 임포트 함수의 문자열은 무엇인가?

The screenshot shows the PEview interface with the file 'Lab03-01.exe' loaded. On the left, the file structure is displayed with nodes like IMAGE\_DOS\_HEADER, MS-DOS Stub Program, IMAGE\_NT\_HEADERS, IMAGE\_SECTION\_HEADER .text, IMAGE\_SECTION\_HEADER .data, SECTION .text, and IMPORT Address Table. The IMPORT Address Table node is highlighted with a blue selection box. On the right, a table lists the imports:

pFile	Data	Description	Value
00000200	0000024C	Hint/Name RVA	0080 ExitProcess
00000204	00000000	End of Imports	kernel32.dll

### 3. 악성코드 기초 분석



- **실습 3-1**

- 2. 악성코드임을 의미하는 호스트 기반 표시자는 무엇인가?
- 3. 악성코드를 인식할 수 있는 유용한 네트워크 기반의 시그니처가 존재하는가? 있다면 무엇인가?

#### Lab03-01.exe Strings

```
CONNECT %s:%i HTTP/1.0
[중략]
StubPath
SOFTWARE\Classes\http\shell\open\command\
Software\Microsoft\Active Setup\Installed Components\
test
www.practicalmalwareanalysis.com
admin
VideoDriver
WinVMX32-
vmx32to64.exe
U
SOFTWARE\Microsoft\Windows\CurrentVersion\Run
Ph?
V5h
V1?
V)V
U
SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\Shell Folders
```

### 3. 악성코드 기초 분석



- **실습 3-2**
  - 기초 동적 분석 도구를 이용해 Lab03-02.exe 파일에서 발견된 악성코드를 분석하라.
- **질문**
  - 1. 악성코드 자체가 어떻게 설치되었는가?
  - 2. 설치 후 악성코드를 어떻게 실행할 수 있는가?
  - 3. 악성코드가 동작할 때 어떤 프로세스를 발견할 수 있는가?
  - 4. 정보를 수집하는 ProcMon을 사용하기 위해 어떤 필터를 설정했는가?
  - 5. 악성코드임을 의미하는 호스트 기반 표시자는 무엇인가?
  - 6. 악성코드에서 유용한 네트워크 기반 시그니처가 존재하는가?

### 3. 악성코드 기초 분석



- **실습 3-3**
  - 기초 동적 분석 도구를 이용해 Lab03-03.exe 파일에서 발견된 악성코드를 분석하라.
- **질문**
  - 1. Process Explorer로 이 악성코드를 모니터링했을 때 무엇을 알아냈는가?
  - 2. 실시간 메모리 변조를 확인할 수 있는가?
  - 3. 악성코드임을 의미하는 호스트 기반 표시자는 무엇인가?
  - 4. 이 프로그램의 목적은 무엇인가?

### 3. 악성코드 기초 분석



- **실습 3-4**
  - 기초 동적 분석 도구를 이용해 Lab03-04.exe 파일에서 발견된 악성코드를 분석하라.
  - (이 프로그램은 나중에 9장에서 실습에서 분석한다)
- **질문**
  - 1. 이 파일을 실행했을 때 어떤 일이 발생했는가?
  - 2. 동적 분석 시 장애물이 무엇인가?
  - 3. 이 파일을 실행시키는 다른 방법이 있는가?



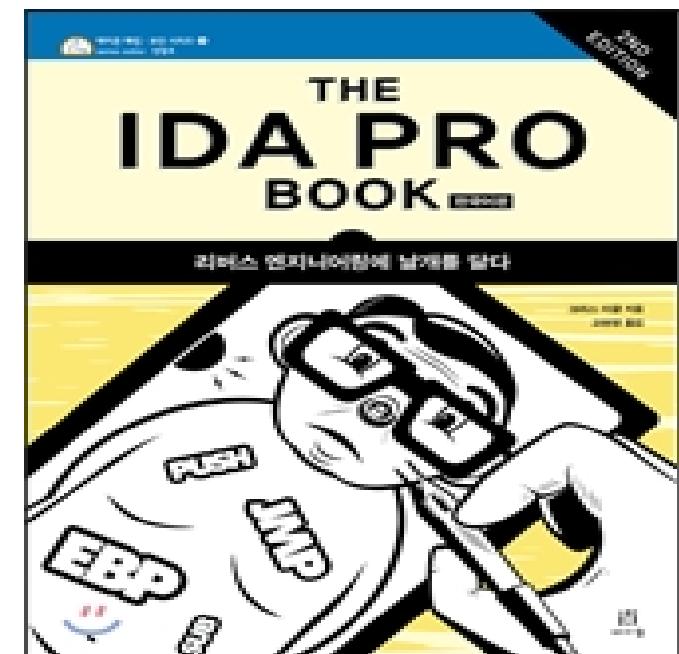
## 4. 악성코드 고급 정적 분석: 아이다 활용



## 4. 악성코드 고급 정적 분석: 아이다 활용

- ✓ 헥스레이(Hex-Rays) 사에서 배포
- ✓ PE(Portable Executable), COFF(Common Object File Format, 유닉스 공용 라이브러리 포맷), ELF(Executable and Linking Format, 유닉스용 실행 파일 포맷) 지원
- ✓ x86 / x64 지원
- ✓ 함수 발견, 스택 분석, 지역 변수 확인 등 수 많은 기능 제공
- ✓ 과정의 모든 부분을 수정, 조작, 재배치, 재정의
- ✓ 분석 진행 상황 저장 기능(주석 작성, 라벨링, 함수 이름 붙이기)
- ✓ 막강한 플러그인 지원

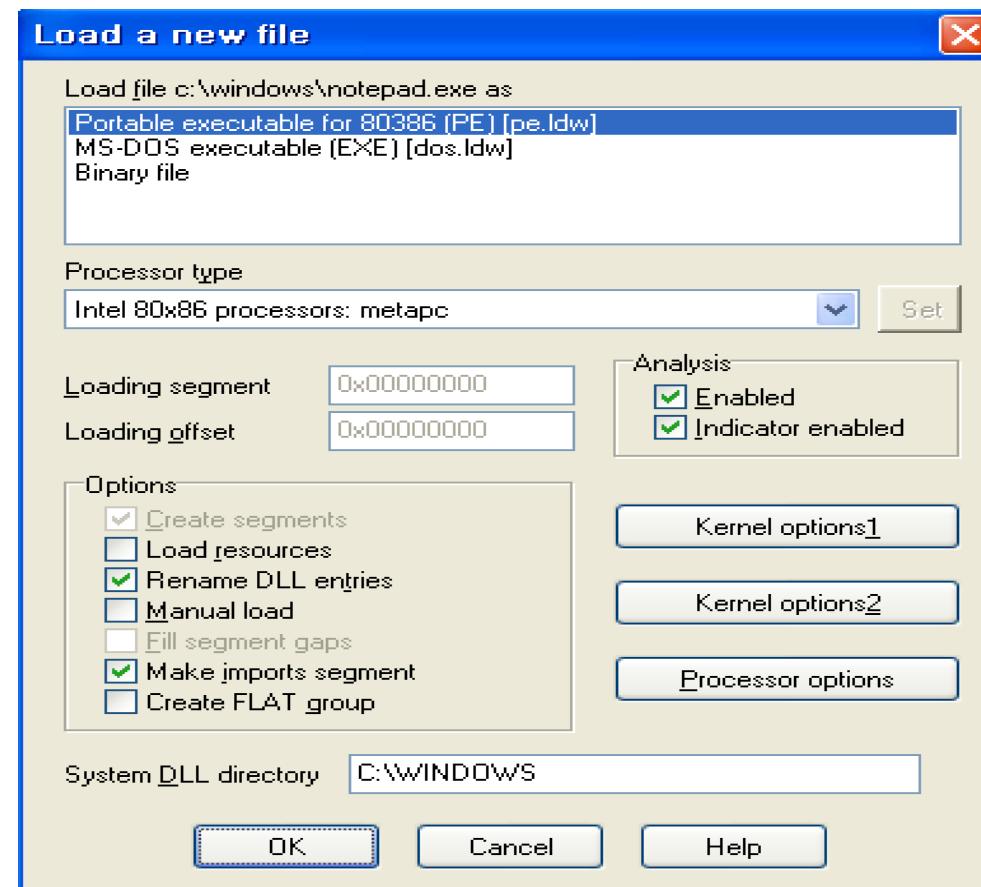
The screenshot shows the Hex-Rays website with a dark background featuring blue light rays. At the top, there's a navigation bar with links: IDA, Overview, News, Processors, Formats, Debuggers, Tech, Sales, and Support. Below the navigation, the text "Hex-Rays Home > IDA" is displayed. The main title "Hex-Rays" is in a large, stylized font, followed by "IDA: About". Underneath, there's a section titled "IDA: About" with the sub-section "What is IDA all about?". A paragraph describes IDA as a multi-processor disassembler and debugger with many features. At the bottom, it says "An executive summary is provided for the non-technical user."





