

Xero

박준혁 (한국디지털미디어고등학교 1학년)

2011.10.16

[C] Trivial 100P

Subject

secu2011

Comment

0xC73B9452

0xC73B95C6

0xC73B95E6

C변은 간단한 문제였다.

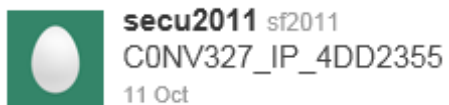
0xC73B9452 값을 보고 10진수로 변환하니 3342570578 값이 나왔다.

최근에 ip를 10진수, 16진수로도 표현할 수 있다는 것을 알게 되어서 myip.kr에서 ip주소로 변환해 199.59.148.82 라는 주소를 얻었다.

들어가보니 트위터 사이트가 나타났고, 다른 주소들도 마찬가지로였다.

계속 고민하다가 subject가 secu2011 인것을 보고 <http://twitter.com/secu2011> 으로 들어가 보았다.

그러자 다음과 같이 키 값을 발견했고 인증에 성공했다.



Key : C0NV327_IP_4DD2355

[D] Trivial 100P

Subject

This is crypto

Comment

ThisIsCrypto

By. Rust

File : ThisIsCrypto.zip

처음에는 원지 예상조차못해서 포기했었는데 힌트파일이 추가되었다.

추가된 pdf 힌트파일을 읽어도 딱히 도움될 만한 글은 없었다.

그러나 요즘 유행하는 세로드립으로 쓴 Fuck You 라는 글을 발견했다.

그리고 문제 텍스트파일을 세로로 읽다가 5번째 컬럼에서 세로로 읽어 키로 생각되는 문장을 발견하였다.

대회가 끝나고 이 문제에 대해 문서를 쓰다가 다른 사실들도 알아내었다.

캘리포니아 주의회 의원인 Tom Ammiano가 슈왈제네거에게 "Kiss my gay ass"라고 욕을 하곤 주지사 연설 도중 나간 일이 있다고 한다. 슈왈제네거는 그에 대한 대답으로 해당 의원이 낸 법안에 거부권을 행사하면서 다음과 같은 글을 썼다고 한다.

물론 내용은 문제될 게 없지만 일부러 세로로 욕을 쓴 것이다.

미리 이 사실을 알았다면 세로로 읽으면 된다는 것을 알 수 있었을 텐데 아쉬움이 남는다.

Key : YouActivateTrapCard!

[E] Analysis 100P

Subject

♥0♥

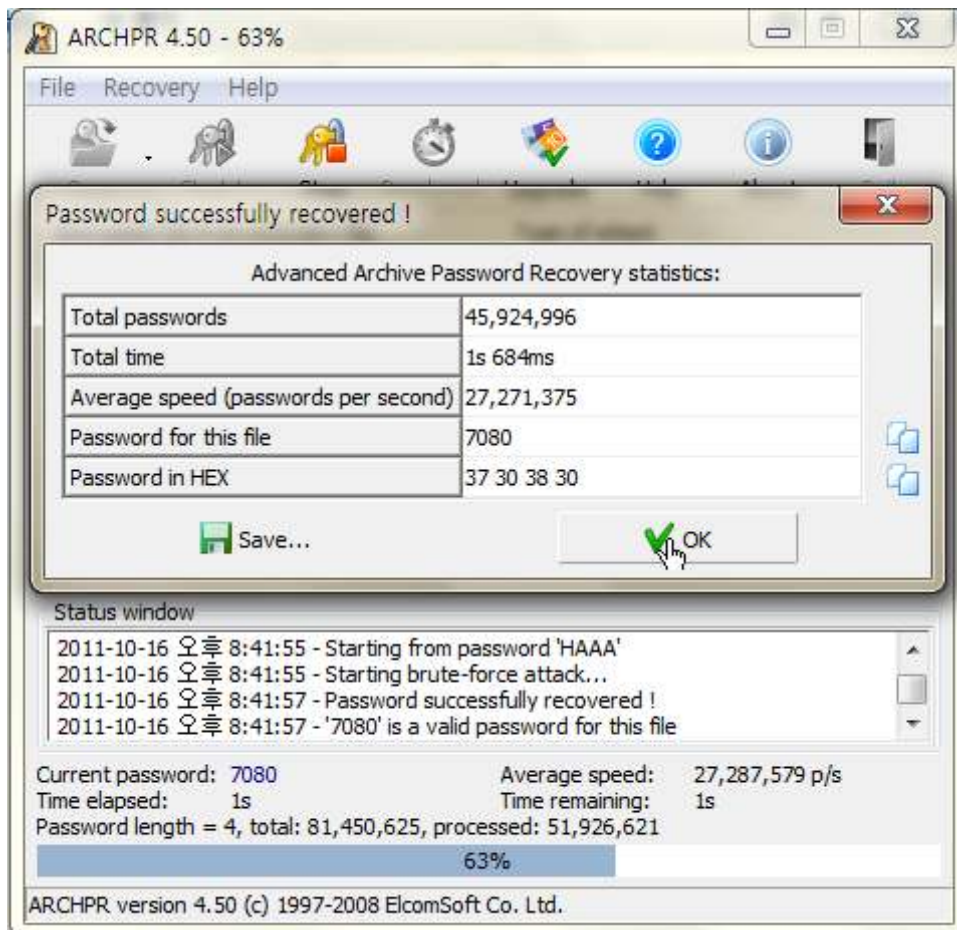
Comment

소느님 사랑해여♥0♥

By. Hak6a6y

File : wOw.zip

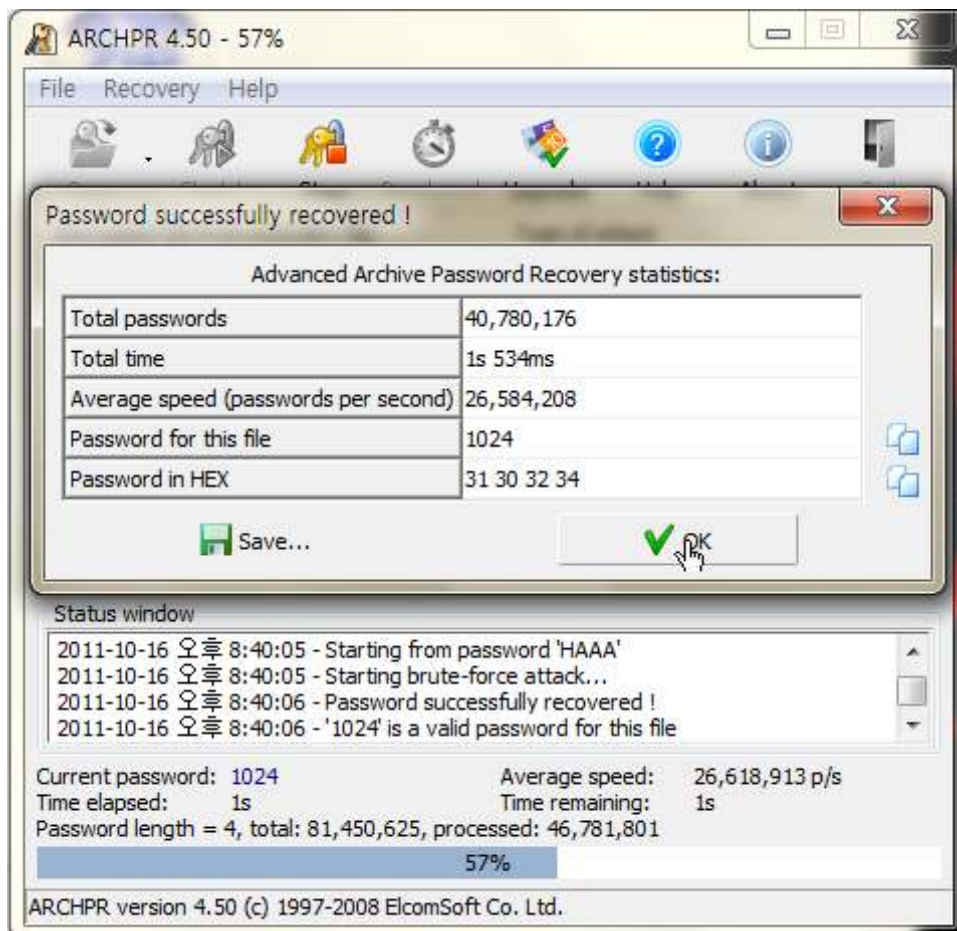
wOw.zip 파일에 압축 암호가 걸려있길래 ARCHPR 툴을 이용하여 7080이라는 비밀번호를 알아내었다.



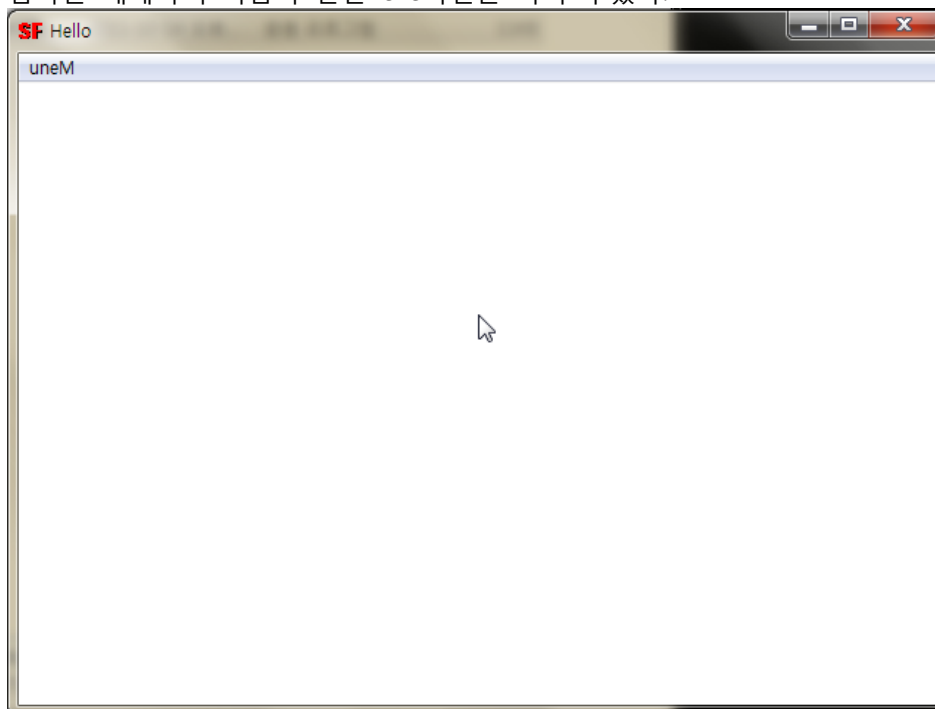
압축을 해제하니 두 개의 이미지와 또 다른 압축 파일 하나를 주었다.

