POC2008 Reverse Engineering Report

Reverse Engineering 문제 분석 결과 보고서

2008.11.14

목 차 (총 51페이지)

1. Reversing 문제 분석

❑ 문제목표

— 코드 보호를 위하여 실행 압축된 파일을 분석하여 원하는 Key 값 확인

— Process 점검을 이용한 Anti-debugging 기법을 우회하여 원하는 Key 값 확인

❑ 풀이환경

— 분석도구 : Olly-debugger, IDA Pro

❑ 풀이방법

— 실행압축 알고리즘을 감추기 위해 Signature를 변조하더라도 프로그램, 실행 시 메모리에는 압축이 풀린 상태로 적재된 다는 점에 착안하여 Debugging 수행.

— 프로그램 실행 후 Debugging을 수행하는 방법을 이용하여, 실행 초기에 수행되는 Anti-debugging 기법을 우회.

❑ 풀이결과

— 1번 문제의 답 : Beginner

— 2번 문제의 답 : POCACHIP

❑ Question1.exe 문제 상황

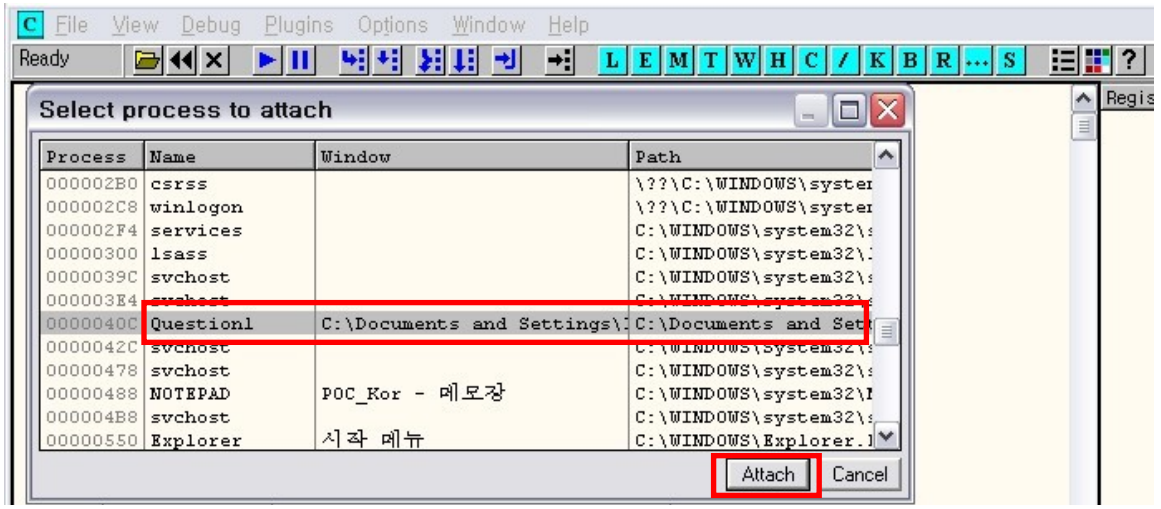
— Question1.exe를 Disassemble하기 위하여 Ollydbg나 IDA pro를 이용하여 분석을 시도할 경우 실행파일 압축으로 인하여 분석 진행 불가

— Unpack을 위한 Peid 등을 이용한 압축알고리즘을 확인 불가

(실행 압축 후 Signature에 해당 되는 부분이 변조 됨) – (MUP 지양)

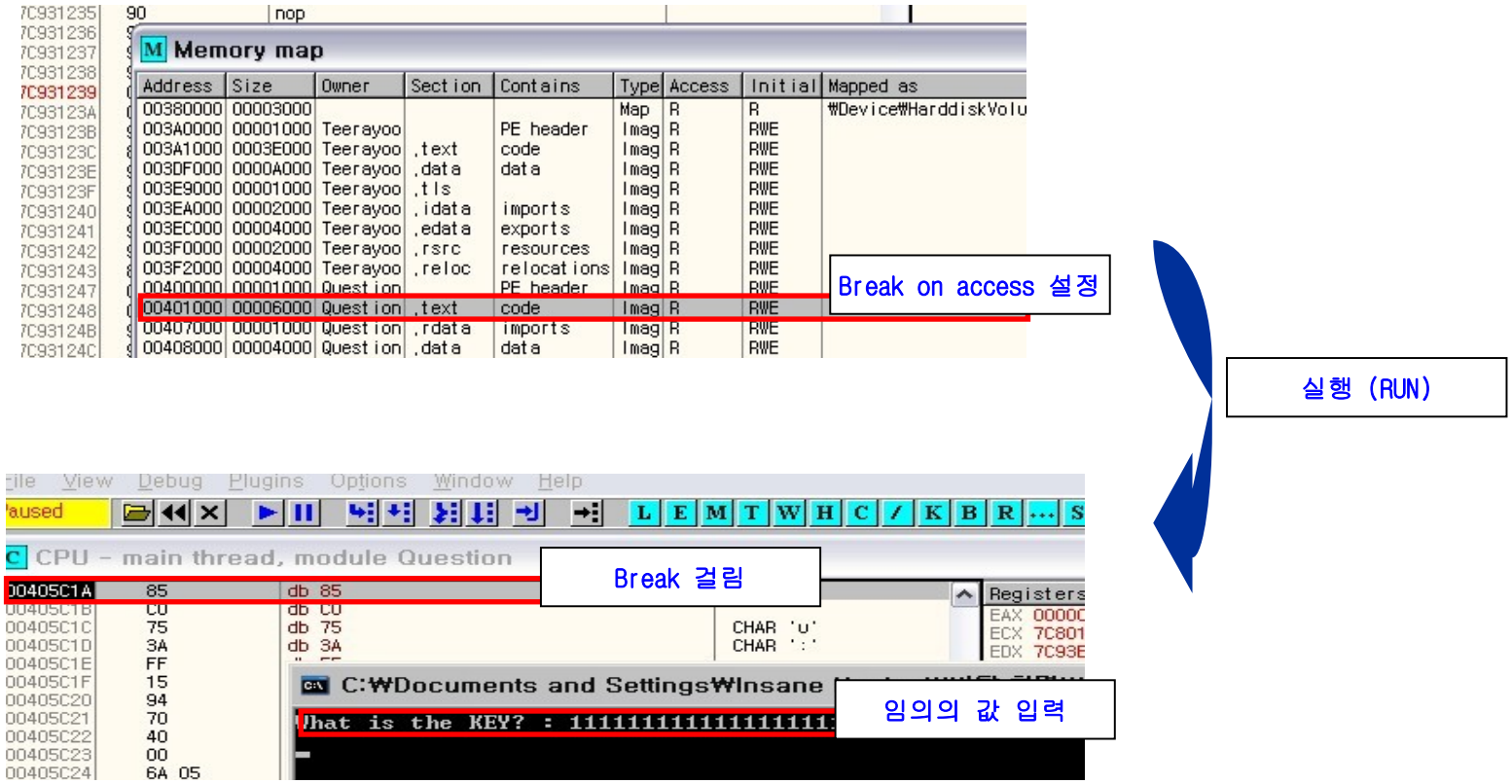
❑ 상세 풀이 방법

— Question1.exe 실행 후 Ollydbg의 Attach 기능을 이용하여 압축이 풀린 상태로 메모리에 적대된 실행코드에 접근.



❑ Question1.exe 상세분석 (계속)

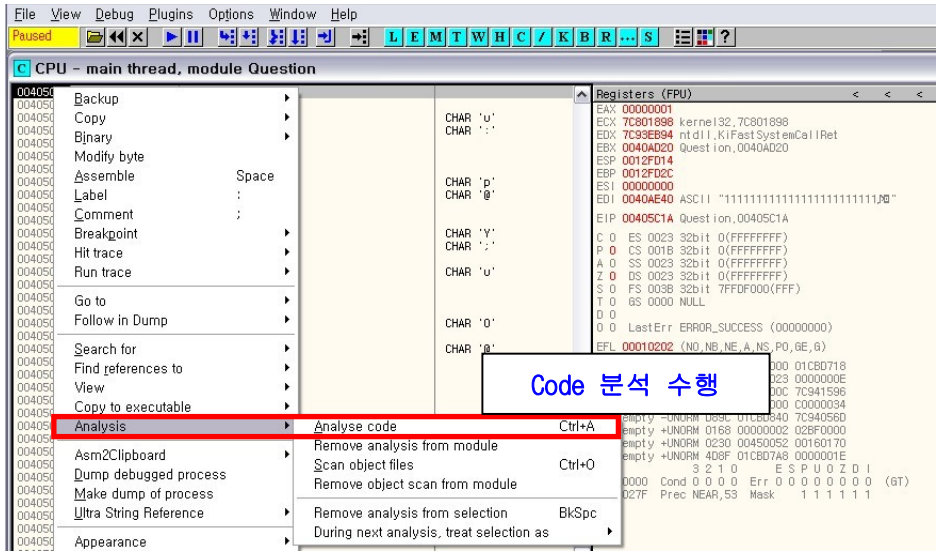
- Attach 후 Process를 Run시킨 후 Memory map을 통하여 Question1.exe의 .code 섹션에 Breakepoint 설정.
- 임의의 입력 값과 숨겨진 Serial 번호를 비교하기 위해 Question1.exe의 .code 섹션에서 비교 Code가 실행되는 순간에 프로그램이 Break 되도록 설정할 수 있음.

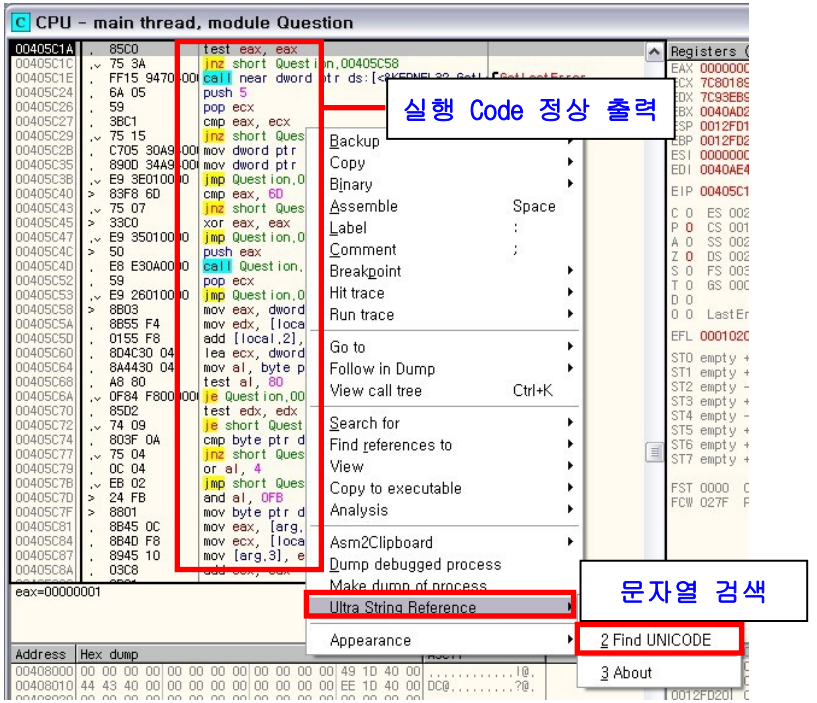


❑ Question1.exe 상세분석 (계속)

— “Analyse code”를 이용하여 .code 섹션 부분의 실행코드 분석 수행.

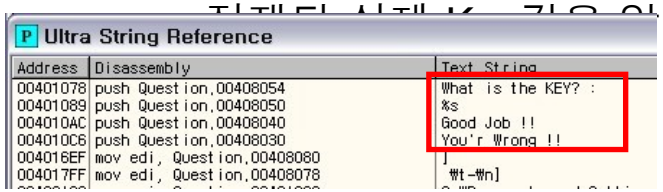
— 분석 된 실행 코드 내의 문자열 검색을 이용하여 정답 및 오답에 대한 문자열이 위치하는 부분으로 이동.

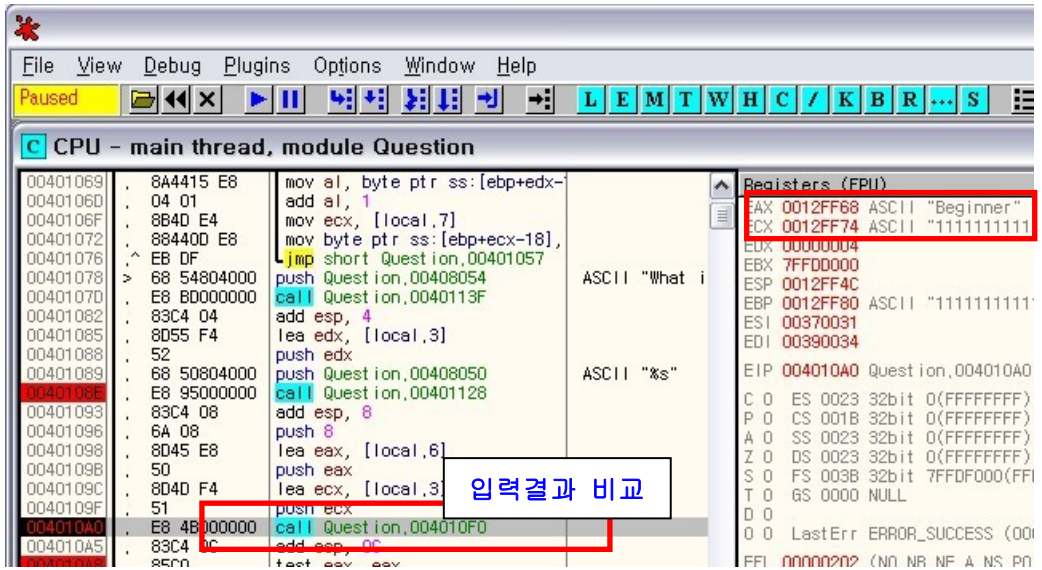


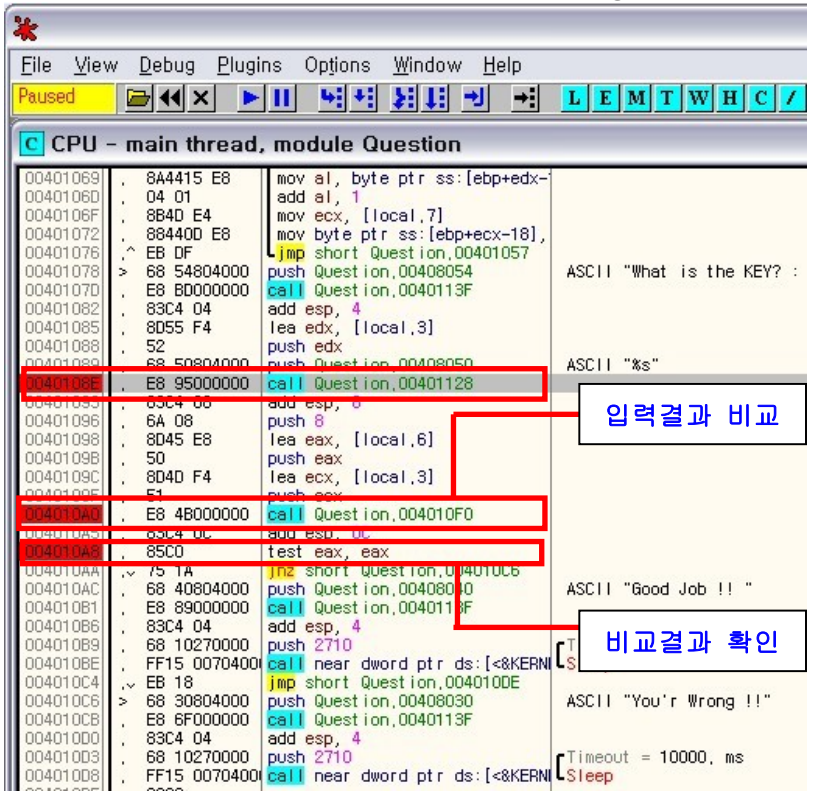


❑ Question1.exe 상세분석 (계속)

- 해당 위치로 이동하면, 임의로 입력한 값과 실제 Key값을 비교하는 부분을 확인할 수 있음 .
- 최종 Key 값을 비교하는 함수에 Breakpoint를 걸고 프로그램을 실행시키면 Register 內에 저장된 실제 Key 값을 알 수 있음.





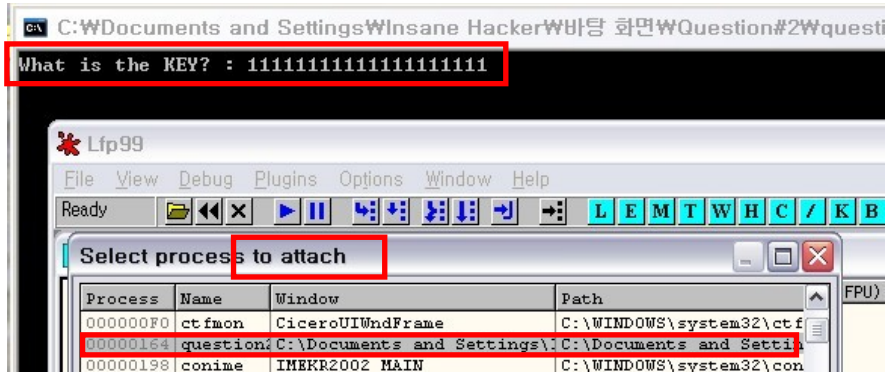


❑ Question2.exe 문제 상황

- Question2.exe를 Disassemble하기 위하여 Ollydbg나 IDA pro를 이용하여 분석을 시도할 경우 실행파일 압축으로 인하여 분석 진행 불가
- Unpack을 위한 Peid 등을 이용한 압축알고리즘을 확인 불가
  (실행 압축 후 Signature에 해당 되는 부분이 변조 됨)
- Ollydbg가 실행된 상태에서 Question2.exe 실행 시 Ollydbg 실행 종료
  (몇 개의 특정 Process 확인 후 실행중단 시키는 Anti-Debugging 기법 적용 됨)

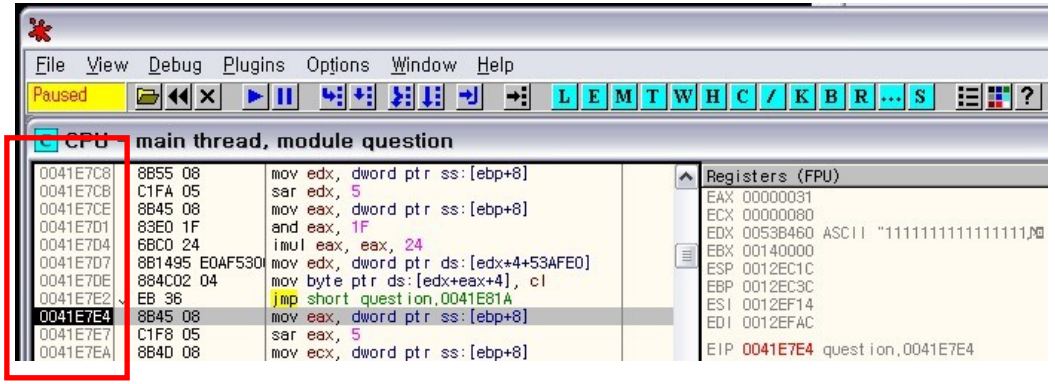
❑ Question2.exe 상세 분석

- Question2.exe 실행 후 Ollydbg의 Attach 기능을 이용하여 압축이 풀린 상태로 메모리에 적대된 실행코드에 접근.