## 4. 악성코드 고급 정적 분석: 아이다 활용

1. PE/MS-Dos/Binary 파일 선택
2. x86 / x64 선택
3. Rebase 시 메모리 로드 가상 주소 설정 가능

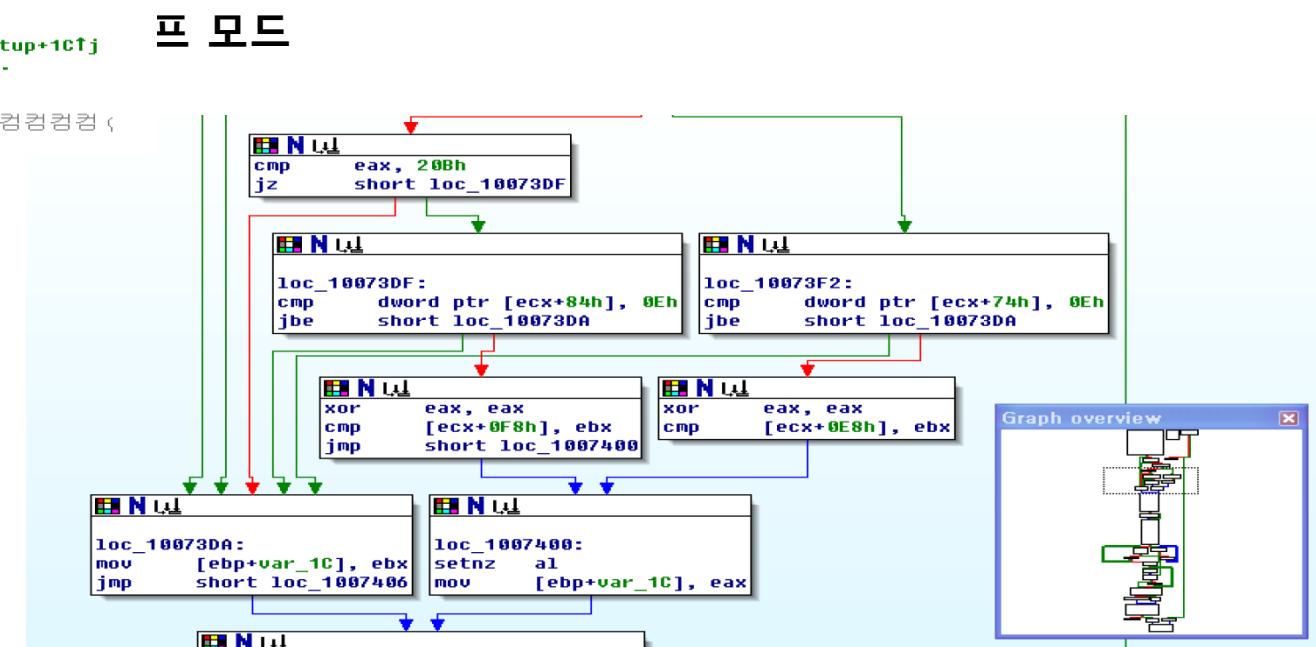


## 4. 악성코드 고급 정적 분석: 아이다 활용



**디스어셈블리 윈도우 모드 (스페이스 바 키로 전환)**

텍스트 모드





## 4. 악성코드 고급 정적 분석: 아이다 활용

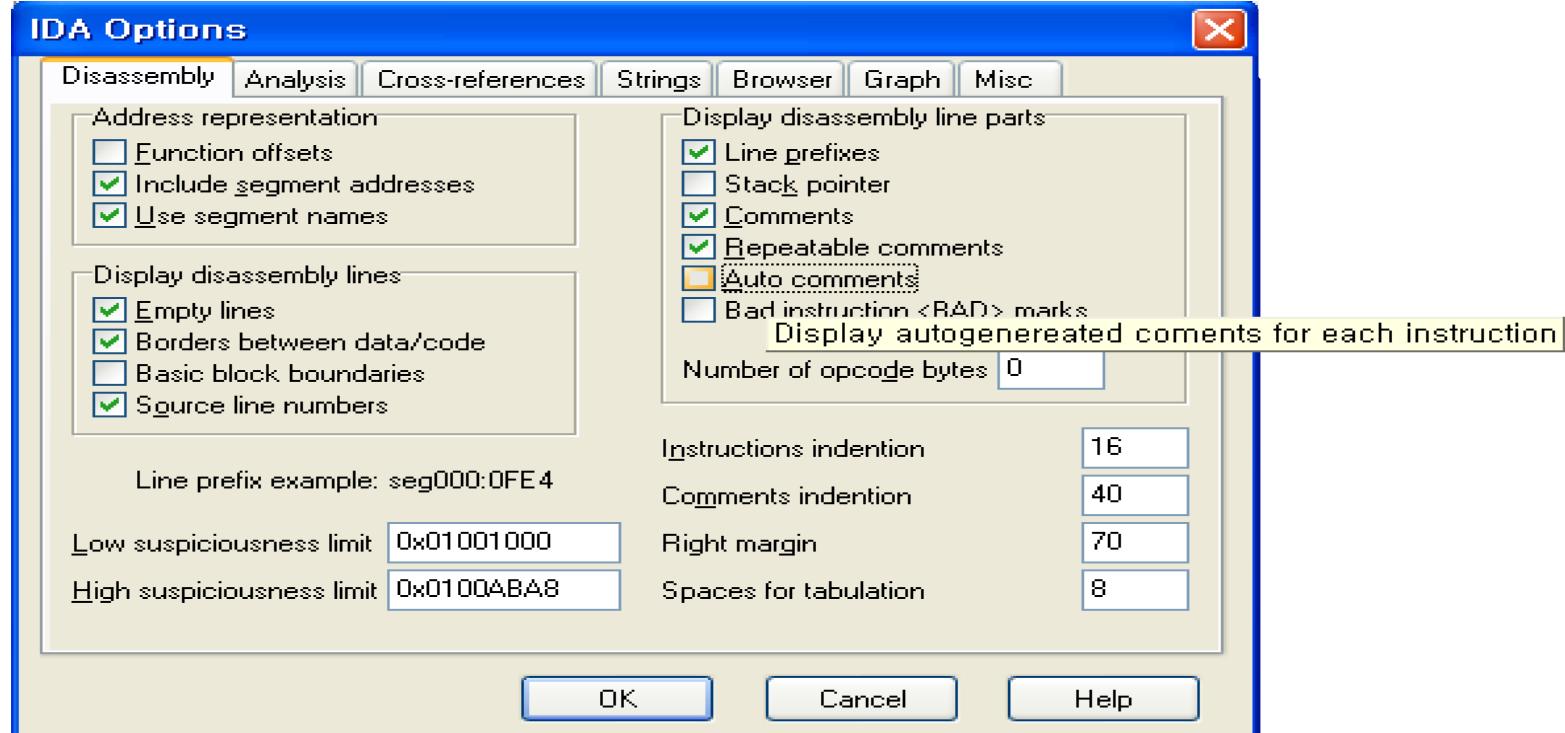
### 그래프 모드 vs 텍스트 모드 특징 비교

그래프 모드	텍스트 모드
<ul style="list-style-type: none"><li>• 그래프 형태로 출력</li><li>• 화살표 색깔과 방향을 이용 → 분석 도중 프로그램의 흐름 파악</li><li>• 빨강 : 조건점프 거짓 시</li><li>• 초록 : 조건점프 참 시</li><li>• 파랑 : 무조건 점프 시</li></ul>	<ul style="list-style-type: none"><li>• 기존의 보기 방식</li><li>• 메모리 주소, 옵코드, 섹션명 등을 출력</li><li>• 실선 : 무조건 점프</li><li>• 점선 : 조건 점프</li></ul>



# 4. 악성코드 고급 정적 분석: 아이다 활용

## Auto Comments (Options > General)



```
rd_1001898
og ; int
; Logical Exclusive OR
; lpModuleName
imp__GetModuleHandleA@4 ; GetModuleHandleA(x)
oduleHandleA(x) ; GetModuleHandleA(x)

word ptr [eax], 5A4Dh ; Compare Two Operands
short loc_10073DA ; Jump if Not Zero (ZF=0)
ecx, [eax+3Ch]
add
ecx, eax ; Add
dword ptr [ecx], 4550h ; Compare Two Operands
short loc_10073DA ; Jump if Not Zero (ZF=0)
movzx
eax, word ptr [ecx+18h] ; Move with Zero-Extend
cmp
eax, 10Bh ; Compare Two Operands
jz
short loc_10073F2 ; Jump if Zero (ZF=1)
cmp
eax, 20Bh ; Compare Two Operands
jz
short loc_10073DF ; Jump if Zero (ZF=1)
```



## 4. 악성코드 고급 정적 분석: 아이다 활용

### 분석에 유용한 윈도우

윈도우	설명
함수 윈도우(Function Window)	실행 파일 내의 모든 함수를 목록화
이름 윈도우(Names Window)	함수, 명명된 코드, 명명된 데이터와 문자열을 포함해 이름 관련 모든 주소를 목록화
문자열 윈도우(Strings Window)	5자 이상의 ASCII 문자열 출력 (문자열 윈도우의 Setup에서 설정 변경 가능)
임포트 윈도우(Imports Window)	임포트 되는 모든 함수 목록
익스포트 윈도우(Export Window)	익스포트 되는 모든 함수 목록 (DLL분석 시 용이)
구조체 윈도우(Structures Window)	데이터 구조 레이아웃 목록



## 4. 악성코드 고급 정적 분석: 아이다 활용

### 원래 보기로 되돌리기

윈도우	설명
함수 윈도우(Function Window)	실행 파일 내의 모든 함수를 목록화
이름 윈도우(Names Window)	함수, 명명된 코드, 명명된 데이터와 문자열을 포함해 이름 관련 모든 주소를 목록화
문자열 윈도우(Strings Window)	5자 이상의 ASCII 문자열 출력 (문자열 윈도우의 Setup에서 설정 변경 가능)
임포트 윈도우(Imports Window)	임포트 되는 모든 함수 목록
익스포트 윈도우(Export Window)	익스포트 되는 모든 함수 목록 (DLL분석 시 용이)
구조체 윈도우(Structures Window)	데이터 구조 레이아웃 목록

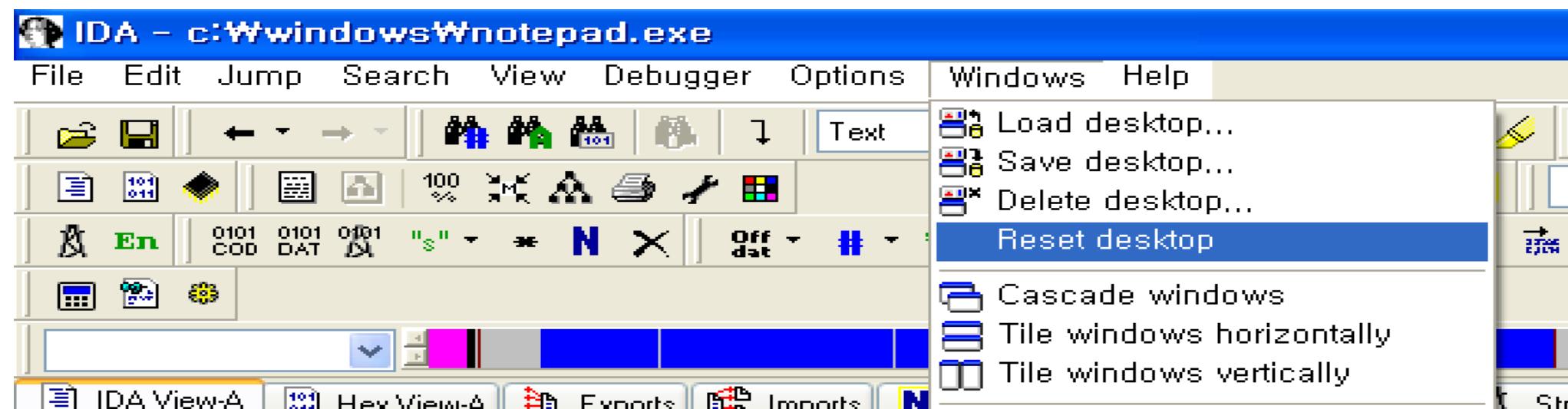


## 4. 악성코드 고급 정적 분석: 아이다 활용

원래 보기로 되돌리기 : Windows > Reset Desktop

→ 윈도우와 GUI 인터페이스만 기본으로 복구

→ 원하는 보기 저장 시 : Windows > Save Desktop



## 4. 악성코드 고급 정적 분석: 아이다 활용



## IDA Pro 탐색 > 링크와 상호 참조 사용

- Sub : 함수 시작 링크
  - Loc : 목적지로 점프하는 링크
  - Offset : 메모리 내의 오프셋 링크



## 4. 악성코드 고급 정적 분석: 아이다 활용

### 검색

- Search > Next Code

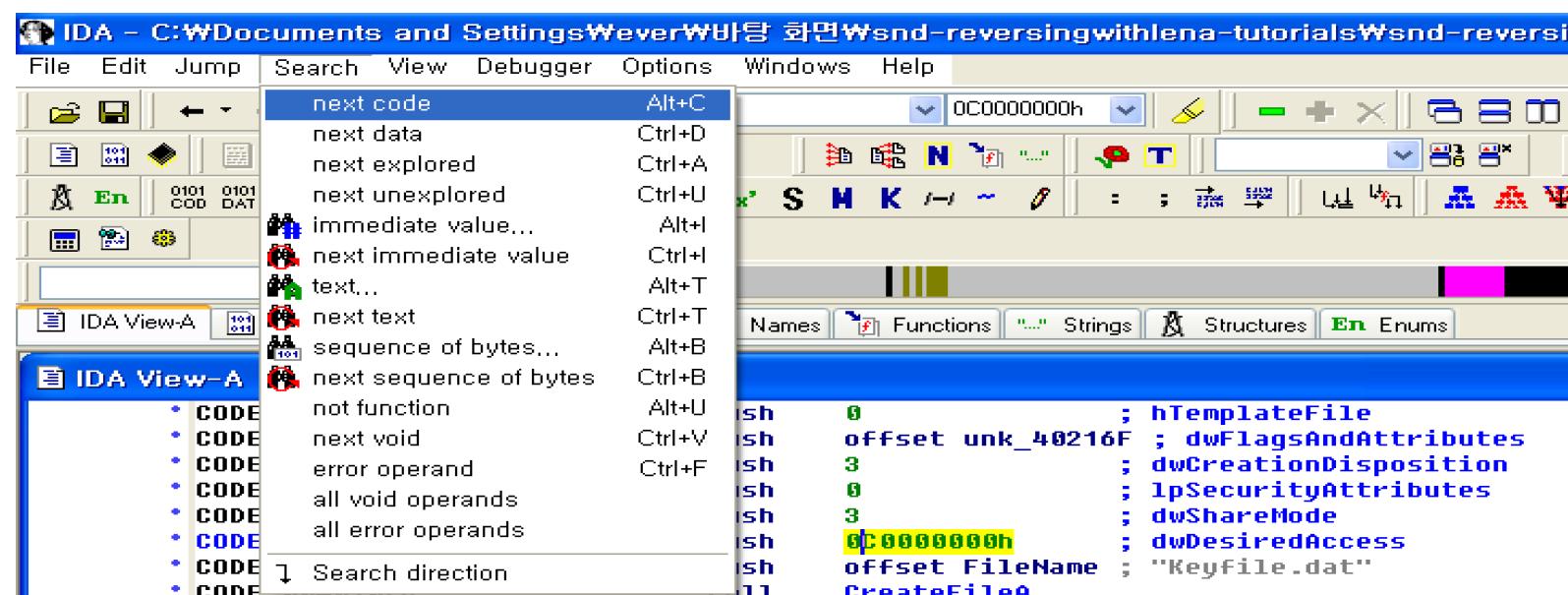
저장한 명령어를 담고 있는 다음 위치 커서로 이동

- Search > Text

전체 디스어셈블리 윈도우에서 특정 문자열 검색

- Search > Sequence of Bytes

특정 바이트 순서로 16진수로 보기 윈도우에 있는 바이너리 검색





## 4. 악성코드 고급 정적 분석: 아이다 활용

1. xref로 표기
2. 함수를 호출한 위치나 사용한 문자열 위치를 알림
3. 함수의 호출된 파라미터로 신속히 이동

Call stack:

