기술문서	'09. 11. 05. 작성

Binary Diffing

작성자 : 영남대학교 @Xpert 곽철홍 kchcjcj@yu.ac.kr

목 차

0. 소개

binary diffing이라는 용어는 어떻게 보면 생소할 수 있으나, 리버스 엔지니어링처럼 파일의 구조를 분석하는 학문을 배우는 입장에서는 당연히 알게 되는 개념이다. 이 문서를 참고함으로써 그 개념이 이러한 용어를 가지고 있다는 것을 알게 되는 사람도 있을 것이라 생각했고, 갓 파일분석에 입문한 사람도 참고하면 도움이 될 만한 자료를 만들고 싶어서 문서로 남기게 되었다. 컴퓨터에 대해서 잘 모르는 사람도 단순한 호기심에 이 비교방법을 접할 수 있으며 이해하는 것도 어렵지 않다. 적극적으로 툴을 구해서 사용해보고 이해하는 것이 큰 도움이 될 것이라 생각한다.

1. 정 의

가. 정 의

뜻을 간략히 풀어보면 이진파일의 차이를 의미한다. binary diffing은 크게 이진파일들의 차이를 찾는 작업을 의미한다. 또, binary differ는 이진파일들의 차이를 찾아주는 툴을 의미한다.

두 객체의 차이를 찾는 것은 사람에게도 익숙한 일이다. 컴퓨터로는 더 쉽게 이 작업을 행할 수 있다. 이 작업을 통하여 소프트웨어의 패치전과 후의 차이를 직접 알아볼 수 있고, 어떤 상황으로 인해 파일이 변경되었는지도 확인할 수 있다. 또한 변경되는 부분을 조사하면 그 부분이 어떤 역할을 하는지도 알 수 있기 때문에 임의로 조작도 가능하다.

두 파일의 차이를 찾는 알고리즘은 많이 알려져 있다. UNIX에는 UNIX시스템이 만들어질 때부터 diff라는 파일비교 유틸리티가 있었다.(오래전부터 있었으므로 기능이 추가되고 보완되어서 기능이 아주 많다.) DOS기본 명령어 중에도 fc(file compare)라는 명령어가 있다.(UNIX에서의 명령은 cmp이다.)

나. diff의 간단한 예제분석

유닉스시스템에 기본으로 있는 간단한 툴인 diff를 이용하면 단순한 파일의 차이를 알 수 있다. 특히 텍스트파일비교기능이 강력하여서 지금까지도 패치내역으로 diff의 출력파일을 내놓는다. 텍스트환경의 인터페이스에서도 알아보기 쉽도록 구성되어 있다.