또 압축파일에 비밀번호가 걸려있었다.

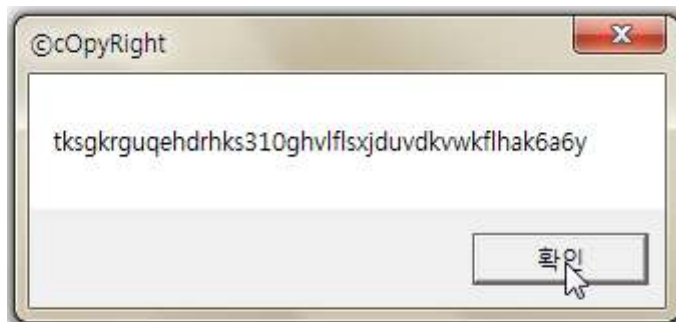
또 툴을 이용해 브루트포싱하니 1024라는 비밀번호가 나왔다.



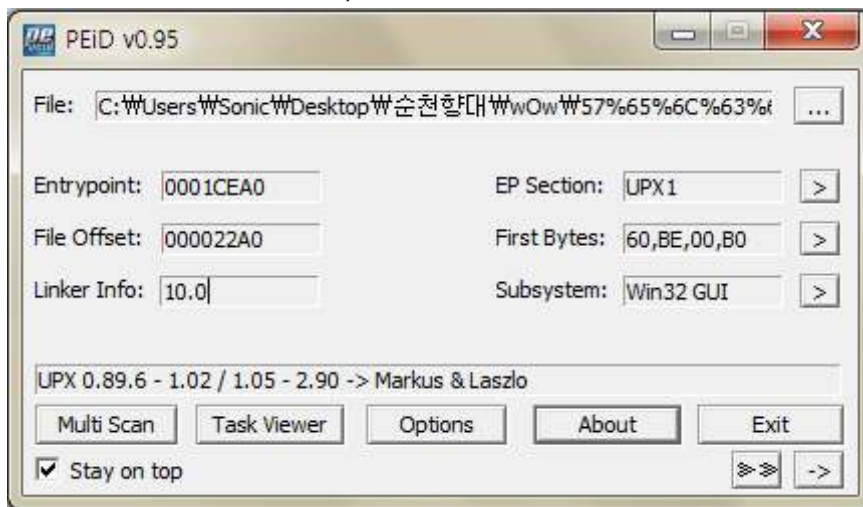
압축을 해제하니 다음과 같은 exe파일을 하나 주었다.



메뉴의 copyright를 눌러서 뜯은 메시지박스의 값을 인증해보았지만 키가 아니었고 그냥 리버싱을 하기로 했다.



PEiD로 보면 다음과 같이 upx로 패킹되어있는것을 확인할 수 있다.



툴로 언패킹해도 되지만 수동으로 언패킹하기로 했다.

UPX 언패킹 방법은 어셈 코드의 맨뒤 DB 00 코드 세 줄 위의 JMP 구문에 F2를 눌러 브레이크를 걸고 F9로 실행하여 그 부분에서 멈추면 F8을 한번 누르면 패킹이 풀린 실제 코드 부분으로 들어가게된다.

0041CFE3	, 54	PUSH ESP
0041CFE4	, 50	PUSH EAX
0041CFE5	, 53	PUSH EBX
0041CFE6	, 57	PUSH EDI
0041CFE7	, FF05	CALL EBP
0041CFE9	, 58	POP EAX
0041CFEA	, 61	POPAD
0041CFEB	, 804424 80	LEA EAX,DWORD PTR SS:[ESP-80]
0041CFEF	> 6A 00	PUSH 0
0041CFF1	, 39C4	CMP ESP,EAX
0041CFF3	^ 75 FA	JNZ SHORT Qnkgd2Fp,0041CFEF
0041CFF5	, 83EC 80	SUB ESP,-80
0041CFF8	,- E9 8941FFFF	JMP Qnkgd2Fp,00411186
0041CFFD	00	DB 00
0041CFFE	00	DB 00
0041CFFF	00	DB 00

다음이 실제 코드부분이다.

00411186	✓ E9 450C0000	JMP Qnkgd2Fp,00411DD0	
0041118B	✓ E9 22290000	JMP Qnkgd2Fp,00413AB2	JMP to MSVCR100, _controlfp_s
00411190	✓ E9 C5290000	JMP Qnkgd2Fp,00413B5A	JMP to kernel32, GetSystemTimeAsFileTime
00411195	✓ E9 A2290000	JMP Qnkgd2Fp,00413B3C	JMP to ntdll, RtlDecodePointer
0041119A	✓ E9 19290000	JMP Qnkgd2Fp,00413AB8	JMP to MSVCR100, _invoke_watson
0041119F	✓ E9 2C230000	JMP Qnkgd2Fp,004134D0	
004111A4	✓ E9 C7020000	JMP Qnkgd2Fp,00411470	
004111A9	✓ E9 881A0000	JMP Qnkgd2Fp,00412C36	JMP to MSVCR100, _CRT_RTC_INITW
004111AE	✓ E9 95290000	JMP Qnkgd2Fp,00413B48	JMP to kernel32, GetTickCount
004111B3	✓ E9 38080000	JMP Qnkgd2Fp,004119F0	JMP to USER32, KillTimer
004111B8	✓ E9 63210000	JMP Qnkgd2Fp,00413320	
004111BD	✓ E9 A4290000	JMP Qnkgd2Fp,00413B66	JMP to ntdll, RtlAllocateHeap
004111C2	✓ E9 E31E0000	JMP Qnkgd2Fp,004130AA	JMP to MSVCR100, _amsg_exit
004111C7	✓ E9 12200000	JMP Qnkgd2Fp,004131DE	JMP to MSVCR100, _XcptFilter
004111CC	✓ E9 43080000	JMP Qnkgd2Fp,00411A14	JMP to USER32, SendMessageW
004111D1	✓ E9 20200000	JMP Qnkgd2Fp,004131F6	JMP to MSVCR100, _CrtSetCheckCount
004111D6	✓ E9 01290000	JMP Qnkgd2Fp,00413ADC	JMP to kernel32, InterlockedExchange
004111DB	✓ E9 50290000	JMP Qnkgd2Fp,00413B30	JMP to kernel32, UnhandledExceptionFilter

Search for에서 All intermodular calls 로 호출되는 함수들을 보았다.

Backup	▶	Command	Ctrl+F
Copy	▶	Sequence of commands	Ctrl+S
Binary	▶	Constant	
Assemble	Space	Binary string	Ctrl+B
Label	:	All intermodular calls	
Comment	;	All commands	
Breakpoint	▶	All sequences	
Run trace	▶	All constants	
Follow	Enter	All switches	
Go to	▶	All referenced text strings	
Follow in Dump	▶	User-defined label	
Search for	▶	User-defined comment	

다음과 같이 호출되는 함수들이 보인다.

Found intermodular calls		
Address	Disassembly	Destination
00411186	JMP Qnkgd2Fp,00411DD0	(Initial CPU selection)
004114A8	CALL DWORD PTR DS:[418284]	GD132, GetStockObject
004114C1	CALL DWORD PTR DS:[41844C]	USER32, LoadCursorW
004114D9	CALL DWORD PTR DS:[418450]	USER32, LoadIconW
00411512	CALL DWORD PTR DS:[418454]	USER32, RegisterClassW
00411554	CALL DWORD PTR DS:[418458]	USER32, CreateWindowExW
0041156E	CALL DWORD PTR DS:[41845C]	USER32, ShowWindow
00411587	CALL DWORD PTR DS:[418460]	USER32, GetMessageW
0041159E	CALL DWORD PTR DS:[418464]	USER32, TranslateMessage
004115B1	CALL DWORD PTR DS:[418468]	USER32, DispatchMessageW
00411C4A	CALL Qnkgd2Fp,004111A9	MSVCR100, _CRT_RTC_INITW
00411C7A	CALL Qnkgd2Fp,004111A9	MSVCR100, _CRT_RTC_INITW
00411CC1	CALL DWORD PTR DS:[4183C8]	MSVCR100, __set_app_type
00411CCC	CALL DWORD PTR DS:[41832C]	ntdll, RtlEncodePointer
00411D14	CALL DWORD PTR DS:[4183D4]	MSVCR100, __setusermatherr
00411D2D	CALL DWORD PTR DS:[4183D8]	MSVCR100, _configthreadlocale

CrackMe나 여러 리버싱 문제들에서 키 값을 메시지박스로 표시하므로 키 값이 그 부분에 있다고 생각하여 MessageBox 함수를 호출하는 부분을 찾아가보았다.

그러자 SNNHAK1215.DLL 이라는 문자열을 출력하는 구문을 발견하였다.

Address	Hex dump	Disassembly	Comment
00413D81	3BF4	CMP ESI,ESP	
00413D83	E8 E5D3FFFF	CALL Qnkgd2Fp,0041116D	
00413D88	EB 32	JMP SHORT Qnkgd2Fp,00413DBC	
00413D8A	8BF4	MOV ESI,ESP	
00413D8C	6A 00	PUSH 0	
00413D8E	FF15 34844100	CALL DWORD PTR DS:[418434]	USER32,MessageBeep
00413D94	3BF4	CMP ESI,ESP	
00413D96	E8 D2D3FFFF	CALL Qnkgd2Fp,0041116D	
00413D9B	EB 1F	JMP SHORT Qnkgd2Fp,00413DBC	
00413D9D	8BF4	MOV ESI,ESP	
00413D9F	6A 00	PUSH 0	
00413DA1	68 A0624100	PUSH Qnkgd2Fp,004162A0	
00413DA6	68 385A4100	PUSH Qnkgd2Fp,00415A38	UNICODE "SNNHAK1215.DLL"
00413DAB	8B45 08	MOV EAX,DWORD PTR SS:[EBP+8]	
00413DAE	50	PUSH EAX	
00413DAF	FF15 6C844100	CALL DWORD PTR DS:[41846C]	USER32,MessageBoxW

Key : SNNHAK1215.DLL

위에서 '는 한글이 아니고 또한 마땅히 치환할 만한 문자가 없어서 그냥 두니 'ㅎ' 라는 문자열로 되고, 뒤의 ㄹ 이 ㅎ으로 해독되어 ㅎㅎ로 딱 맞게 떨어진다.