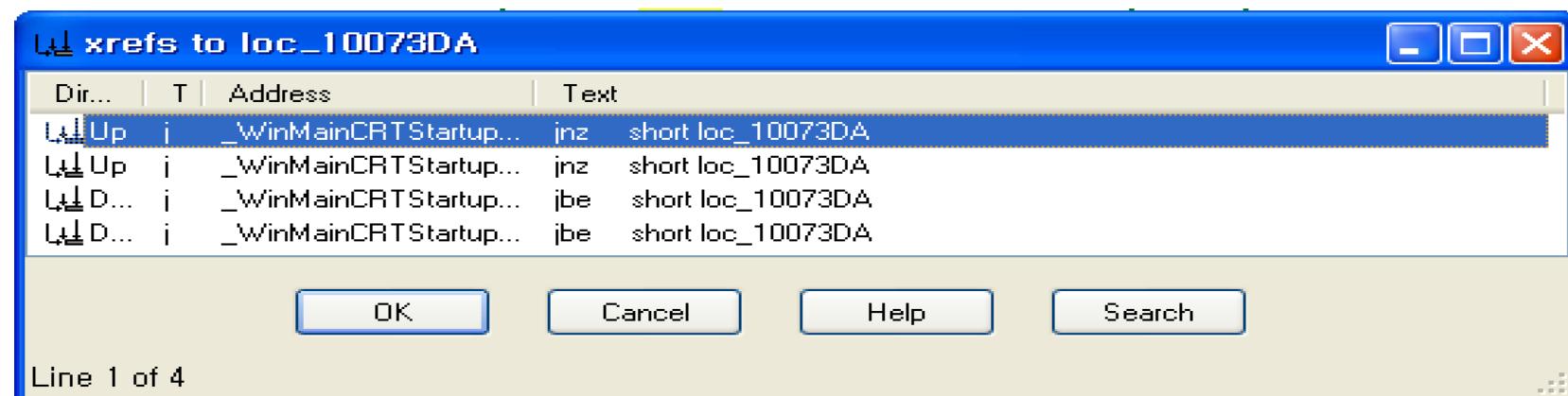
- .text:010073DA
- .text:010073DA loc\_10073DA:
- .text:010073DA
- .text:010073DA
- .text:010073DD

Assembly code:

```
mov    [ebp+var_1C], ebx
jmp    short loc_1007406
```

Annotations:

- ; CODE XREF: \_WinMainCRTStartup+1C↑j
- ; \_WinMainCRTStartup+29↑j ...





## 4. 악성코드 고급 정적 분석: 아이다 활용

- 함수를 인식하고 이름을 기입하여 지역 변수와 파라미터로 분리하는 기능

```
; Attributes: bp-based frame

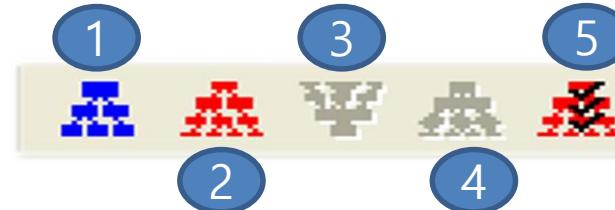
; BOOL __stdcall DllEntryPoint(HINSTANCE hinstDLL, DWORD fdwReason,
public DllEntryPoint
DllEntryPoint proc near

var_424= dword ptr -424h      지역변수
var_420= dword ptr -420h
var_41C= dword ptr -41Ch
var_418= dword ptr -418h
var_414= dword ptr -414h
var_40D= dword ptr -40Dh
var_204= word ptr -204h
var_4= dword ptr -4
hinstDLL= dword ptr 8          파라미터
fdwReason= dword ptr 0Ch
lpReserved= dword ptr 10h
```



## 4. 악성코드 고급 정적 분석: 아이다 활용

### 그래프 옵션

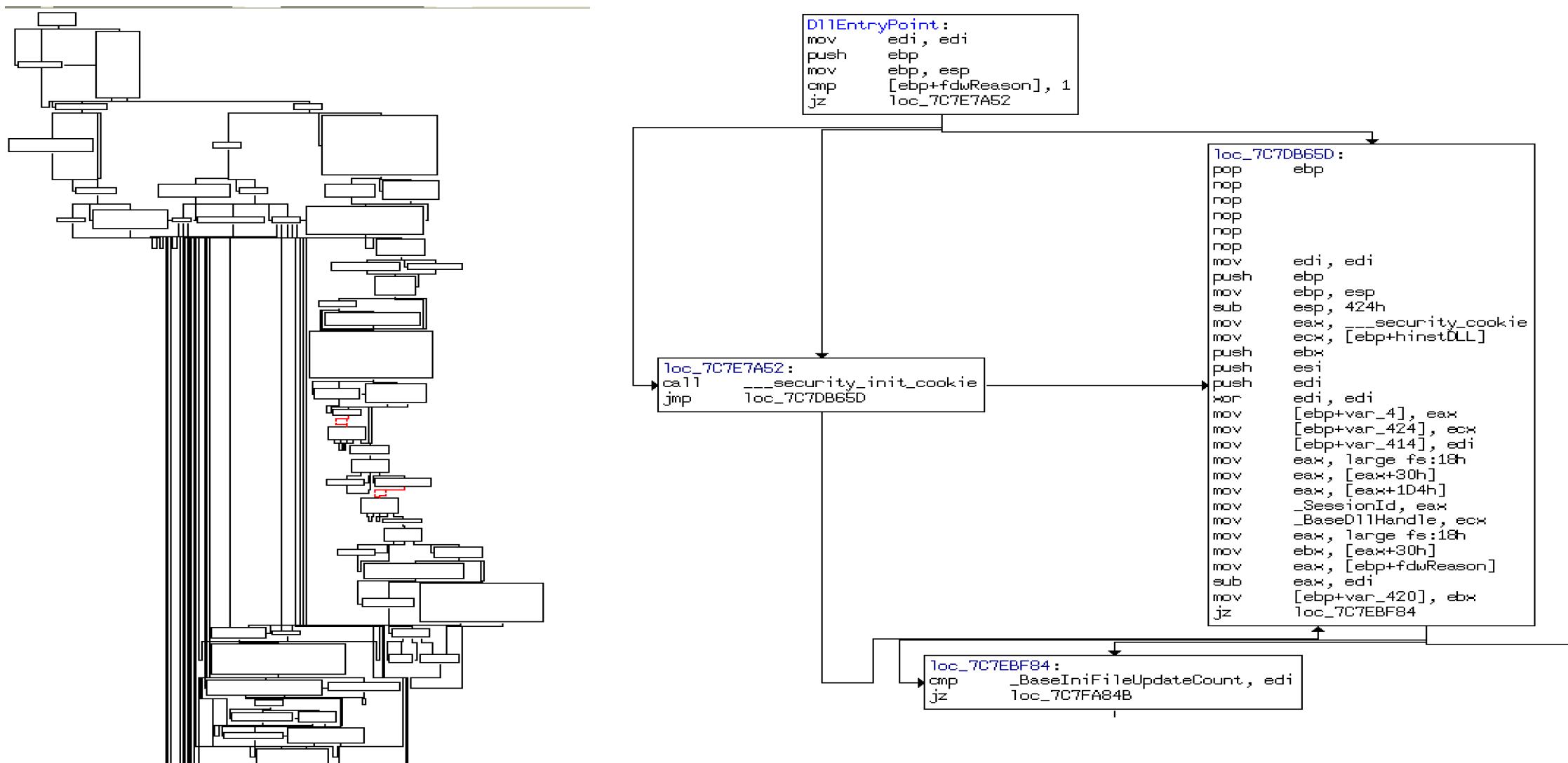


번호	기능
1	현재 함수의 플로우 차트 작성
2	전체 프로그램의 그래프 함수 호출
3	현재 선택한 상호 참조를 알아낼 수 있게 상호 참조 그래프 작성
4	현재 선택한 심볼에서 상호 참조 그래프 작성
5	사용자가 정의한 상호 참조 그래프 작성



# 4. 악성코드 고급 정적 분석: 아이다 활용

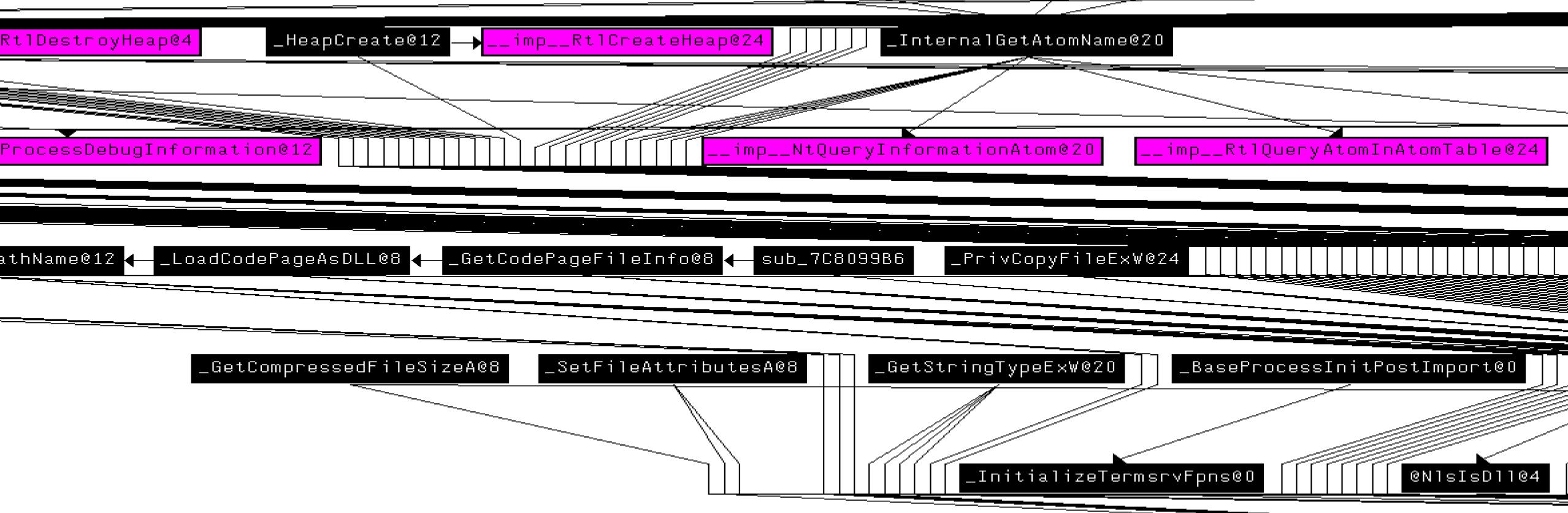
## 1. 현재 함수의 플로우차트 작성





# 4. 악성코드 고급 정적 분석: 아이다 활용

## 2. 전체 프로그램의 그래프 함수 호출





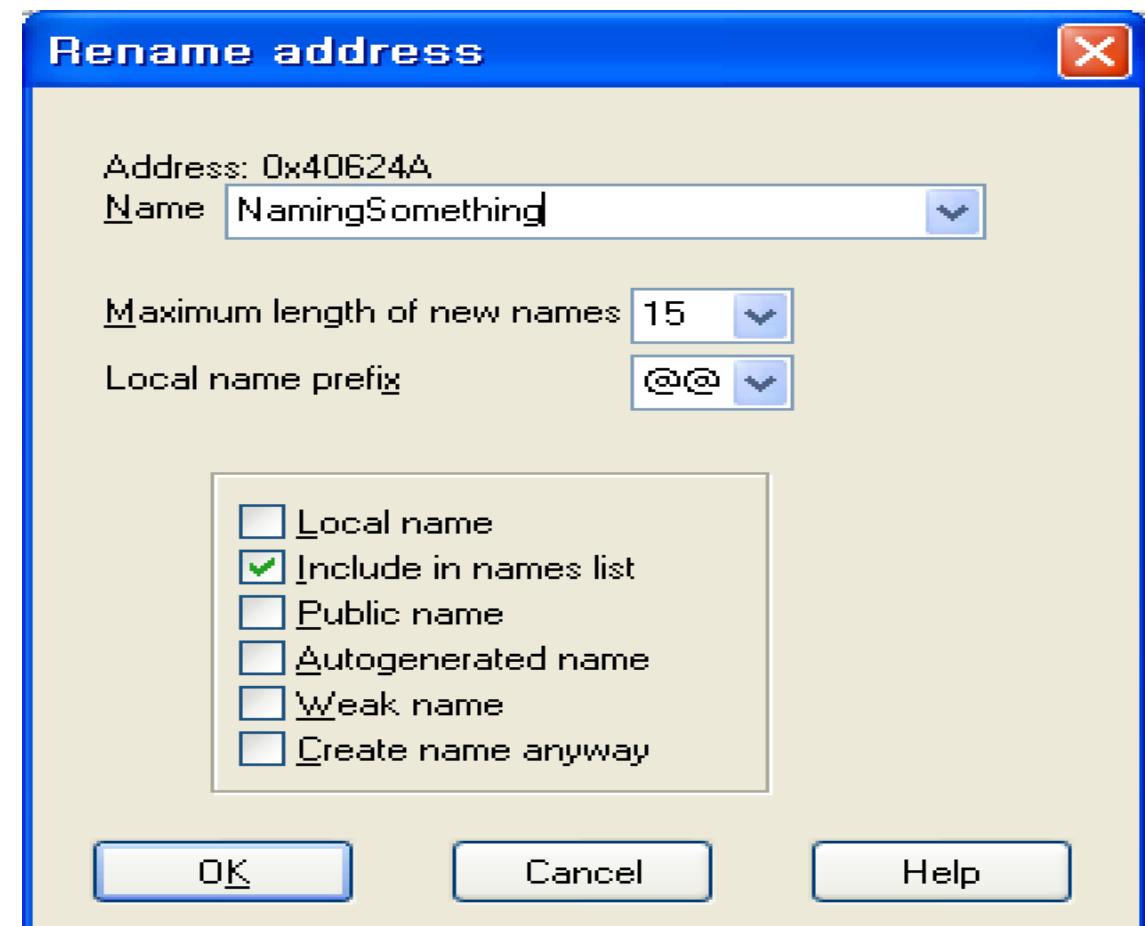
## 4. 악성코드 고급 정적 분석: 아이디 활용

- 함수 이름 변경

IDA는 가상 주소와 스택 변수에 자동으로 이름을 정하지만 분석가가 임의대로 더 좋은 이름으로 수정할 수 있다.