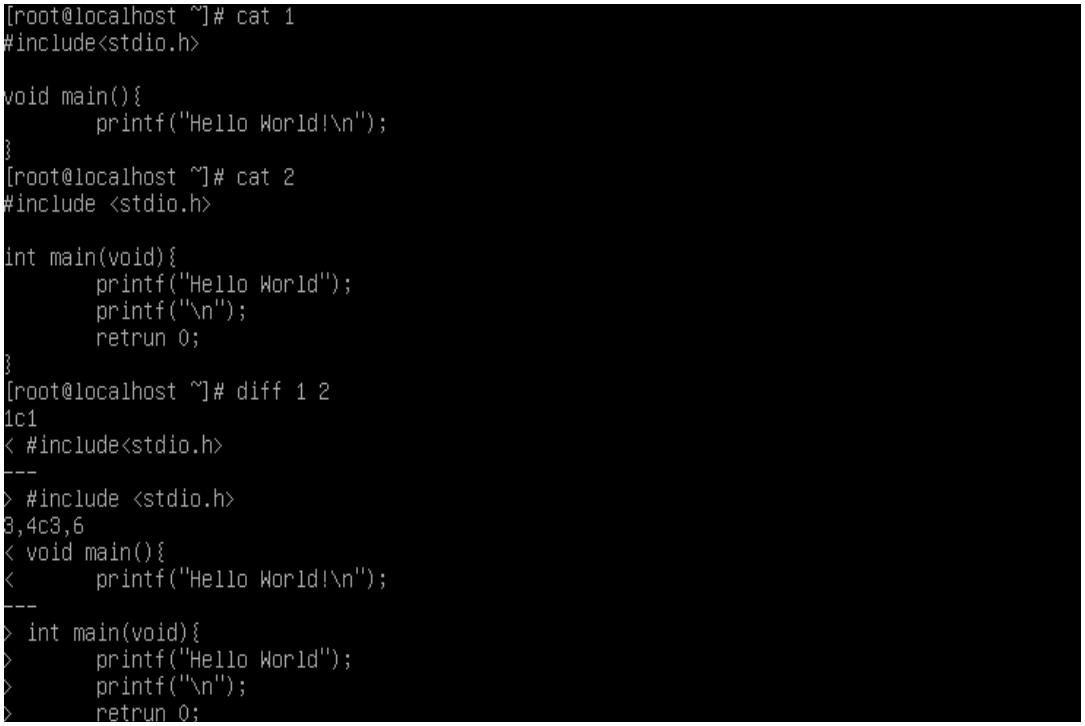

그림 1 diff의 간단한 사용예제

설명이 없어도 이해하기 쉽도록 구성되어 있지만 간략히 설명을 달아보자면, 1 이라는 파일과 2라는 파일을 비교하는 과정인데 소스의 첫 번째 문자가 <, >로 되어있는 것이 보인다. <기호는 첫 번째 인자로 준 1 이라는 파일의 코드를 나타내고 >는 2의 코드를 나타낸다. ---으로 텍스트를 끊어주며 아래위로 비교해보면 된다. 1c1이라고 표시되어 있는 것은 c를 사이에 두고 첫 번째 인자의 첫 번째 줄과 두 번째 인자의 두 번째 줄을 비교해보라는 것이며, 아래의 3,4c3,6은 첫 번째 인자의 3~4줄과 두 번째 인자의 3~6줄이 다르므로 비교해보라는 것이다.

단순히 텍스트파일을 비교하는 것 외에도 다른 기능들도 있으니 더 자세한 것은 man페이지를 확인해보자.

다. diff의 사용이유

비교를 하려면 일단 비교할 2개의 파일이 필요하다. 나중에 어떤 프로그램이 수정되었을 경우 원본과 패치후의 소스를 비교하자면 용량이나 가독성면에서 효율이 떨어진다. 위 예제에 있듯이 diff는 소스의 바뀐 부분만 따로 출력해주므로 원본과 바뀐 소스만 가지고도 패치된 내용을 알 수 있다. 개발자들은 패치와 diff의 출력파일만 내놓으면 패치내역을 손쉽게 공개하는 셈이 된다. 또한 바뀐 부분만 출력하기 때문에 특정루틴이 어느 부분에서 동작하는지도 알 수 있다. 분석하는 입장에서는 하나하나씩 분석해 볼 필요가 없다.

그림 2 http://wiki.kldp.org/wiki.php/DiffAndPatch 에서는 오픈소스에 참여하려는 사람들을 위해 diff와 patch의 개념과 사용방법을 소개하고 있다.

2. 툴 사용예제

지금은 fc(cmp), diff보다 더 강력한 기능을 가진 툴이 많다. 자신이 툴을 가지고 있지 않을 경우 시스템마다 위와 같은 differ는 기본으로 존재하므로 언제든지 비교는 가능하다. 하지만 가능하다면 손쉽고 강력한 툴을 쓰는 것이 효율적일 것이다.

가. 헥스에디터

바이너리파일에 직접 접근할 수 있는 툴을 의미한다. 디버깅 툴이나 텍스트 편집 툴과 같이 쓰면 유용하다. 툴마다 다르겠지만 대부분은 코드를 ASCII나 다른 진수로 변환, 값의 수정을 지원한다. 그러나 대개 파일의 형식까지는 분석하지 못하므로 어떤 값을 변경하면 원하는 결과를 얻을 수 있는지 생각하는 것은 사용자가 해야 할 일이다.

이름	크기	종류	수정한 날짜
MSAVE1.R3S	9KB	R3S 파일	2009-11-05 오후 ...
ESAVE1.R3S	2KB	R3S 파일	2009-11-05 오후 ...
HERO	1KB	MS-DOS 프로그...	1999-06-21 오후 ...
PLAY.BAT	1KB	MS-DOS 일괄 파일	1995-11-17 오후 ...
MSAVE0.R3S	9KB	R3S 파일	1995-10-17 오전 ...
ESAVE0.R3S	2KB	R3S 파일	1995-10-17 오전 ...
MSAVE3.R3S	9KB	R3S 파일	1995-10-05 오후 ...
ESAVE3.R3S	2KB	R3S 파일	1995-10-05 오후 ...
SMARTCHK.CPS	1KB	CPS 파일	1995-09-30 오후 ...
MSAVE4.R3S	9KB	R3S 파일	1995-09-24 오후 ...
ESAVE4.R3S	2KB	R3S 파일	1995-09-24 오후 ...
MAIN.EXE	249KB	응용 프로그램	1995-09-18 오후 ...
OPGRP.R3	721KB	R3 파일	1995-08-30 오전 ...

```
C:₩WINDOWS₩system32₩cmd.exe
C:₩HERO>fc /b MSAVE0.R3S MSAVE1.R3S
파일을 비교합니다: MSAVE0.R3S - MSAVE1.R3S
00000000: 95 09
00000001: A0 B0
00000002: 17 05
00000003: 03 19
00000004: 20 06
00000023: C2 90
0000003E: 00 01
0000029F: 0F 02
000002AB: 14 11
000002AC: 07 0A
000002B2: 01 0E
000002B5: 01 00
000002B6: 00 01
000002D3: 0F 02
000002E6: FE 0C
000002ED: 0F 02
000002F9: 0B 0F
00000300: FE 04
00000307: 0F 02
0000031A: 0A 04
00000321: 0F 02
00000334: 04 0A
0000033B: 00 02
```