출제자분의 센스를 잘 표현한 것 같다.

위의 디코드 표대로 해독하면 다음의 문장이 나온다.

안녕하세요. 러스트입니다.

대회는 재미있게 즐기고 계신가요? ㅎㅎ

너무 모니터와 키보드만 보시는 것 같아서 종이 위 펜으로도 풀 수 있는 문제를 준비하였습니다 ㅎㅎ

그럼 키값을 알려드리겠습니다. (영타로 인증해주세요) 키값은 내가바로해독왕이그입니다.

Key : sorkqkfhgoehrdhkddlrml

[G] Forensic 200P

Subject

Energy

Comment

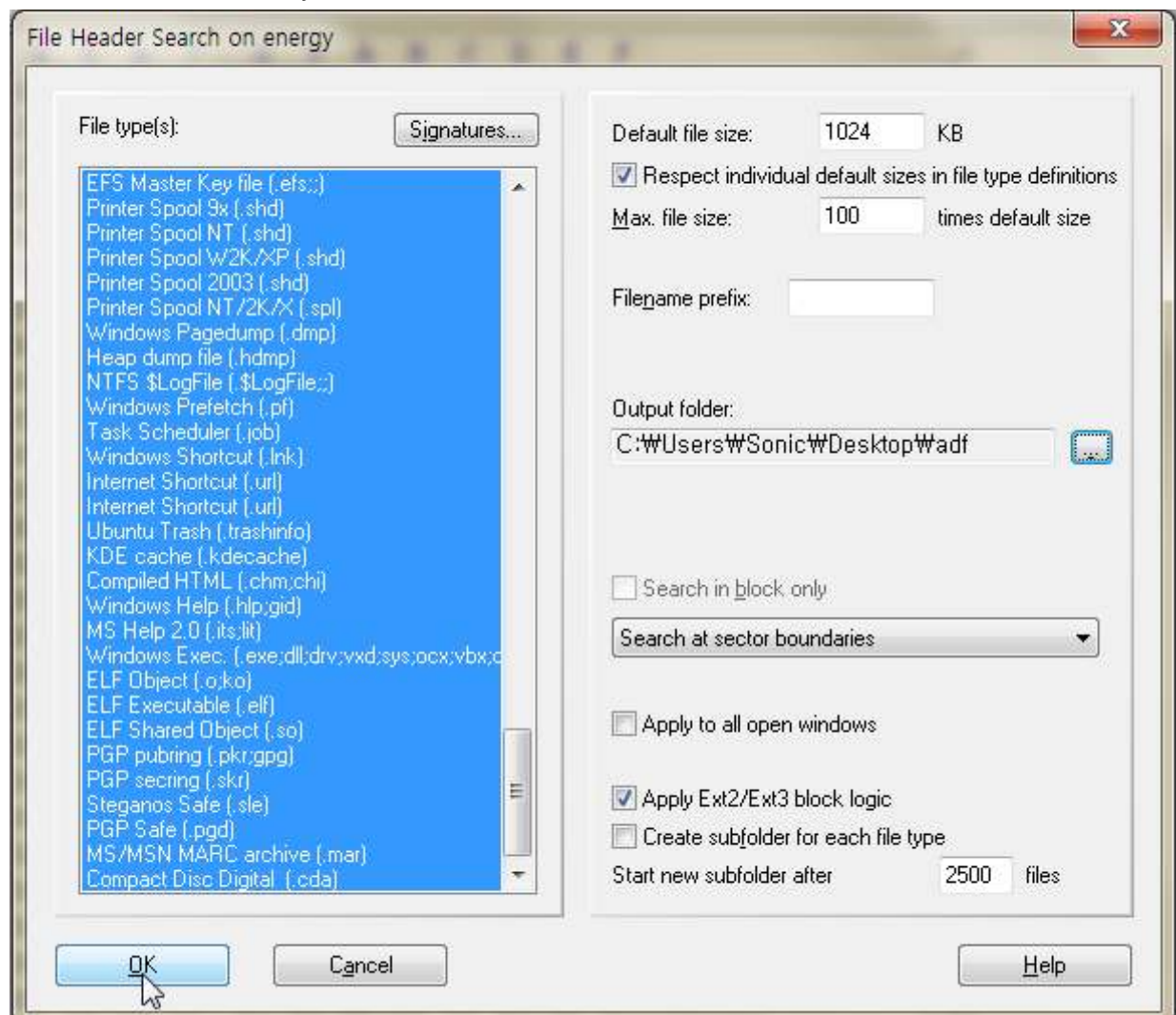
I'm Your Energy - J.S Park

By. lwmr

File : energy.zip

energy.zip 파일을 압축해제하면 energy 라는 파일이 나온다.

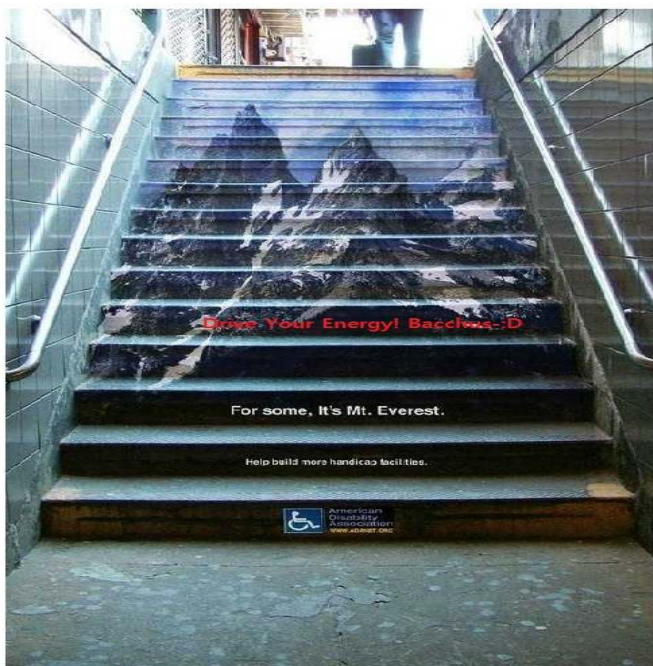
winhex를 이용해 recovery하였다.



다음과 같이 20개의 파일들이 추출되었다.



파일들을 둘러보다가 00008.jpg 에 적혀있는 글귀에 energy라는 글이 있길래 인증하니 성공하였다.



Key : Drive Your Energy! Bacchus-:D

[N] Forensic 200P

Subject

남친 털기

Comment

남자친구의 가방에서 USB를 획득한 영희.

USB의 한 폴더엔 여자의 사진들로 가득했는데...

그 모습을 본 남자친구는 화들짝 놀라며 폴더를 지워버렸다.

영희가 다른 폴더를 뒤질려고 하자 남자친구는 화를 내며 USB를 포맷시켜 버렸다.

화가난 영희는 포맷된 USB를 가져왔는데.....

By. MANO

File : 남친 털기

처음에는 winhex로 recovery를 하여 파일들을 추출하니 30개의 이미지가 나왔고, 이미지마다 문자열들이 적혀있었다.

이미지의 이름 순서대로 문자열을 모으니 다음의 값이 나왔다.

c6568ac7a43ec681a6a0a0c2940eca470daf2999917d13b77d1a38ca6ed35bd106628ebc6828a59f062f8660450e366b

그러나 인증에 실패하였고, 검은 글씨와 흰 글씨 따로 로도 시도해보았으나 실패하였다.

그래서 다른 포렌식 툴을 찾아보기로 했다.

리눅스의 Autopsy forensic browser 이라는 툴을 이용하기로 했다.

우선 su root 명령어로 root 권한을 얻은 후 autopsy를 실행시켰다.

(처음엔 autopsy가 깔려 있지 않아 apt-get install autopsy 명령어로 설치를 하였다.)

```
root@Xero-vm: /home/sonic
파일(F) 편집(E) 보기(V) 검색(S) 터미널(T) 도움말(H)
root@Xero-vm:/home/sonic# autopsy

=====
Autopsy Forensic Browser
http://www.sleuthkit.org/autopsy/
ver 2.24
=====

Evidence Locker: /var/lib/autopsy
Start Time: Tue Oct 18 00:01:29 2011
Remote Host: localhost
Local Port: 9999

Open an HTML browser on the remote host and paste this URL in it:

http://localhost:9999/autopsy

Keep this process running and use <ctrl-c> to exit
█
```

위와 같이 정상적으로 autopsy가 실행되었고, <http://localhost:9999/autopsy> 주소로 들어가면 아래와 같이 autopsy forensic browser를 사용할 수 있다.

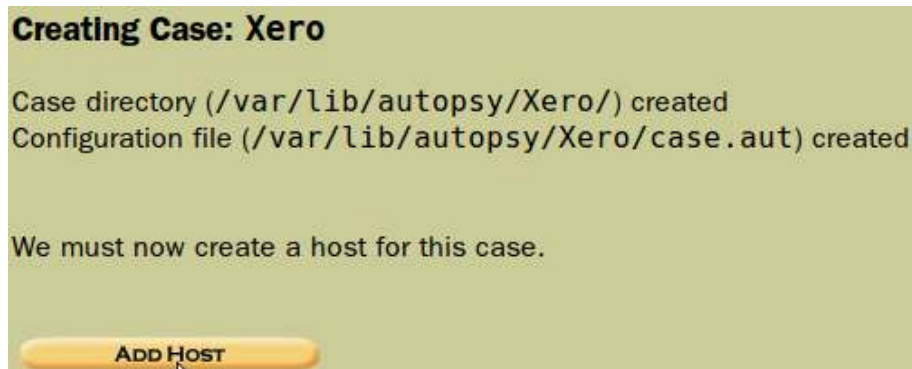
NEW CASE 로 새로운 케이스를 하나 만들었다.