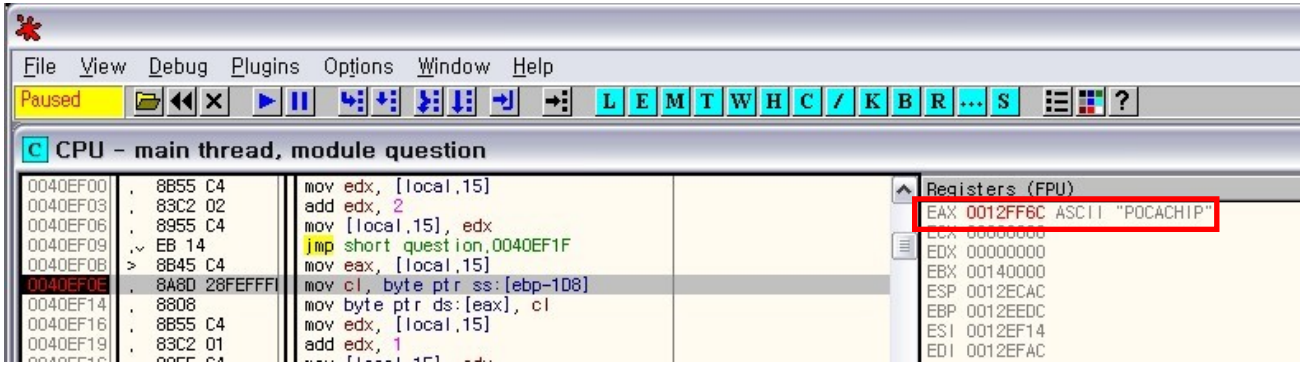
❑ Question2.exe 상세분석 (계속)

- 현재 코드의 실행이 Kernel 부분에 멈춰 있으나, 실제 정답과 사용자 입력 값을 비교하는 부분은 .code 섹션 內 존재하므로 'F7' 혹은 'F8'과 같은 단계별 실행 키를 이용하여 Question2.exe의 .code 섹션으로 이동.
- Question2.exe의 .code 섹션으로 이동 후 지속적으로 단계별 수행을 하며 Register 값의 변화를 살피다 보면, 원하는 문자열을 찾을 수 있음.



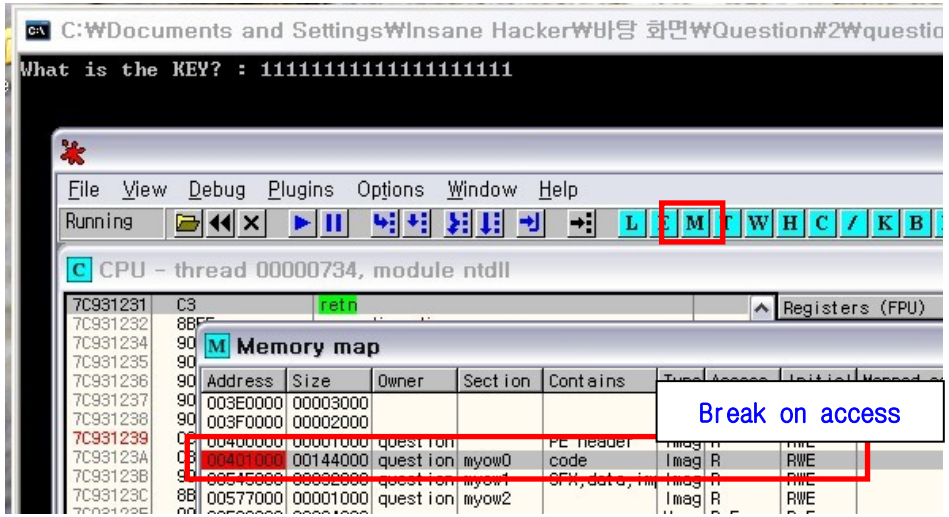
단계별 실행

.code 섹션으로 이동

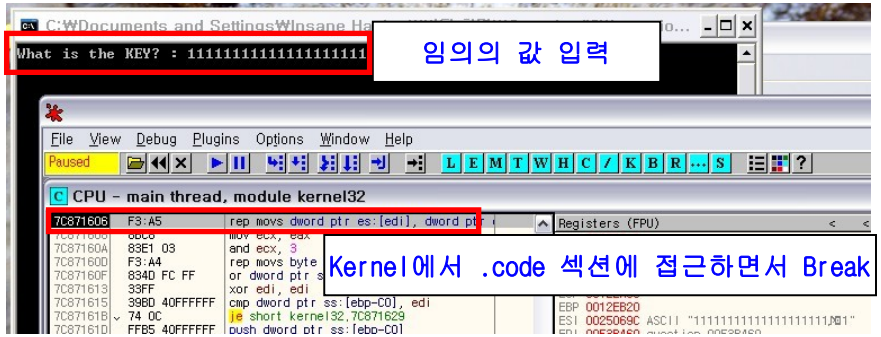


❑ Question2.exe 상세분석 (계속)

- Attach 후 Process를 Run시킨 후 Memory map을 통하여 Question1.exe의 .code 섹션에 Breakepoint 설정.
- 임의의 입력 값 입력 후 숨겨진 Serial 번호와 비교하기 위해 Kernel 에서 Question2.exe의 .code 섹션에 접근하는 순간 Break가 됨.

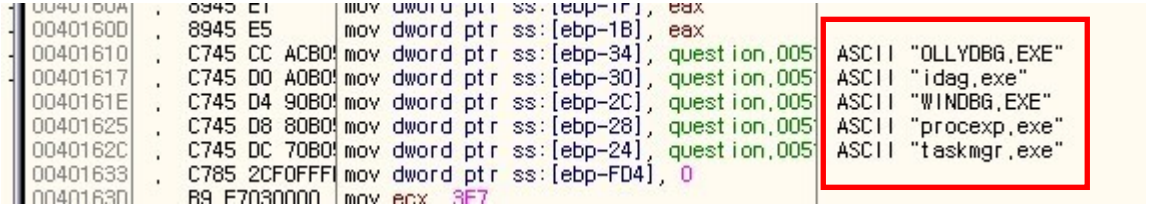


실행 (RUN)

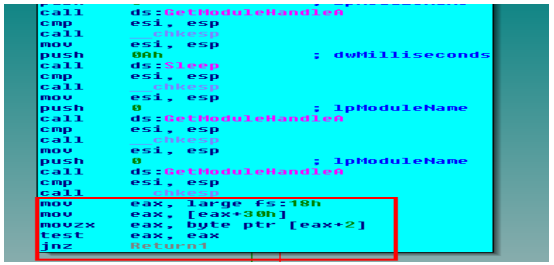


❑ Question2.exe 상세분석 (계속)

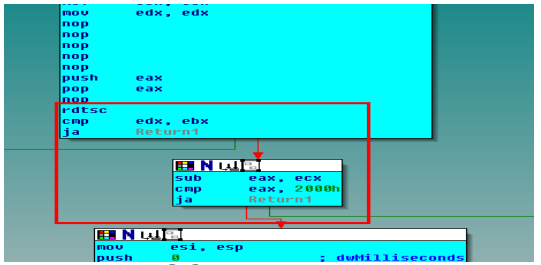
— 또한, Question2.exe의 .code 섹션 상단으로 이동하여 보면, 실행압축 해제 후 새로 만들어진 코드의 OEP를 찾을 수 있으며, 그 밑으로 실행 흐름을 따라가 보면, 아래와 같이 Question2.exe의 Anti-debugging 기법에 의하여 종료되는 대상 Process의 목록도 확인 가능함.



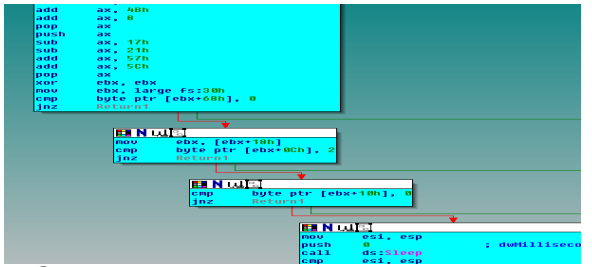
— 상세 Anti-Debugging 기법이 적용된 부분들…



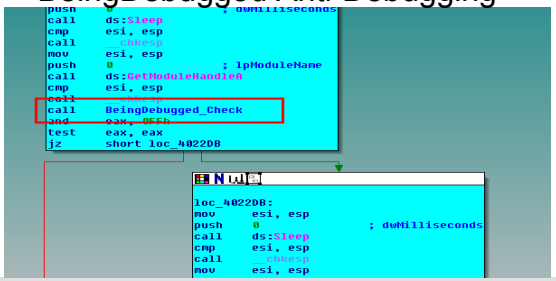
BeingDebugged Anti-Debugging



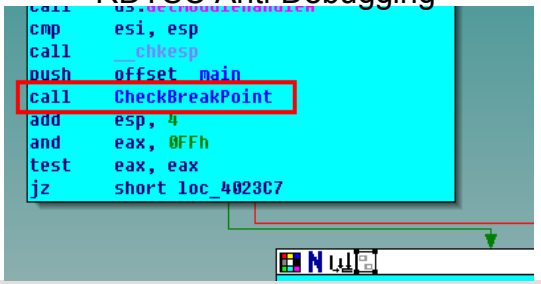
RDTSC Anti-Debugging



NtGlobalFlag, HeapFlags Anti-Debugging



BeingDebugged Anti-Debugging



Break point check Anti-Debugging

2. Script 문제 분석

❑ 문제목표

— Encoding된 HTML 파일을 Decoding하여 얻은 Javascript 코드를 분석하여 원하는 Password 획득

— Adobe Reader의 Buffer overflow 취약점을 이용한 공격을 위하여 임의로 삽입된 Java script를 추출, 분석하여 원하는 Password 획득

❑ 풀이환경

— 분석도구 : Internet Explorer, notepad, Ultraedit, pdftk(Linux)

❑ 풀이방법

— US-ASCII 및 escape로 encoding된 source code를 decoding.

— Pdftk를 이용하여 pdf 內 은닉/압축된 Stream을 추출 및 decoding 수행.

❑ 풀이결과

— 1번 문제의 답 : ASEC is AhnLab Security E-response Center

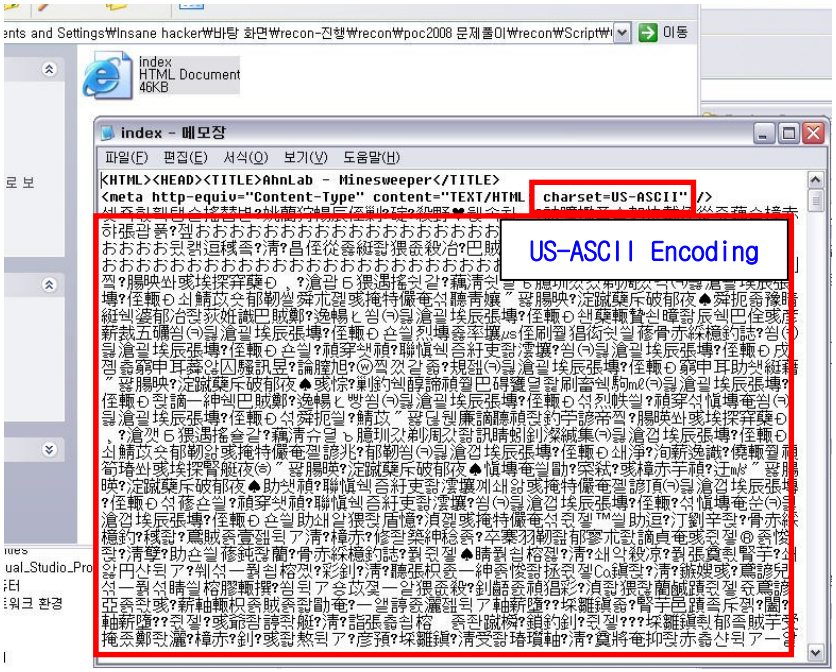
— 2번 문제의 답 : Do U Know ASEC?

2. Script 문제 분석

❑ Script 1번 문제 상세 분석

— Notepad.exe를 이용하여 문제의 파일을 열어보면 US-ASCII 형식으로 Encoding 된 HTML파일이라는 것을 알 수 있음.

— Internet Explorer에서 실행되는 Minesweeper를 통해 문제는 US-ASCII로 Encoding된 Java script라는 사실을 유추할 수 있음.



※ 부연설명: US-ASCII 인코딩 방식은 구형 컴퓨터에서 문자를 표현할 때 쓰이던 방식이며, 1개의 문자는 8비트(1바이트)로 표현

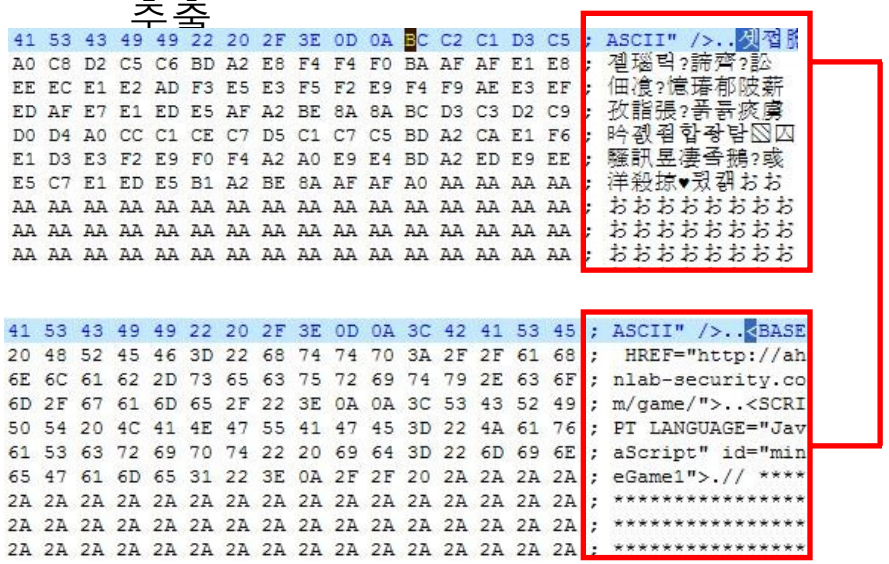
되는데, US-ASCII 방식은 이중 7비트를 취하고 나머지 1비트를 버린다. 예를 들어, 화면상에 문자를 표현하고자 할

때 10110101과 00110101 은 같은 결과를 보여준다.

❑ Script 1번 문제 상세 분석 (계속)

— Ultra Editor를 이용하여 해당 문자열을 수작업을 통해 일부 검증해 보면 Plain text를 볼 수
있음. (10000000은 Hexcode로 0x80이므로, 각 문자의 Hex값에서 0x80만큼만 빼주면 원하는 문자를 얻을 수 있음)

— 간단한 C code를 이용하여 전체 Code를 US-ASCII 방식으로 Decoding하여 Source code 추출



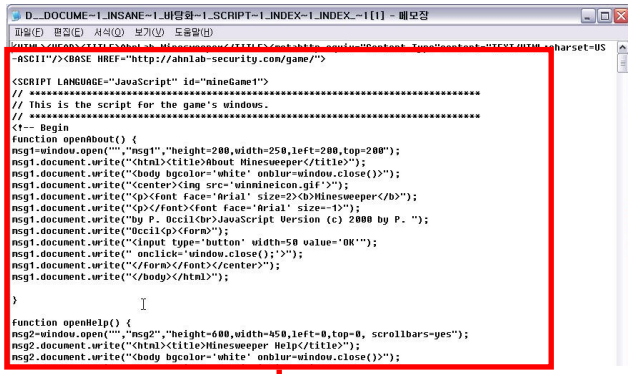

```
#include <iostream>
#include <fstream>

using namespace std;
int main() {
        unsigned char c;
        fstream in("index.html",ios::in); // 괴상한 파일 읽기
        fstream out("modified.html",ios::out); // 결과물.
        while(1) {
                in >> c; // 한글자씩 읽기.
                if (in.eof())
                        break;
                else if (c > 0x80)
                        c -= 0x80;
                out << c; // 파일에 출력
        }
        in.close();
        out.close();
        return 0;
}
```

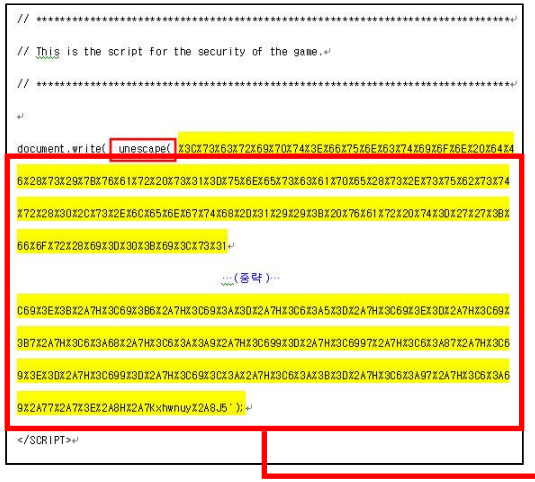
수작업 검증

추출 코드

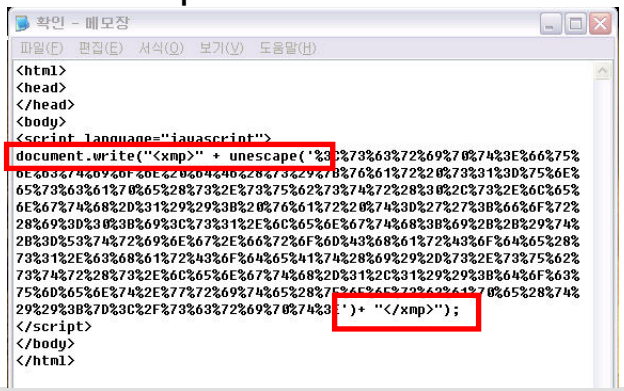
❑ Script 1번 문제 상세 분석 (계속)

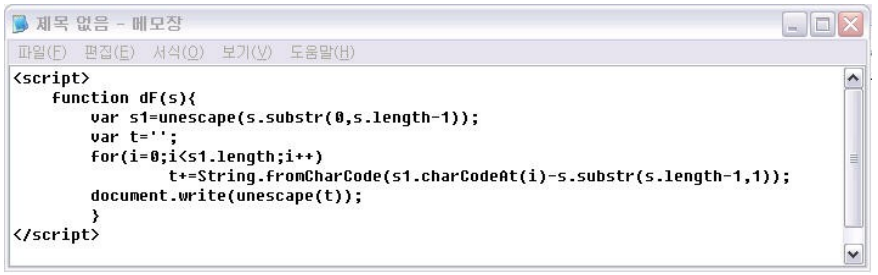
— Index.html을 decoding하면 plain-text code를 얻을 수 있으나, 내용 중 다시 Java script의 escape() 함수로 encoding된 부분을 확인할 수 있음.





— unescape(‘ ……..’) 부분의 문자열을 확인하기 위하여 unescape()를 사용하는 간단한 Java script를 작성하여 소스코드 확인. (dF()라는 함수가 문자열을 처리하는 Code 획득)

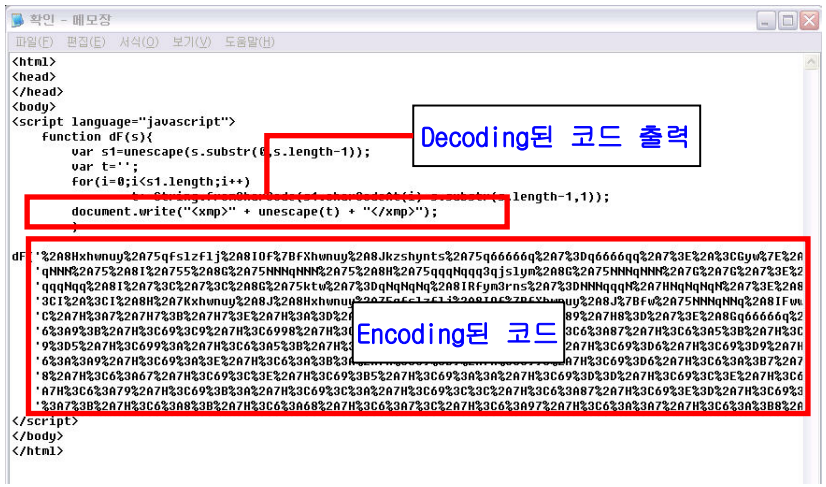


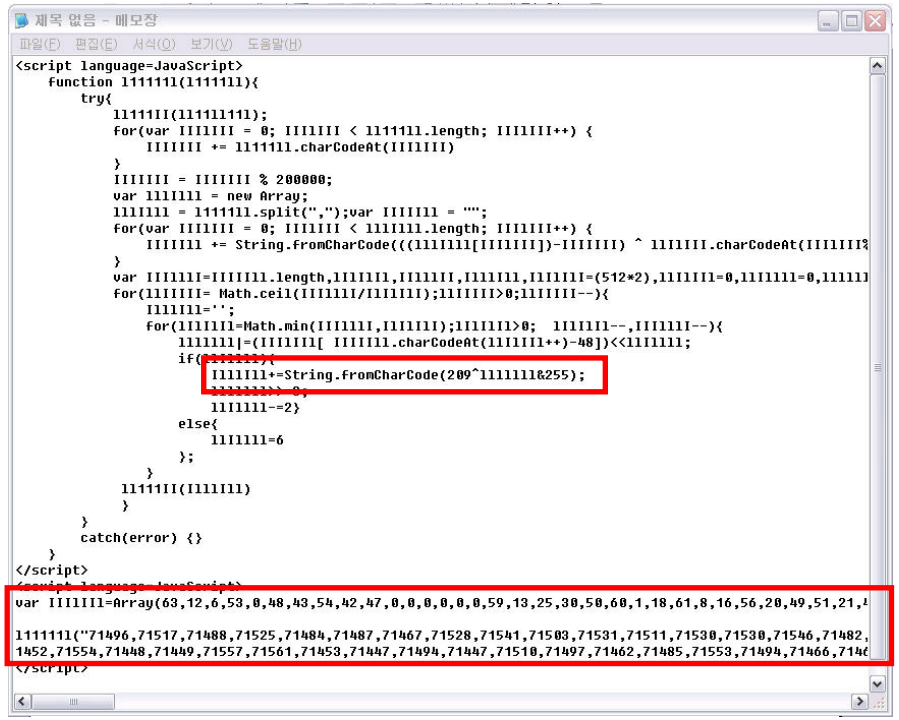


❑ Script 1번 문제 상세 분석 (계속)

— dF(s)를 아래와 같은 Java script를 이용하여 실행하면 새로운 Code를 생성함을 알 수 있음.