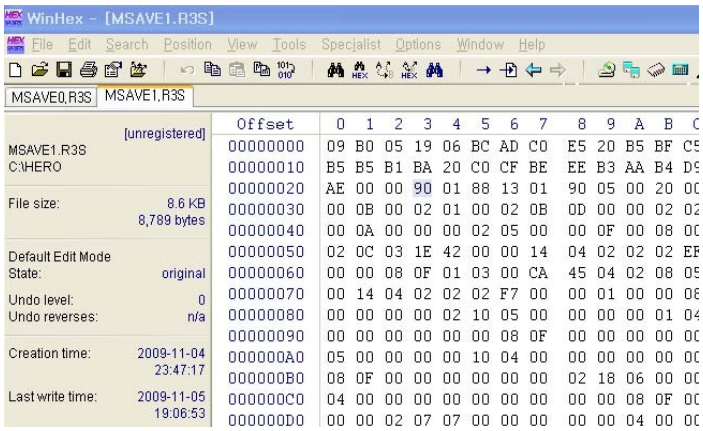

그림 3 헥스에디터로 간단한 게임의 세이브파일에 접근해 보았다. 파일 0번과 1번은 돈이 다르도록 저장하고 파일들의 수정날짜를 확인하여서 값이 저장되는 파일을 알아내었다. fc /b로 간단한 바이너리비교를 하고 돈이 변경된 위치를 확인하였다. 헥스에디터로 그 값을 바꾸니 게임에서도 적용되었다. 옛날게임이라서 가능하지만 요즘의 게임은 훨씬 복잡한 과정을 거쳐야 할 것이다.

나. 파일시스템모니터링

앞서 살펴보았던 것은 개별파일이 변했는지는 알 수 있지만 파티션, 디렉터리의 전반적인 변화는 감지하기 힘들다. 파일들의 전반적인 변화를 감지하는 툴이 이 부류에 속한다.

게임의 세이브파일이 2개가 생겨서 관리된다고 하면 어떤 파일들이 변경되는지 알아야 할 것이다. 파일들의 변화를 감지하는 방법을 알아보면 다음과 같다.

1) 수동비교

파일을 변경하기 전 백업을 해두고 어떤 작업을 한 뒤 복사본과 원본을 비교해보는 방법이다. 시간이 많이 들고 비효율적이기는 하지만 바뀐 부분을 확실하게 알 수 있다. 윈도우의 regedit툴을 이용하여 레지스트리를 텍스트파일로 백업해두는 방법은 레지스트리모니터링 프로그램이 없을 경우 유용하다.

2) 파일속성 비교

내용은 생략하고 파일의 쓰기시간, 크기등 속성만 비교하는 방법이 있다. 작업 전의 디렉터리내용의 속성을 텍스트파일로 저장해두고 작업 후에 비교하여 속성이 변한 파일을 분석하는 방법이다. 리눅스에서는 ls -alt(최근순서정렬)를 텍스트파일로 저장하고 윈도우에서는 dir /o:d(오래된 순서정렬)로 저장해둔다.

3) Archive속성

```
C:₩Documents and Settings₩koopa₩바탕 화면₩big>attrib
A           C:₩Documents and Settings₩koopa₩바탕 화면₩big₩big.c
A           C:₩Documents and Settings₩koopa₩바탕 화면₩big₩big.dsp
A           C:₩Documents and Settings₩koopa₩바탕 화면₩big₩big.dsw
A           C:₩Documents and Settings₩koopa₩바탕 화면₩big₩big.ncb
A           C:₩Documents and Settings₩koopa₩바탕 화면₩big₩big.opt
A           C:₩Documents and Settings₩koopa₩바탕 화면₩big₩big.plg

C:₩Documents and Settings₩koopa₩바탕 화면₩big>attrib -a *.*

C:₩Documents and Settings₩koopa₩바탕 화면₩big>attrib
            C:₩Documents and Settings₩koopa₩바탕 화면₩big₩big.c
            C:₩Documents and Settings₩koopa₩바탕 화면₩big₩big.dsp
            C:₩Documents and Settings₩koopa₩바탕 화면₩big₩big.dsw
            C:₩Documents and Settings₩koopa₩바탕 화면₩big₩big.ncb
            C:₩Documents and Settings₩koopa₩바탕 화면₩big₩big.opt
            C:₩Documents and Settings₩koopa₩바탕 화면₩big₩big.plg

C:₩Documents and Settings₩koopa₩바탕 화면₩big>
```

그림 4 attrib는 파일의 속성을 표시한다. -a옵션으로 archive를 업데이트할 수 있다.