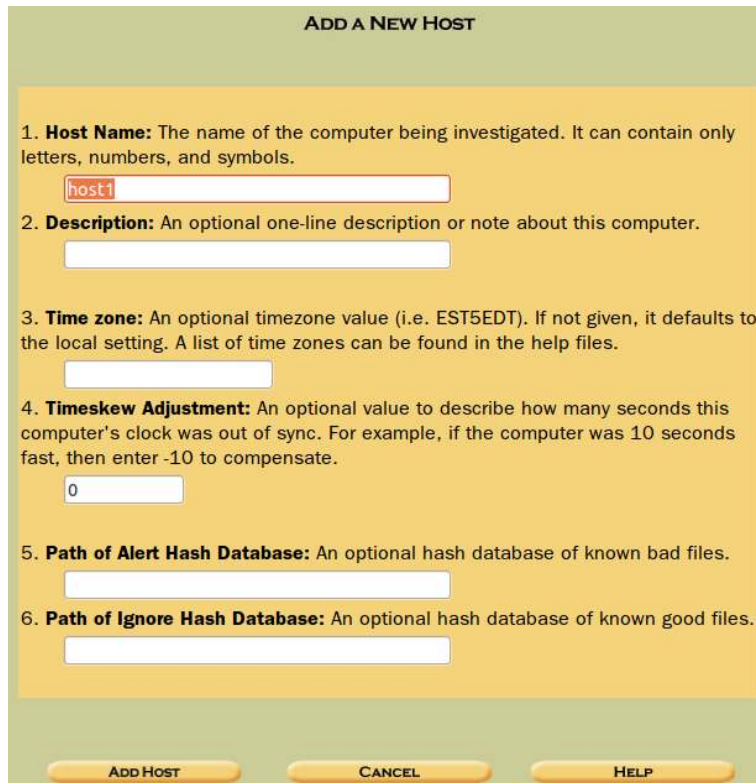
아래와 같이 Case Name을 입력하고 NEW CASE 버튼을 눌러서 만들었다.

The screenshot shows the "CREATE A NEW CASE" dialog box. The title bar reads "CREATE A NEW CASE". The dialog has a yellow background. It contains three sections: 1. **Case Name:** The name of this investigation. It can contain only letters, numbers, and symbols. Below this is a text input field containing the word "Xero". 2. **Description:** An optional, one line description of this case. Below this is a single-line text input field. 3. **Investigator Names:** The optional names (with no spaces) of the investigators for this case. Below this are two columns of five text input fields each, labeled a. through j. At the bottom, there are three yellow buttons: "NEW CASE", "CANCEL", and "HELP". A mouse cursor is pointing at the "NEW CASE" button.

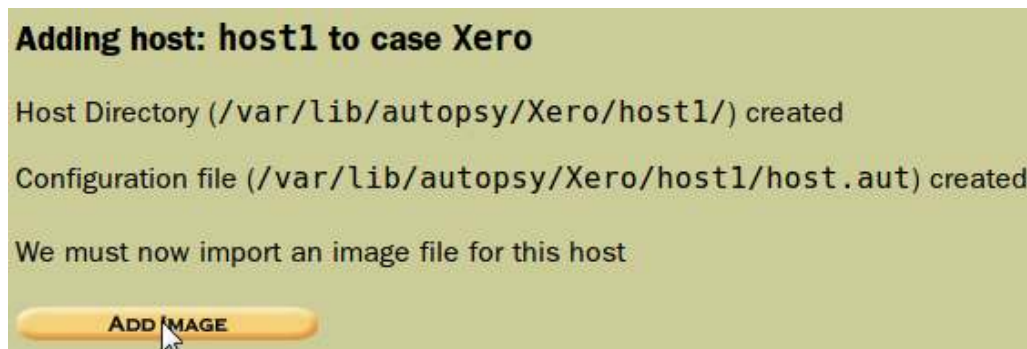
다음과 같이 새로운 케이스가 만들어졌고 ADD HOST 버튼으로 HOST를 추가하는 페이지로 넘어간다.



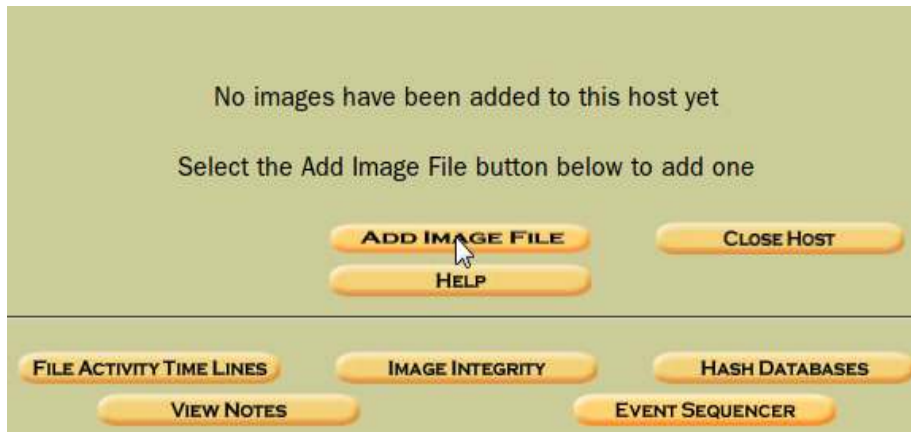
Host Name 을 입력하고 ADD HOST 버튼을 눌러 HOST를 추가하였다.



HOST가 추가되었고, ADD IMAGE 버튼을 눌러 이미지를 추가하는 페이지로 넘어갔다.



ADD IMAGE FILE 버튼을 눌러서 이미지파일을 추가했다.



Location 폼에 대상 파일의 full path를 넣고 Type을 Partition으로 선택하고 NEXT버튼으로 다음 과정으로 넘어갔다.

(‘남친털기’ 라는 한글 파일 이름을 그대로 넣었더니 에러가 나서 파일 이름을 영어로 변경하였다.)

A screenshot of a dialog box titled "ADD A NEW IMAGE". It has a yellow background. The first section is "1. Location" with instructions to enter the full path (starting with /) to the image file. If the image is split, to enter '*' for the extension. A text input field contains "/home/sonic/forensic/asdf". The second section is "2. Type" with instructions to select if the image file is for a disk or a single partition. There are two radio buttons: "Disk" (unselected) and "Partition" (selected). The third section is "3. Import Method" with instructions on how to import the image file. There are three radio buttons: "Symlink" (selected), "Copy" (unselected), and "Move" (unselected). At the bottom, there are three buttons: "NEXT" (with a mouse cursor clicking it), "CANCEL", and "HELP".

Data Integrity를 Calculate로 선택하고 ADD 버튼을 눌러 추가하였다.

Image File Details

Local Name: images/asdf

Data Integrity: An MD5 hash can be used to verify the integrity of the image. (With split images, this hash is for the full image file)

☐ Ignore the hash value for this image.

☒ Calculate the hash value for this image.

☐ Add the following MD5 hash value for this image:

☐ Verify hash after importing?

File System Details

Analysis of the image file shows the following partitions:

Partition 1 (Type: fat12)

Mount Point: File System Type:

Buttons: **ADD**, **CANCEL**, **HELP**

OK 버튼을 누르면 이미지가 추가된다.

Calculating MD5 (this could take a while)
Current MD5: 8CB8940E307C1652FDD782930429BF6D
Testing partitions
Linking image(s) into evidence locker
Image file added with ID img1
Volume image (0 to 0 - fat12 - C:) added with ID vol1

Buttons: **OK**, **ADD IMAGE**

ANALYZE 버튼을 눌러서 분석 해보았다.

Select a volume to analyze or add a new image file.

CASE GALLERY **HOST GALLERY** **HOST MANAGER**

mount	name	fs type	
C: /	asdf-0-0	fat12	details

Buttons: **ANALYZE**, **ADD IMAGE FILE**, **CLOSE HOST**, **HELP**

Buttons: **FILE ACTIVITY TIME LINES**, **IMAGE INTEGRITY**, **HASH DATABASES**, **VIEW NOTES**, **EVENT SEQUENCER**

다음과 같이 나타나는데 FILE ANALYSIS 를 눌러 파일 분석을 시도하였다.



다음과 같이 에러없이 분석이 되었고, \$OrphanFiles/ 로 들어가보았다.

Current Directory: [C:/](#)

[ADD NOTE](#) [GENERATE MD5 LIST OF FILES](#)

DEL	Type dir / in	NAME	WRITTEN	ACCESSED	CREATED	SIZE	UID	GID
v / v		\$FAT1	0000-00-00 00:00:00 (UTC)	0000-00-00 00:00:00 (UTC)	0000-00-00 00:00:00 (UTC)	6144	0	0
v / v		\$FAT2	0000-00-00 00:00:00 (UTC)	0000-00-00 00:00:00 (UTC)	0000-00-00 00:00:00 (UTC)	6144	0	0
v / v		\$MBR	0000-00-00 00:00:00 (UTC)	0000-00-00 00:00:00 (UTC)	0000-00-00 00:00:00 (UTC)	512	0	0
d / d		\$OrphanFiles/	0000-00-00 00:00:00 (UTC)	0000-00-00 00:00:00 (UTC)	0000-00-00 00:00:00 (UTC)	0	0	0
r / r		00 0000 (Volume Label Entry)	2011-08-22 21:43:58 (KST)	0000-00-00 00:00:00 (UTC)	0000-00-00 00:00:00 (UTC)	0	0	0

\$OrphanFiles/ 로 들어가면 다음과 같이 이미지들이 이름까지 완벽하게 복원된 것을 볼 수 있다.

Current Directory: [C:/](#) /\$OrphanFiles/

[ADD NOTE](#) [GENERATE MD5 LIST OF FILES](#)

DEL	Type dir / in	NAME	WRITTEN	ACCESSED	CREATED	SIZE	UID	GID
✓	- / r	ANUARY.JPG	2011-08-22 11:33:50 (KST)	2011-08-22 00:00:00 (KST)	2011-08-22 21:43:26 (KST)	44163	0	0
✓	- / r	ARCH.JPG	2011-08-22 11:34:12 (KST)	2011-08-22 00:00:00 (KST)	2011-08-22 21:43:26 (KST)	82284	0	0
✓	- / r	AY.JPG	2011-08-22 11:37:58 (KST)	2011-08-22 00:00:00 (KST)	2011-08-22 21:43:26 (KST)	191062	0	0
✓	- / r	CTOBER.JPG	2011-08-22 11:36:52 (KST)	2011-08-22 00:00:00 (KST)	2011-08-22 21:43:26 (KST)	96773	0	0
✓	- / r	EBRUARY.JPG	2011-08-22 11:34:02 (KST)	2011-08-22 00:00:00 (KST)	2011-08-22 21:43:26 (KST)	175318	0	0
✓	- / r	ECEMBER.JPG	2011-08-22 11:37:12 (KST)	2011-08-22 00:00:00 (KST)	2011-08-22 21:43:26 (KST)	111964	0	0

다음과 같이 월별, 요일별, 숫자별 등등으로 나타나있다.