— 새로 생성된 Code는 다시 하단 부분의 배열을 Seed로 하여 특정 문자열을 생성해 내는 코드임을 확인할 수 있으나 몇몇의 미정의된 함수와 인자로 인하여 정상적인 실행은 불가.

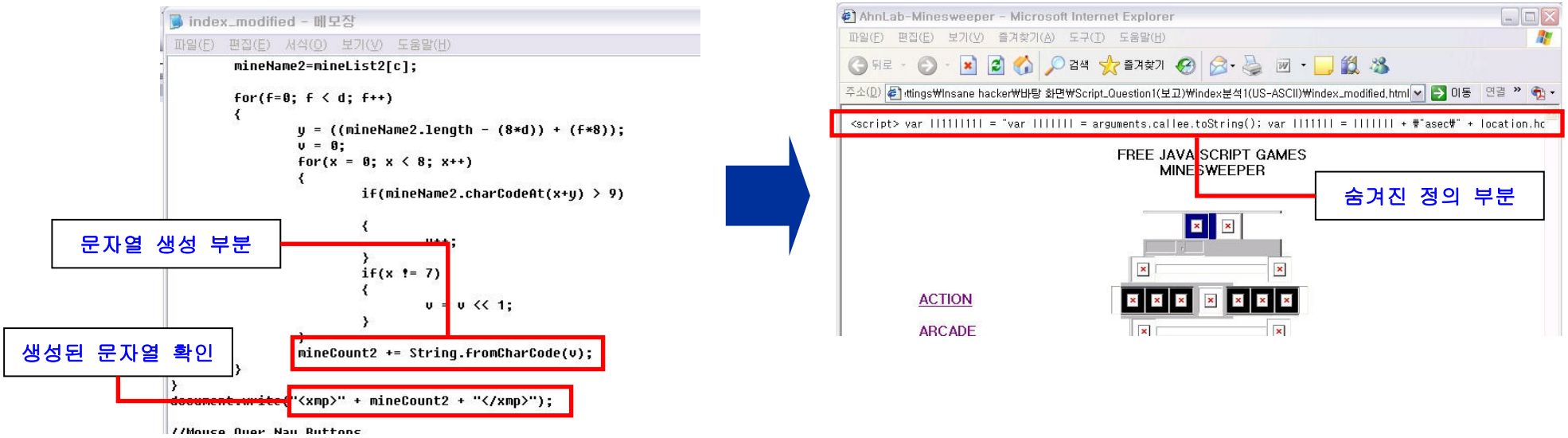




❑ Script 1번 문제 상세 분석 (계속)

— 분석된 Encoding 문자열에서는 미정의된 함수 및 변수에 대한 정보가 없으므로, 문제의 전체 실행코드를 분석하여 나머지 미정의 함수 및 변수에 대한 선언 부분을 검색해 보면, “mineGame2” 부분의 mineCount2 변수가 이름과 다르게 특정 문자열을 저장하는 변수로 사용되고 있음을 알 수 있음.

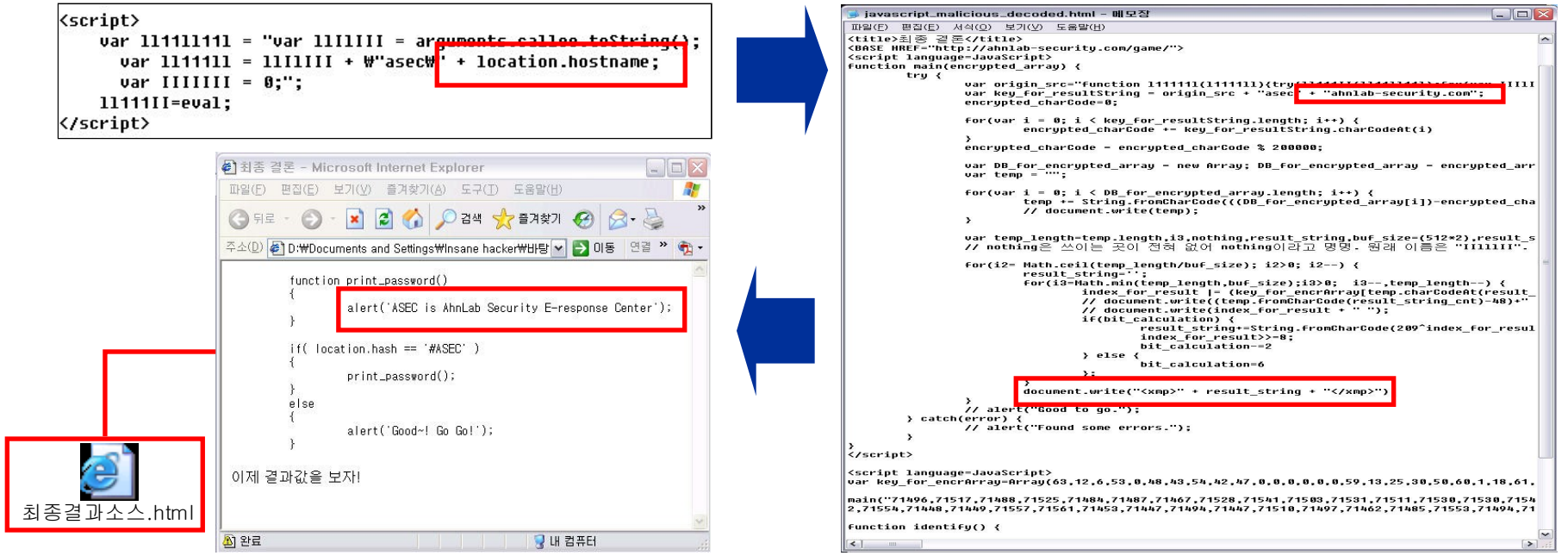
— 이 변수의 내용을 출력하도록 Java script 작성 후 웹브라우저를 통해 결과를 확인해보면, 숨겨진 변수 및 함수를 정의하는 source code를 얻을 수 있음.



❑ Script 1번 문제 상세 분석 (계속)

- 얻어진 source code를 모두 조합하여 정리해 보면 미정의 함수 및 변수의 문제는 해결이 되지만, 테스트를 수행하는 환경이 사용자의PC이므로 location.hostname을 읽는 부분에서 문제가 발생할 것이라는 것을 짐작할 수 있음.

- Index.html을 decoding하여 얻은 source code의 최상단에서 정의된 <base href=http://ahnlab-security.com/game/>에 착안하여 location.hostname을 "ahnlab-secrity.com"으로 치환한 후 웹브라우져를 통해 최종 결과를 확인하면 원하는 Pasword를 얻을 수 있음.

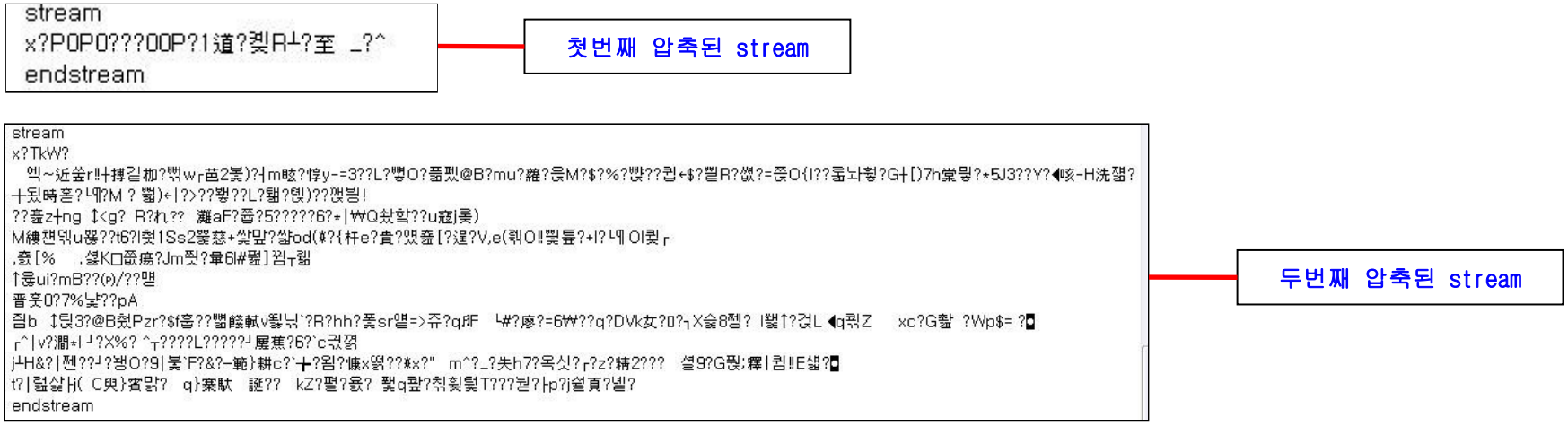


❑ Script 2번 문제 상세 분석

— 문제의 PDF 파일을 열어보면 아무 내용도 없는 백지를 출력하거나, 최신버젼의 Adobe Reader에서는 임의의 code 삽입에 대한 경고 메시지가 출력 됨.

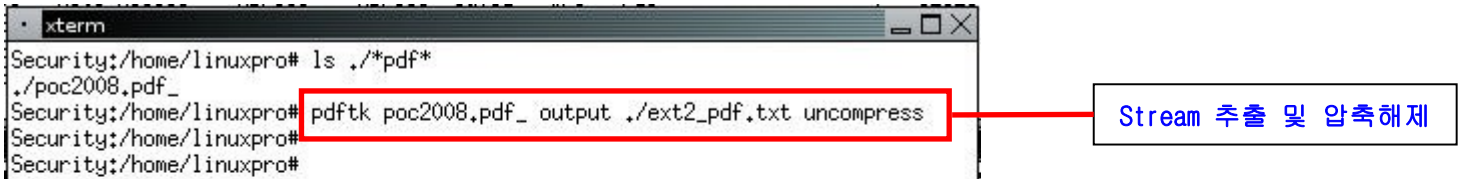
— Notepad 혹은 Wordpad를 이용하여 내용을 살펴보면, pdf 파일 內에 삽입된 두 개의 암호화된 Stream을 확인할 수 있음.

— 이 와 더불어 동일 파일 내의 “/FlateDecode”와 “<< /S /JavaScript /JS 13 0 R >>” 문자열을 통하여 해당 Stream이 ZLIB을 이용하여 압축된 Java Script라는 것을 짐작할 수 있음

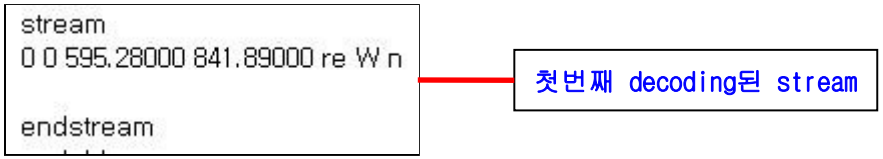


❑ Script 2번 문제 상세 분석 (계속)

— 문제의 Stream을 decoding하기 위하여 Linux 기반의 pdftk라는 유틸리티를 이용하면, 평문화된 문자열을 얻을 수 있음.



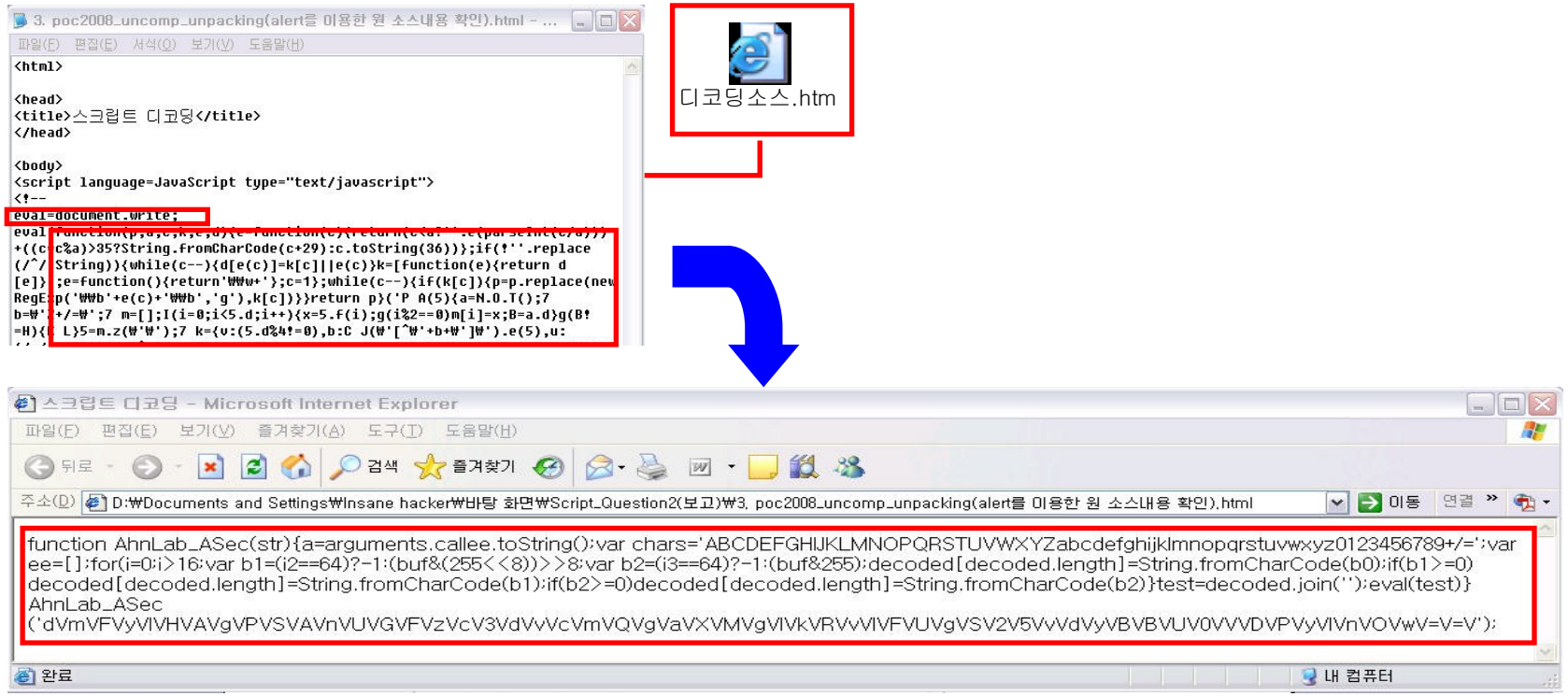
— Plain-Text로 변환된 문서를 Wordpad를 이용하여 확인하면 아래와 같이 두개의 decoding된 code를 확인할 수 있음.



❑ Script 2번 문제 상세 분석 (계속)

— 얻어진 Script는 “eval(function(p,a,c,k,e,d) {e=function(c) …” 이 후 분석이 난해하도록 packing된 형태로 출력되어 있음.

— eval() 함수는 변수의 문자열을 Script의 일부로 사용하는 함수 임에 착안하여, script부분만 추출하여 HTML형태로 만든 뒤, “eval=document.write”와 같이 함수를 재선언하는 방법을 통해 packing된 source code의 decoding 형태를 얻을 수 있음.



❑ Script 2번 문제 상세 분석 (계속)

— 최종적으로 얻어진 Script는 하단의 “dVmVFVyVIVHVAVgVPVSVAVnVUVGVFVzVcV3VdVvVcVmVQVgVaVXVMVgVIVkVRVvVIVFVUVgVSV2V5VvVdVyVBVBVUV0VVVDVPVyVIVnVOVwV=V=V”라는 문자열을 argument로 받아 decoding하는 방식의 함수이며, 마지막 부분의 “eval(test)”가 모든 decoding 완료된 결과를 실행하는 부분이라 추측 할 수 있음.

— 아래의 세 줄의 code는 스크립트 변조 유/무를 점검하는 부분이므로 주석처리를 하여 결과

출력에 방해가 되지 않도록 수정하고 eval(test) 부분을 document.write(“<xmp>” + test + “</xmp>”);로 수정하여 최종 결과를 확인할

```
a=arguments.callee.toString();
ss=a.length
if(ss!=1154) {return false} << 주석처리
```

```
주소(D) ;tion2(보고)₩4. poc2008_result(최종 결론 확인).h
var p = 'Password is "Do U Know ASEC?"';
```

```
        /* if(ss!=1154) {return false} */
        str=ee.join('');

        var invalid={strlen:(str.length%4!=0),chars:new RegExp('[^'+chars+']').test(str),equals:
(/=/.test(str)&&(/=[^=]/.test(str)||/={3}/.test(str))));

        if(invalid.strlen||invalid.chars||invalid.equals) throw new Error('Invalid Data');

        var decoded=[];
        var c=0;

        while(c<str.length) {
                var i0=chars.indexOf(str.charAt(c++));
                var i1=chars.indexOf(str.charAt(c++));
                var i2=chars.indexOf(str.charAt(c++));
                var i3=chars.indexOf(str.charAt(c++));
                var buf=(i0<<18)+(i1<<12)+((i2&63)<<6)+(i3&63);
                var b0=(buf&(255<<16)))>>16;
                var b1=(i2==64)?-1:(buf&(255<<8)))>>8;
                var b2=(i3==64)?-1:(buf&255);
                decoded[decoded.length]=String.fromCharCode(b0);

                if(b1>=0)decoded[decoded.length]=String.fromCharCode(b1);
                if(b2>=0)decoded[decoded.length]=String.fromCharCode(b2)
        }

        test=decoded.join('');
        document.write("<xmp>"+test+"</xmp>");
}
//-->
</script>
</head>
```

3. Network 문제 분석

❑ 문제목표

- 주어진 Packet dump file을 분석하여, 수행된 공격기법 파악 및 숨겨진 Hint 파악
  .
- Hint를 이용하여 두 번째 Packet dump file에서 탐지된 공격기법 파악 및 최종 Password 획득

❑ 풀이환경

- 분석도구 : Wireshark, XVI32, file, cat, editcap, picozip

❑ 풀이방법

- Dump file의 encapsulation 형태를 분석 가능한 형식으로 변환하고, 주요 Packet 內 data payload를 재조합하여 Hint가 포함된 Binary(Jpeg) 추출
- Dump file의 변조된 Header를 복구하고, Mail 서버의 Login buffer overflow 취약점을 이용한 공격 Packet에서 임의의 Script를 추출하여 최종 Password 획득