FAT파일시스템은 archive비트라는 것이 있다. 마지막 백업이후로 파일이 변경되었는지 알아보기 위한 비트이다. 백업작업을 하고 파일의 속성이 변경될 경우 비트가 활성화되므로 파일속성비교에 비해 간편하게 변경여부를 알 수 있다.

4) 체크섬과 해시값검사

보통은 불법적인 침입이 있었는지 알아보는 방법으로 무결성검사를 한다. 하지만 단순히 파일을 비교하는데도 이용할 수 있다. 이 방법은 각각의 파일별로 해시함수로 해시값을 만들어서 따로 텍스트파일로 테이블을 만들고 파일이 변경되면 그 값을 비교해 보는 방법이다. 무엇이 바뀌었는지 자세히 알기가 어렵다는 단점이 있다.

```
[root@localhost tmp]# ls -al > before
[root@localhost tmp]# ./program
[root@localhost tmp]# ls -al > after
[root@localhost tmp]# diff before after
1c1
< 합계 32
---
> 합계 40
6,7c6,8
< -rw-r--r-- 1 root root    0 11월  5 02:36 3
< -rw-r--r-- 1 root root    0 11월  5 02:37 before
---
> -rw-r--r-- 1 root root   31 11월  5 02:37 3
> -rw-r--r-- 1 root root    0 11월  5 02:37 after
> -rw-r--r-- 1 root root  390 11월  5 02:37 before
[root@localhost tmp]# md5sum 1 >> table
[root@localhost tmp]# md5sum 2 >> table
[root@localhost tmp]# md5sum 3 >> table
[root@localhost tmp]# ./program
[root@localhost tmp]# md5sum 1 >> aftertable
[root@localhost tmp]# md5sum 2 >> aftertable
[root@localhost tmp]# md5sum 3 >> aftertable
[root@localhost tmp]# diff table aftertable
3c3
< cc985ae43e6abeb60b74f87cabb0340e  3
---
> fb98056e373a5a6956c12618344654e8  3
[root@localhost tmp]#
```

그림 5 리눅스에서의 파일속성비교와 해시값 검사를 나타내고 있다. program이라는 파일을 실행하면 3이라는 파일에 내용이 추가된다.

다. 다른 툴

두 파일을 비교하여서 자신이 값을 변경하고자하는 부분을 알아낸 다음 그 값을 변경하는 것이 크래커의 목적이다. 이 비교방법을 자동으로 해주고 값 변경을 쉽게 할 수 있도록 만들어진 툴이 몇 가지 존재한다. 그중에서도 파일에 저장되어 있는 값이 아니라 실시간으로 프로세스에 올라가 있는 값을 찾아서 변경하는 유명한 툴을 한번 써 보았다.