- Ex)

`sub_40624A → NamingSomething`





## 4. 악성코드 고급 정적 분석: 아이다 활용

- 주석

세미콜론(;)을 활용하여 주석을 추가, 자동 주석 추가 기능

- 오퍼랜드 포맷

The screenshot shows the IDA View-A window displaying assembly code. A specific line of code, ".text:00403C53 var\_10", is highlighted in yellow. A modal dialog box is overlaid on the window, prompting the user to "Please enter a string". Inside the dialog, there is a text input field containing the value "var\_10". Below the input field are three buttons: "OK", "Cancel", and "Help".

```
.text:00403C53 uExitCode      = dword ptr -48h
.text:00403C53 var_20         = dword ptr -2Ch
.text:00403C53 var_28         = dword ptr -28h
.text:00403C53 lpText          = dword ptr -20h
.text:00403C53 var_10         = byte ptr -1Ch
.text:00403C53 var_18         = dword ptr -18h
.text:00403C53 lpString2       = dword ptr -14h
```



## 4. 악성코드 고급 정적 분석: 아이다 활용

- 이름 있는 상수 사용

CreateFileA의 인자 값 설정

```
.text:00409325      push    0
.text:00409327      push    0
.text:00409329      push    3
.text:0040932B      push    0
.text:0040932D      push    3
.text:0040932F      push    80000000h
.text:00409334      push    ecx
.text:00409335      call    ds:CreateFileA

.text:00409325      push    0
.text:00409327      push    0
.text:00409329      push    OPEN_EXISTING
.text:0040932B      push    0
.text:0040932D      push    3          ; FILE_SHARE_READ & FILE_SHARE_WRITE
.text:0040932F      push    80000000h
.text:00409334      push    ecx
.text:00409335      call    ds:CreateFileA
```

- 원하는 특정 표준 심볼 상수가 보이지 않을 때

→ View > Open Subviews > Type Libraries 를 사용하여 수동으로 로딩



## 4. 악성코드 고급 정적 분석: 아이다 활용

실습 5-1 IDA Pro만을 이용해 파일 Lab05-01.dll 내의 악성코드를 분석하라. 이 실습의 목적은 IDA Pro를 직접 다루는 데 있다. 이미 IDA Pro를 사용해 본 적이 있으면 다음 문제를 무시하고 악성코드 리버싱에 초점을 맞춰도 좋다.



## 4. 악성코드 고급 정적 분석: 아이다 활용

1. DllMain의 주소는 무엇인가?

```
.text:10000D02E ; ===== S U B R O U T I N E =====
.text:10000D02E
.text:10000D02E
.text:10000D02E ; BOOL __stdcall DllMain(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID
.text:10000D02E _DllMain@12 proc near ; CODE XREF: DllEntryPoint
.text:10000D02E ; DATA XREF: sub_100110FF+
.text:10000D02E
.text:10000D02E hinstDLL      = dword ptr  4
.text:10000D02E fdwReason     = dword ptr  8
.text:10000D02E lpvReserved   = dword ptr  0Ch
.text:10000D02E
.text:10000D02E         mov    eax, [esp+fdwReason]
.text:10000D032         dec    eax
.text:10000D033         jnz    loc_10000D107
.text:10000D039         mov    eax, [esp+hinstDLL]
.text:10000D03D         push   ebx
.text:10000D03E         mov    ds:hModule, eax
```



## 4. 악성코드 고급 정적 분석: 아이다 활용

2. Imports 윈도우를 이용해 gethostbyname을 탐색해보자. 임포트 위치는 어디인가?

100162A0	fwrite	MSVCRT
100163CC	52 gethostbyname	WS2_32
100163E4	9 htons	WS2_32
100163F0	11 socket	WS2_32



## 4. 악성코드 고급 정적 분석: 아이다 활용

3. gethostbyname에 함수는 몇 개인가?

xrefs to gethostbyname				
Dir...	T	Address	Text	
Up	p	sub_10001074:loc_1...	call ds:gethostbyname	
Up	p	sub_10001074+1D3	call ds:gethostbyname	
Up	p	sub_10001074+26B	call ds:gethostbyname	
Up	p	sub_10001365:loc_1...	call ds:gethostbyname	
Up	p	sub_10001365+1D3	call ds:gethostbyname	
Up	p	sub_10001365+26B	call ds:gethostbyname	
Up	p	sub_10001656+101	call ds:gethostbyname	
Up	p	sub_1000208F+3A1	call ds:gethostbyname	
Up	p	sub_10002CCE+4F7	call ds:gethostbyname	
Up	r	sub_10001074:loc_1...	call ds:gethostbyname	
Up	r	sub_10001074+1D3	call ds:gethostbyname	
Up	r	sub_10001074+26B	call ds:gethostbyname	
Up	r	sub_10001365:loc_1...	call ds:gethostbyname	
Up	r	sub_10001365+1D3	call ds:gethostbyname	
Up	r	sub_10001365+26B	call ds:gethostbyname	
Up	r	sub_10001656+101	call ds:gethostbyname	
Up	r	sub_1000208F+3A1	call ds:gethostbyname	
Up	r	sub_10002CCE+4F7	call ds:gethostbyname	



## 4. 악성코드 고급 정적 분석: 아이다 활용

4. 0x10001757에 위치한 gethostbyname 호출을 보면 어떤 DNS 요청이 이뤄지는지 알 수 있는가?

10019191	20 00 00 5B 54 68 69 73 20 69 73 20 52 44 4F 5D	..-[This is RDO]
100191A1	70 69 63 73 2E 70 72 61 74 69 63 61 6C 6D 61 6C	pics.practicalmal
100191B1	77 61 72 65 61 6E 61 6C 79 73 69 73 2E 63 6F 6D	wareanalysis.com
100191C1	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	- - - - -

```
• |.text:10001737           add    esp, 0Ch
• |.text:1000173A           test   eax, eax
• |.text:1000173C           jz    loc_100017ED
• |.text:10001742           cmp    dword_1008E5CC, ebx
• |.text:10001748           jnz   loc_100017ED
• |.text:1000174E           mov    eax, off_10019040
• |.text:10001753           add    eax, 0Dh
• |.text:10001756           push   eax
• |.text:10001757           call   ds:gethostbyname ; name
```



## 4. 악성코드 고급 정적 분석: 아이다 활용

5. 0x10001656에 있는 서브루틴에서 IDA Pro는 지역 변수 몇 개를 인지하고 있는가?

6. 0x10001656에 있는 서브루틴에서 IDA Pro는 파라미터 몇 개를 인지하고 있는가?

.text:10001656 var_675	= byte ptr -675h
.text:10001656 var_674	= dword ptr -674h
.text:10001656 hLibModule	= dword ptr -670h
.text:10001656 timeout	= timeval ptr -66Ch
.text:10001656 name	= sockaddr ptr -664h
.text:10001656 var_654	= word ptr -654h
.text:10001656 Dst	= dword ptr -650h
.text:10001656 Parameter	= byte ptr -644h
.text:10001656 var_640	= byte ptr -640h
.text:10001656 CommandLine	= byte ptr -63Fh
.text:10001656 Source	= byte ptr -63Dh
.text:10001656 Data	= byte ptr -638h
.text:10001656 var_637	= byte ptr -637h
.text:10001656 var_544	= dword ptr -544h
.text:10001656 var_50C	= dword ptr -50Ch
.text:10001656 var_500	= dword ptr -500h
.text:10001656 Buf2	= byte ptr -4FCh
.text:10001656 readfds	= fd_set ptr -4BCh
.text:10001656 phkResult	= byte ptr -3B8h
.text:10001656 var_3B0	= dword ptr -3B0h
.text:10001656 var_1A4	= dword ptr -1A4h
.text:10001656 var_194	= dword ptr -194h
.text:10001656 WSADATA	= WSADATA ptr -190h
.text:10001656 arg_0	= dword ptr 4
text:10001656	



## 4. 악성코드 고급 정적 분석: 아이다 활용

7. Strings 원도우를 이용해 디스어셈블리 내의 문자열 ¶cmd.exe /c를 찾아보자. 어디에 있는가?
8. ¶cmd.exe /c를 참조하는 코드 영역에서 무슨 일이 발생하는가?

```
.text:100101C8      cmp     dword_1008E5C4, ebx
.text:100101CE      jz      short loc_100101D7
.text:100101D0      push    offset aCmd_execC ; "¶cmd.exe /c"
.text:100101D5      jmp     short loc_100101DC
```



## 4. 악성코드 고급 정적 분석: 아이다 활용

10. 0x1000FF58에서 서브루틴으로 수백 라인은 문자열을 비교하기 위한 일련의 memcmp 비교다. robotwork와 문자열 비교가 성공적으로 이뤄지면 무슨 일이 일어나는가? (memcmp가 0을 반환)

The screenshot shows assembly code from the IDA Pro debugger. A green arrow points to the first instruction of the subroutine at address 0x10010444. The assembly code is as follows:

```
loc_10010444: ; size_t
push    9
lea     eax, [ebp+var_5C0]
push    offset aRobotwork ; "robotwork"
push    eax           ; void *
call    memcmp
add    esp, 0Ch
test   eax, eax
jnz    short loc_10010468
```

The code pushes the value 9 onto the stack, then loads the address of the string "robotwork" into EAX. It then calls the `memcmp` function, which compares the memory starting at the current stack pointer (ESP) with the string "robotwork". If the comparison fails (EAX is not zero), it jumps to the label `loc_10010468`. Otherwise, it continues execution.