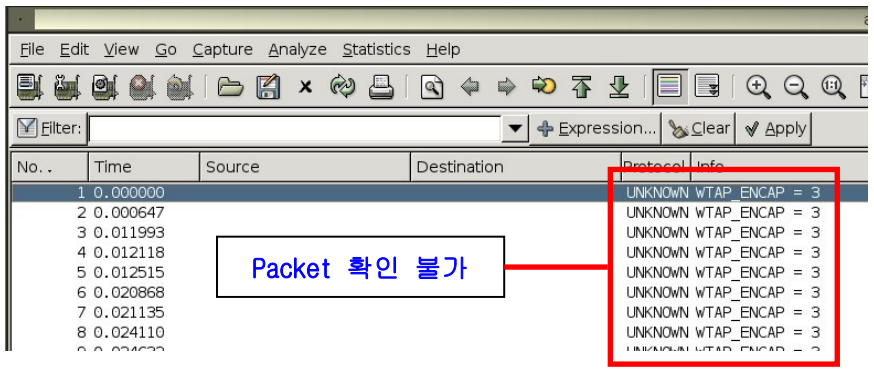
❑ 풀이결과

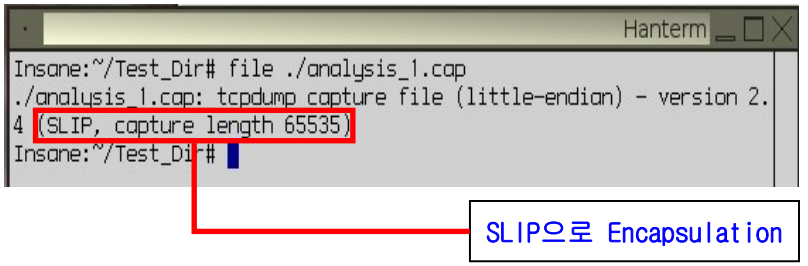
- 1번 문제의 답 : Hint 88
- 2번 문제의 답 : NPqnstjr!@#

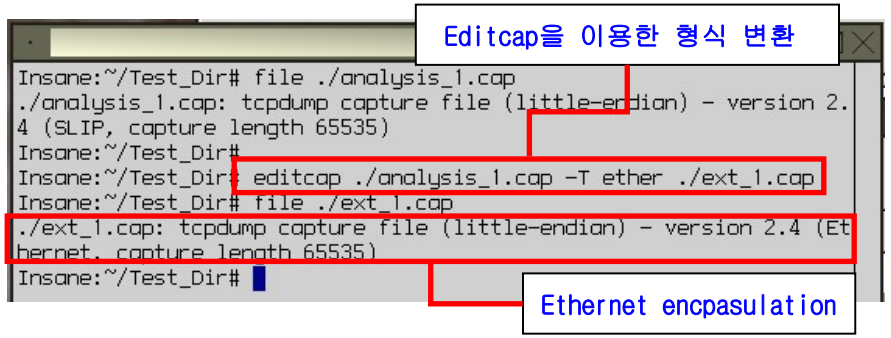
❑ Network 1번 문제 상세 분석

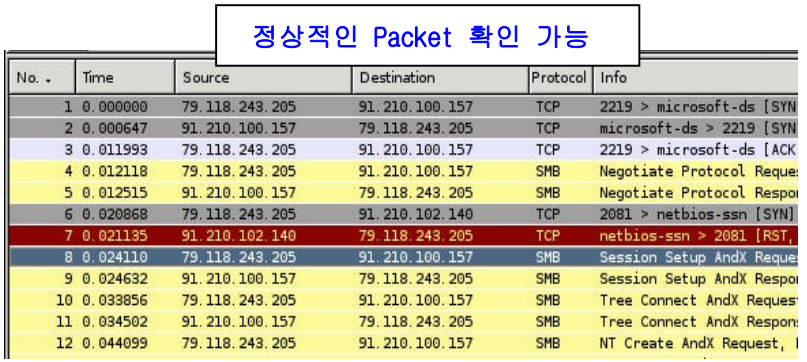
— 분석을 위하여 주어진 Dump file을 Wireshark를 이용하여 열어보면 정상적인 Packet 분석이 불가능 함을 알 수 있음. (file 유틸을 이용하여 dump file 형식 확인)

— Wireshark와 함께 제공되는 editcap을 이용하여 Dump file을 SLIP → ethernet encapsulation 형식으로 변환하면 정상적인 Packet 확인 가능.







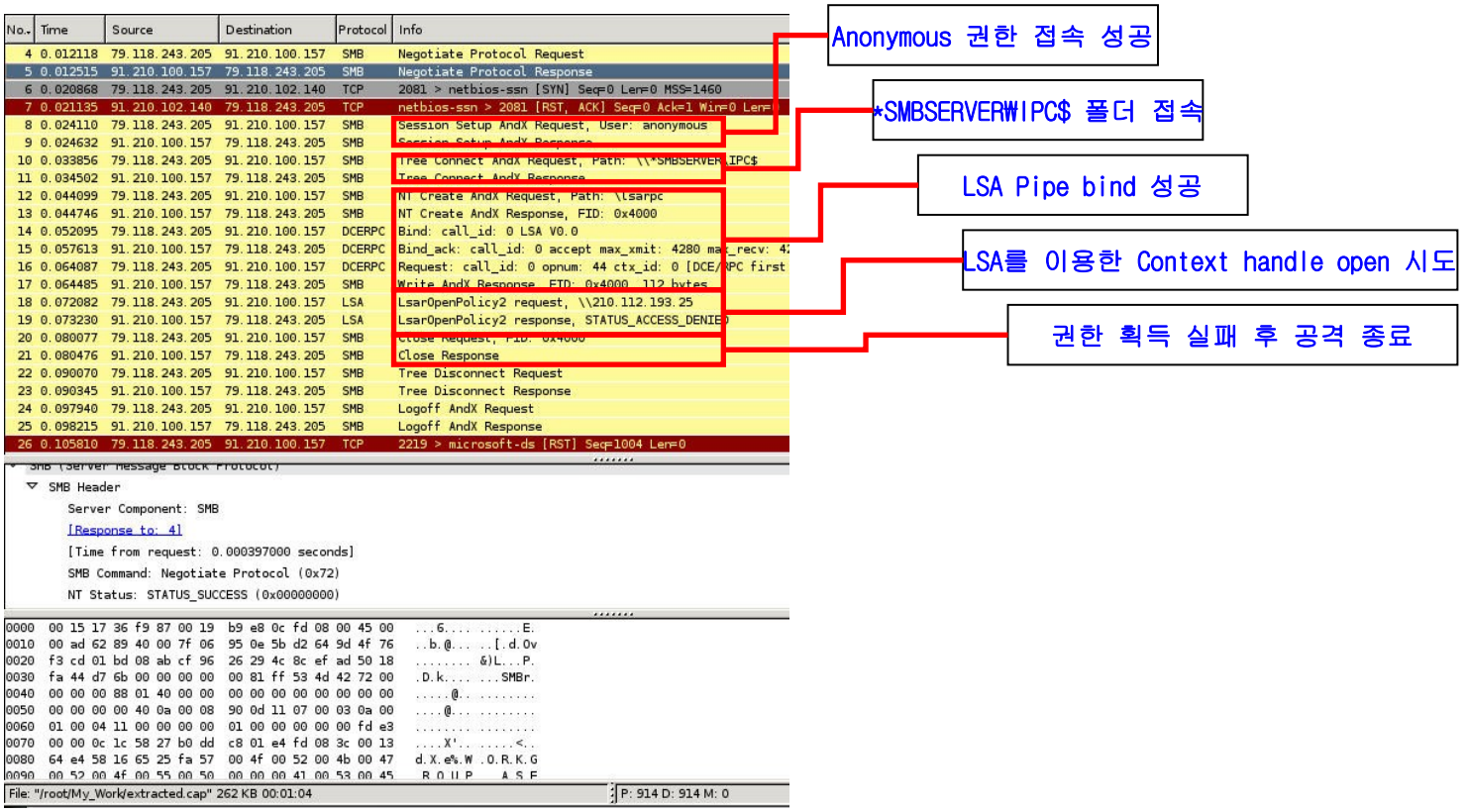


※ 부연설명: editcap은 wireshark와 함께 제공되는 Utility로서, packet encapsulation 변환, packet 조합, filtering, 분리 등
dump file을 재조합하는데 사용되는 Utility 임.

❑ Network 1번 문제 상세 분석 (계속)

— 주요상황 분석 (1) : 79.118.243.205 ↔ 91.210.100.157

- SMB Protocol을 이용하여 Anonymous 권한을 통해 취약 서버에 접속 후 *SMBSERVER\IPC$ 기본공유 폴더를 경유, Lsarpc pipe를 이용한 타 서버에 대한 파일생성 권한 획득을 시도하지만, 권한판정에 의하여 접속 거부됨. Lsarpc pipe를 이용하는 점으로 미루어 Worm에 의한 공격이라 판단 됨.



❑ Network 1번 문제 상세 분석 (계속)

— 주요상황 분석 (2) : 91.236.114.204에 의한 Host Sweep 발생

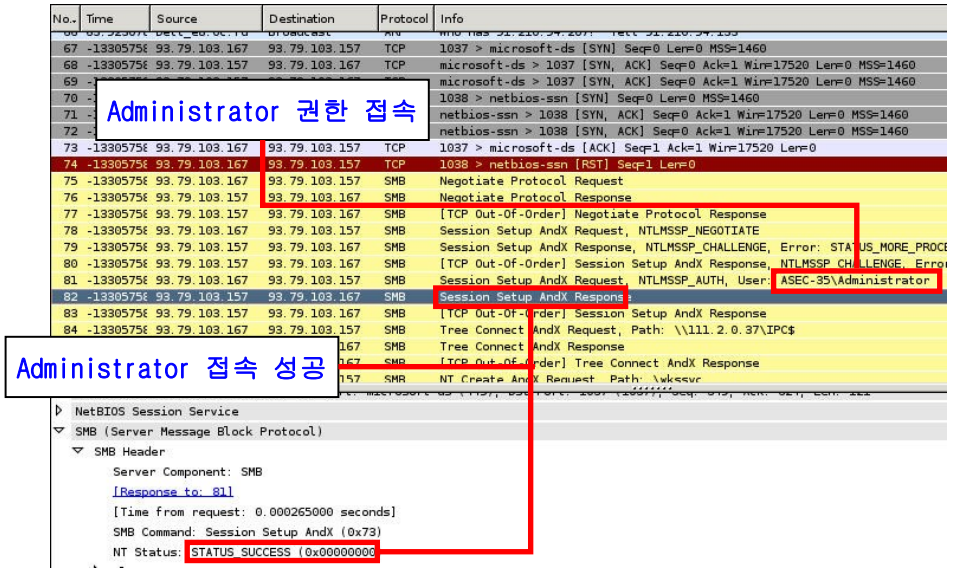
- 91.236.114.204에 의하여 139 Port에 대한 Host Sweep 발생 : 이 후 특별한 공격 수행 없음

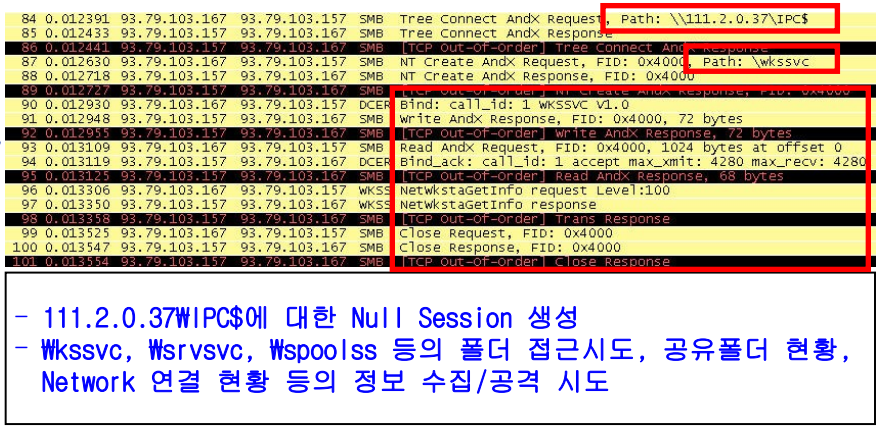
No.	Time	Source	Destination	Protocol	Info
35	0.257393	93.85.93.246	91.210.92.238	ICMP	Destination unreachable (Host unreachable)
36	0.362190	91.210.92.238	91.216.239.140	TCP	loc-srv > 4637 [SYN, ACK] Seq=0 Ack=1 Win=25200 Len=0 MSS=1
37	63.81958	91.236.114.204	91.210.94.207	TCP	fax > netbios-ssn [SYN] Seq=0 Len=0 MSS=1460
38	63.81958	91.236.114.204	91.210.94.206	TCP	4555 > netbios-ssn [SYN] Seq=0 Len=0 MSS=1460
39	63.81970	91.236.114.204	91.210.94.205	TCP	4556 > netbios-ssn [SYN] Seq=0 Len=0 MSS=1460
40	63.81983	91.236.114.204	91.210.94.198	TCP	4562 > netbios-ssn [SYN] Seq=0 Len=0 MSS=1460
41	63.81983	91.236.114.204	91.210.94.196	TCP	4560 > netbios-ssn [SYN] Seq=0 Len=0 MSS=1460
42	63.81984	91.236.114.204	91.210.94.197	TCP	4561 > netbios-ssn [SYN] Seq=0 Len=0 MSS=1460
43	63.81988	91.210.94.207	91.236.114.204	TCP	netbios-ssn > fax [RST, ACK] Seq=0 Ack=1 Win=0 Len=0
44	63.81988	91.210.94.206	91.236.114.204	TCP	netbios-ssn > 4555 [RST, ACK] Seq=0 Ack=1 Win=0 Len=0
45	63.82000	91.210.94.205	91.236.114.204	TCP	netbios-ssn > 4556 [RST, ACK] Seq=0 Ack=1 Win=0 Len=0
46	63.82013	91.210.94.198	91.236.114.204	TCP	netbios-ssn > 4562 [RST, ACK] Seq=0 Ack=1 Win=0 Len=0
47	63.82025	91.210.94.196	91.236.114.204	TCP	netbios-ssn > 4560 [RST, ACK] Seq=0 Ack=1 Win=0 Len=0
48	63.82025	91.210.94.197	91.236.114.204	TCP	netbios-ssn > 4561 [RST, ACK] Seq=0 Ack=1 Win=0 Len=0
49	63.91702	91.236.114.204	91.210.94.199	TCP	4567 > netbios-ssn [SYN] Seq=0 Len=0 MSS=1460
50	63.91732	91.210.94.199	91.236.114.204	TCP	netbios-ssn > 4567 [RST, ACK] Seq=0 Ack=1 Win=0 Len=0
51	63.91802	91.236.114.204	91.210.94.212	TCP	4568 > netbios-ssn [SYN] Seq=0 Len=0 MSS=1460
52	63.91864	91.236.114.204	91.210.94.213	TCP	iax > netbios-ssn [SYN] Seq=0 Len=0 MSS=1460
53	63.91882	91.210.94.213	91.236.114.204	TCP	netbios-ssn > iax [RST, ACK] Seq=0 Ack=1 Win=0 Len=0
54	63.91907	Dell_e8:0c:fd	Broadcast	ARP	Who has 91.210.94.212? Tell 91.210.94.133
55	63.91944	91.210.94.212	91.236.114.204	TCP	netbios-ssn > 4568 [RST, ACK] Seq=0 Ack=1 Win=0 Len=0
56	63.91952	91.236.114.204	91.210.94.214	TCP	4570 > netbios-ssn [SYN] Seq=0 Len=0 MSS=1460
57	63.92001	91.236.114.204	91.210.94.215	TCP	4571 > netbios-ssn [SYN] Seq=0 Len=0 MSS=1460
58	63.92039	91.236.114.204	91.210.94.205	TCP	4572 > netbios-ssn [SYN] Seq=0 Len=0 MSS=1460
59	63.92052	91.236.114.204	91.210.94.207	TCP	4574 > netbios-ssn [SYN] Seq=0 Len=0 MSS=1460
60	63.92052	91.236.114.204	91.210.94.206	TCP	4573 > netbios-ssn [SYN] Seq=0 Len=0 MSS=1460
61	63.92076	91.236.114.204	91.210.94.204	TCP	4575 > netbios-ssn [SYN] Seq=0 Len=0 MSS=1460
62	63.92094	91.210.94.204	91.236.114.204	TCP	netbios-ssn > 4575 [RST, ACK] Seq=0 Ack=1 Win=0 Len=0

❑ Network 1번 문제 상세 분석 (계속)

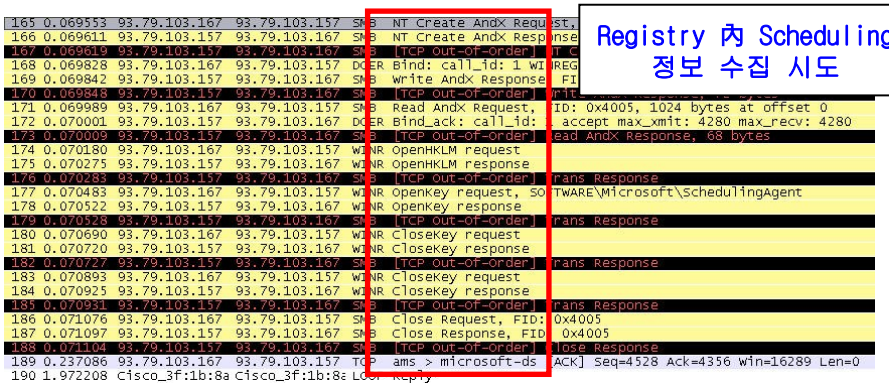
— 주요상황 분석 (3) : 93.79.103.167 → 93.79.103.157의 공유 Port(TCP-445) 취약점을 이용한 공격 발생

- Administrator 권한에 대한 NTLM 인증 후 Enumeration 발생

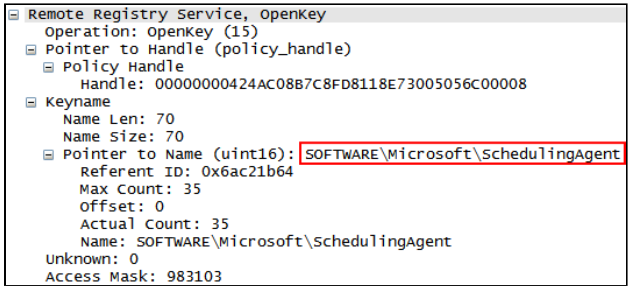




- 111.2.0.37₩IPC$에 대한 Null Session 생성
- ₩kssvc, ₩srvsvc, ₩spoolss 등의 폴더 접근시도, 공유폴더 현황, Network 연결 현황 등의 정보 수집/공격 시도



Registry 內 Scheduling 정보 수집 시도



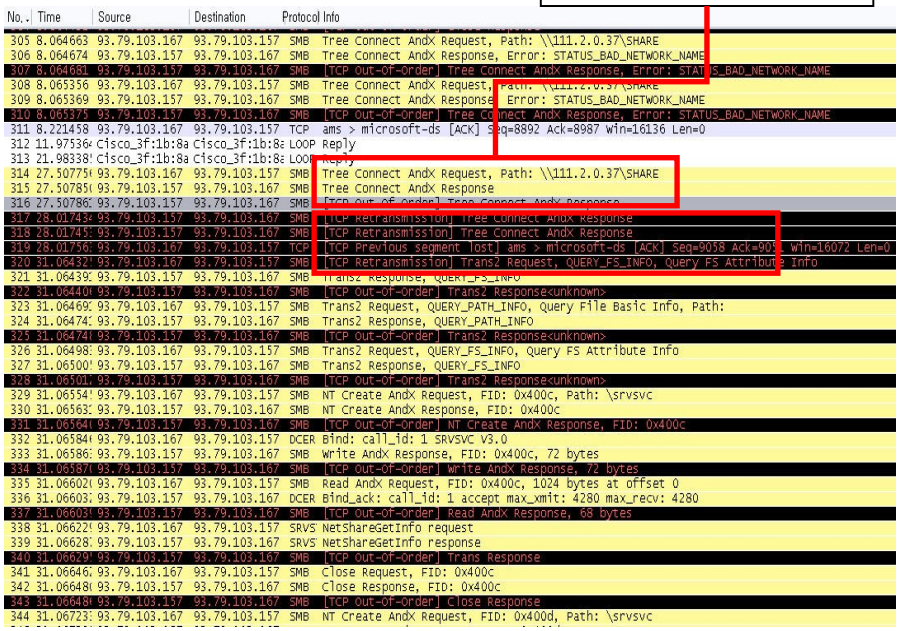
❑ Network 1번 문제 상세 분석 (계속)

— 주요상황 분석 (4) : 93.79.103.167 → 93.79.103.157의 공유 Port(TCP-445) 취약점을 이용한 공격 발생 (접속 권한이 Administrator → \ 로 변경)

- / 권한에 대한 NTLM 인증 후 Enumeration 발생, 314번째 Packet에서 Share 폴더에 접속 허가 되며

  공격 양상이 바뀌게 됨.