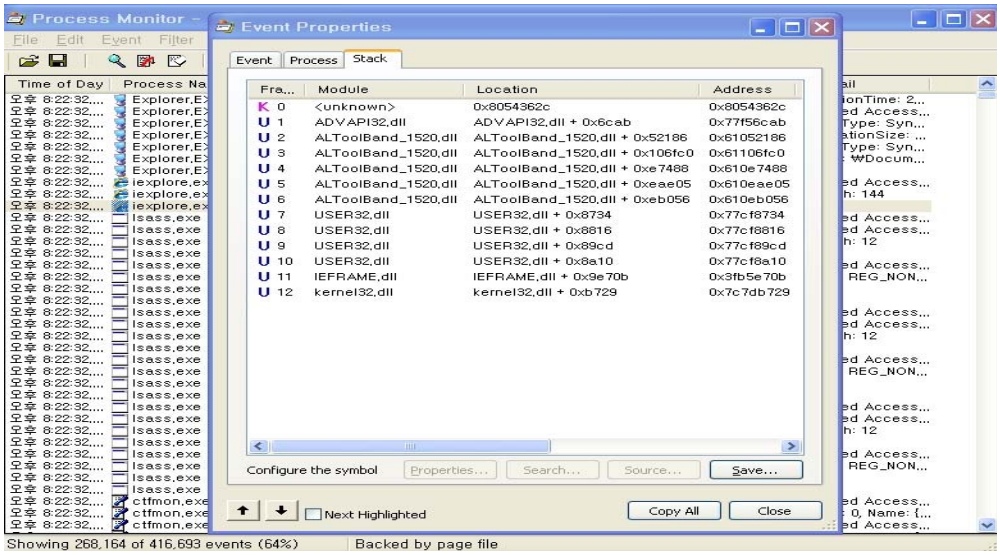

그림 6 실시간 프로세스 모니터링 툴인 Process Monitor

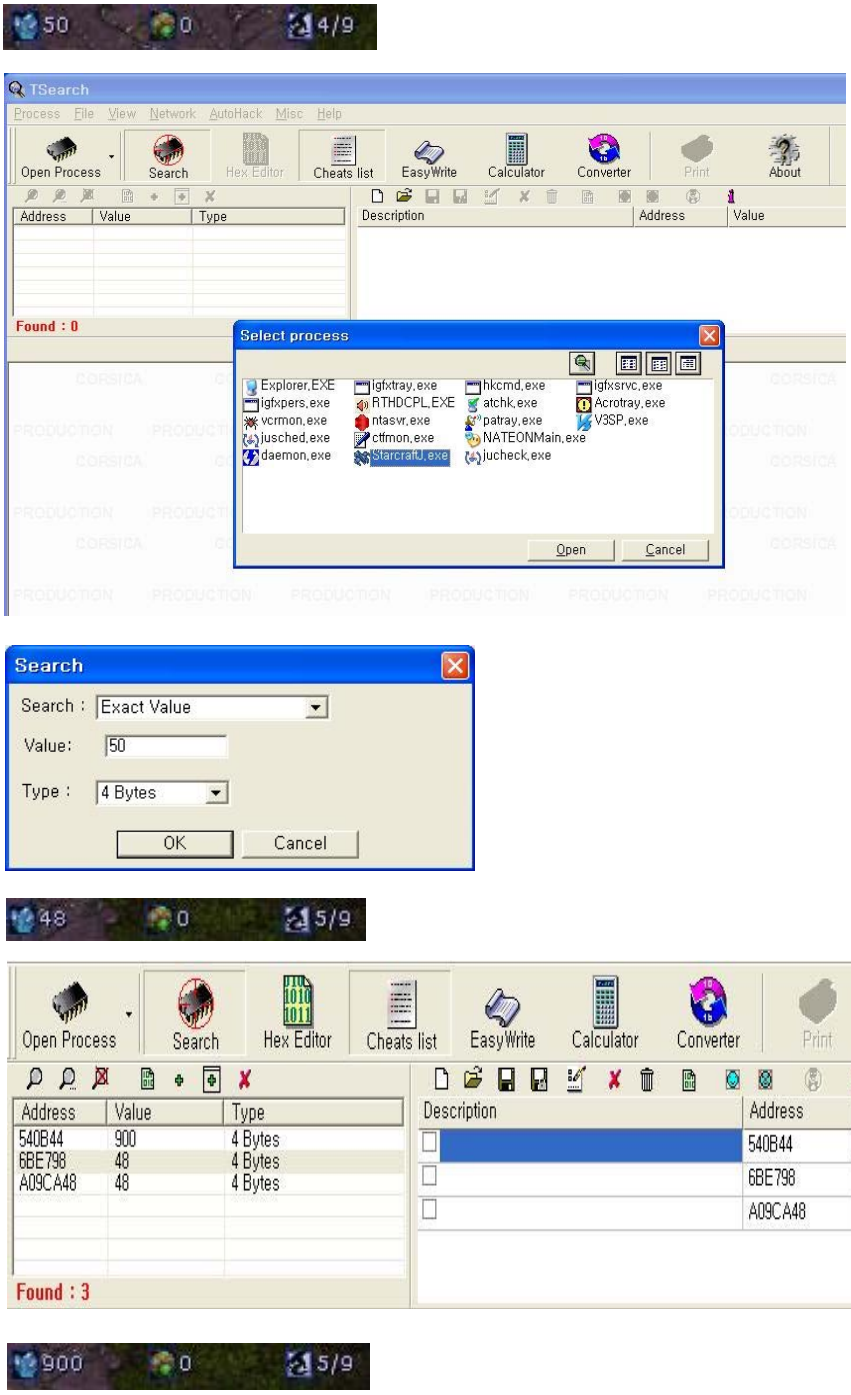

그림 7 TSearch를 사용하여 메모리에 올라가 있는 값을 변경해보았다. 변경되는 값만 입력해주면 자동으로 찾아주므로 사용자는 원하는 값으로 변경만 하면 된다.

유명한 binary diffing툴인 다른그림(DarunGrim)을 이용하여 윈도우패치의 내용을 살펴보는 예제를 해보았다. 이 방법을 통하여 직접적인 패치의 루틴들을 살펴볼 수 있으며 취약점 분석/보완 작업에 도움을 줄 수 있다.

Apr 10, 2007	**Vulnerability in Microsoft Agent Could Allow Remote Code Execution (932168): MS07-020** **Affected Software:** Windows 2000 Server, Windows 2000 Professional, Windows 2000 Datacenter Server, Windows 2000 Advanced Server, Windows XP Home Edition, Windows XP Professional, Windows XP Professional 64-Bit Edition, Windows Server 2003 for Small Business Server, Windows Server 2003, Datacenter Edition, Windows Server 2003, Enterprise Edition, Windows Server 2003, Standard Edition, Windows Server 2003, Web Edition, Windows Server 2003 Datacenter Edition for Itanium-based Systems, Windows Server 2003 Enterprise Edition for Itanium-based Systems, Windows Server 2003 Datacenter x64 Edition, Windows Server 2003 Enterprise x64 Edition, Windows Server 2003 Standard x64 Edition	Windows 2000 Service Pack 4, Windows XP Service Pack 2, Windows XP Professional x64 Gold, Windows Server 2003 Gold, Windows Server 2003 SP1, Windows Server 2003 for Itanium-based Systems Gold, Windows Server 2003 for Itanium-based Systems SP1, Windows Server 2003 x64 Gold, Windows XP Professional x64 SP2, Windows Server 2003 SP2, Windows Server 2003 x64 SP2, Windows Server 2003 for Itanium-based Systems SP2	Critical

그림 8 http://www.microsoft.com/technet/security/current.aspx 에서 윈도우 패치들을 다운로드할 수 있다. 예전 파일을 다운로드하여 현재의 파일과 비교해보았다.