## 4. 악성코드

### 11. PSLIST 악스

```
push    ebp
mov     ebp, esp
mov     eax, 1634h
call    __alloca_probe
and    [ebp+buf], 0
push    ebx
push    edi
mov     ecx, 0FFh
xor    eax, eax
lea     edi, [ebp+var_633]
rep stosd
stosw
stosb
push    49h
xor    ebx, ebx
pop    ecx
xor    eax, eax
lea     edi, [ebp+pe.cntUsage]
mov     [ebp+pe.dwSize], ebx
rep stosd
mov     ecx, 3FFh
lea     edi, [ebp+var_1630]
mov     [ebp+hModule], ebx
push    ebx          ; th32ProcessID
rep stosd
push    2           ; dwFlags
call    CreateToolhelp32Snapshot
cmp    eax, 0FFFFFFFh
mov     [ebp+hObject], eax
jnz    short loc_100066DF
call    ds:GetLastError
push    eax
lea     eax, [ebp+buf]
push    offset aCreatetoolhe_0 ; "WrWnWrWnCreateToolhelp32Snapshot Fail:Error"...
push    eax          ; Dest
call    ds:sprintf
lea     eax, [ebp+buf]
push    eax          ; buf
push    [ebp+s]        ; s
call    sub_100038BB
add    esp, 14h
push    1
pop    eax
jmp    loc_1000689B
```

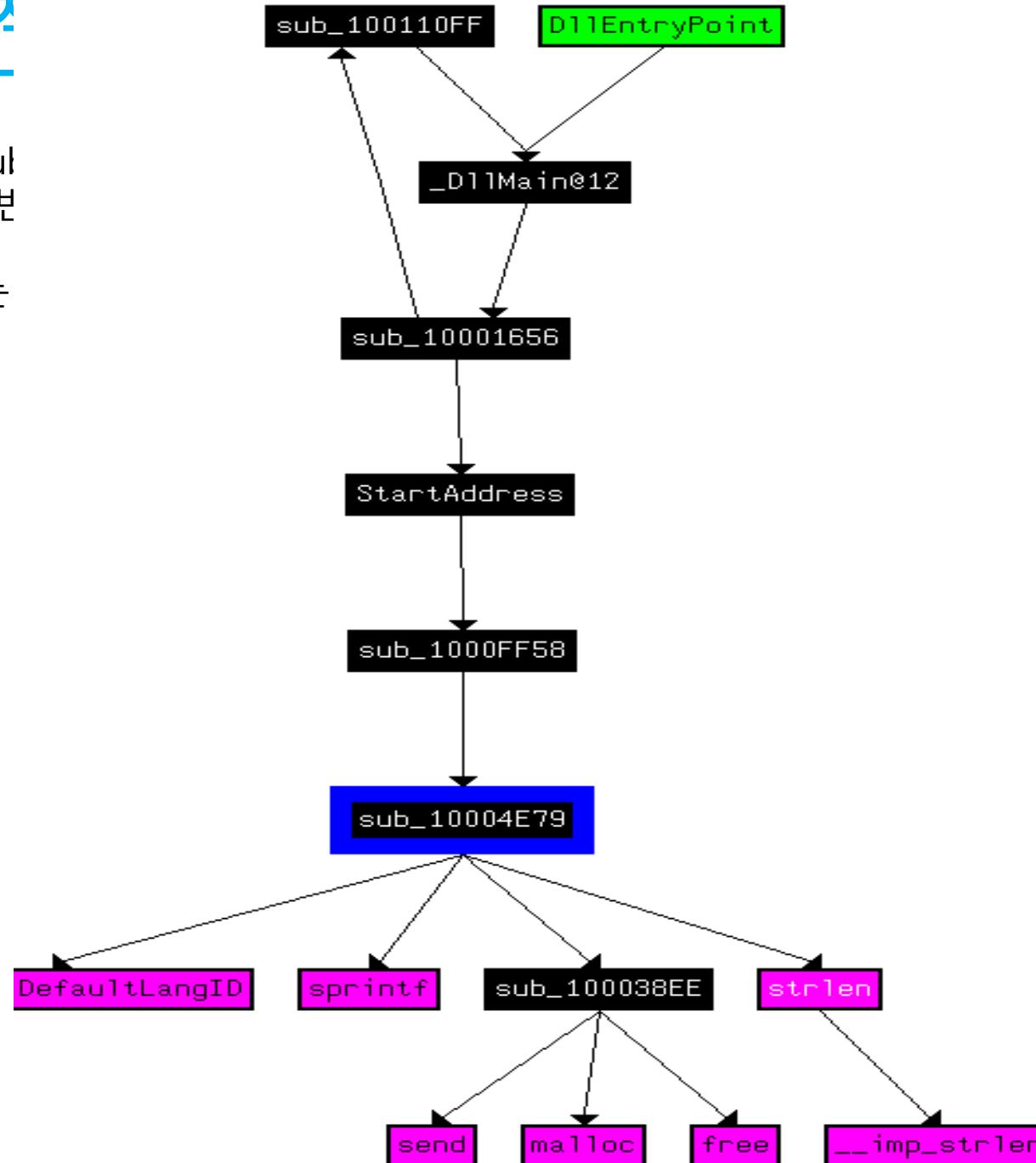


## 4. 악성코드 고급 정지

12. 그래프 모드를 이용해 sub\_100110FF가 호출하는 API 함수는 무엇인가? 해당 API 함수에만 기본

13. DllMain이 직접 호출하는

| 호출하는 API 함수는 무엇  
인가?  
인가?





## 4. 악성코드 고급 정적 분석: 아이다 활용

14. 0x10001358에서 Sleep 호출이 존재한다. (sleep까지 수밀리초 값을 파라미터로 갖는 API 함수) 코드 후반부를 보면 이 코드가 수행되려면 프로그램이 얼마 동안 Sleep하는가?

```
loc_10001341:
mov    eax, OFF_10019020
add    eax, 0Dh
push   eax          ; Str
call   ds:atoi
imul   eax, 3E8h
pop    ecx
push   eax          ; dwMilliseconds
call   ds:Sleep
xor    ebp, ebp
jmp    loc_100010B4
sub_10001074 endp
```

after atoi eax = 30  
30 \* 1000(3E8h) = 30000  
30초



## 4. 악성코드 고급 정적 분석: 아이다 활용

15. 0x10001701에서 소켓을 호출한다. 세 가지 파라미터는 무엇인가?

```
loc_100016FB:          ; protocol
push    IPPROTO_TCP
push    SOCK_STREAM      ; type
push    AF_INET          ; af
call    ds:socket
mov     edi, eax
cmp     edi, 0xFFFFFFFFh
jnz    short loc_10001722
```

16. 소켓과 IDA Pro에서 명명한 심볼 상수 기능을 이용해 이 파라미터를 좀더 유용하게 할 수 있겠는가? 변경 후 파라미터는 무엇인가?

```
loc_100016FB:          ; protocol
push    IPPROTO_TCP
push    SOCK_STREAM      ; type
push    AF_INET          ; af
call    ds:socket
mov     edi, eax
cmp     edi, 0xFFFFFFFFh
jnz    short loc_10001722
```



## 4. 악성코드 고급 정적 분석: 아이다 활용

17. 명령어 옵코드 0xED의 사용법을 찾아보자. 이 명령어는 VMware 탐지를 수행하는 VMXh 매직 문자열로 사용한다. 이 악성코드는 이를 이용하고 있는가? VMware를 탐지하는 다른 증거가 있는가?

D9	ED	5		s	FLDLN2	ST			0123	.1..	0.23	Load Constant log <sub>2</sub>
----	----	---	--	---	--------	----	--	--	------	------	------	--------------------------------

coder32 edition | X86 Opcode and Instruction Reference 1.11  
<http://ref.x86asm.net/coder32.html#xDE>

```
.text:10003851 loc_10003851:          ; CODE XREF: sub_1000+1000
.text:10003851
.text:10003857      lea     ebx, [esi+10Ch]    ; "VMware Virt
.text:1000385C      push    offset aVMwareVirtualE ; "VMware Virt
.text:1000385D      push    ebx             ; Str
.text:1000385E      call    edi             ; strstr
.text:10003860      pop    ebx
```

## 4. 악성코드 고급 정적 분석: 아이다 활용



18. 0x1001D988로 점프해보자. 무엇을 찾을 수 있는가?

19. IDA 파이썬 플러그인을 설치했다면(IDA Pro 상용 버전에는 포함되어 있음) Lab05-01.py를 실행해보자. IDA 파이썬 스크립트는 이 책의 악성코드와 함께 제공한다.(커서가 0x1001D988에 위치해야 함) 스크립트 실행 후 무슨 일이 일어났는가?

20. 동일한 위치에 커서를 두고 이 데이터를 ASCII 문자열로 어떻게 변환할 수 있는가?



## 4. 악성코드 고급 정적 분석: 아이다 활용

21. 문자 편집기로 스크립트를 열어보자. 어떻게 동작하는가?

The screenshot shows a Windows Notepad window with the title bar '76 Lab05-01.py - C:\Documents and Settings\Wever\바탕 화면\PracticalMalwar...' and the menu bar 'File Edit Format Run Options Windows Help'. The main content area contains the following Python code:

```
sea = ScreenEA()

for i in range(0x00,0x50):
    b = Byte(sea+i)
    decoded_byte = b ^ 0x55
    PatchByte(sea+i,decoded_byte)
```



# 악성코드 주요 행위 분석



# 악성코드 주요 행위 분석

- 원도우 API

- 대부분의 악성코드는 윈도우 플랫폼에서 동작
- 운영체제와 밀접하게 상호작용



<http://soen.kr>

**프로그래머와 예비 개발자의 학습 및 놀이 동아리**  
Software Engineering(소프트웨어 공학 연구소)

[SoEn 소개](#)
[SoEn 연혁](#)
[운영 방침](#)
[웹서버 소개](#)

<p><b>공지사항</b></p> <p>15-08-18 winapi 도메인 호스팅을 중지합니다.          13-04-20 soen.kr로 도메인명 확정합니다.          12-11-04 SoEnLab.com 도메인명 변경했습니다.          12-07-15 SoEnLab.kr 도메인을 등록했습니다.          12-06-25 게시판을 복구하였습니다.          11-09-15 DevCafe.kr 도메인을 확보했습니다.          11-06-11 WinApi 도메인을 변경할 예정입니다.          10-06-29 WinApi 오픈 10주년이 되었습니다.          10-06-10 경찰서 가서 조사 받고 왔습니다.          09-07-13 서버 이전 완료 보고 및 이후의 정책 변화          09-07-03 아너림 글판 특허 받았습니다.          09-06-29 새로운 서버로 이전했습니다.</p>	<p><b>업데이트 소식</b></p> <p>11-09-15 휴게실 프로그래밍 컬럼 업데이트          11-07-11 OpenGL 초안 강좌 올렸습니다.          11-07-01 서식 편집 컨트롤 실행 파일만 올렸습니다.          09-07-20 당근 1.22를 릴리즈했습니다.          09-06-02 C++개발자를 위한 자바 강좌          09-04-20 아너림 글판을 공개합니다.          08-10-25 GDI+ 강좌 업데이트          08-10-14 FreeType 초안 강좌 업로드          08-07-02 WTL 8.0 강좌를 올렸습니다.          08-06-22 딪넷 강좌 일부를 올렸습니다.          08-03-01 당근 1.21을 릴리즈했습니다.          08-02-03 C/C++ 강좌의 정오표를 수정했습니다.</p>
---	--

알립니다.  
 soen.kr은 winapi.co.kr의 새 도메인입니다. 모든 강좌와 자료를 SoEn으로 옮겨 왔으며 이전 사이트인 WinApi는 점차적으로 폐쇄할 예정입니다. 링크를 걸어 놓으신 분들은 새 도메인 주소로 링크를 수정해 주시기 바랍니다. SoEn은 언제까지나 이 모습 그대로 변함없이 운영됩니다.

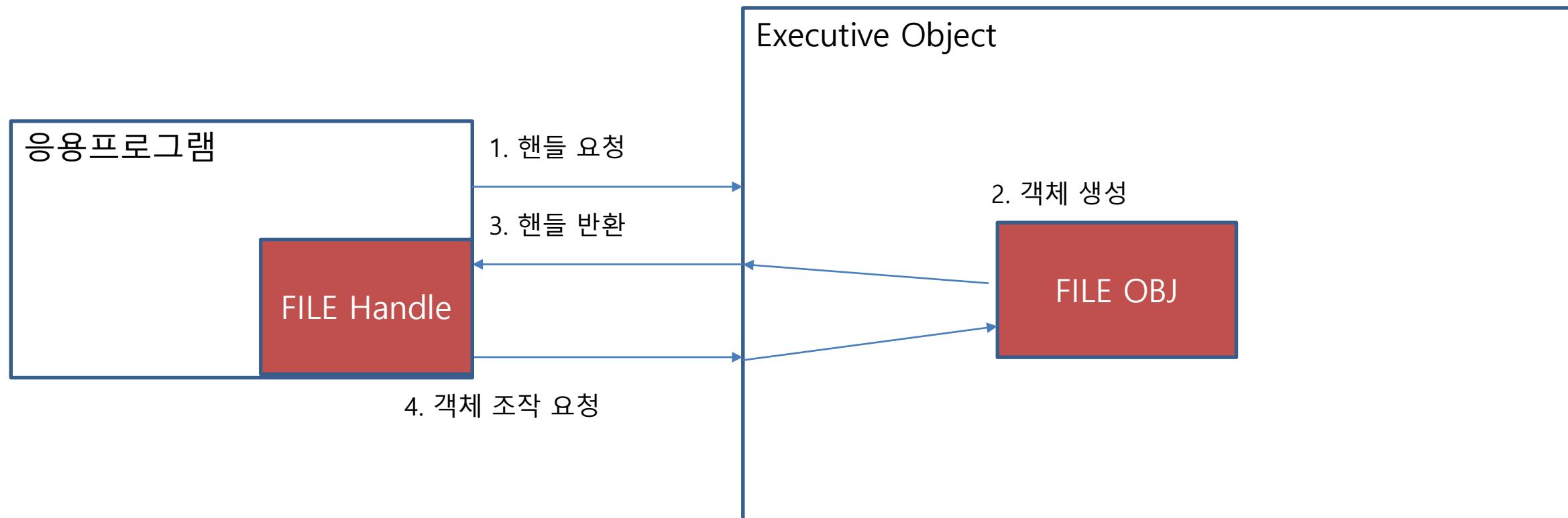
총 방문자 (2000년 6월 29일 이후) : 957만 8068명, 오늘 방문자(중복 제거) : 277명