파일명의 첫 글자는 복구하지 못했지만 딱 봐도 알 수 있게 되어있다.
월별로 이미지의 글자들을 이어붙이니 다음의 값이 나왔고 인증에 성공했다.
7d1991d1a0da38c773bf2bd1d35a6e99

Key : 7d1991d1a0da38c773bf2bd1d35a6e99

[R] Trivial 200P

Subject

Trivial is fun

Comment

By. Rust

File : trivia_is_fun.png

다음과 같이 TRiViA iS FUN! 이라는 글자가 일정한 패턴으로 이루어져 있다.

TRiViA iS FUN!

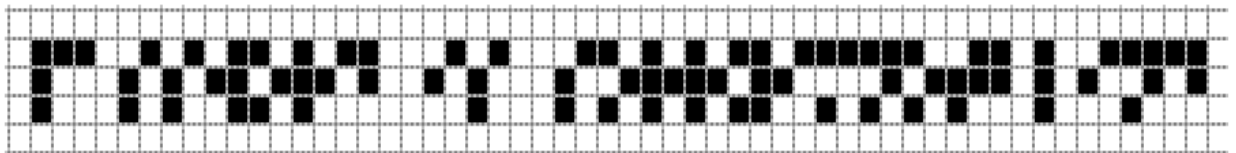
확대하여 글자들의 밑쪽을 보면 hint is blind 라는 글의 글자들이 한 글자씩 나와 있는 것을 볼 수 있다.

힌트가 보이지 않는 것이라고 해서 LSB 워터마크도 시도해보고 명도 채도 값, 레벨 값 변경도 시도해보았는데 아무것도 숨겨져 있지 않았다.

힌트로 점자가 나오자 그제서야 blind를 장님으로 보고 문제를 풀어야 한다는 걸 알았다.

글자들이 일정한 패턴으로 이루어져 있으므로 일정한 패턴을 뽑아보았다.

다음이 그 일정한 패턴이다.



영어 점자 표로 해독하니 password is sorryimnotblind 가 나왔다.

Key : sorryimnotblind

[T] Analysis 200P

Subject

I'm on a local

Comment

난 고약한 해커다!

길을가며 Wifi를 찾다가 취약한 AP를 발견하여 접속을 해보니

어떤 사람이 무슨 작업을 하고 있는듯 한데..?

By. Rust

File : T6133d45d592a75c103d30a3c3f33dcdb.zip

다음과 같이 패킷들이 잡히게 보인다.

1	0.000000	120.50.133.190	192.168.0.37	TCP	62	avt-prc
2	0.000014	192.168.0.37	120.50.133.190	TCP	60	54811 >
3	0.000076	192.168.0.37	120.50.133.190	TCP	54	[TCP DU
4	0.000179	192.168.0.37	120.50.133.190	TCP	54	[TCP DU
5	0.000431	192.168.0.37	120.50.133.190	TCP	66	54811 >
6	0.000469	192.168.0.37	120.50.133.190	TCP	66	[TCP Re
7	0.000517	192.168.0.37	120.50.133.190	TCP	66	[TCP Re
8	0.006255	120.50.133.190	192.168.0.37	TCP	60	avt-prc
9	0.006278	120.50.133.190	192.168.0.37	TCP	60	[TCP DU

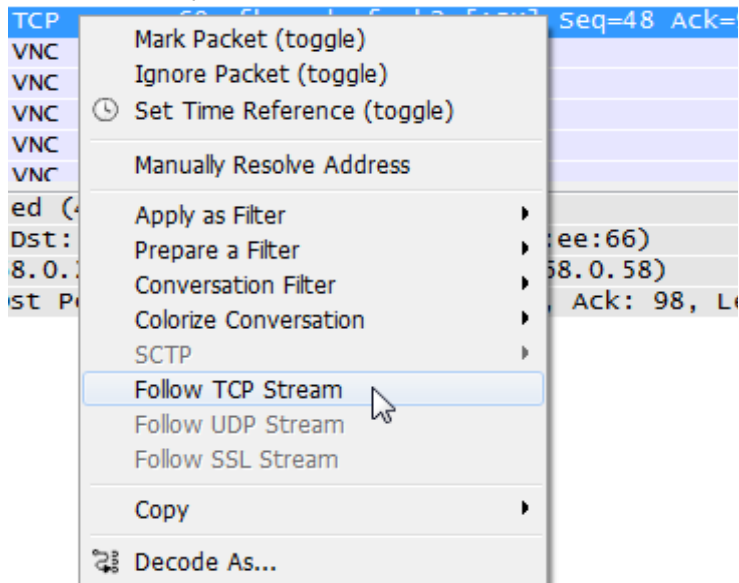
필터를 tcp로 하여 tcp만 뽑아보았다.

Filter: tcp

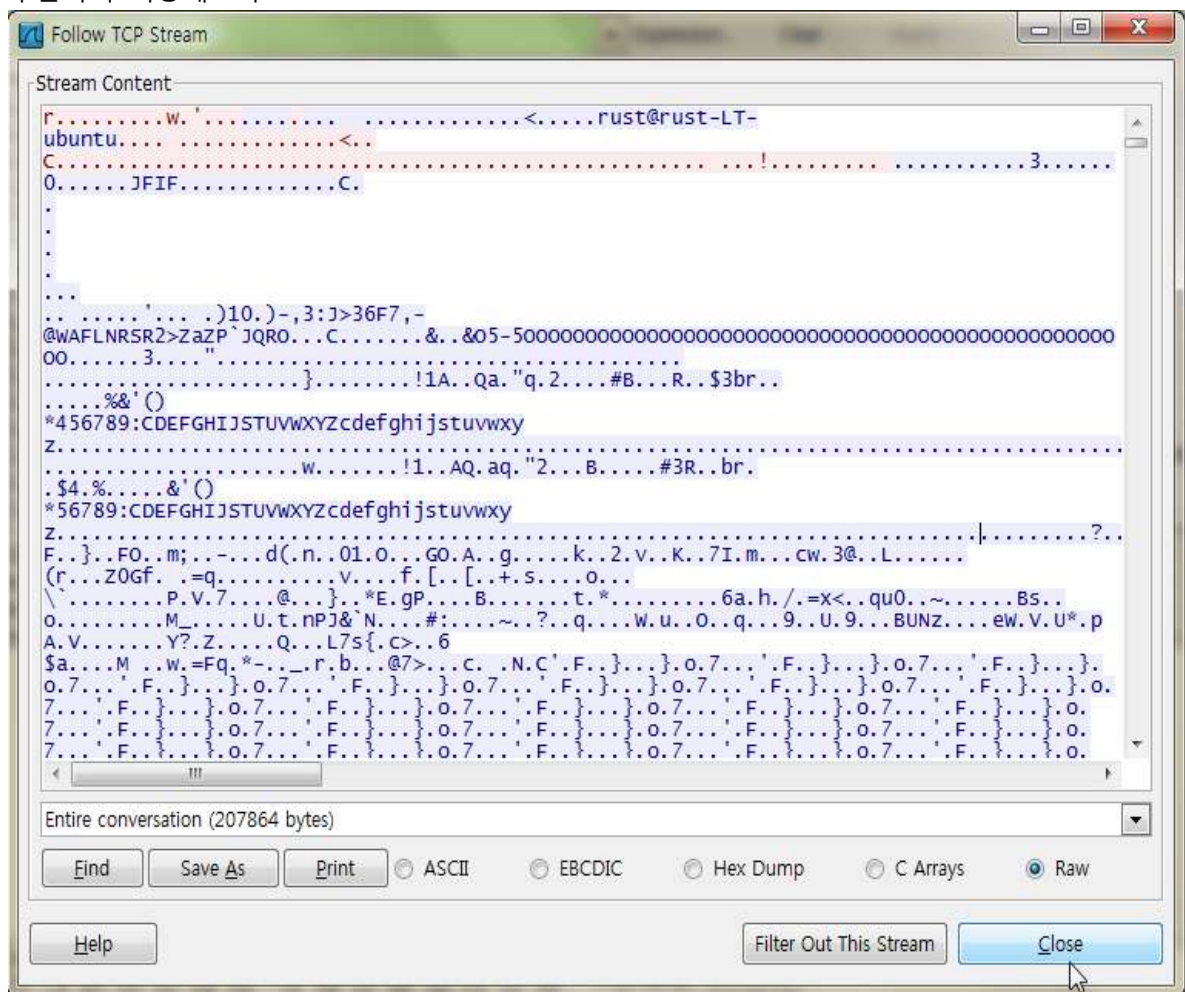
tcp 패킷들의 stream을 보던 도중 vnc와 함께 많은 패킷들이 오고가는 부분을 보았다.

36	11.776962	192.168.0.37	78.131.193.150	TCP	60	56514 > 40489 [ACK] Seq=1 Ack=1 win=65392 Len=1
37	11.949380	192.168.0.58	192.168.0.29	VNC	70	
38	11.954620	192.168.0.29	192.168.0.58	VNC	60	
39	11.971668	192.168.0.58	192.168.0.29	VNC	60	
40	11.975261	192.168.0.29	192.168.0.58	VNC	97	
41	12.062230	192.168.0.58	192.168.0.29	VNC	74	
42	12.062312	192.168.0.58	192.168.0.29	VNC	114	
43	12.063329	192.168.0.29	192.168.0.58	TCP	60	rfb > danf-ak2 [ACK] Seq=48 Ack=98 win=14600 Len=0
44	12.065050	192.168.0.58	192.168.0.29	VNC	64	
45	12.065876	192.168.0.29	192.168.0.58	VNC	60	
46	12.077085	192.168.0.29	192.168.0.58	VNC	1514	

다음과 같이 tcp stream을 보았다.