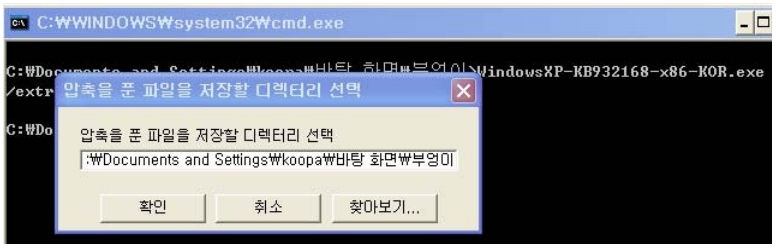

그림 9 받은 파일을 실행하면 바로 설치하게 된다. 뒤에 /extract옵션을 주어서 압축을 푼다.

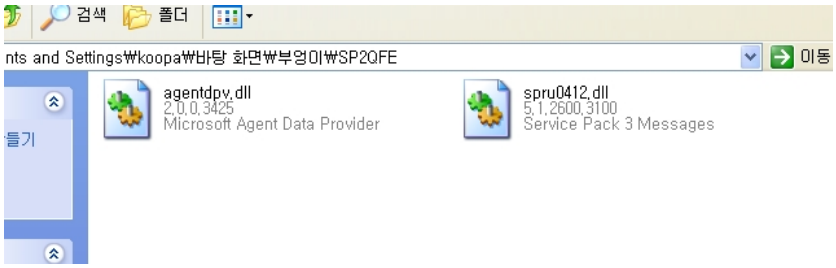

그림 10 파일 중에서 agentdpv.dll파일을 비교해보았다.

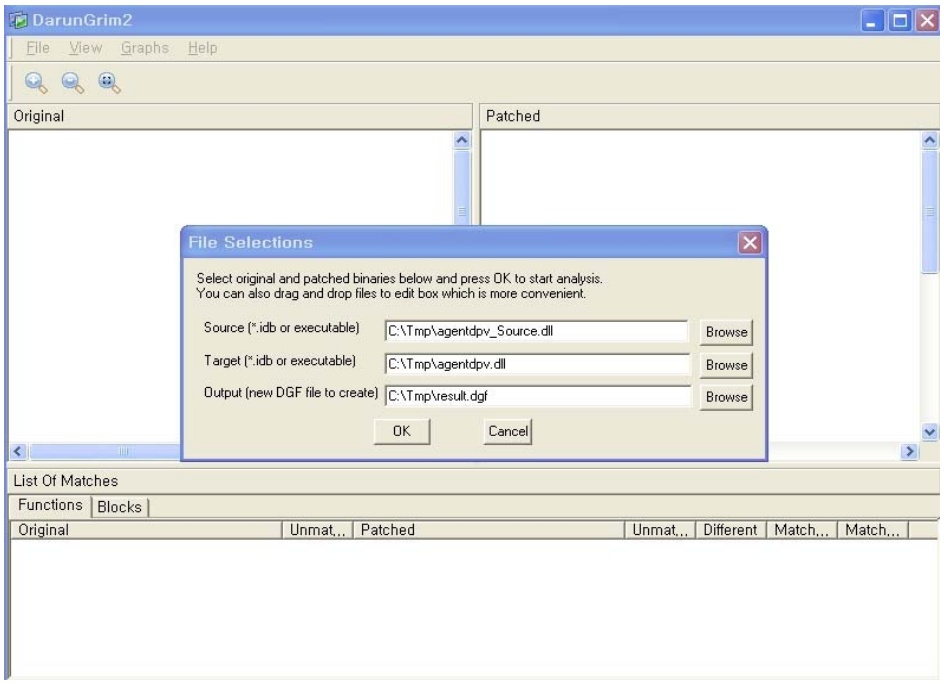

그림 11 다른그림2를 사용해보았다. 다른그림은 IDA5.0이상이 설치되어야 이상없이 잘 동작한다.