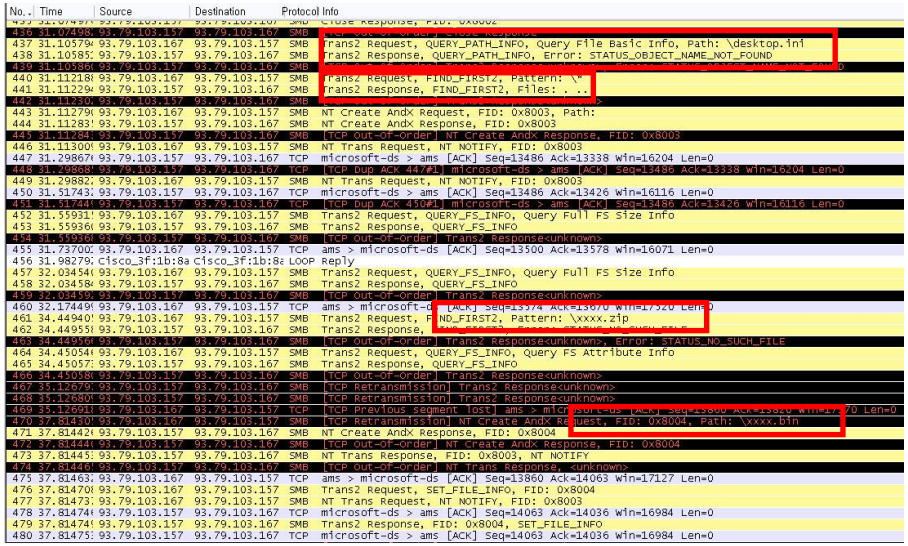
위와 다르게 STATUS_BAD 에러가 발생하지 않음



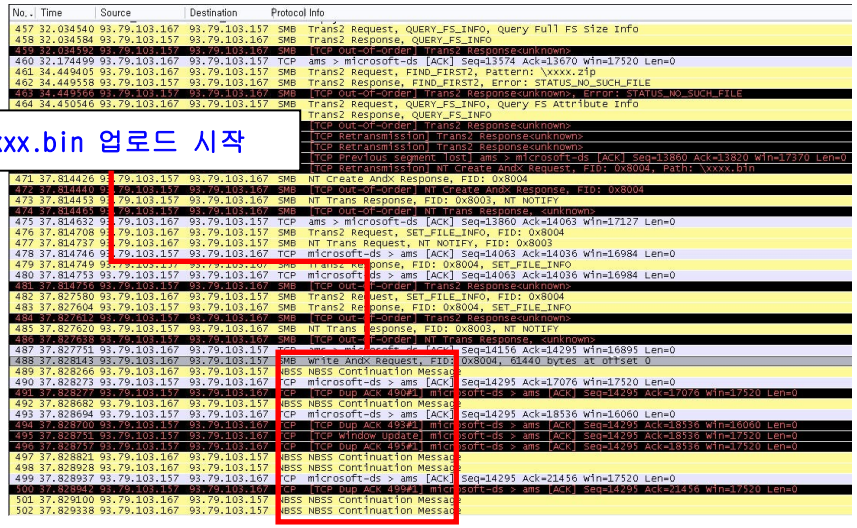
❑ Network 1번 문제 상세 분석 (계속)

- 주요상황분석(4) : 다량의 정보 수집 후 Desktop.ini, xxxx.bin 등의 파일 존재 유/무를 확인 후 존재하지 않을 경우 xxxx.bin 파일을 Upload하는 공격을 수행 함.

- 전체적으로 짧은 시간 내 다량의 정보수집 및 공격이 이루어지는 것으로 볼 때 , Window2000 계열의 공유폴더 취약점을 대상으로 한 Exploit 혹은 Worm을 이용한 공격으로 판단 됨.



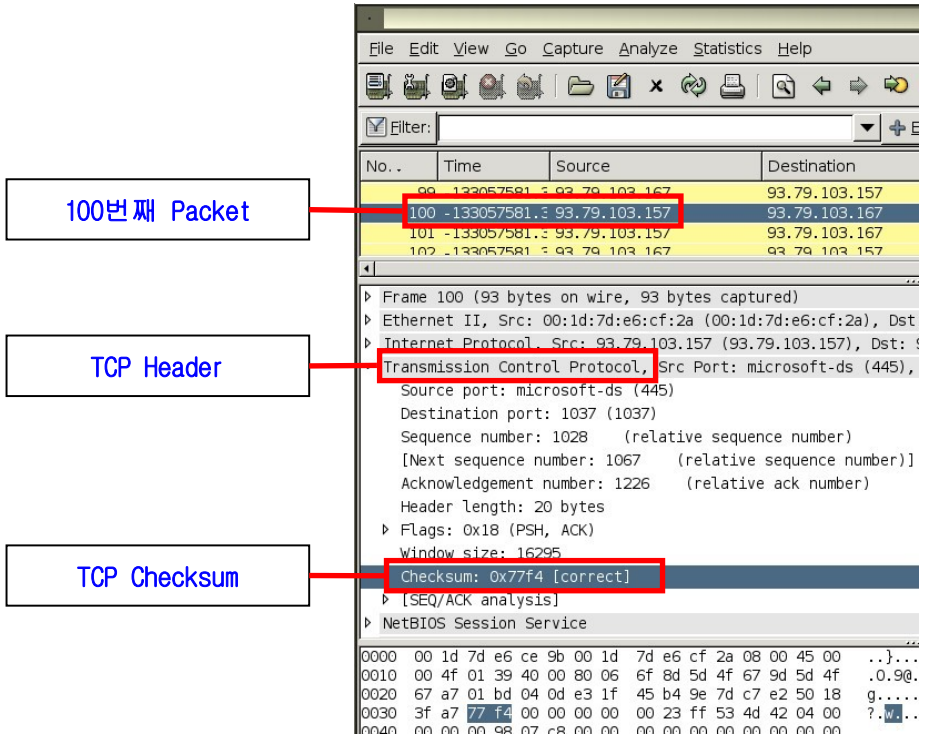
xxxx.bin 업로드 시작

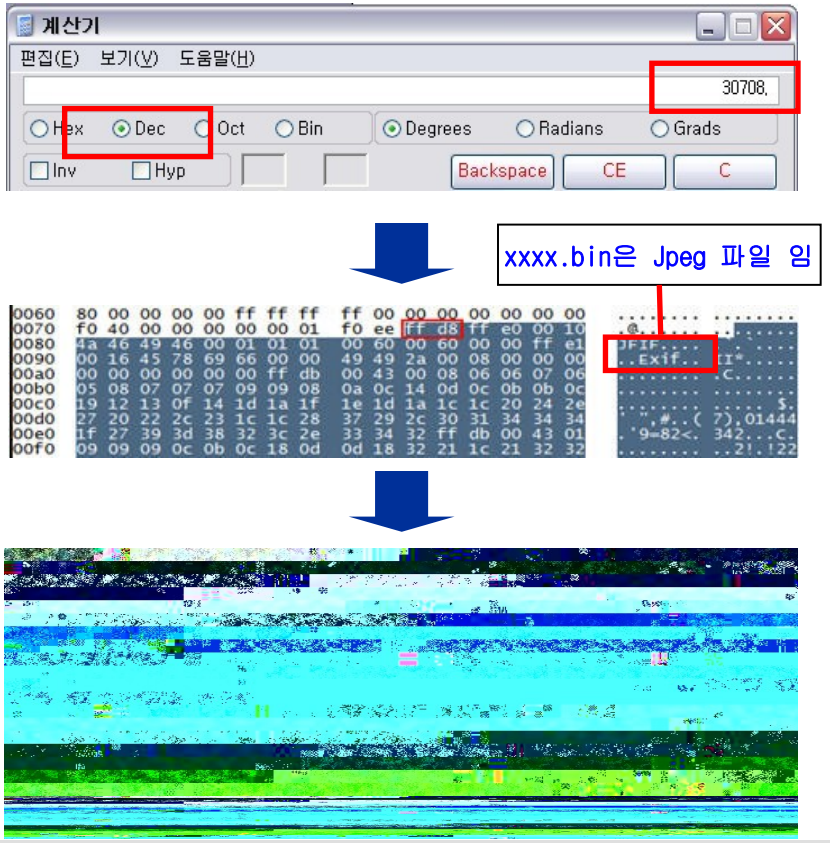


❑ Network 1번 문제 상세 분석 (계속)

— 문제풀이 : Analysis_1.cap 內 100번째 Packet의 TCP Header checksum 값 확인

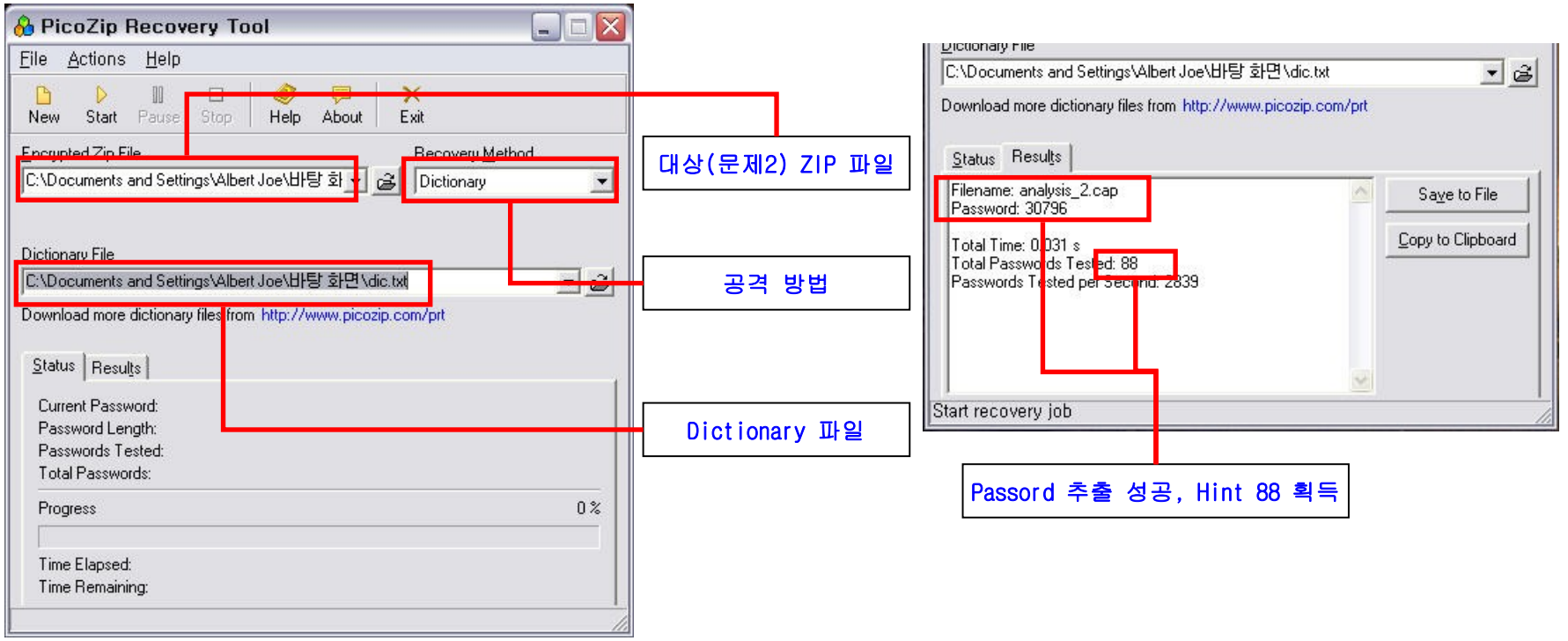
— 앞에서 설명한 93.79.103.167 → 93.79.103.157로 보내는 Data payload를 조합하는 방법을 이용하여 xxxx.bin을 재구성 (Jpeg 파일 임 – 문제의 오류로 구분 불가)





❑ Network 1번 문제 상세 분석 (계속)

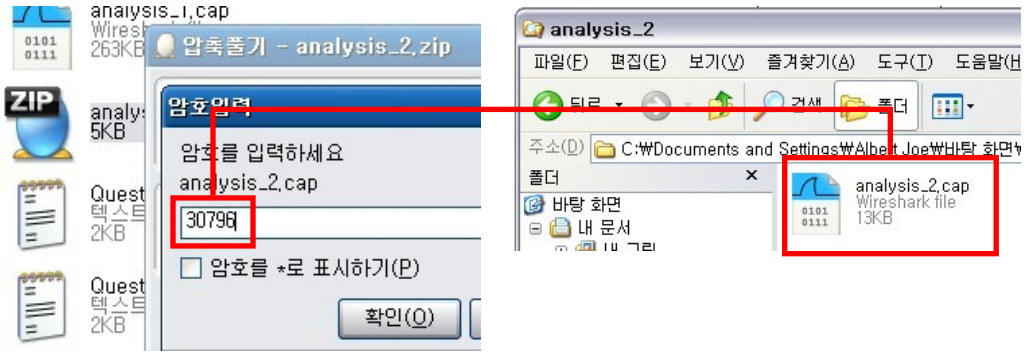
— 문제풀이 : 주최 측에서 새롭게 xxxx.bin 파일을 제공하기 전까지 “Hint 88”을 확인할 수 없어, 자동 zip password recovery 툴을 이용하여 Dictionary 공격으로 두번째 문제의 zip 파일 압축암호(30796)을 얻을 수 있었음.

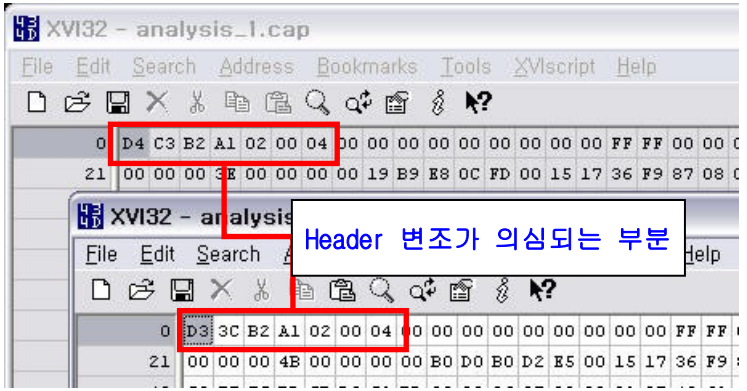


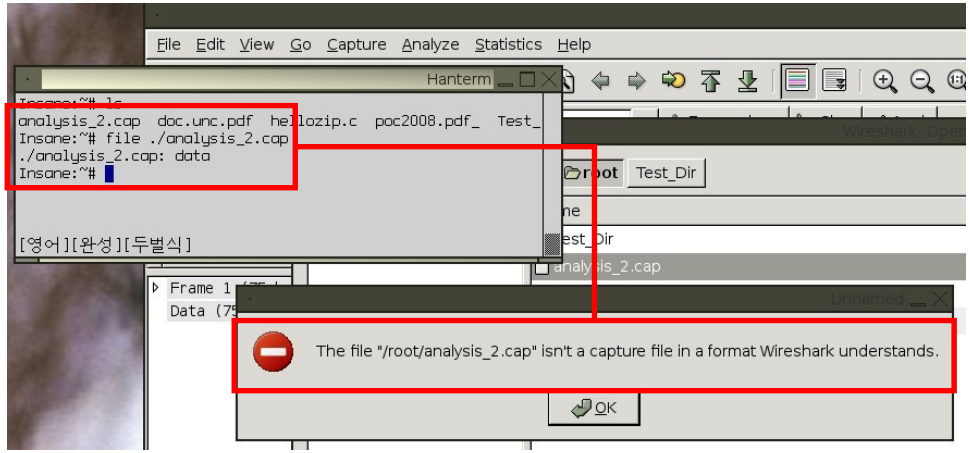
❑ Network 2번 문제 상세 분석

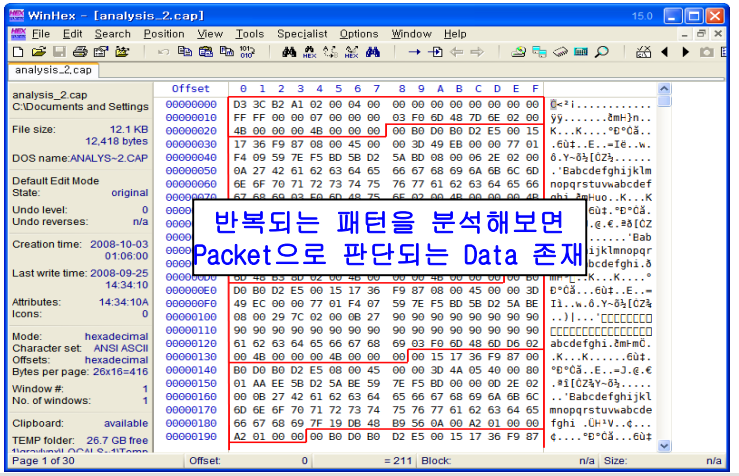
— ‘30796’을 이용하여 압축 해제 후 file이나 Wireshark를 이용한 분석시도 실패.

— 문제의 file을 winhex나 Xvi32를 이용하여 확인하면 Dump 파일임을 확인할 수 있음.





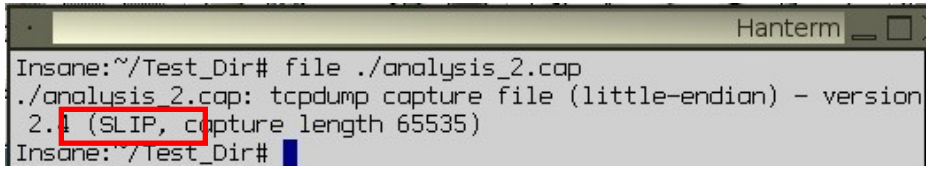


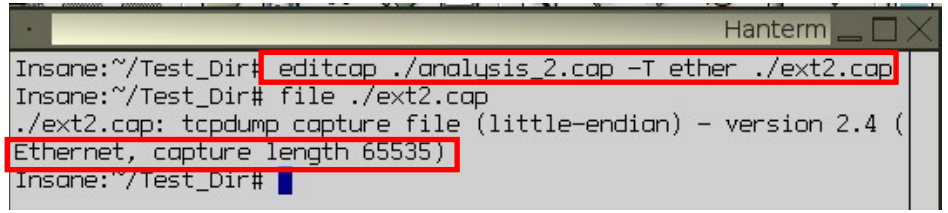


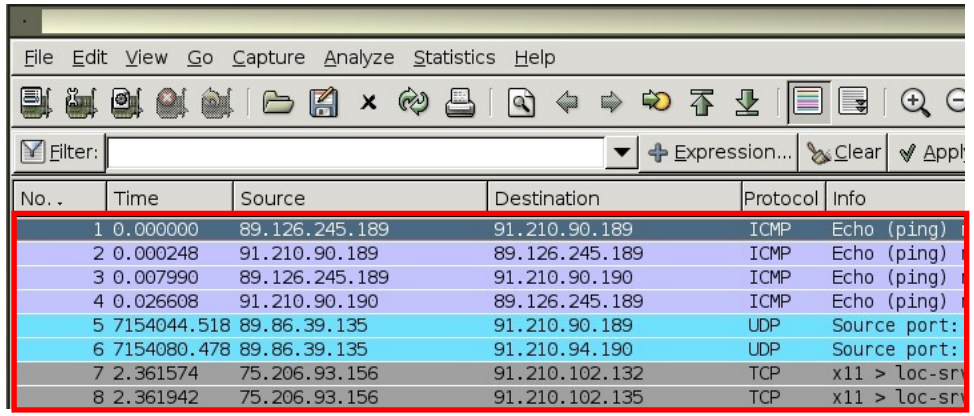
❑ Network 2번 문제 상세 분석

— 변조된 Header 복구 후 file utility를 이용하여 encapsulation type을 확인.