다음과 같이 JFIF 라는 헤더가 보이고 jpg라는 것을 유추 할 수 있다.
추출하여 저장해보자.



JFIF 이므로 jpg 파일이다.

jpg 파일은 파일 시그니처가 FF D8 FF로 시작하므로 Ctrl+F 로 찾아 앞쪽의 불필요한 헤더들을 제거하였다.

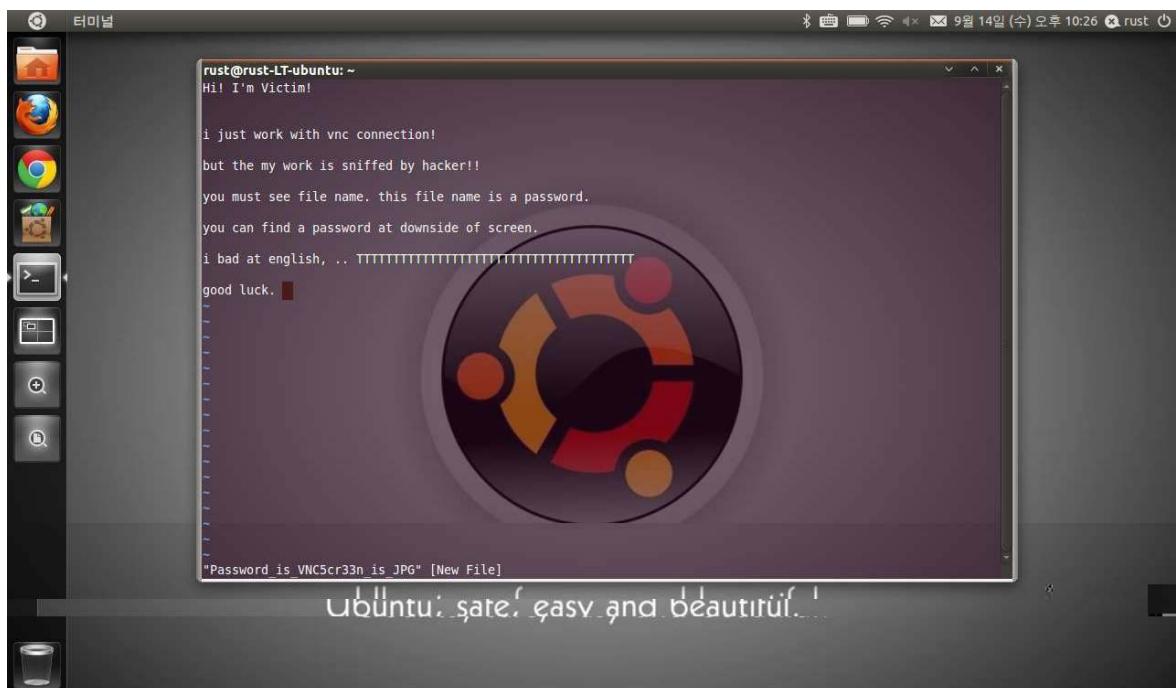
72 01 F6 19 E6 A1 A2 A5 AC 0B 77 8A 27 BF CC B8	r.ö.æ;¢¥¬.wŠ'¿İ,
00 00 00 00 01 05 00 03 20 20 18 00 01 00 FF 00ÿ.
FF 00 FF 10 08 00 FB 3C 01 00 00 00 13 72 75 73	ÿ.ÿ...û<.....rus
74 40 72 75 73 74 2D 4C 54 2D 75 62 75 6E 74 75	t@rust-LT-ubuntu
00 00 01 01 20 18 00 01 00 FF 00 FF 00 FF 10 08ÿ.ÿ.ÿ..
00 FB 3C 01 02 43 00 0E 00 00 00 07 00 00 00 08	.û<..C.....
00 00 00 06 00 00 00 05 00 00 00 04 00 00 00 02
00 00 00 01 00 00 00 00 FF FF FF 10 FF FF FF 11ÿÿÿ.ÿÿÿ.
FF FF FF 18 FF FF FF E6 FF FF FF 20 FF FF FF 21	ÿÿÿ.ÿÿÿæÿÿÿ ÿÿÿ!
03 00 00 00 00 00 05 00 03 20 00 00 FF FF 00 00ÿÿ..
00 00 05 00 00 33 00 00 00 07 90 EF 30 FF D8 FF3.....10ÿÿ

winhex로 recovery를 시도해보았지만 실패하였다.

그래서 그냥 jfif 파일 시그니처를 직접 찾으며 수동 카빙하였다.

수동 카빙을 통해 여러 이미지들이 나왔고, 바탕화면이 15장의 사진으로 나뉘어 표현되어있었다.

다음이 잘린 사진을 이어붙인 바탕화면 사진이다.



밑쪽에 Password_is_VNC5cr33n_is_JPG 라는 글귀가 보이고 인증에 성공했다.

Key : VNC5cr33n_is_JPG

[V] Trivial 300P

Subject

ASCII storm

Comment

By. Rust

File : ASCII_STORM.pdf

이 문제는 어려웠지만 재미있게 푼 문제이다.

주어진 pdf를 열어보면 아무 것도 없고 단지 Watch Number 7 :D 라는 글귀만이 있다.

숨겨진 글자를 찾다가 실패했고, 헥스로 뜯어보던 중 수상한 헤더를 발견하였다.

..endstream.. 앞의 ~>는 base85 암호화이다.

```
3E 3E 0D 0A 73 74 72 65 61 6D 0D 0A 36 3C 23 27 >>..stream..6<#'  
5C 37 50 51 23 46 2B 42 32 71 71 30 65 61 5F 29 \7PQ#F+B2qq0ea_)  
30 48 61 3E 2A 2B 3D 4B 40 24 36 3B 5E 5A 47 33 0Ha>*+=K@$6;^ZG3  
26 3D 66 37 2F 54 4F 2D 70 40 6A 63 2B 4B 3F 56 &=f7/TO~p@jc+K?V  
3C 37 63 30 35 72 3D 4F 39 31 44 3C 63 31 2E 45 <7c05r=O91D<c1.E  
70 37 33 2B 6B 39 2B 31 64 51 31 4F 35 57 56 59 p73+k9+1dQ1O5WVY  
5E 2D 6E 6F 37 2C 46 26 6B 25 46 3B 44 55 66 74 ^~no7,F&k%F;DUft  
41 39 31 25 50 32 30 3A 23 6A 30 2F 2E 21 46 2E A91%P20:#j0/.!F.  
53 29 3F 44 34 46 54 49 43 2B 42 33 28 75 37 38 S)?D4FTIC+B3(u78s  
73 7E 3E 0D 0A 65 6E 64 73 74 72 65 61 6D 0D 0A s~>..endstream..
```

그러나 앞의 <~ 가 없어서 base85가 아니라 여기고 여러 시도를 하던 중 ..stream.. 뒤쪽인 6<#W7PQ#F+B2qq0ea_)0Ha>*+=K@\$6;^ZG3&=f7/TO~p@jc+K?V<7c05r=O91D<c1.Ep73+k9+1dQ1O5WVY^~no7,F&k%F;DUftA91%P20:#j0/.!F.S)?D4FTIC+B3(u78s~> 전체를 base85 디코드 해보았다.

디코드 하니 다음의 값이 나왔다.

BT /F1 8 Tf 10 10 Td (<~BQS?83WN~rAnc'm2_K5b/p(fKFDI2F/n8g:04AsE@:Nt(0fLsV2)R3G1dsAk5t"/0f_*H3(<~>) Tj ET

이번에는 <~ ~> 모두 있어 바로 해독하니 다음의 주소가 나왔다.

<http://cfile7.uf.tistory.com/attach/175B55424E8CADCE19528F>

위의 주소로 들어가 txt 파일 하나를 다운받았다.

열어보니 헥스같이 보이는 값들이 나와있었다.

아스키를 16진수로 바꿀까, 16진수를 아스키로 바꿀까 하다가 그냥 아스키를 16진수로 바꿔서 헥스 에디터에 집어넣기로 했다.

다음이 파이썬으로 코딩한 소스이다.

```
# -*- coding: cp949 -*-
f=file('C:/Users/Sonic/Desktop/45c!!t0l-leX.txt') #파일 열기
sProblem=f.read() #파일에서 전체 읽음
f.close() #파일 닫기
lAnswer=[] #배열 정의
sSplit=sProblem.split() #공백단위로 나눔
for i in range(len(sSplit)): #len함수로 배열의 갯수를 구해 for문을 돌림
    if(len(sSplit[i])==1): #헥스와 아스키를 구분하는것이므로 길이가 1이면 아스키일것이다
        lAnswer.insert(i,hex(ord(sSplit[i]))[2:]) #아스키를 10진수로바꿔 헥스로 바꾼후 슬라
이싱을 통해 3자리부터 끊는다 (0x를 없애기위해)
    else: #만약 헥스라면
        lAnswer.insert(i,sSplit[i]) #그대로 추가
f=file('C:/Users/Sonic/Desktop/Answer.txt','w') #파일 열기
f.write("".join(lAnswer)) #배열 전체 출력
f.close() #파일 닫기
```

위의 소스를 돌려서 헥스 값을 얻었고 헥스 에디터에 넣었다.

파일 시그니처가 43 57 53 이므로 확장자를 swf로 바꿔보았다.

```
43 57 53 08 49 B8 00 00 78 9C ED 7D 07 5C 53 C9 CWS.I,...xœi}.\SÉ
F6 FF A4 41 42 00 91 5E 15 91 AE 40 A8 36 4A 28 öÿ×AB.'^.'@`"6J(
D2 09 08 88 20 08 04 08 10 09 04 43 68 56 76 AD Ò..^ .....ChVv.
6B 17 29 22 2A 58 96 15 7B 05 DB 2A 6B 17 75 5D k.)"*X-.{.Û*k.u]
5D 75 C5 DE 7B 77 ED 85 FF BD 73 6F 92 9B 10 44 ]uÅP{wi...ÿ×so'>.D
```

다음과 같이 정상적으로 재생되었고 인증에 성공했다.



Key : Do_U_L0V3_C0de?