# 악성코드 주요 행위 분석

- **핸들(Handle)**

- 윈도우, 프로세스, 모듈, 메뉴, 파일 등과 같이 운영체제에서 오픈되거나 생성
- 객체나 메모리 위치를 참조한다는 점에서 포인터와 같음





# 악성코드 주요 행위 분석

## • 파일 시스템 함수

- 악성코드가 시스템과 상호작용하는 가장 일반적인 방식은 파일을 생성하거나 수정해 파일명을 구별하거나 기존 파일명을 변경
- 식별자로 사용 가능
- 파일을 생성해 그 파일에 웹 브라우징 내용을 저장한다면 해당 스파이웨어 형태일 가능성이 높음

함수	설명
CreateFile	파일을 생성하고 열 때 사용 기존 파일, 파일, 스트림, I/O 장치를 열고 새로운 파일을 생성 dwCreationDisposition 인자는 이 함수가 새로운 파일을 생성하는지, 기존 파일을 오픈하는지 여부 제어
ReadFile WriteFile	파일을 읽고 쓰는 데 사용 두 함수 모두 파일을 스트림 형태로 운영 ReadFile은 파일에서 몇 바이트를 읽고 나면, 그 다음 몇 바이트를 읽는 형식을 가짐
CreateFileMapping MapViewOfFile	파일 매핑은 파일을 메모리로 로드해 사용할 때 사용 CreateFileMapping : 파일을 MapViewOfFile : 매핑된 베이스 주소를 가져온다 <b>참고</b> 파일 매핑은 윈도우 로더 기능을 복제할 때 주로 사용 파일 맵을 획득한 후 악성코드는 PE 헤더를 파싱해 메모리 내의 파일에 필요한 부분을 변경 → 마치 OS 로더에 의해 로드된 것처럼 PE파일을 실행



# 악성코드 주요 행위 분석

- **특수 파일**
  - 드라이브 문자와 폴더명으로 접근할 수 없는 파일도 존재
- **공유 파일**
  - WW\$ServerName\share나 WW?W\$ServerName\share로 시작하는 이름을 가짐
  - 네트워크 상에 저장된 공유 폴더에서 디렉터리나 파일에 접근
- **네임스페이스를 통해 접근 가능한 파일**
  - 운영체제 내에서 네임스페이스를 통해 접근
  - 네임스페이스: 고정된 숫자의 폴더와 각각 저장하는 다른 유형의 객체
  - Win32 장치 네임스페이스는 접두사 WW.W를 이용 → 물리적 장치 접근하여 파일처럼 읽음

## 네임 스페이스를 이용한 사례

파일 시스템을 무시하여 PhysicalDisk1나 Physical Memory에 바로 접근하기 위해  
WW.WPhysicalDisk1나 WDevice\PhysicalMemory를 사용  
→ 파일을 직접 생성  
→ 비할당 영역에 악성코드 생성 (백신 우회)



# 악성코드 주요 행위 분석

- ADS(Alternate Data Streams) : NTFS 내의 기존 파일에 데이터를 추가 기능 제공
  - 한 파일에 다른 파일을 추가
  - 추가 데이터는 디렉터리 목록에 나오지 않고, 파일 내용 출력이 안됨
  - ADS 데이터는 normalFile.txt:Stream:\$Data라는 규칙에을 가짐

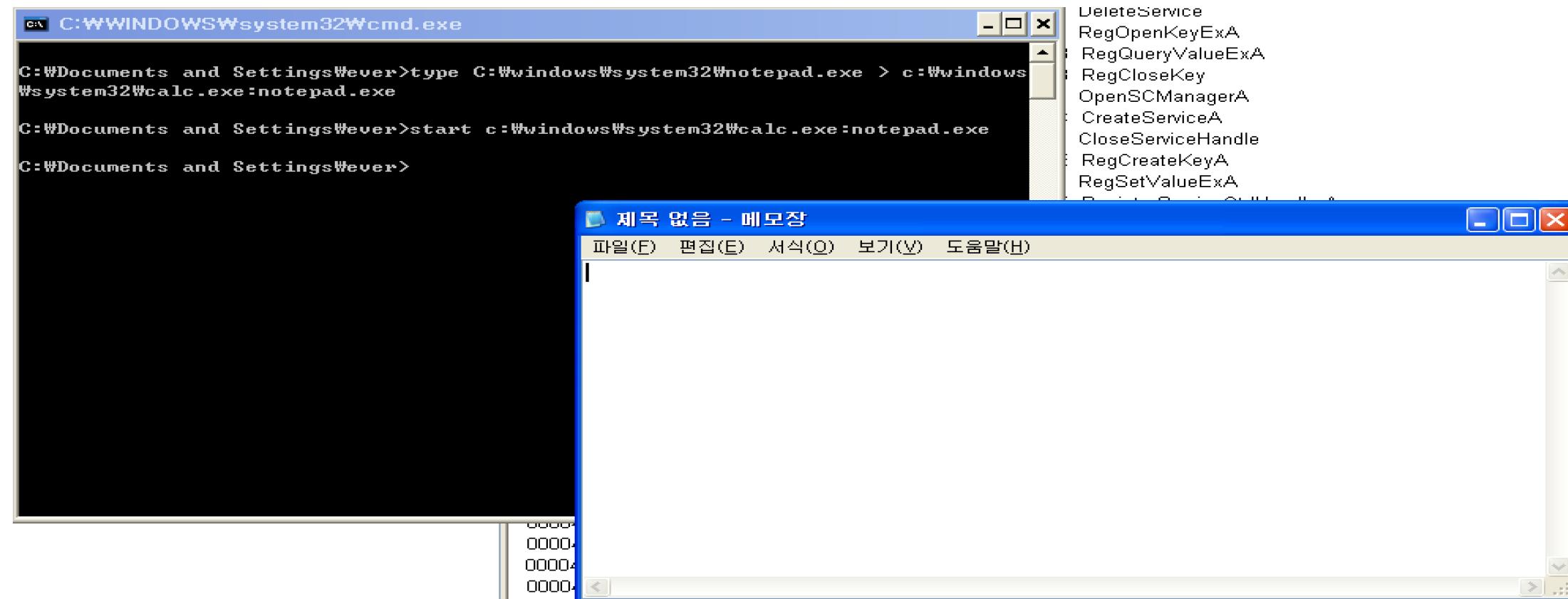
The screenshot illustrates the creation and reading of an alternate data stream (\$Data) on a file named 'README.txt'.  
1. A screenshot of a Windows File Explorer window shows a file named 'README.txt' with a size of 1KB. The file details pane indicates it is a '텍스트 문서' (Text Document) from '2015-03-22 오후...'.  
2. Below the file explorer is a screenshot of a Windows Notepad window titled 'README.txt - 메모장' (Notepad). It displays the content of the file.  
3. A command prompt window titled 'cmd.exe' shows the command 'echo "codeblack" >> README.txt:codeblack' being run, which creates the alternate data stream.  
4. Another command prompt window shows the command 'more < README.txt:codeblack', which reads the contents of the alternate data stream.  
5. A screenshot of a Windows File Explorer window shows the file 'README.txt' again, with its properties indicating it contains 1 file and 2 streams, totaling 92,908,789,760 bytes.  
6. At the bottom, a URL is provided: <http://www.hahwul.com/2015/03/ntfs-file-system-ads/>.



# 악성코드 주요 행위 분석

- ADS를 사용한 프로그램 감추기 및 실행 실습

- C:\> type C:\Windows\system32\notepad.exe > c:\Windows\system32\calc.exe:notepad.exe
- C:\> start c:\Windows\system32\calc.exe:notepad.exe





# 악성코드 주요 행위 분석

## • 윈도우 레지스트리

- 운영체제와 설정이나 옵션 같은 프로그램 구성 정보를 저장
- 수직 구조의 정보 데이터베이스 → 성능 향상!
- 네트워킹, 드라이버, 시작, 사용자 계정, 다른 정보 등 거의 모든 윈도우 구성 정보 저장
- 레지스트리를 활용하는 악성코드
  - 영구 데이터나 설정 데이터 저장
  - 컴퓨터 부팅 시마다 자동으로 동작할 수 있게 수정

레지스트리 구성 값	설명
루트 키 (root key)	루트키라 부르는 다섯 가지 최상위 부분으로 나눠짐 때로는 HKEY와 하이브란 용어를 사용
서브 키 (subkey)	서브키는 폴더 내의 서브폴더 역할
키 (key)	키는 또 다른 폴더나 값을 저장할 수 있는 레지스트리 내 폴더 (루트키와 서브키는 모두 키다)
값 엔트리 (value entry)	값 엔트리는 순차적인 이름과 값 쌍 임.
값이나 데이터 (value or data)	레지스트리 엔트리 내에 저장된 데이터



# 악성코드 주요 행위 분석

- 레지스트리 루트 키

루트키 종류	설명
HKEY_LOCAL_MACHINE (HKLM)	시스템 전역 설정 저장
HKEY_CURRENT_USER (HKCU)	현재 사용자에 특화된 설정 저장
HKEY_CLASSES_ROOT	정의한 유형 정보를 저장
HKEY_CURRENT_CONFIG	현재 하드웨어 구성 설정, 특히 현재 설정과 표준 설정의 차이를 저장
HKEY_USERS	기본 사용자, 새로운 사용자, 현재 사용자의 설정을 정의



# 악성코드 주요 행위 분석

- 레지스트리 편집기

레지스트리 편집기

파일(F) 편집(E) 보기(V) 즐겨찾기(A) 도움말(H)

이름	종류	데이터
ab(기본값)	REG_SZ	(값 설정 안됨)
abAPSDaemon	REG_SZ	"C:\Program Files\Common rundll32.exe bthprops.cpl,,Blue"
abBluetoothAuthenticationAgent	REG_SZ	"C:\Program Files\Everything"
abEverything	REG_SZ	"C:\Windows\IME\imjp8_1"
abIMJP8I8,1	REG_SZ	"C:\Program Files\iTunes\iTunes"
abiTunesHelper	REG_SZ	"C:\Windows\system32\IM"
abPHIME2002A	REG_SZ	C:\Windows\system32\IM
abPHIME2002ASync	REG_SZ	C:\Windows\system32\IM
abSunJavaUpdateSched	REG_SZ	"C:\Program Files\Common"
abVMware User Process	REG_SZ	"C:\Program Files\VMware\VMware"

키	항목
루트 키	HKLM
서브키	SOFTWARE, Microsoft, Windows, CurrentVersion, Run
값 엔트리	ASPDaemon, BluetoothAuthenticationAgent ...

내 컴퓨터\HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run



# 악성코드 주요 행위 분석

- 서비스

- 악성코드가 새로운 코드를 실행하는 또 다른 방식은 서비스로 설치하는 방법
- 백그라운드 애플리케이션으로 실행하는 서비스를 사용, 프로세스나 스레드 없이 실행  
(코드가 스케줄링돼 사용자 입력 없이 윈도우 서비스 관리자가 실행)
- 서비스 이용의 이점
  - SYSTEM 권한으로 실행 (administrator나 사용자 계정보다 상위 권한)
  - 운영체제 시작 시 자동으로 실행
  - 작업관리자(Task Manager)에 프로세스가 보이지 않을 수 있음
- 서비스 API
  - OpenSCManager: 서비스 제어 관리자(Service control manager)에 핸들 반환
  - CreateService: 신규 서비스로 등록, 부팅 시 서비스의 자동/수동 시작 여부를 호출자가 지정
  - StartService: 서비스를 시작하고 서비스가 수동으로 시작하게 설정되어 있을 때만 사용



# 악성코드 주요 행위 분석

- 서비스 제어 sc.exe

```
C:\Documents and Settings\ever>sc qc "UMTools"
[SC] GetServiceConfig SUCCESS

SERVICE_NAME: UMTools
        TYPE               : 110  WIN32_OWN_PROCESS <interactive>
        START_TYPE          : 2    AUTO_START
        ERROR_CONTROL       : 1    NORMAL
        BINARY_PATH_NAME    : "C:\Program Files\VMware\VMware Tools\umtoolsd.exe"

        LOAD_ORDER_GROUP    :
        TAG                : 0
        DISPLAY_NAME        : VMware Tools
        DEPENDENCIES        :
        SERVICE_START_NAME  : LocalSystem
```

- 서비스 등록/시작

- 등록 : sc create [서비스명] binpath= [서비스 파일 경로]
- 시작 : sc start/stop [서비스명]
- 삭제 : sc delete [서비스명]
- 조회 : sc query [서비스명]



# 악성코드 주요 행위 분석

- 자동으로 시작하는 프로그램

- Autoruns 는 운영체제가 시작 시 자동으로 실행하는 코드를 목록화
- IE, 다른 프로그램으로 로드되는 DLL, 커널로 로드되는 드라이버 목록화
- 자동으로 코드가 실행하게 되는 25~30 군데 위치를 조사
- 전체 리스트가 아닐 수 있음

The screenshot shows the Autoruns application window with a blue title bar and a standard Windows-style menu bar (File, Entry, Options, User, Help). Below the menu is a toolbar with various icons corresponding to the tabs above the main table. The main area is a table with the following columns: Autorun Entry, Description, Publisher, Image Path, and Timestamp.