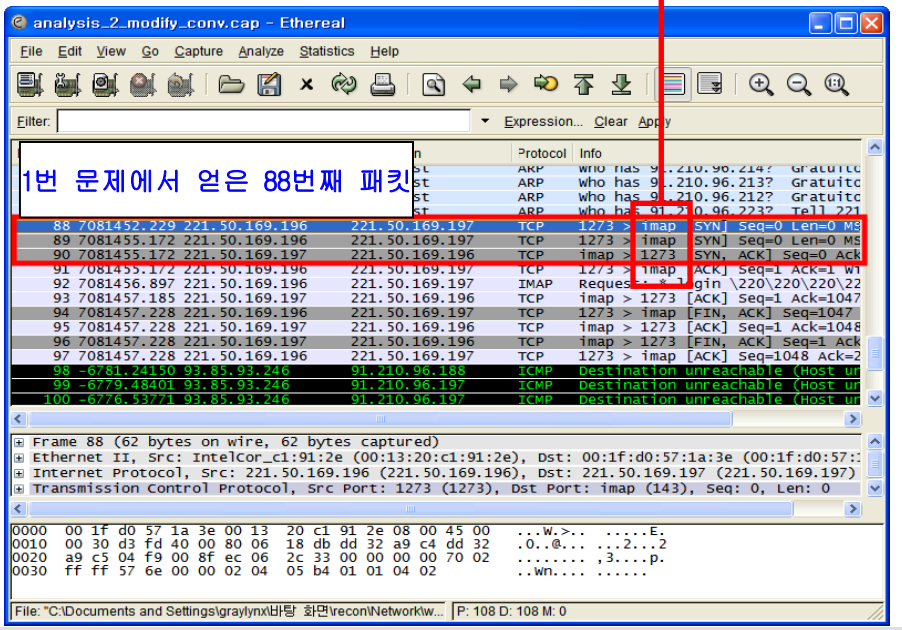
— 1번 문제와 동일하게 SLIP → Ethernet으로 Encapsulation 변환하면 정상 Dump 확인 가능





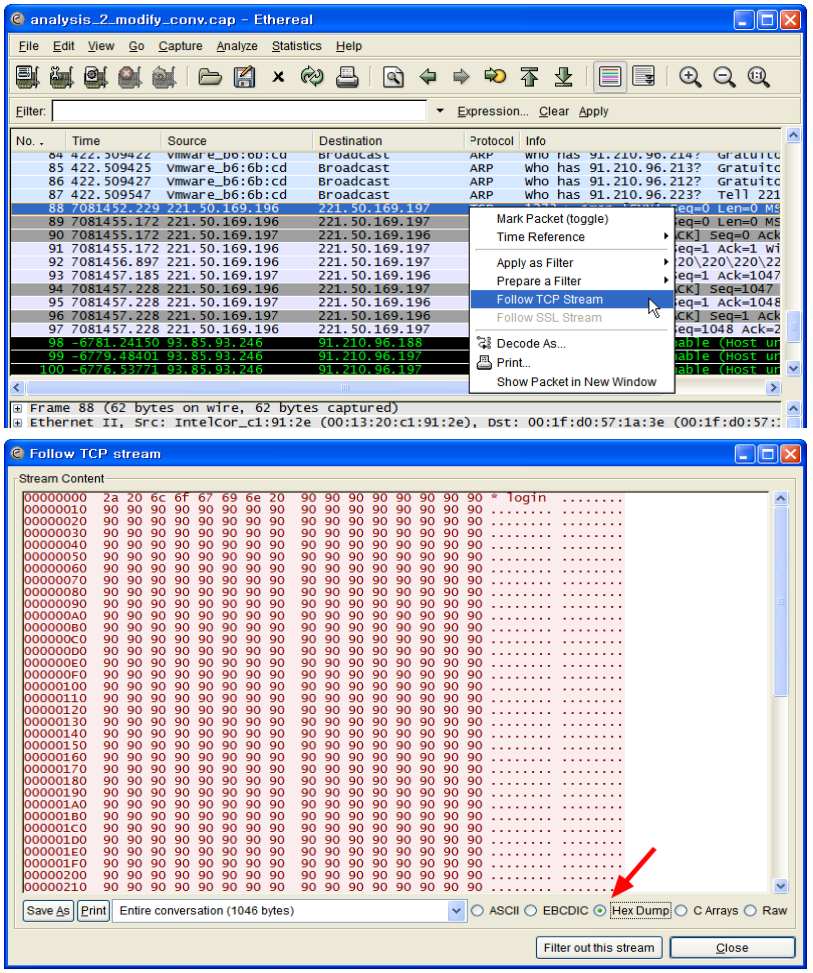


IMAP 관련 취약점을 이용한 공격 추정



❑ Network 2번 문제 상세 분석

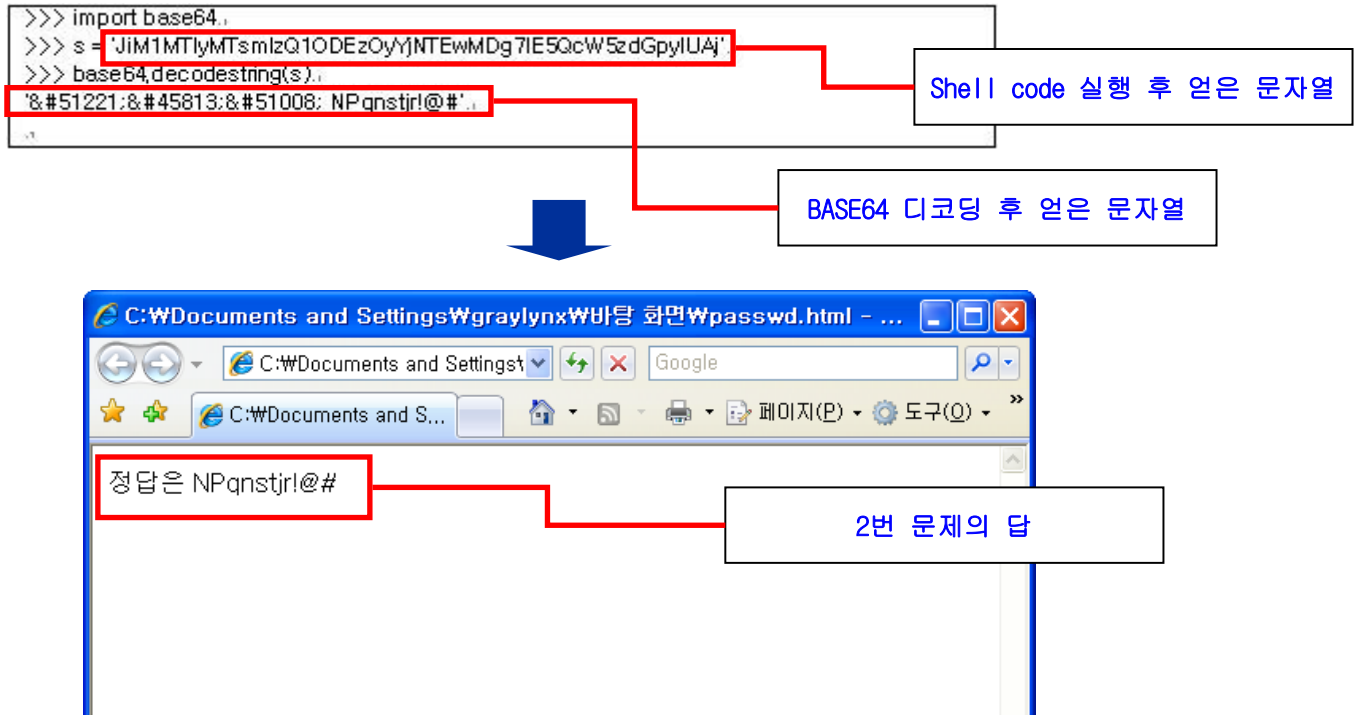
— IMAP관련 접속에 대한 자세한 분석을 위해 Follow TCP Stream을 이용하여 IMAP Session에 대한 분석 시도



❑ Network 2번 문제 상세 분석

— 추출된 Shell code를 실행시킬 경우 BASE64로 인코딩 된 것으로 판단되는 문자열을 얻을 수 있으며, Python을 이용한 간단한 Script로 해당 문자열을 디코딩할 수 있음.

— 디코딩된 문자열을 HTML파일로 저장한 뒤 웹 브라우져로 열면 정답을 확인할 수 있음.



4. Spyware 문제 분석

❑ 문제목표

— Ugo Boy BHO에 대한 진단/분석/조치/예방 방법 기술

— Spyware ‘sapkin.exe’에 대한 진단/분석/조치/예방 방법 기술

❑ 풀이환경

— 분석도구 : Wireshark, winanalysis, Ollydbg, IDA Pro, TCPView, Process explorer 등

❑ 풀이방법

— IDA Pro 및 Ollydbg를 이용한 Sample.dll 분석 및 Password 출력 조건 분석

— Winanalysis 및 Sysinternal Suite를 이용하여 sapkin.exe의 실행 전/후 시스템 변경 사항 점검, IDA Pro를 이용한 sapkin.exe의 행동방식 분석

❑ 풀이결과

— 1번 문제의 답 : AS long as you love me

— 2번 문제의 답 : 복사 작업 중 Clipboard에 저장되는 정보를 sapkin.log 파일로 저장하는 일종의 Key logger.

❑ Spyware 1번 - “진단”

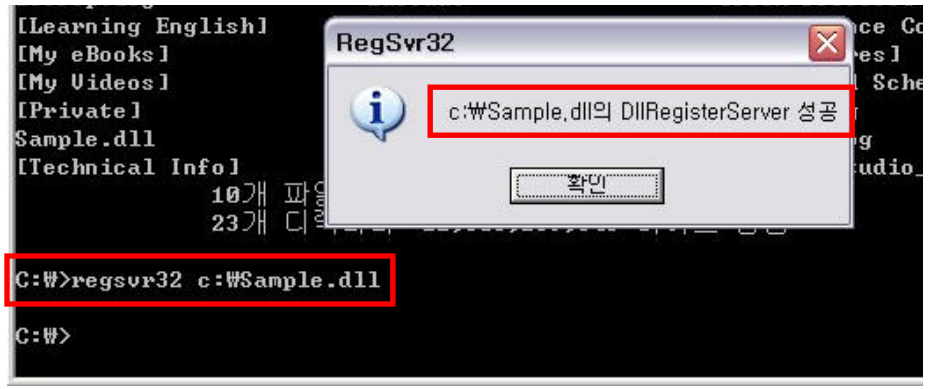
— BHO의 개념

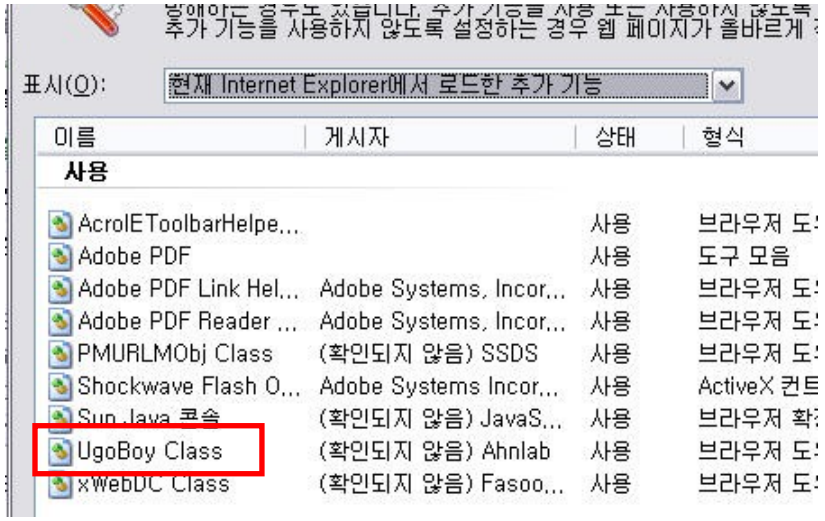
- BHO(Browser Helper Object)는 Internet Explorer Browser에서 기본적으로 지원하는 기능 외에 별도의 기능을 지원하기 위하여 Plug-in 형태로 Internet Explorer에 추가되는 DLL 모듈.
- BHO가 제대로 동작하기 위해서는 Registry 內의,

(1) \HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\Browser helper Object 부분에 CLSID가 등록되어야 하며,

(2) \HKCR\CLSID\에 Class ID가 등록되어 있어야 한다.

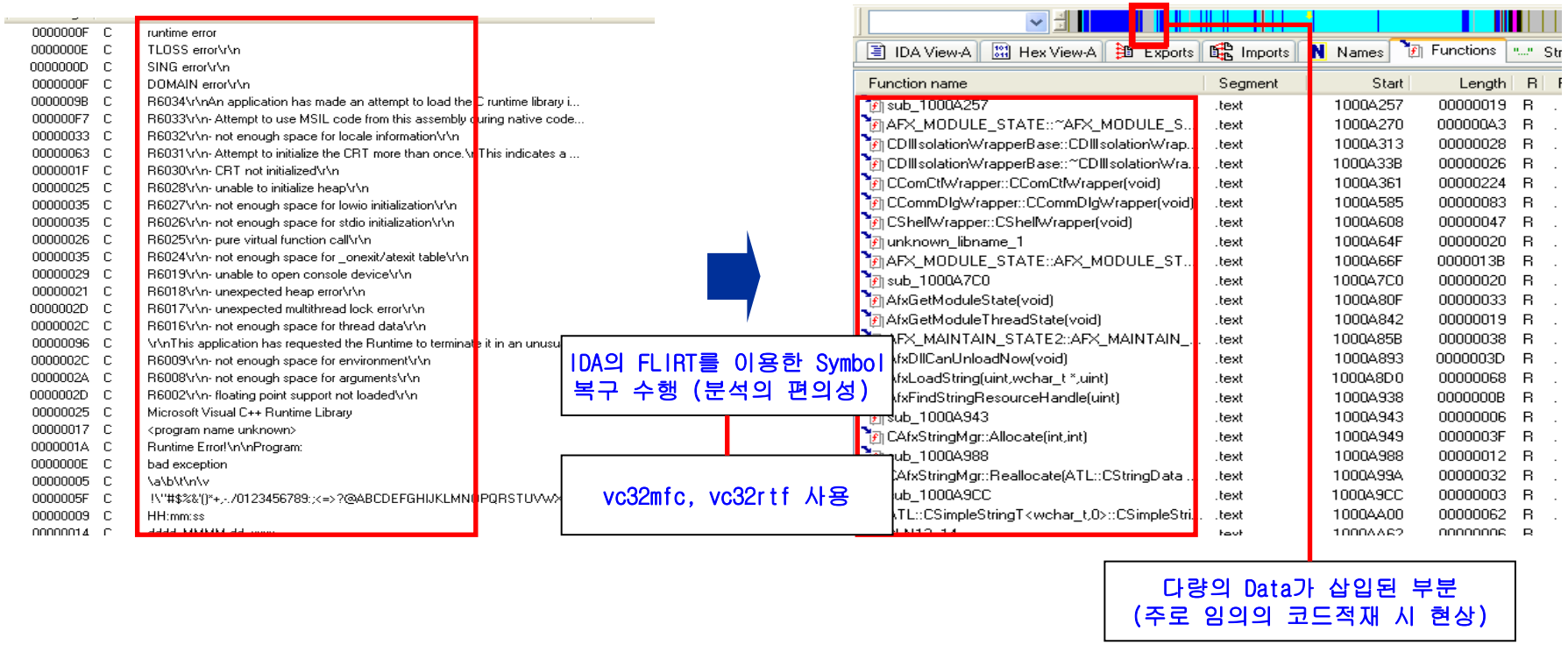
— 문제의 DLL을 Regsvr32를 이용하여 등록하면 Internet Explorer 실행 시 UgoBoy BHO가 Load되는 것을 확인할 수 있음. 그러나 System 內 특이한 변동사항은 없음





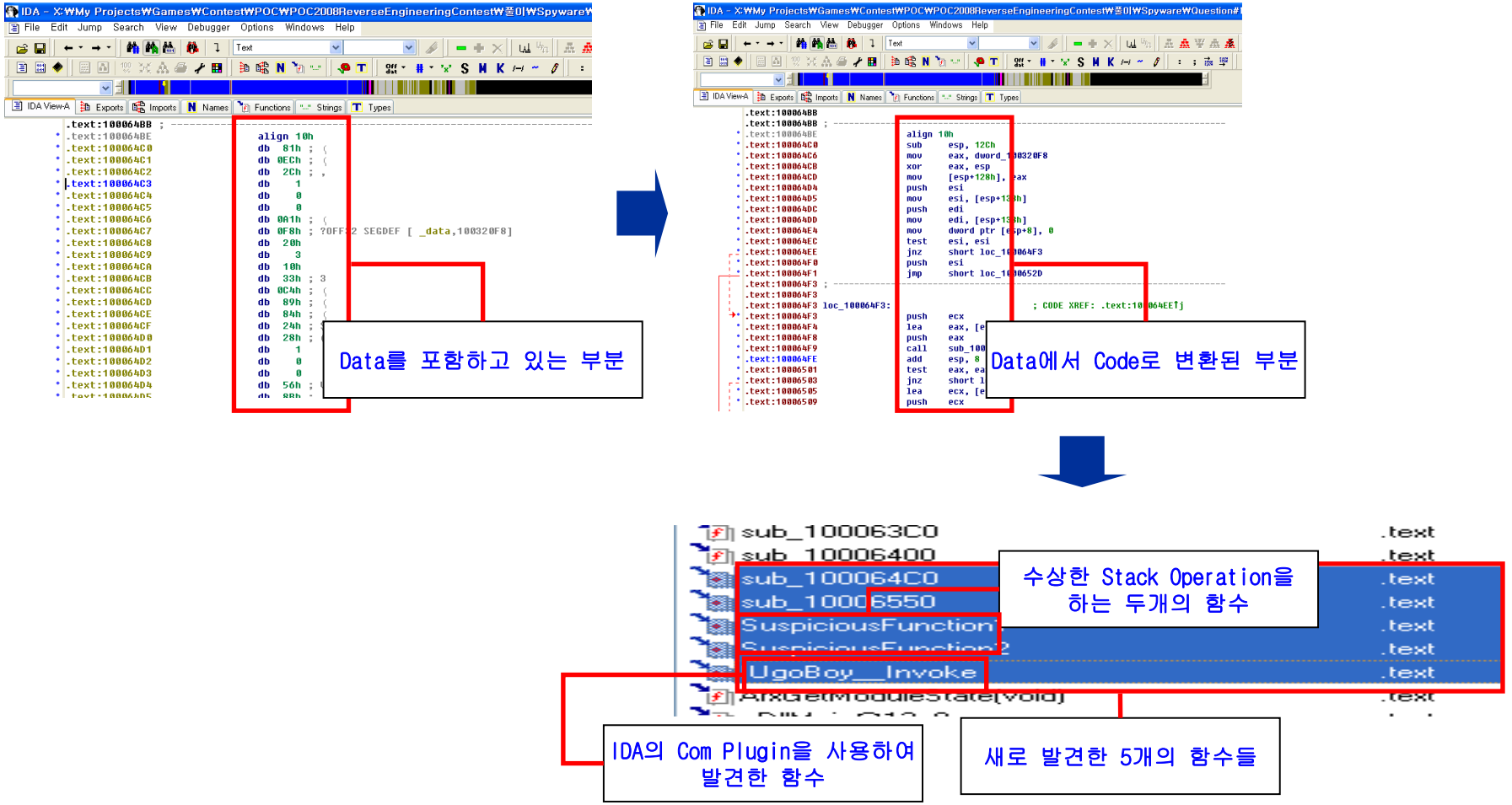
4. Spyware 문제 분석

- Spyware 1번 - “분석”
  - IDA Pro를 이용한 Sample.dll의 String 섹션 분석을 통하여 Statically Linked 된 Binary임을 확인할 수 있음. (경험적 분석)
  - 분석의 편의를 위하여 vc32mfc 및 vc32rtf를 사용하여 MFC 라이브러리의 Symbol 복구



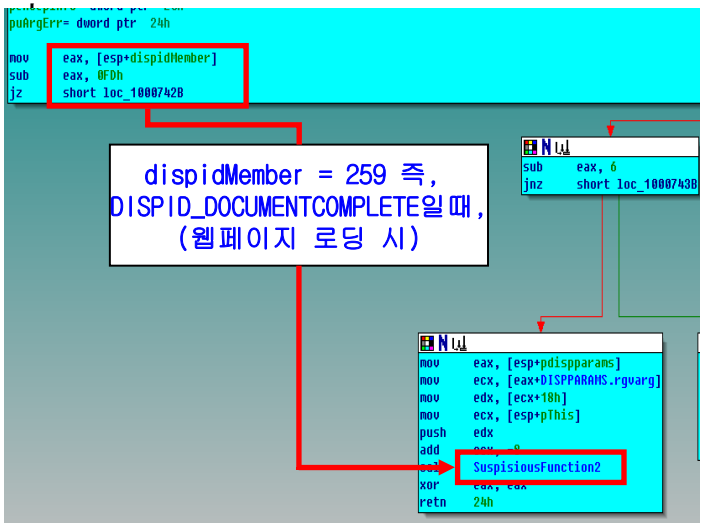
❑ Spyware 1번 - “분석”

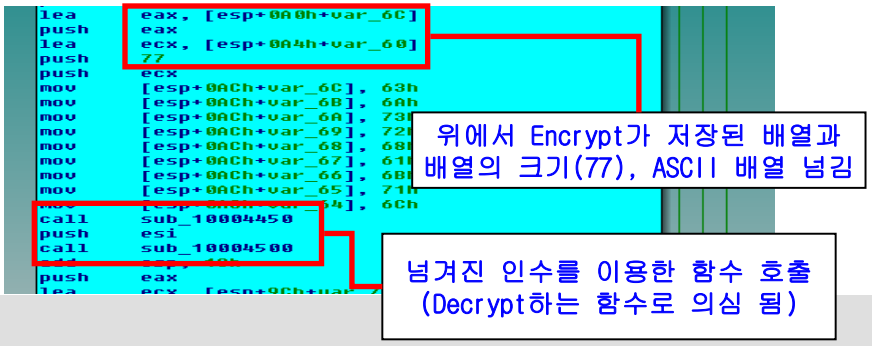
— Code 속에 다량의 데이터가 있는 부분을 검색하여 해당 부분이 임의의 Code가 아닌지 확인 수행 → Data부분을 Code로 변환해 보면, 대략 5개의 함수가 존재함을 알 수 있음.