[W] Analysis 300P

Subject

Time is gold

Comment

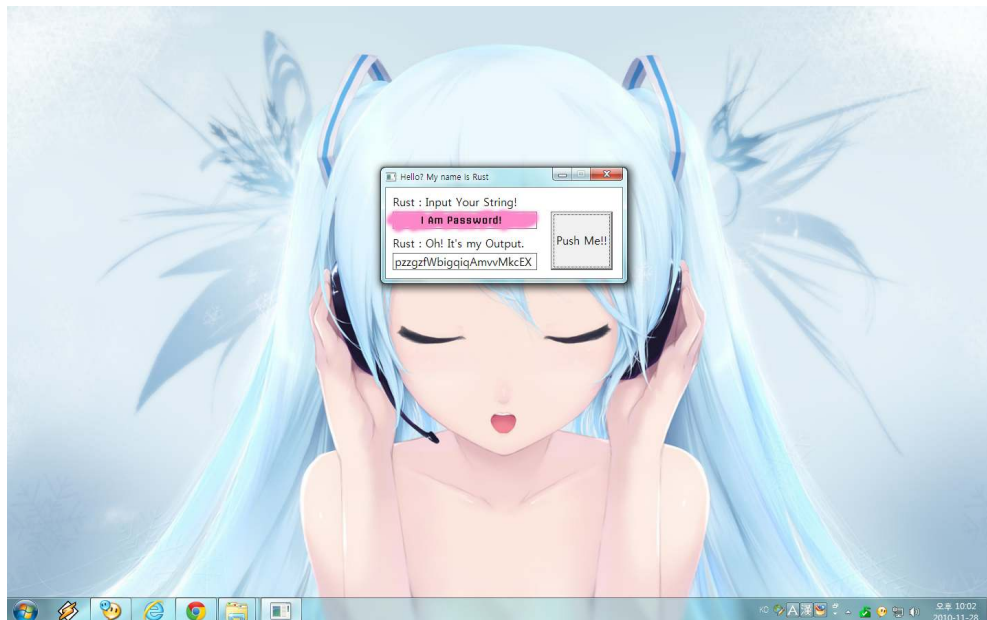
By. Rust

File : ReverseMe.zip

압축을 해제하면 ReverseMe.exe 파일과 함께 다음의 사진 한 장이 주어진다.

여타 리버싱 문제와 같이 문자열을 입력하면 내부의 소스에 따라 처리되어 암호화 된 문장이 나온다.

우리가 할 일은 입력한 문자열을 리버싱을 통해 찾는것이다.



올리디버거로 열어서 소스를 보았다.

GetWindowText 함수로 텍스트를 입력받는다.

그리고 문자열의 길이가 0x1A 이상이면 에러창을 띄운다.

```
00911146 . 6A 1A      PUSH 1A
00911148 . 8D4D E0    LEA ECX,DWORD PTR SS:[EBP-20]
0091114B . 51         PUSH ECX
0091114C . 52         PUSH EDX
0091114D . FF15 FC209101 CALL DWORD PTR DS:[<&USER32.GetWindowTextA
00911153 . 83F8 1A    CMP EAX,1A
00911156 . 7C 29      JL  SHORT ReverseM,00911181
00911158 . A1 78339100 MOV EAX,DWORD PTR DS:[913378]
0091115D . 6A 00      PUSH 0
0091115F . 68 8C219100 PUSH ReverseM,0091218C
00911164 . 68 94219100 PUSH ReverseM,00912194
00911169 . 50         PUSH EAX
0091116A . FF15 00219101 CALL DWORD PTR DS:[<&USER32.MessageBoxA
00911170 . 8B0D 74339101 MOV ECX,DWORD PTR DS:[913374]
00911176 . 68 A8219100 PUSH ReverseM,009121A8
0091117B . 51         PUSH ECX
0091117C . E9 42020000 JMP  ReverseM,009113C3
00911181 . 807D E0 00  CMP BYTE PTR SS:[EBP-20],0
Count = 1A (26,)
Buffer
hWnd => NULL
GetWindowTextA
Style = MB_OK|MB_APPLMODAL
Title = "ERROR"
Text = "Don't Edit This!"
hOwner => NULL
MessageBoxA
ASCII "X_X!"
```

한 글자씩 차례대로 EBX+1 씩 증가시킨다.

EBX 또한 1씩 증가한다.

isapha 함수로 증가시킨 값이 알파벳이 맞는지 확인한다.

올바르지 않다면 원래의 문자에서 EBX-0x19를 증가시킨다.

(BYTE 형식이므로 EBX-0x19의 DWORD 형에서 하위 1BYTE만 사용한다.)

올바르지 않을때 위와 같은 루틴을 거치는 이유는 z 같은 경우 EBX+1 만큼 증가시킬 경우 알파벳이 나오지 않기 때문이다.

009111AE	> 86FF	MOV EDI,EDI	
009111B0	> 0FBE5410 E0	MOVSX EDX,BYTE PTR SS:[EBP+EBX-20]	
009111B5	> 8D441A 01	LEA EAX,DWORD PTR DS:[EDX+EBX+1]	
009111B9	> 50	PUSH EAX	
009111BA	> FF15 6C209101	CALL DWORD PTR DS:[<&MSVCR100,isalpha>]	^C isalpha
009111C0	> 83C4 04	ADD ESP,4	
009111C3	> 85C0	TEST EAX,EAX	
009111C5	> 74 1A	JE SHORT ReverseM,009111E1	
009111C7	> 8D4B 01	LEA ECX,DWORD PTR DS:[EBX+1]	
009111CA	> 004C1D E0	ADD BYTE PTR SS:[EBP+EBX-20],CL	
009111CE	> EB 18	JMP SHORT ReverseM,009111E8	
009111D0	> 8B0D 74339101	MOV ECX,DWORD PTR DS:[913374]	
009111D6	> 68 80219100	PUSH ReverseM,009121B0	ASCII "Use Only Alphabet!"
009111D8	> 51	PUSH ECX	
009111DC	> E9 E2010000	JMP ReverseM,009113C3	
009111E1	> 8D53 E7	LEA EDX,DWORD PTR DS:[EBX-19]	
009111E4	> 00541D E0	ADD BYTE PTR SS:[EBP+EBX-20],DL	
009111E8	> 43	INC EBX	
009111E9	> 8D7C1D E0 00	CMP BYTE PTR SS:[EBP+EBX-20],0	
009111EE	> 75 C0	JNZ SHORT ReverseM,009111B0	

여기서부터 다음 암호화 루틴이 나올 때까지는 현재 시간을 구해서 특정 테이블을 구성한다.

00911226	> 6A 00	PUSH 0	
00911228	> FF15 68209101	CALL DWORD PTR DS:[<&MSVCR100,_time64>]	MSVCR100,_time64
0091122E	> 8945 A0	MOV DWORD PTR SS:[EBP-60],EAX	
00911231	> 8D45 A0	LEA EAX,DWORD PTR SS:[EBP-60]	
00911234	> 50	PUSH EAX	
00911235	> 8955 A4	MOV DWORD PTR SS:[EBP-5C],EDX	
00911238	> FF15 64209101	CALL DWORD PTR DS:[<&MSVCR100,_localtime64>]	MSVCR100,_localtime64
0091123E	> 8BD8	MOV EBX,EAX	
00911240	> 8B73 14	MOV ESI,DWORD PTR DS:[EBX+14]	
00911243	> 81C6 6C070001	ADD ESI,76C	
00911249	> 83C4 08	ADD ESP,8	
0091124C	> 33FF	XOR EDI,EDI	
0091124E	> 8975 A8	MOV DWORD PTR SS:[EBP-58],ESI	
00911251	> 897D AC	MOV DWORD PTR SS:[EBP-54],EDI	
00911254	> DB45 AC	FILD QWORD PTR SS:[EBP-54]	
00911257	> DC2D 30229101	FSUBR QWORD PTR DS:[91223D]	
0091125D	> DD05 28229101	FLD QWORD PTR DS:[91222B]	
00911263	> 09C9	EXCH ST(1)	

다음은 시간 테이블을 짜는 루틴의 최종 부분이다.

0015F788에 생성하므로 Ctrl+G로 0015F788 로 가보았다.

0091132C	> 0FBE0E	MOVSX ECX,BYTE PTR DS:[ESI]	
0091133F	> 034CBD B0	ADD ECX,DWORD PTR SS:[EBP+EDI+4-50]	
00911393	> 51	PUSH ECX	
00911394	> FF15 6C209101	CALL DWORD PTR DS:[<&MSVCR100,isalpha>]	^C isalpha
0091139A	> 83C4 04	ADD ESP,4	
0091139D	> 85C0	TEST EAX,EAX	
0091139F	> 74 08	JE SHORT ReverseM,009113A9	
009113A1	> 8A54BD B0	MOV DL,BYTE PTR SS:[EBP+EDI+4-50]	
009113A5	> 0016	ADD BYTE PTR DS:[ESI],DL	

Stack SS:[0015F788]=00000002
ECX=00000068

그러면 다음과 같은 테이블을 볼 수 있다.

Address	Hex dump	ASCII
0015F788	02 00 00 00 00 00 00 00 01 00 00 00 01 00 00 00F...F...
0015F798	01 00 00 00 00 00 00 00 01 00 00 00 07 00 00 00F...*
0015F7A8	02 00 00 00 01 00 00 00 00 00 00 00 02 00 00 001...

위의 테이블을 4개 단위로 읽어보면 2 0 1 1 1 0 1 7 2 1 0 2 이다.

이 당시의 시간은 다음과 같다.



즉 2 0 1 1 1 0 1 7 2 1 0 2는 2011년 10월 17일 21시 2분이므로 현재의 시간을 테이블로 생성한다는 것을 알 수 있다.

뒤로 가면 다음의 루틴이 보이는데, 이 루틴은 문자열의 뒤쪽부터 구한 테이블의 값 만큼 증가시킨다.