Autorun Entry	Description	Publisher	Image Path	Timestamp
HKLM\Software\Microsoft\Windows\CurrentVersion\Run				2016-05-21 오전 1:15
APSDaemon	Apple Push	Apple Inc.	c:\program files\com...	2012-02-14 오후 5:22
Everything	Everything	Everything	c:\program files\ever...	2014-08-06 오전 10:01
iTunesHelper	iTunesHelper	Apple Inc.	c:\program files\itun...	2012-03-27 오후 8:28
SunJavaUpd...	Java(TM) Update Sche...	Oracle Corporation	c:\program files\com...	2013-07-03 오전 1:16
VMware Use...	VMware Tools Core Se...	VMware, Inc.	c:\program files\vmw...	2016-02-26 오전 7:08
HKLM\Software\Microsoft\Active Setup\Installed Components				2014-01-22 오후 4:00
Microsoft Ou...	Outlook Express Setup...	Microsoft Corporation	c:\program files\outl...	2008-04-14 오전 3:30
주소록 6	Outlook Express Setup...	Microsoft Corporation	c:\program files\outl...	2008-04-14 오전 3:30
HKCU\Software\Microsoft\Windows\CurrentVersion\Run				2016-06-04 오후 7:50
uTorrent	µTorrent	BitTorrent Inc.	c:\documents and set...	2016-05-04 오전 2:54

At the bottom of the window, there are status messages: "Ready." on the left and "Windows Entries Hidden." on the right.



# 악성코드 주요 행위 분석

- 일반 레지스트리 함

- 악성코드에서 레지스트리를 수정할 목적으로 사용하는 윈도우 API

API	설명
RegOpenKeyEx	편집과 질의용으로 레지스트리를 오픈
RegSetValueEx	레지스트리에 새로운 값을 추가하고 데이터를 설정
RegGetValue	레지스트리 내의 값 엔트리용 데이터를 반환

- Reg Add

```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\ever>reg

Console Registry Tool for Windows - version 3.0
Copyright (C) Microsoft Corp. 1981-2001. All rights reserved

REG 작업 [매개 변수 목록]
작업 [ QUERY : ADD : DELETE : COPY :
        SAVE : LOAD : UNLOAD : RESTORE :
        COMPARE : EXPORT : IMPORT ]
```



# 악성코드 주요 행위 분석

- **.reg 파일을 이용한 레지스트리 스크립트**
  - reg. 확장자를 가진 파일은 가독성이 있는 레지스트리 데이터를 포함
  - 더블 클릭하여 실행하면 담고 있는 정보를 자동으로 레지스트리로 병합해 수정
- 샘플 .reg 파일

```
Windows Registry Editor Version 5.00

[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run]
"MaliciousValue"="C:\Windows\evil.exe"
```



# 악성코드 주요 행위 분석

## • 버클리 호환 소켓

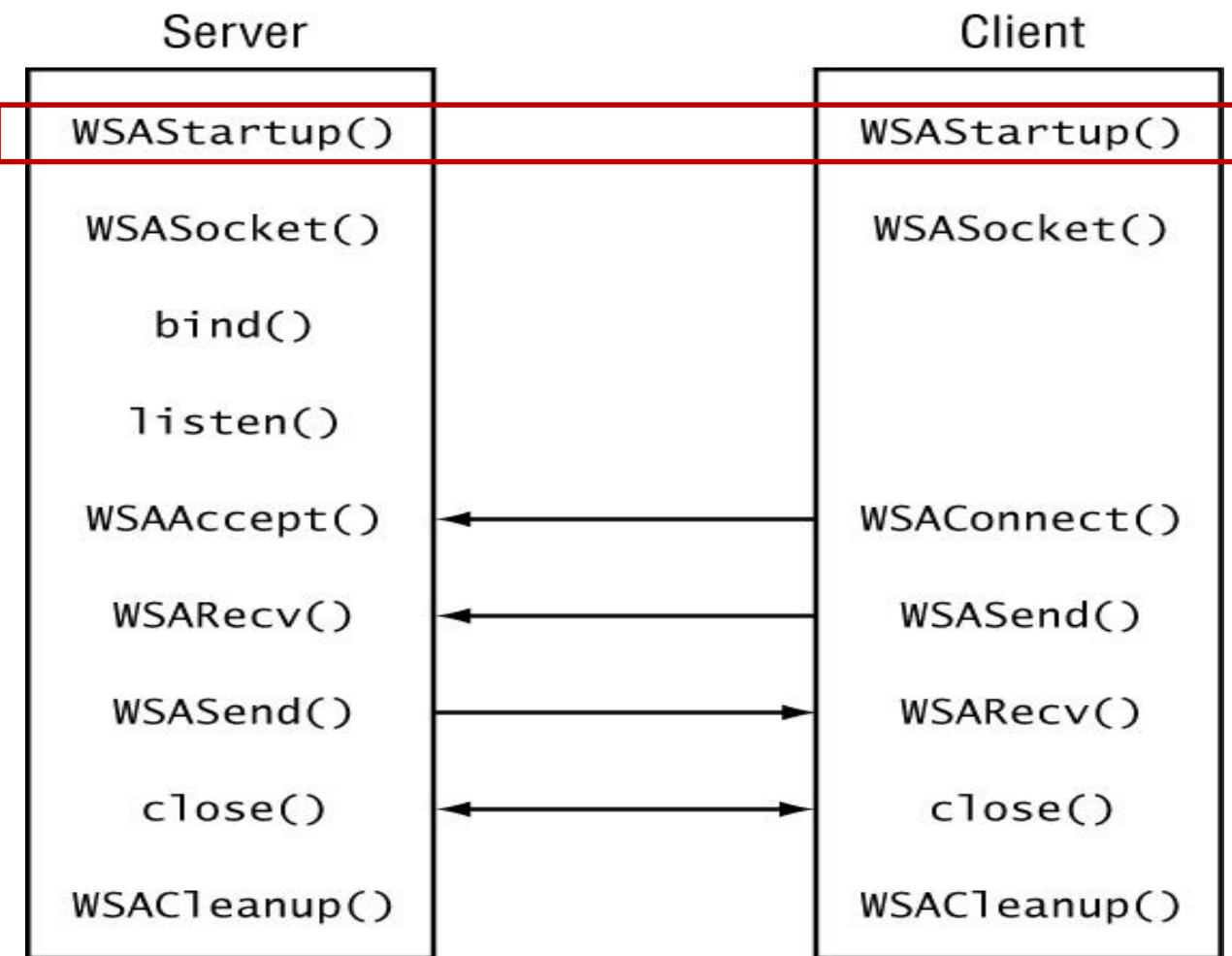
- 버클리 호환 소켓 네트워크 기능은 Winsock 라이브러리에서 주로 ws2\_32.dll에 구현
- 버클리 호환 소켓 네트워크 함수

API	설명
socket	소켓 생성
bind	호출 전에 소켓을 특정 포트로 할당
listen	소켓이 인바운드 연결을 위해 리스닝하고 있음을 나타냄
accept	외부 소켓 연결을 오픈하고 연결을 받아들임
connect	연결을 외부 소켓으로 오픈하고 외부 소켓은 연결을 기다림
recv	외부 소켓에서 데이터를 수신
send	외부 소켓으로 데이터 발송

# 악성코드 주요 행위 분석



- #### • 네트워킹 동작 방식



WSAStartup()는 네트워크 함수 수행을 위해 초기화 하는 함수다.  
여기서부터 네트워크 동작을 수행하기 때문에 BP를 선언하면 유용

그림 출처: <http://www.cnblogs.com/smkipedia/archive/2008/05/26/1207981.html>



# 악성코드 주요 행위 분석

- **WinINet API**

- Winsock API 보다 상위 수준의 API
- Wininet.dll에 저장
- HTTP, FTP와 같은 프로토콜을 구현
- InternetOpen: 인터넷 연결 초기화
- InternetOpenUrl : URL에 연결할 때 사용한다(HTTP 페이지나 FTP 리소스에 사용할 수 있음)
- InternetReadFile : ReadFile 함수 같이 프로그램이 인터넷에서 다운로드 한 파일에서 데이터를 읽음



# 악성코드 주요 행위 분석

- **DLL(Dynamic Link Libraries, 동적 링크 라이브러리)**

- 현재 윈도우에서 다양한 애플리케이션끼리 코드를 공유하는 라이브러리를 사용하는 방식
- 다른 애플리케이션에 의해 실행할 수 있는 익스포트 함수를 포함
- 정적 라이브러리는 DLL이전에 사용하던 표준 → 프로세스 별로 메모리에 로드

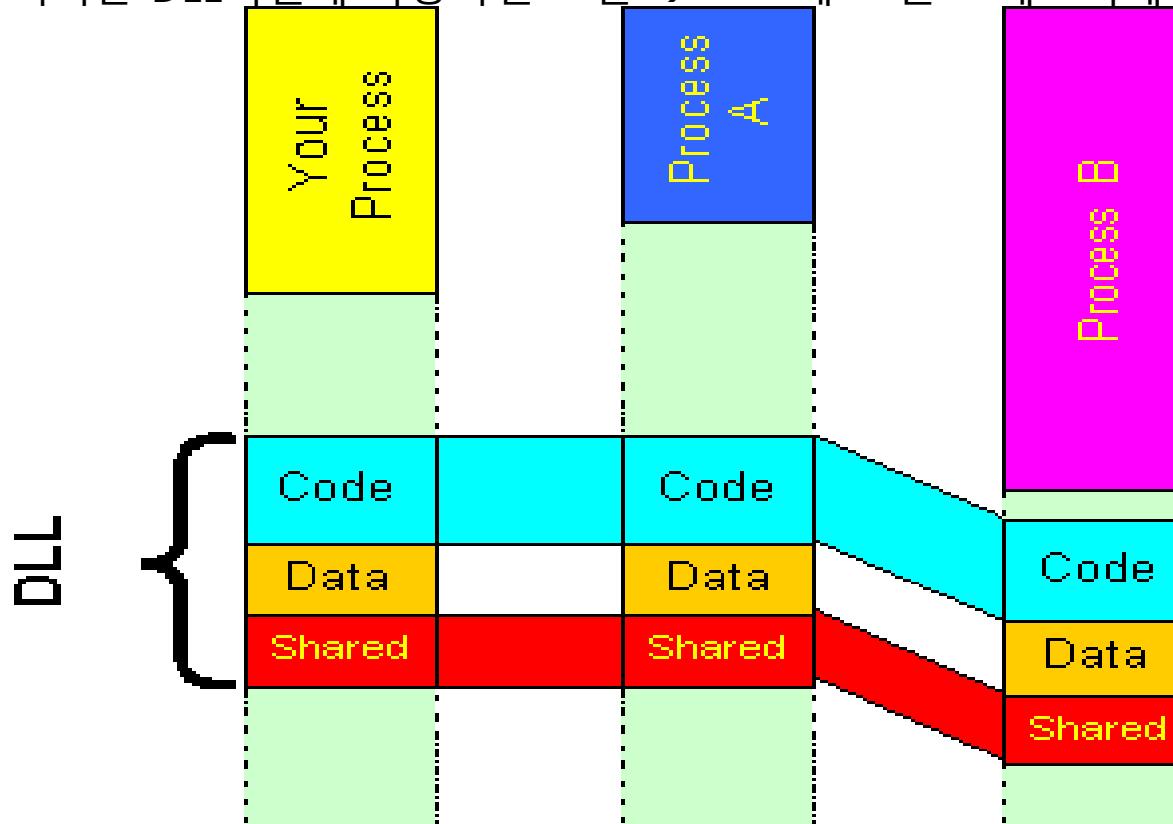


그림 출처 : <http://www.codeproject.com/Articles/1037/Hooks-and-DLLs>

Copyright www.boanproject.com All rights reserved



- **악성코드 제작자가 DLL을 이용하는 법**

- 악성코드 저장용: exe 파일 프로세스당 하나만 가지기 때문에 dll을 사용하여 다른 프로세스에서 실행 가능
- 윈도우 DLL 사용: 거의 모든 악성코드는 윈도우와 상호작용할 수 있는 윈도우 OS DLL을 사용하여 악성코드 분석가에게 상당한 통찰력을 제공
- 외부 DLL 사용: 다른 프로그램과의 상호동작을 위해 외부 DLL을 사용
- 윈도우 API를 활용해 직접 다운로드하지 않고 파이어폭스의 DLL을 이용