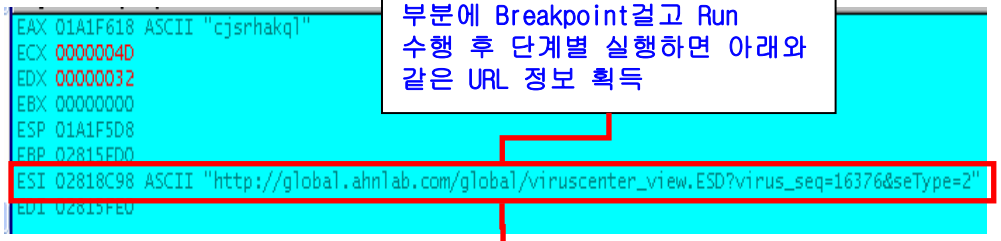


❑ Spyware 1번 - “분석”

— IUgoBoy_Invoke 함수를 분석하여, 웹페이지가 로딩되면 SuspiciousFunction2 함수가 호출됨을 알 수 있으며, Suspicious Function2에는 Encrypt된 code가 저장된 것으로 의심되는 부분을 찾을 수 있음. 이 후에 나오는 부분에서 Decrypt 함수로 의심되는 함수 발견 가

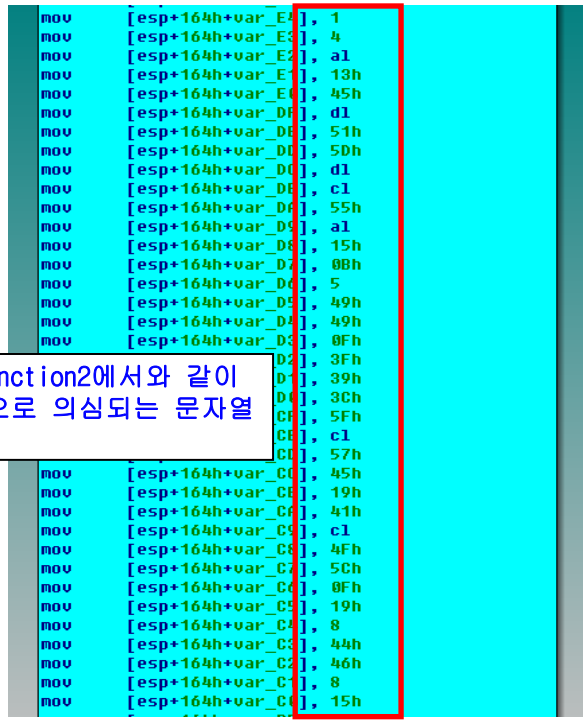






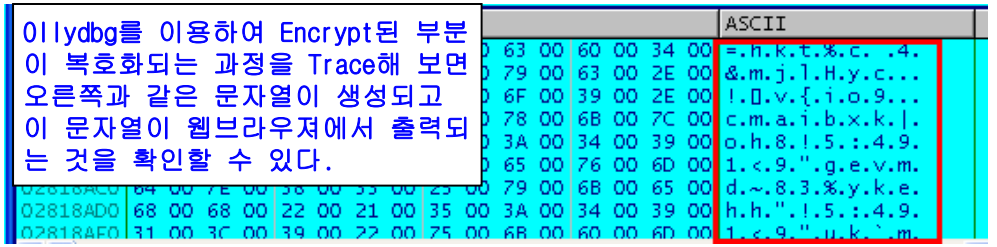
❑ Spyware 1번 - “분석”

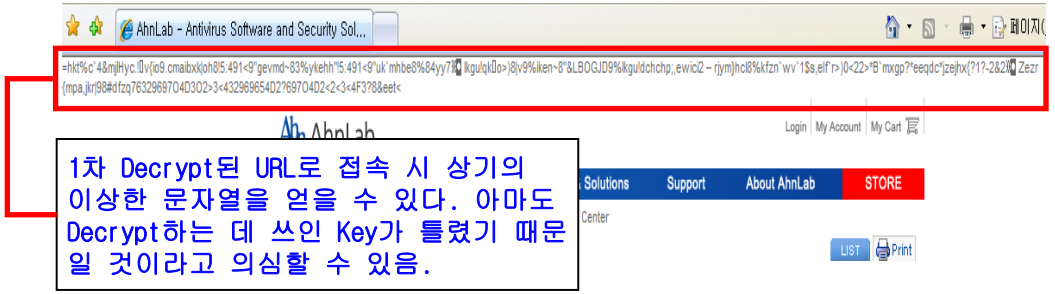
— Suspicious Function2 함수의 분석을 계속하다 보면 밑부분에서 Suspicious Funtion1의 호출부분을 찾을 수 있음 → Decrypt의 결과로 미루어 Suspicious Function1이 최종 Password를 Decrypt하는 함수임을 짐작할 수 있음.



Suspicious function2에서와 같이 Encrypt된 것으로 의심되는 문자열 발견

Ollydbg를 이용하여 Encrypt된 부분이 복호화되는 과정을 Trace해 보면 오른쪽과 같은 문자열이 생성되고 이 문자열이 웹브라우져에서 출력되는 것을 확인할 수 있다.





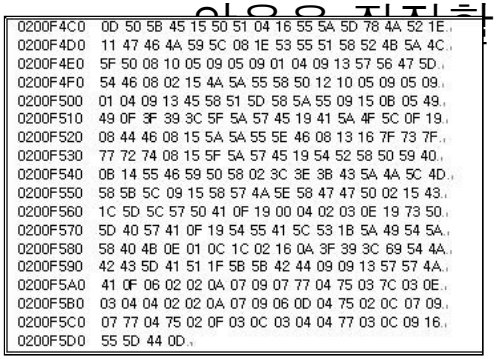
1차 Decrypt된 URL로 접속 시 상기의 이상한 문자열을 얻을 수 있다. 아마도 Decrypt하는 데 쓰인 Key가 틀렸기 때문일 것이라고 의심할 수 있음.

❑ Spyware 1번 - “분석”

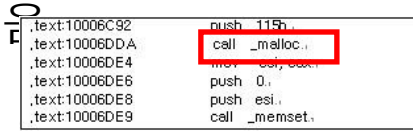
— Suspicious Function 1을 분석해 보면, 앞서 표시한 배열을 이용한 Data 초기화 후, ‘%d%H%M%s’ 형식의 날짜 문자열을 입력 받아 Decrypt 함을 알 수 있고, 최종적으로

XOR 연산을 수행하여 Password를 출력함을 알 수 있음.

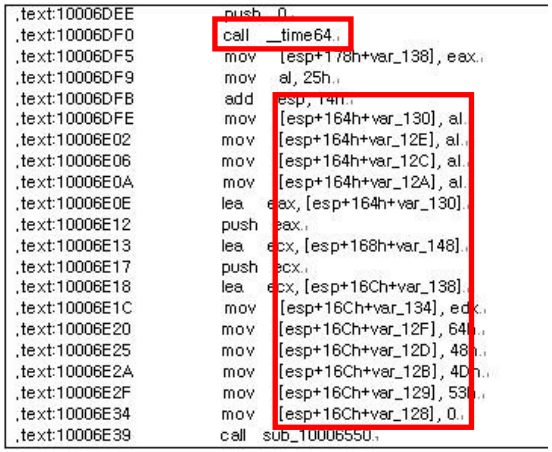
— 이 의미는 제대로 된 날짜가 입력되지 않으면 Decrypt가 제대로 수행되지 않음을 짐작할 수 있음



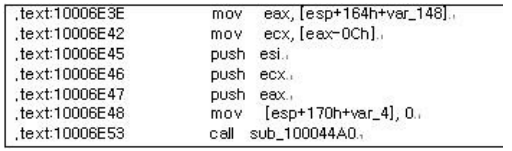
< Stack에 저장된 복호화 대상 >



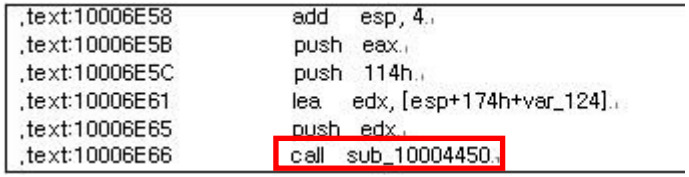
< 복호화 대상용 메모리 할당 >



< 날짜 시간형식의 문자열 변환 >



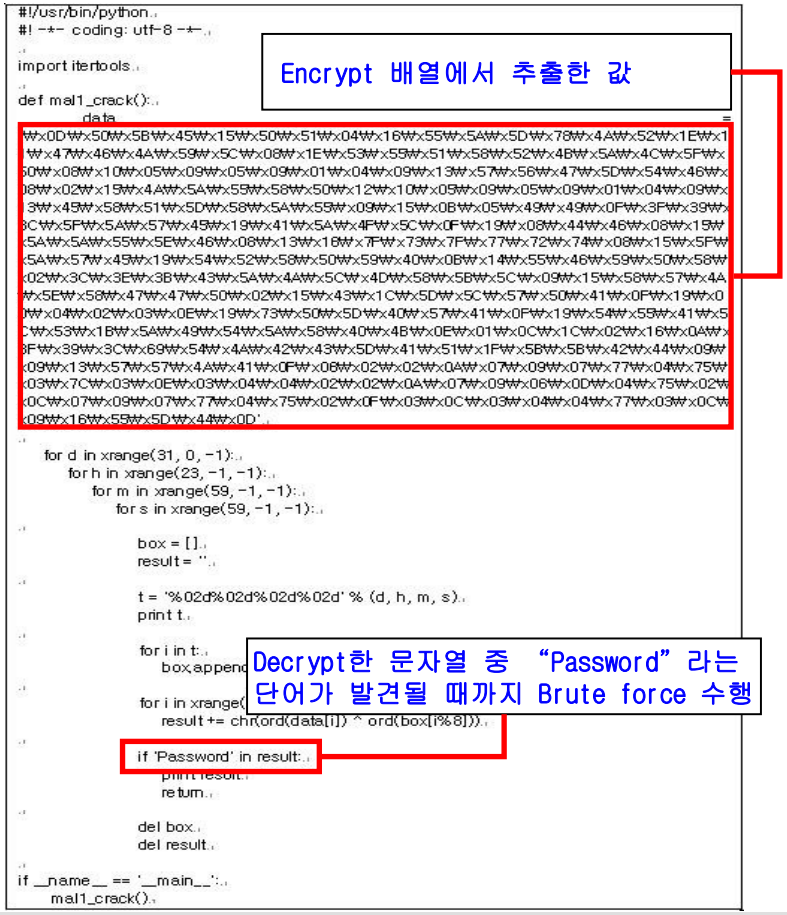
< 변환된 문자열을 ASC코드로 재변환 >

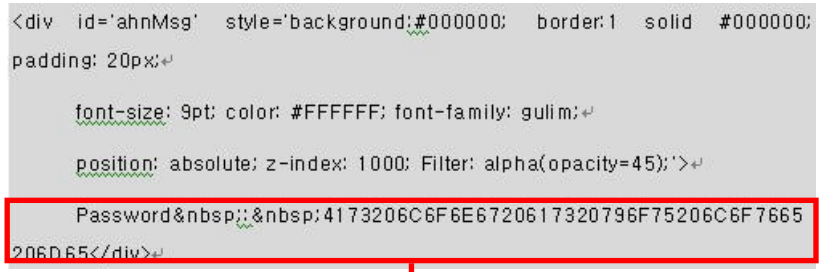


< 복호화 함수 호출 >

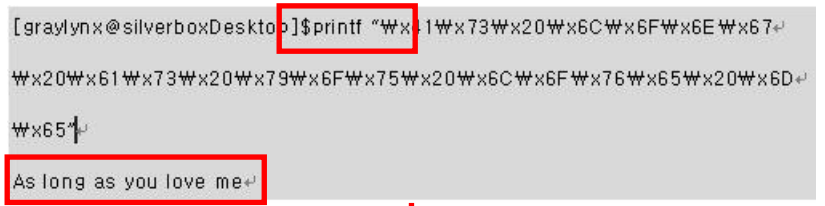
❑ Spyware 1번 - “분석”

— Password는 정확한 시간이 Key값으로 주어졌을 경우 입력되었을 경우 제대로 얻을 수 있으므로, Key값으로 주어진 정확한 시간을 얻기 위해 간단한 Python 프로그램을 이용하여 Brute force 공격을 수행.





Decrypt 성공 후 얻은 문자열, 그러나 해당 문자열 마저도 Hex encoding 된 것으로 추정 됨



Printf 명령을 이용하여 Hex code를 decoding하면 Password “As long as you love me” 를 얻을 수 있다

4. Spyware 문제 분석

❑ Spyware 1번 - “조치”

— 1. 모든 Internet Explorer를 종료

— 2. HKEY_CLASSES_ROOT\CLSID\{202D7790-9289-483E-8E4A-16BD52ADFC2 A}
\InprocServer32 키의 값에 나와있는 경로의 Sample.dll을 삭제.

— 3. HKEY_CLASSES_ROOT\CLSID\{202D7790-9289-483E-8E4A-16BD52ADFC2A}
키 값 삭제

— 4. HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion
\Explorer\Browser Helper Objects\{202D7790-9289-483E-8E4A-16BD52ADFC2A}
키 값 삭제

— 혹은, Internet Explorer → 도구 → ‘추가기능관리’로 이동하여 의심스러운 BHO는 “사용 안 함”에 Check.

❑ Spyware 1번 – “예방”

— 1. 검증되지 않은 Site에서 다운로드 받은 파일을 무턱대고 실행하지 않음.

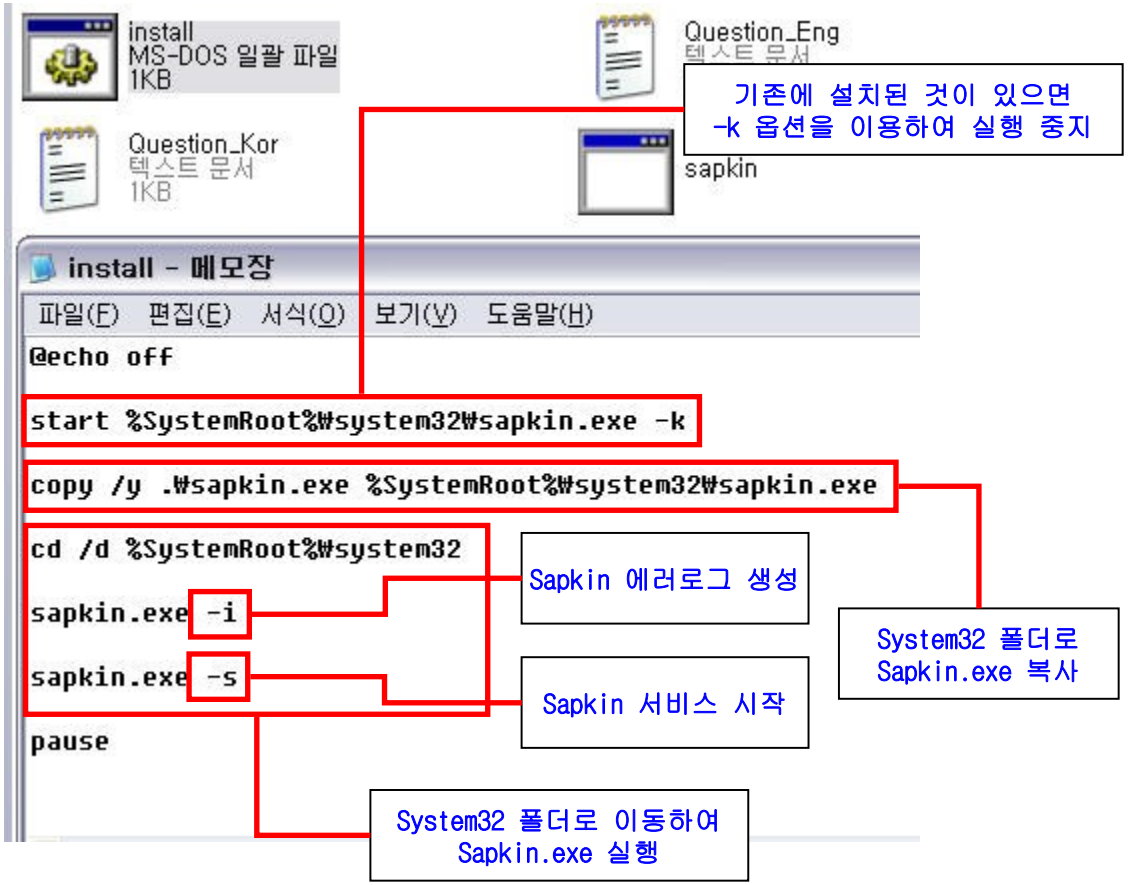
— 2. 검증되지 않은 Site에서 제공하는 ActiveX 등을 무심코 설치하지 않는다.

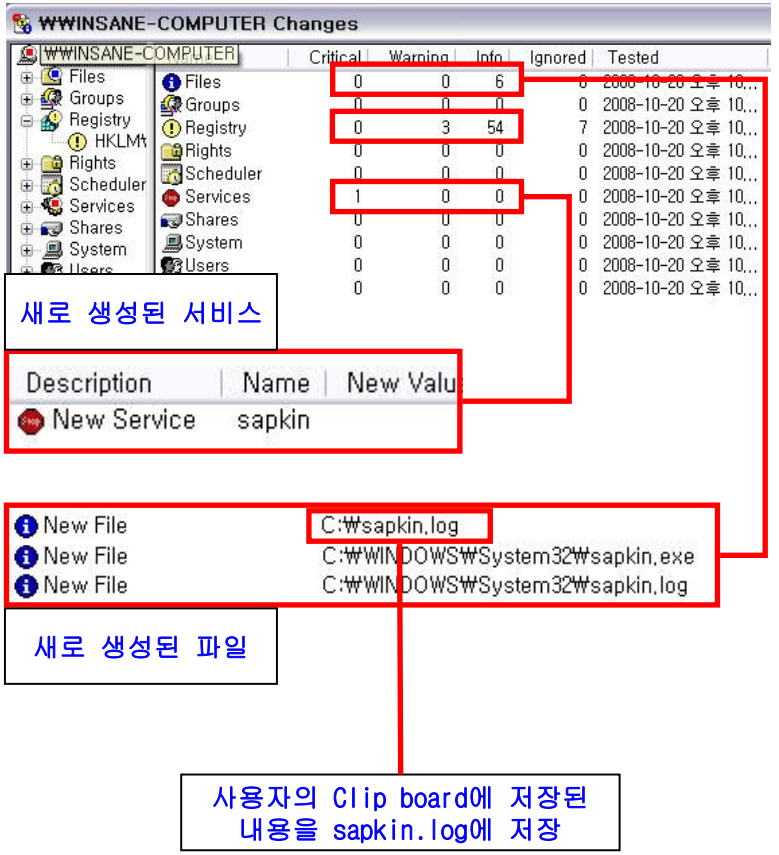
— 3. 백신 S/W는 항상 최신버젼으로 유지하고, 전용 Checker 프로그램이나 Process Explorer 혹은 Internet Explorer의 ‘추가기능관리’를 통하여 의심스러운 BHO가 설치되지 않았는지 수시로 확인한다.