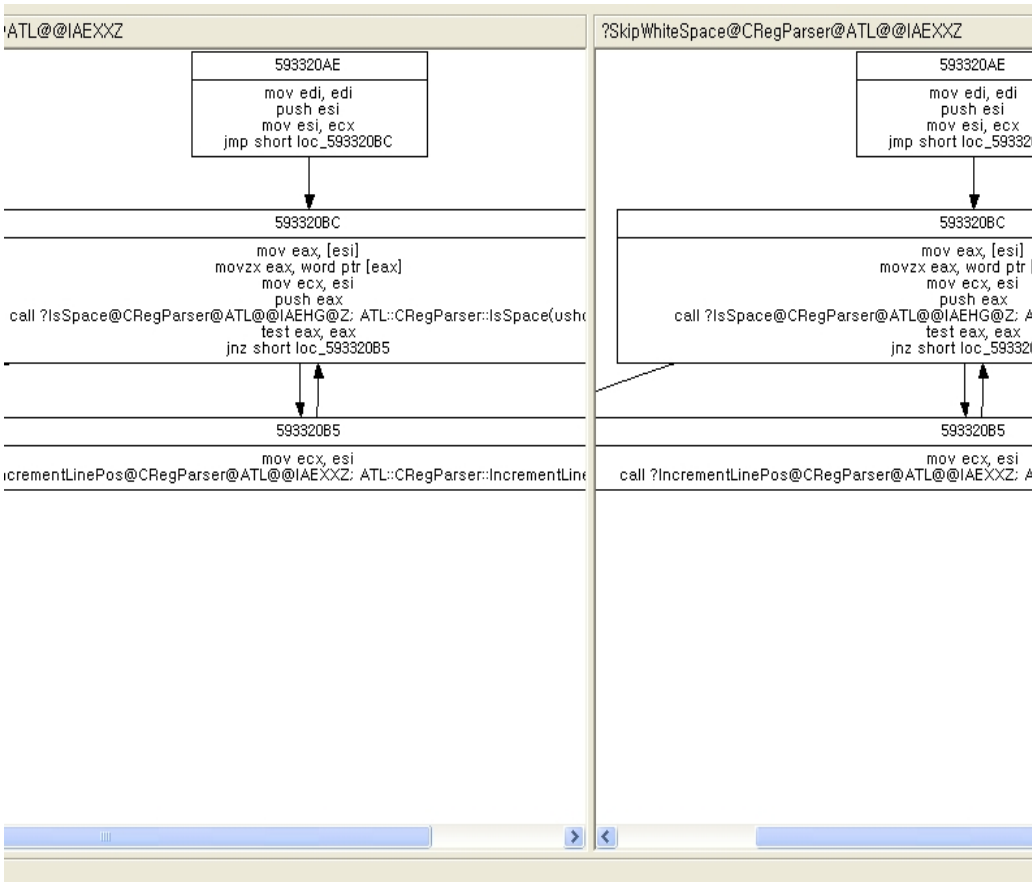

그림 12 오랜 분석시간 끝에 결과가 나왔다. IDA는 그림과 같이 보기 쉽도록 그래프로 표현하는데 다른그림이 이 기능을 활용하여 diffing을 한다.

List Of Matches

Functions | Blocks

Original	Unmatched	Patched	Unmatched	Different	Matched	Match Rate
_IID_IAgentDataProvider	0	_IID_IAgentDataProvider	0	0	8	100%
_CLSID_AgentDocFileProvider	0	_CLSID_AgentDocFileProvider	0	0	2	100%
_IID_IAgentDataProvider2	0	_IID_IAgentDataProvider2	0	0	1	100%
_IID_IAgentCharacterState	0	_IID_IAgentCharacterState	0	0	1	100%
_IID_IAgentDataProviderSink2	0	_IID_IAgentDataProviderSink2	0	0	2	100%
_IID_IAgentAnimationProvider2	0	_IID_IAgentAnimationProvider2	0	0	2	100%
	0	func_5933CB35	0	0	0	0%
	0	func_5933CB31	0	0	0	0%
	0	func_5933CB15	0	0	0	0%
	0	func_5933CB11	0	0	0	0%
	0	func_5933CB0D	0	0	0	0%
	0	sub_59337B18	0	0	0	0%
	0	sub_59336E5B	0	0	0	0%
	0	sub_5933552A	0	0	0	0%
	0	?Release@?$CComObject@VCAgen...	0	0	0	0%
	0	?Release@?$CComObject@VCAgen...	0	0	0	0%

그림 13 아래는 함수들을 표시한다. Match Rate가 0% 혹은 100%인 것으로 보아 함수가 몇 가지 추가만 된 것으로 보인다.