# 악성코드 주요 행위 분석

- **프로세스**

- 새로운 프로세스를 생성하거나 기존 프로세스를 변형해 현재 프로그램 외부에서 코드를 실행
- CreateProcess API(<https://msdn.microsoft.com/en-us/library/windows/desktop/ms682425%28v=vs.85%29.aspx?f=255&MSPPError=-2147217396> )

C++

```
BOOL WINAPI CreateProcess(
    _In_opt_     LPCTSTR          lpApplicationName,           // 프로그램 경로
    _Inout_opt_   LPTSTR          lpCommandLine,             // 커맨드 라인
    _In_opt_     LPSECURITY_ATTRIBUTES lpProcessAttributes,
    _In_opt_     LPSECURITY_ATTRIBUTES lpThreadAttributes,
    _In_          BOOL            bInheritHandles,
    _In_          DWORD           dwCreationFlags,
    _In_opt_     LPVOID           lpEnvironment,
    _In_opt_     LPCTSTR          lpCurrentDirectory,
    _In_          LPSTARTUPINFO   lpStartupInfo,             // 표준 출력, 입력, 에러
    _Out_         LPPROCESS_INFORMATION lpProcessInformation // 프로세스 정보
);
```



# 악성코드 주요 행위 분석

- 스레드
  - 프로세스는 실행 컨테이너, 스레드는 윈도우 운영체제가 실행
  - 프로세스 내의 코드 일부를 실행
- 스레드 문맥
  - 운영체제가 스레드를 교체하기 전에 CPU의 정보를 스레드 문맥에 저장
  - 스레드가 CPU내부 레지스터 값 변경
- **InjectDll.cpp 파일 분석!**

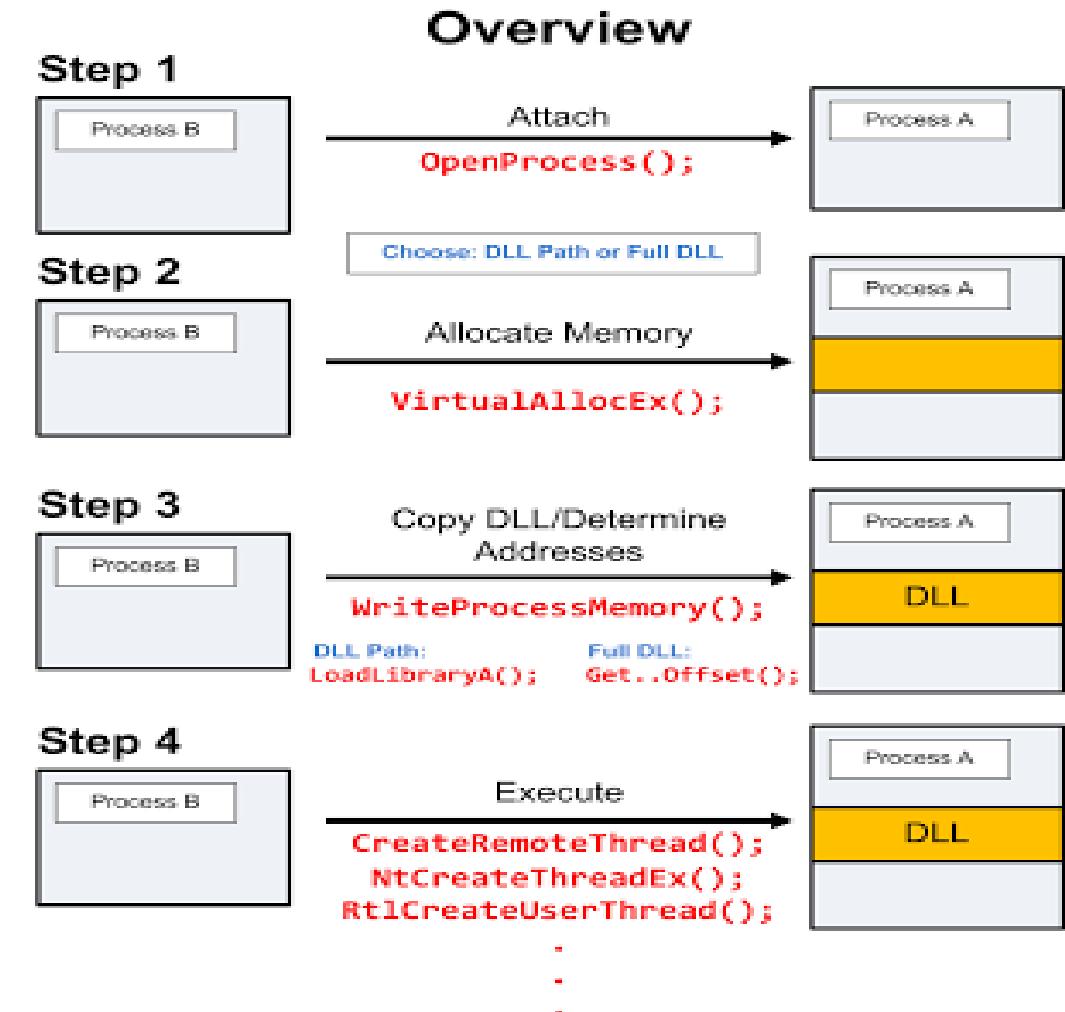


그림 출처 : [http://blog.opensecurityresearch.com/2013\\_01\\_01\\_archive.html](http://blog.opensecurityresearch.com/2013_01_01_archive.html)



# 악성코드 주요 행위 분석

- **뮤텍스를 이용한 내부 프로세스 조정**

- 뮤텍스 : 여러 프로세스와 스레드를 조정하는 전역 개체, 커널에서는 뮤탄트라고 부르기도 함
- 뮤텍스 메커니즘의 특징
  - Atomicity: 하나의 쓰레드가 mutex 를 이용해서 잠금을 “시도하는 도중”에 다른 쓰레드가 mutex 잠금 불가
  - Singularity: 만약 스레드가 mutex “잠금을 했다면” 해당 스레드가 mutex 잠금을 해제하기 전까지 다른 어떠한 쓰레드도 mutex 잠금 불가
  - Non-Busy Wait: 스레드는 다른 쓰레드가 락을 해제하기 전까지 “해당 지점에 머물러 어떠한 CPU 자원도 소비하지 않음”

뮤텍스를 활용한 스레드 동작 원리

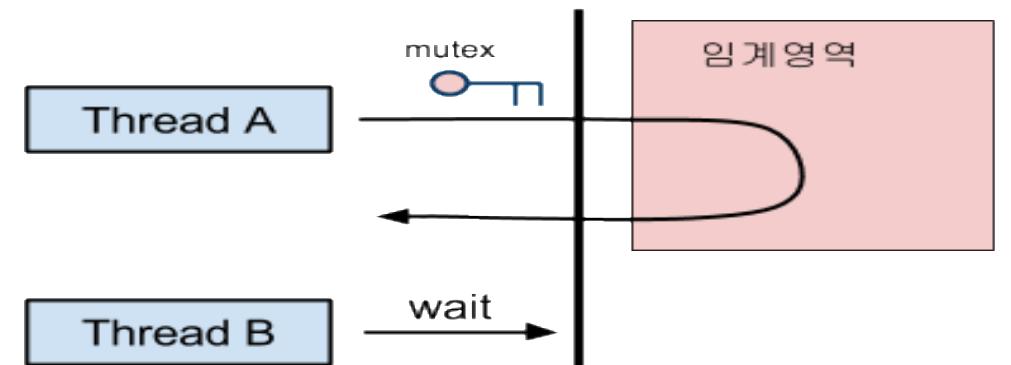


그림 출처: <http://www.joinc.co.kr/w/Site/Thread/Beginning/Mutex>



# 악성코드 주요 행위 분석

## ✓ COM(컴포넌트 객체 모델)

- 이종의 소프트웨어 컴포넌트에서 상호 세부내용을 모르더라도 서로 다른 코드를 호출할 수 있게 하는 인터페이스 표준(ActiveX)
- COM 객체는 클래스 식별자(CLID, Class identifiers) 또는 식별자(IID, Interface Identifiers)라고도 불리는 GUID(Globally Unique Identifiers)를 통해 접근

## ✓ CoCreateInstance

- COM 기능에 접근할 때 사용

```
HRESULT CoCreateInstance(  
    _In_     REFCLSID      rclsid,  
    _In_     LPUNKNOWN     pUnkOuter,  
    _In_     DWORD         dwClsContext,  
    _In_     REFIID        riid,  
    _Out_    LPVOID        *ppv  
);
```

### CLSID 저장소

HKLM\Software\Classes\CLSID  
HKCU\Software\Classes\CLSID

## ✓ COM 서버 악성코드

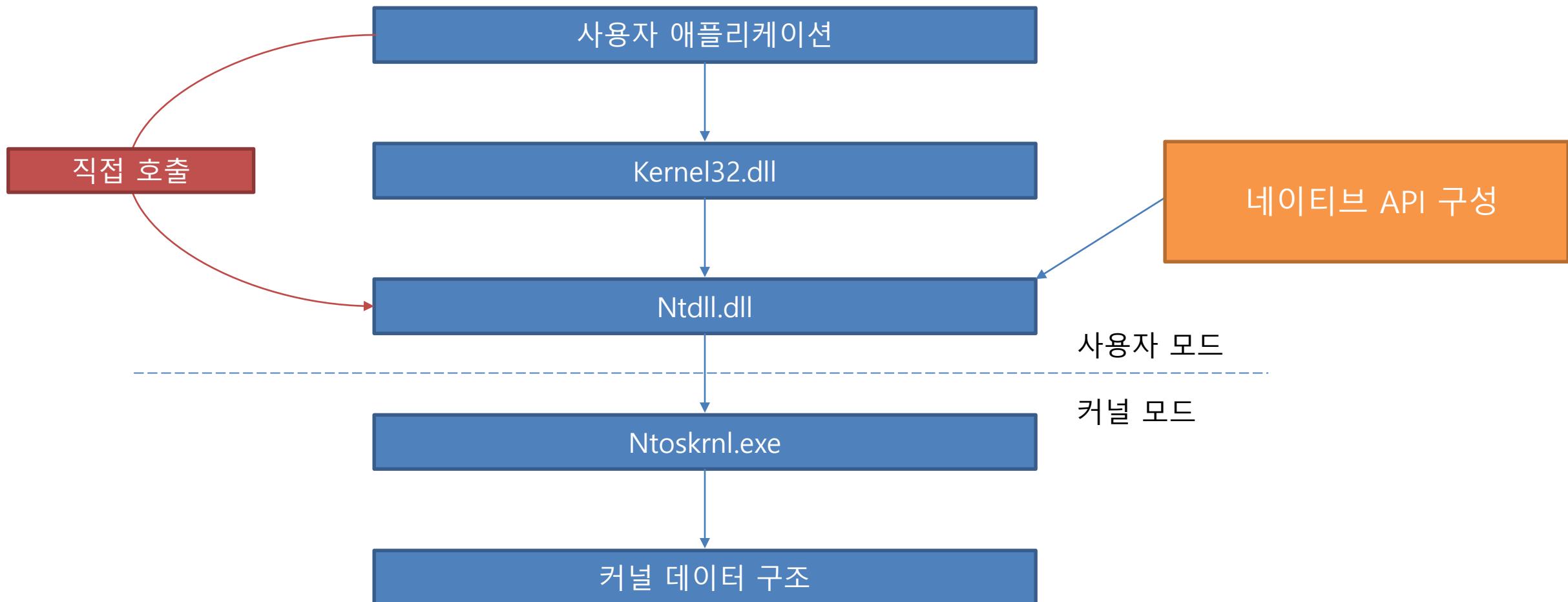
- BHO(Browser Helper Objects)라는 IE의 외부 플러그인을 사용하여 구현
- 특정 프로세스 실행 없이, **인터넷 트래픽 모니터링, 브라우저 사용 추적, 인터넷 통신**
- 다음 함수를 임포트
  - DllCanUnloadNow, DllGetClassObject, DllInstall, DllRegisterServer, DllUnregisterServer



# 악성코드 주요 행위 분석

## ✓ 네이티브 API

- 악성코드 제작자 사이에 흔히 사용하는 하위 수준의 윈도우 상호 작용 인터페이스
- 악의적이지 않은 프로그램은 거의 사용하지 않음





# 악성코드 주요 행위 분석

✓ 시스템, 프로세스, 스레드, 핸들, 기타 항목에 관한 정보 수집에 사용

- NtQuerySystemInformation
- NtQueryInformationProcess
- NtQueryInformationThread
- NtQueryInformationFile
- NtQueryInformationKey



## 6. 실전 악성코드 분석 맛보기

## 6. 실전 악성코드 분석 맛보기