❑ Spyware 2번 - “진단”

— 악성코드를 초기 설치하는 install.bat 분석

— Winanalysis를 이용하여 System 內 변경 점 분석 (클립보드의 내용을 로깅하는 Key logger)

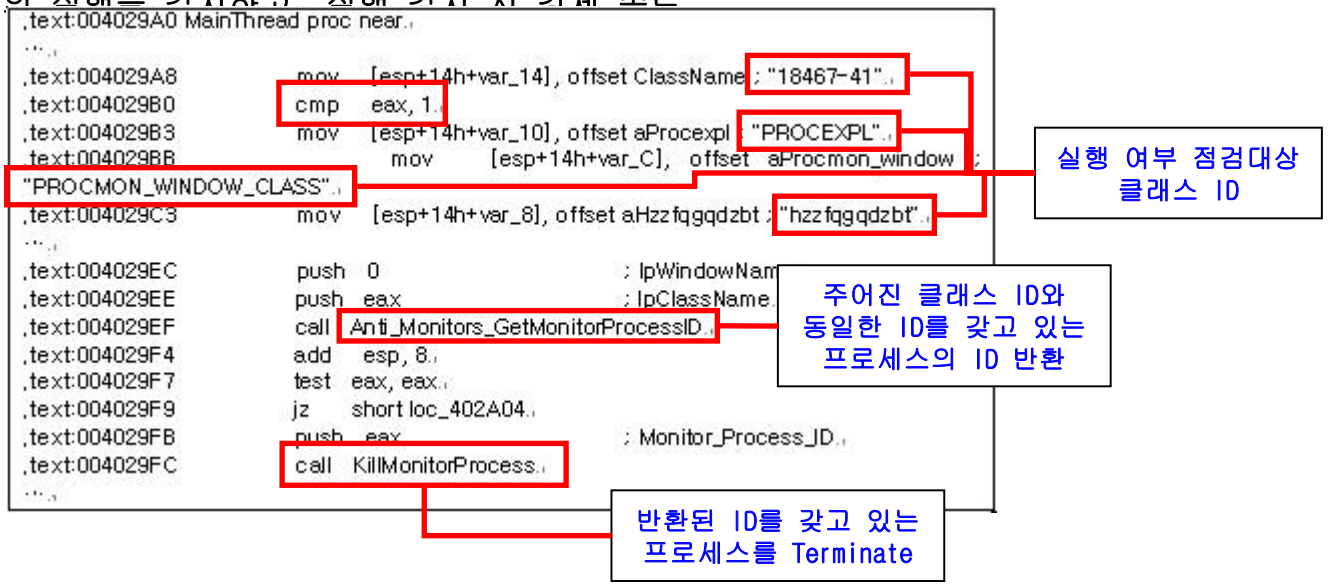




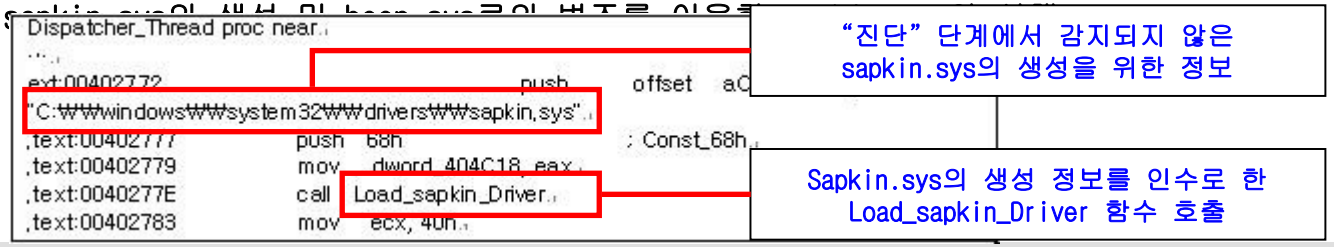
❑ Spyware 2번 - “분석”

— IDA Pro를 이용하여 주요 동작 방식, Anti-debugging 기법 분석

- Regmon (18467-41), Process explorer(PROCEXPL), Process Monitor (PROCMON_WINDOW_CLASS), ?? (hzzfqgqdzbt)



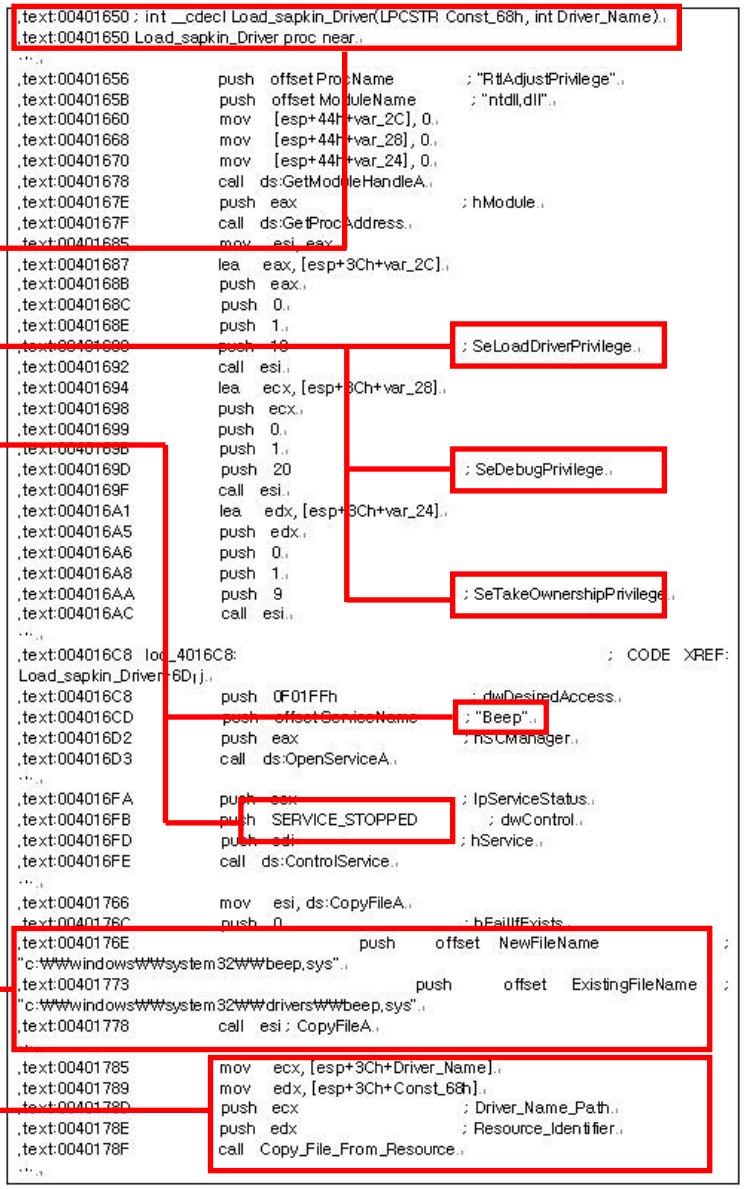
-



❑ Spyware 2번 - “분석”

— Load_sapkin_Driver 함수를 통한 Sapkin.sys실행(1)

- Load_sapkin_Driver 함수의 시작 부분
- Driver Load를 위한 권한 설정
- Beep Service Stop
- c:₩₩windows₩₩system32₩₩drivers₩₩beep.sys 파일을 c:₩₩windows₩₩system32₩₩beep.sys로 복사
- 68h와 ‘C:₩₩windo ws₩₩system32₩₩drivers₩₩sapkin.sys’ 문자열을 인자로 넘겨주면서 Copy_File_From_Resource 함수를 호출



❑ Spyware 2번 - “분석”

— Load_sapkin_Driver 함수를 통한 Sapkin.sys실행(2)

- Copy_File_From_Resource 함수의 시작 부분
- 지정된 ID의 Resource를 검색
- 검색된 Resource를 Load
- c:₩₩windows₩₩system32₩₩drivers₩₩sapkin.sys 파일로, Load된 Resource를 Dump하여 sapkin.sys 파일 생성

```
.text:00401540 Copy_File_From_Resource proc
.text:0040155B                 push  0                 ; lpModuleName
.text:0040155D                 call  ds:GetModuleHandleA
.text:00401563                 mov   edi, eax
.text:00401565                 mov   eax, [esp+28h+Resource_Identifier]
.text:00401569                 push  offset Type       ; "BIN"
.text:0040156E                 push  eax               ; lpName
.text:0040156F                 push  edi               ; hModule
.text:00401570                 call  ds:FindResourceA
.text:00401576                 mov   esi, eax
.text:0040158E                 call  ?AfxGetModuleState@@YGPAVAFX_MODULE_STATE@@XZ ; AfxGetModuleState(void)
.text:00401593                 mov   eax, [eax+0Ch]
.text:00401596                 push  esi               ; hResInfo
.text:00401597                 push  eax               ; hModule
.text:00401598                 call  ds:LoadResource
.text:0040159E                 test  eax, eax
.text:004015D3                 push  esi               ; hResInfo
.text:004015D4                 push  edi               ; hModule
.text:004015D5                 call  ds:SizeofResource
.text:004015DB                 lea   ecx, [esp+28h+var_1C]
.text:004015DF                 mov   esi, eax
.text:004015E1                 call  ??0CFile@@QAE@XZ  ; CFile::CFile(void)
.text:004015E6                 mov   ecx, [esp+28h+Driver_Name_Path]
.text:004015EA                 push  0
.text:004015EC                 push  9001h
.text:004015F1                 push  ecx
.text:004015F2                 lea   ecx, [esp+34h+var_1C]
.text:004015F6                 mov   [esp+34h+var_4], 0
.text:004015FE                 call  ?Open@CFile@@UAEHPBDIPAVCFileException@@@Z ; CFile::Open(char const *,uint,CFileException *)
```

생성된 sapkin.sys를 beep.sys로 덮어 씌워 복사 함 (beep.sys변조)

변조가 완료되었으므로 흔적을 남기지 않기 위해 sapkin.sys 삭제

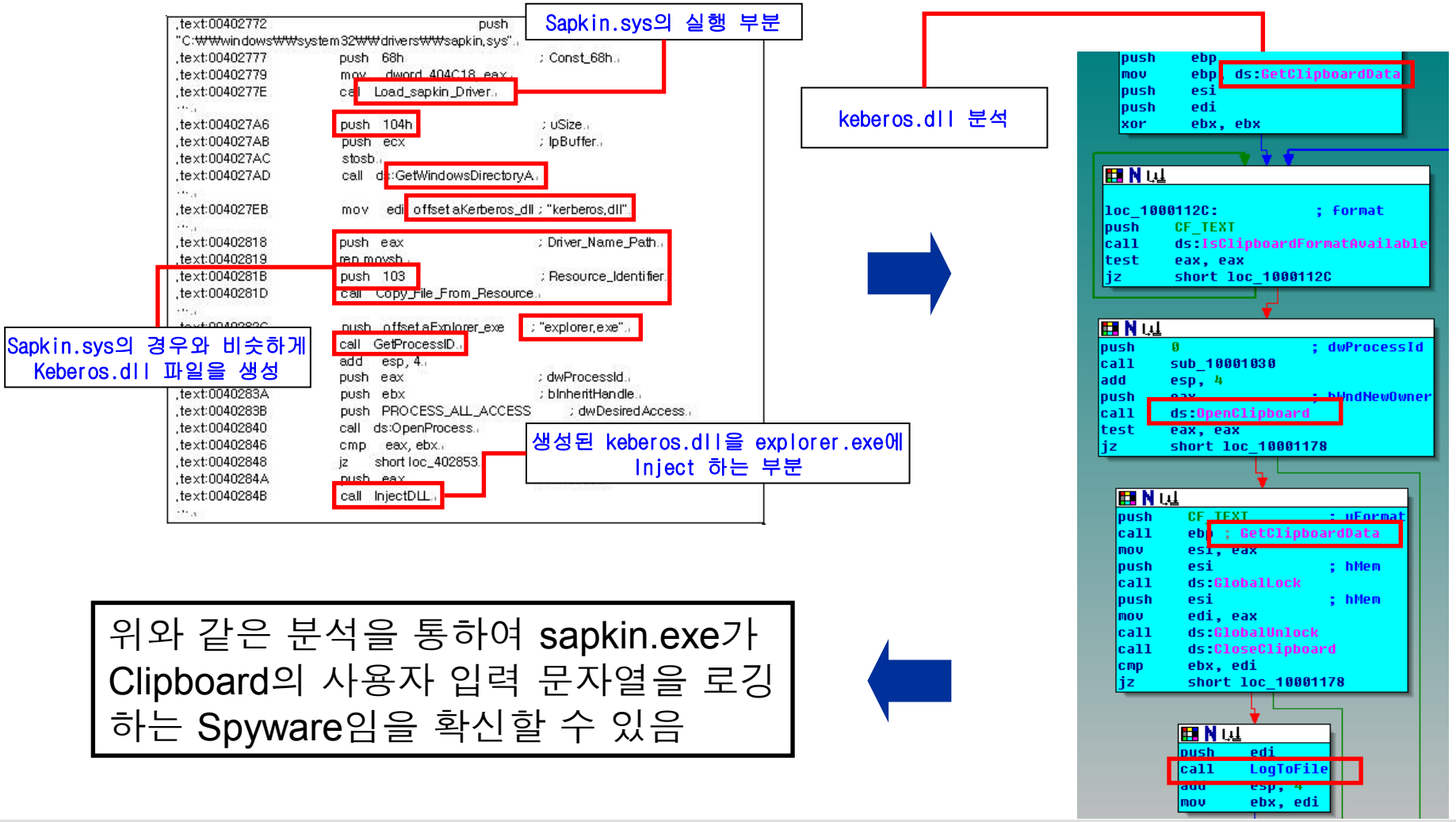
변조된 beep.sys로 beep서비스를 재시작하고 백업했던 beep.sys도 변조된 beep.sys로 교체

```
.text:0040178D                 push  ecx               ; Driver_Name_Path
                                                       ; Resource_Identifier
                                                       ; rom_Resource
.text:004017A2                 push  0                 ; bFailIfExists
.text:004017A4                 push  offset ExistingFileName ;
"c:₩₩windows₩₩system32₩₩drivers₩₩beep.sys"
.text:004017A9                 push  offset aCWindowsSystem ;
"C:₩₩windows₩₩system32₩₩drivers₩₩sapkin.sys"
                               esi ; CopyFileA
                               push  offset aCWindowsSystem ;
                               drivers₩₩sapkin.sys"
.text:004017C0                 call  ds:DeleteFileA
                               push  0                 ; lpServiceArgVectors
.text:004017C8                 push  0                 ; dwNumServiceArgs
                               edi                     ; hService
                               ds:StartServiceA
                               MOVEFILE_REPLACE_EXISTING ; dwFlags
                               push  offset ExistingFileName ;
"c:₩₩windows₩₩system32₩₩drivers₩₩beep.sys"
.text:004017EA                 push  offset NewFileName ;
"c:₩₩windows₩₩system32₩₩beep.sys"
.text:004017EF                 call  ds:MoveFileExA
```

이 후부터 beep.sys 서비스를 이용하여 sapkin.sys가 항시 동작하게 됨

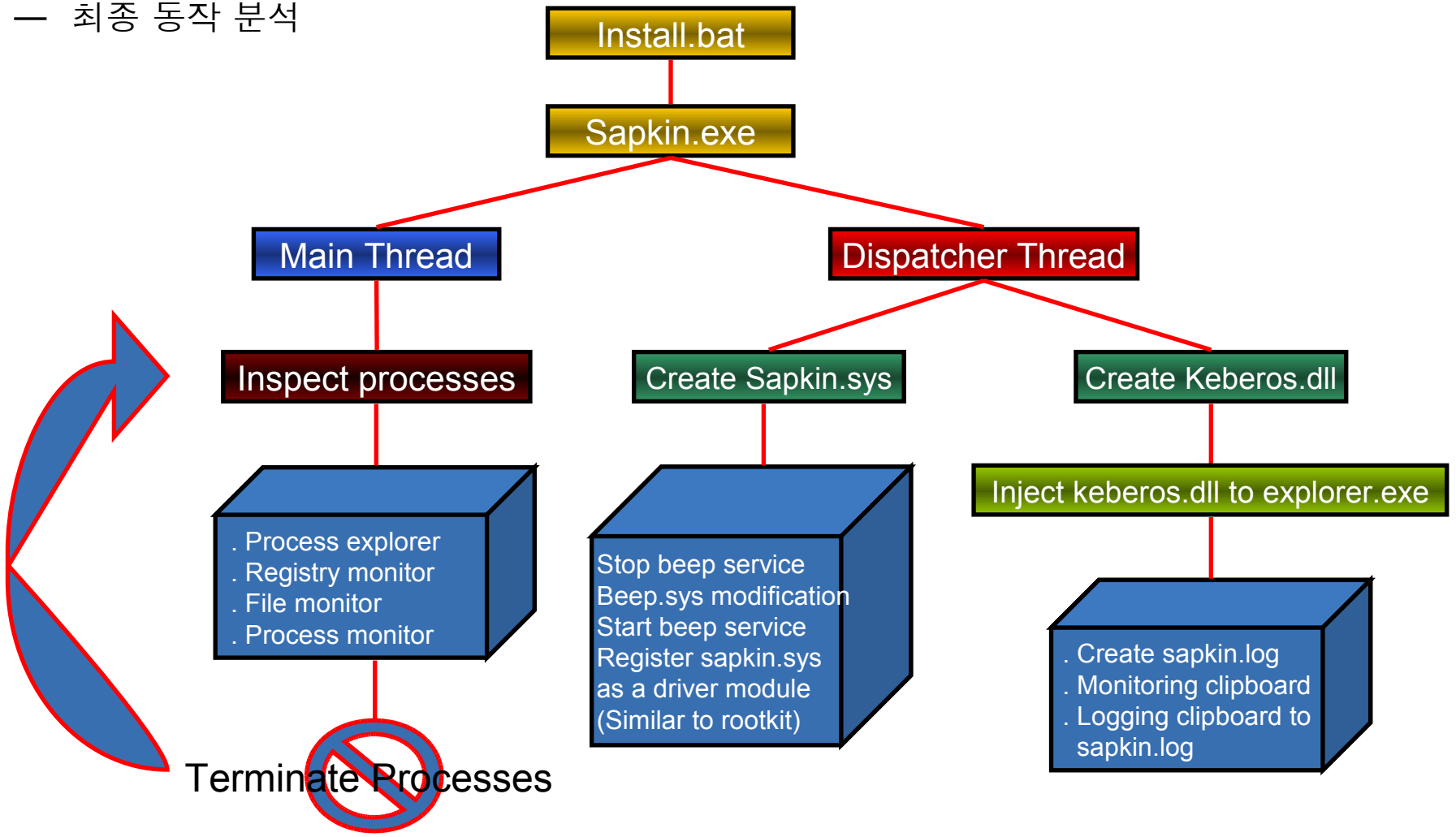
❑ Spyware 2번 - “분석”

— Sapkin.sys 실행 후의 keberos.dll 생성 및 explorer.exe에 Injection



위와 같은 분석을 통하여 sapkin.exe가 Clipboard의 사용자 입력 문자열을 로깅하는 Spyware임을 확신할 수 있음

- Spyware 2번 - “분석”
  - 최종 동작 분석



4. Spyware 문제 분석

❑ Spyware 2번 - “조치”

- 1. Stop sapkin service
- 2. Delete c:\windows\system32\sapkin.exe
- 3. Stop beep service (To unload sapkin.sys from memory)
- 4. Delete c:\windows\sapkin.log
- 5. Delete c:\windows\keberos.dll
- 6. Reboot system

❑ Spyware 2번 – “예방”

- 1. 검증되지 않은 Site에서 다운로드 받은 파일을 무턱대고 실행하지 않는다.
- 2. PC방과 같이 안정성이 검증되지 않은 장소에서는 민감한 정보를 다루는 업무를 수행하지 않는다.
- 3. 백신 S/W는 항상 최신버젼으로 유지한다.

감사합니다.

기타문의 사항은 20silvers@naver.com으로...