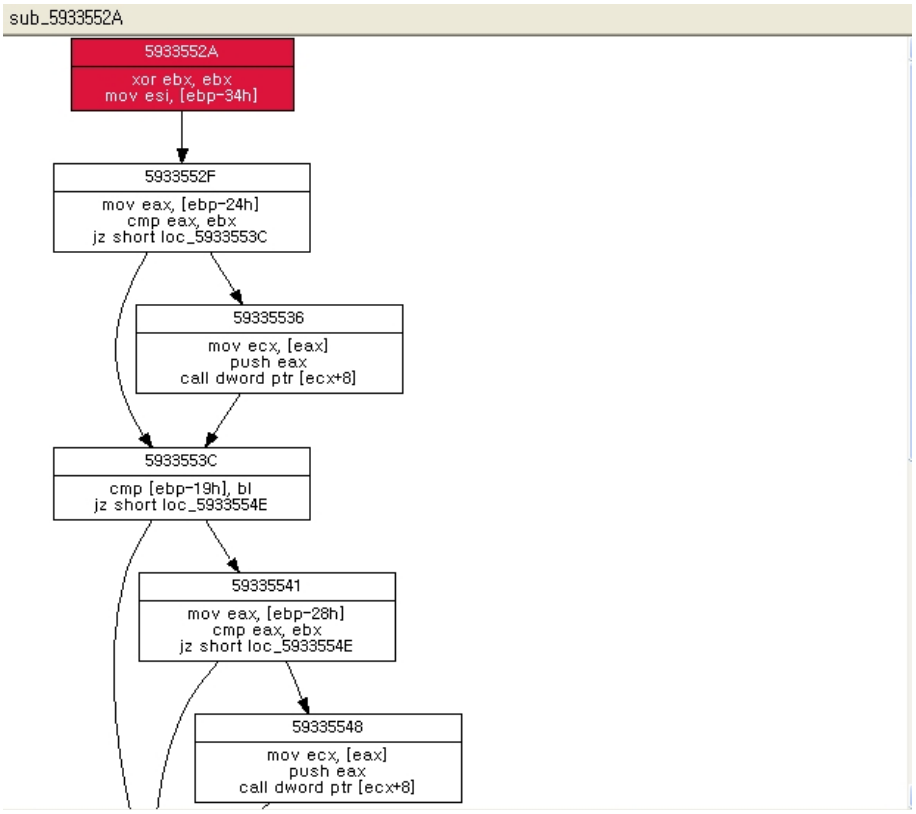

그림 14 추가된 함수중 하나를 그래프로 열어 보았다. 빨간색으로 나타나 있는 부분이 원본과 다르다는 부분이다. 기존에는 없던 것이 추가가 되었음을 나타낸다.

3. 한 계

개발자의 입장에서는 파일의 값을 사용자가 임의로 변경할 수 없도록 대안을 마련하는 것도 중요하지만 애초에 위와 같은 비교가 어렵도록 프로그램을 만들어서 변경의 단계에 도달하기 전에 포기하도록 만드는 것도 필요하다. 파일이 조작되었음을 알려주는 방법과 파일의 조작을 어렵도록 하는 방법을 알아보면 다음과 같다.

가. 체크섬과 해시값

앞에서도 잠깐 언급하였지만 파일의 내용이나 속성이 변하면 체크섬이나 해시값도 변한다. 프로그램은 파일을 읽어 들이면서 특정부분의 체크섬이나 해시값을 구하여 파일의 데이터와 같이 저장할 수 있다. 만약 파일내용이 바뀌었을 경우 프로그램이 파일을 읽어 들이는 과정에서 체크섬과 해시값이 다를 것이므로 정상적인 작동이 불가능하도록 할 수 있다.

개발자가 파일에 그 값들을 저장하는 경우 크래커는 체크섬이나 해시값의 저장부분을

하나하나씩 비교법을 통하여 알아낼 수 있다. 또한 그 값들을 구하는 알고리즘을 잘 알고 있다면 파일조작이 더 쉬울 것이다. 힘든 작업이긴 하나 불가능한 것은 아니라고 생각한다.

나. 압축과 암호화

압축된 파일이나 암호화가 된 파일을 처리되기 전 파일과 비교해보면 아마 파일의 대부분이 바뀌어 있을 것이다. 압축 알고리즘과 암호화 알고리즘의 종류에 따라 다르겠지만 파일을 이러한 방식으로 처리하면 원래대로 복원을 하지 않는 이상 파일조작에 접근조차 어려울 것이다.

하지만 압축 알고리즘의 경우 공개되어 있는 것이 많아서 알고리즘을 알아내기만 한다면 복원작업도 가능하다. 암호화도 마찬가지다. 그렇다고 프로그램을 만든 개발자가 직접 알고리즘을 만든다면 그러한 알고리즘의 전문가가 아닌 이상 허점이 생기기 마련이므로 파일이 오히려 더 취약해질 가능성이 있다.

비교법을 방지하는 어떠한 방법에도 한계점은 있다는 것을 알리기 위해 단순히 가정을 세웠을 뿐이지 실제로는 더 좋은 방법과 알고리즘들이 많아서 개발자가 대처를 잘 한다면 크래커가 건드릴 엄두도 못내는 구조를 만들 수 있을 것이다.

4. 참고 문헌

[1] 네트웍해킹 퇴치비법-에이콘

[2] http://research.hackerschool.org/Datas/Research_Lecture/Binary%20Diffing.pdf

[3] http://technet.microsoft.com/ko-kr/sysinternals/default(en-us).aspx

[4] http://www.darungrim.org/