(이 때 테이블은 순차적으로 사용하고, 끝까지 사용하면 처음 값부터 다시 반복한다.)

```

00911370 > 83FF 0C    CMP EDI,0C
00911373 > 75 02      JNZ SHORT ReverseM,00911377
00911375 > 33FF      XOR EDI,EDI
00911377 > 8B44BD B0    MOV EAX,DWORD PTR SS:[EBP+EDI+4-50]
0091137B > 83F8 1A    CMP EAX,1A
0091137E > 7E 0C      JLE SHORT ReverseM,0091138C
00911380 > 83E8 1A    SUB EAX,1A
00911383 > 8944BD B0    MOV DWORD PTR SS:[EBP+EDI+4-50],EAX
00911387 > 83F8 1A    CMP EAX,1A
0091138A > 7F F4      JG SHORT ReverseM,00911380
0091138C > 0FBED0E    MOVSX ECX,BYTE PTR DS:[ESI]
0091138F > 034C8D B0    ADD ECX,DWORD PTR SS:[EBP+EDI+4-50]
00911393 > 51         PUSH ECX
00911394 > FF15 6C20910 CALL DWORD PTR DS:[<MSVCRT00.isalpha>]
0091139A > 83C4 04    ADD ESP,4
0091139D > 85C0      TEST EAX,EAX
0091139F > 74 08      JE SHORT ReverseM,009113A9
009113A1 > 8A54BD B0    MOV DL,BYTE PTR SS:[EBP+EDI+4-50]
009113A5 > 0016      ADD BYTE PTR DS:[ESI],DL
009113A7 > EB 08      JMF SHORT ReverseM,009113B1
009113A9 > 8A44BD B0    MOV AL,BYTE PTR SS:[EBP+EDI+4-50]
009113AD > 2C 1A      SUB AL,1A
009113AF > 0006      ADD BYTE PTR DS:[ESI],AL
009113B1 > 46        INC ESI
009113B2 > 47        INC EDI
009113B3 > 803E 00    CMP BYTE PTR DS:[ESI],0
009113B6 > 75 B8      JNZ SHORT ReverseM,00911370

```

그 때의 시간으로 암호화를 하니 그림의 시간으로 테이블을 구성해야한다.

2010년 11월 28일 22시 2분이므로 테이블을 다음과 같이 구성하였다.

테이블을 201011282202 까지만 하고 반복시켜도 돼지만 귀찮아서 그냥 초기값을 반복시켜 놓았다.

int table[] = {2, 0, 1, 0, 1, 1, 2, 8, 2, 2, 0, 2, 2, 0, 1, 0, 1, 1, 2, 8, 2, 2, 0, 2};

정확한 복호화 소스를 짜려면 중간 중간 isalpha 함수로 검사하여야 하고 여타 귀찮은 점이 많아서 두 가지 방법으로 나눠 코딩하였다.

isalpha를 검사해서 알파벳이 아닌 것은 EBX-0x19를 빼고, 알파벳인 것은 그대로 둘 테니 두 가지로 코딩하여 조합하기로 했다.

다음은 깨진 알파벳을 EBX-0x19 더하여 복호화 시킨 문자를 추출하는 소스이다.

```
#include <stdio.h>
#include <string.h>
#include <windows.h>

int main()
{
    char ss[100] = "pzzgzfWbigqiqAmvvMkcEX";
    char cTemp;
    int table[] = {2, 0, 1, 0, 1, 1, 2, 8, 2, 2, 0, 2, 2, 0, 1, 0, 1, 1, 2, 8, 2, 2, 0, 2};
    int i, j;
    int nLen;
    int nBuf;
    int aa = 1;
    nLen = (int)strlen(ss);
    if (aa == 1) {
        for (i=0; i<nLen; i++) {
            ss[i] = ss[i] - table[i];
        }

        for (i=nLen-1,j=0; i>=nLen/2; i--,j++) {
            cTemp = ss[i];
            ss[i] = ss[j];
            ss[j] = cTemp;
        }

        for (i=0; i<nLen; i++) {
            nBuf = LOBYTE((DWORD)0x19 - (DWORD)i);
            ss[i] = BYTE((BYTE)ss[i] + (BYTE)nBuf);
        }
    }

    printf("%s Wn", ss);

    return 0;
}
```

다음은 깨지지 않은 문자를 추출하는 소스이다.

```
#include <stdio.h>
```

```
#include <string.h>
```

```
#include <windows.h>
```

```
int main()
```

```
{
```

```
    char ss[100] = "pzzgzfWbigqiqAmvvMkcEX";
```

```
    char cTemp;
```

```
    int table[] = {2, 0, 1, 0, 1, 1, 2, 8, 2, 2, 0, 2, 2, 0, 1, 0, 1, 1, 2, 8, 2, 2, 0, 2};
```

```
    int i, j;
```

```
    int nLen;
```

```
    int nBuf;
```

```
    int aa = 1;
```

```
    nLen = (int)strlen(ss);
```

```
    if (aa == 1) {
```

```
        for (i=0; i<nLen; i++) {
```

```
            ss[i] = ss[i] - table[i];
```

```
        }
```

```
        for (i=nLen-1,j=0; i>=nLen/2; i--,j++) {
```

```
            cTemp = ss[i];
```

```
            ss[i] = ss[j];
```

```
            ss[j] = cTemp;
```

```
        }
```

```
        for (i=0; i<nLen; i++) {
```

```
            if (isalpha(ss[i] - i - 1)) {
```

```
                ss[i] = ss[i] - i - 1;
```

```
            } else {
```

```
                nBuf = LOBYTE((DWORD)0x19 - (DWORD)i);
```

```
                ss[i] = BYTE((BYTE)ss[i] + (BYTE)nBuf);
```

```
            }
```

```
        }
```

```
    }
```

```
    printf("%s Wn", ss);
```

```
    return 0;
```

```
}
```

처음의 소스로 코딩을 하면 깨진 문자를 원상복귀 시키므로 다음의 값이 나온다.

o[r0a~R0v0rse_n값00r

두 번째 소스로 코딩을 하면 알파벳인 문자가 나오므로 다음의 값이 나온다.

UAXeGoodReveXYKETgTeeX

두 문자열에서 온전한 것들을 찾아 조합하면 다음의 값이 나온다.

UAreGoodReverseEng?eer

? 부분을 몰라서 바로 인증을 하지 못하여서 브루트 포싱을 통해 한 글자를 찾았다.

암호화 소스로 UAreGoodReverseEngineer 를 암호화 시키니 사진 속의 암호화 된 값이 나왔고, 인증에 성공했다.

Key : UAreGoodReverseEngineer

[Y] Trivial 300P

Subject

천국을 보여줄게

Comment

정말 천사같은 여자야

By. MinAmi33

File : MinAmi33.zip

압축을 해제하면 다음의 이미지 하나가 나타난다.



헥스에디터로 열어 보던 도중 안에 압축파일이 숨어있는 것을 발견했다.

```
4D C0 5C 18 D0 BF FF D9 50 4B 03 04 14 00 00 00 MA\.\.yÜPK....  
08 00 D1 92 43 3F 20 14 B3 0A 50 04 00 00 8C 06 ..Ñ'C? .'.P...E.
```

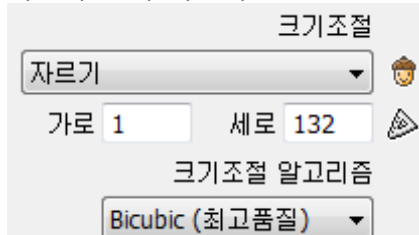
압축을 풀니 컬러 바코드 같아 보이는 이미지들이 132개가 나타났고, 크기 또한 132x132였다.

컬러 바코드를 검색하여 계속 찾았으나 실패하였다.

옆으로도 이어보고 흑백으로 바꿔 겹쳐도 보다가 1x132로 이미지를 잘라 옆으로 이어붙여서 132x132의 이미지를 새롭게 만들어보았다.

일괄 편집을 하기 위해 포토스케이프 툴을 이용했다.

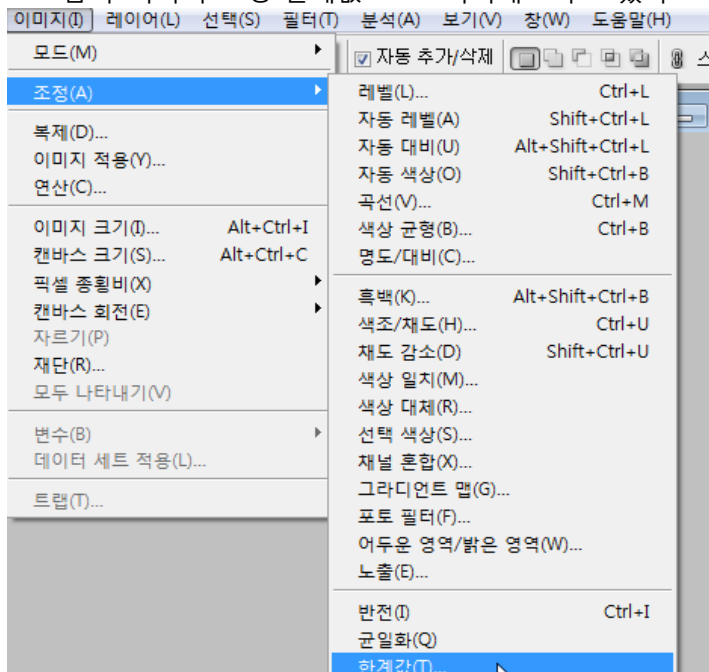
다음과 같이 자르기로 1x132를 지정하여 잘랐다.



그리고 이어붙이기를 하니 다음과 같은 이미지가 나왔고 QR코드라는 것을 알 수 있었다.



포토샵의 이미지-조정-한계값 으로 처리해보기로 했다.



한계값을 223 정도로 지정해서 QR코드가 인식 될 정도로 만들었다.



그렇게 만든 다음의 QR코드를 아이폰의 QR스캐너 어플로 스캔했다.



QR코드를 찍자 아래의 주소가 나왔다.

<http://m.site.naver.com/01Yzy>

위의 주소로 들어가자 다음의 글귀가 있었고 인증에 성공했다.

Key is : C_ProgramFiles_Pruna_Incoming_EBS

Key : C_ProgramFiles_Pruna_Incoming_EBS

후기

6위를 하여 입상을 하게 되었다.

여러 분야들의 문제가 나왔고 정말 재미있게 풀었다.

특히 네이투 암호화 문제는 최근 네이트 해킹 유출 이슈를 다뤄서 재미있었다.

시스템 분야가 매우 약하다는 사실을 알게 되었고, 앞으로는 더욱 열심히 공부해야겠다고 느꼈다.

다음은 대회 종료 당시의 랭킹이다.

2200 점으로 6등을 하였다.

Rank	Nick Name	Challenge Point	Last Auth Time
1	인간남케홀마	5400	[16] 04:26:54
2	pwn3r	3140	[16] 10:00:57
3	SecuRex0	2930	[16] 09:24:27
4	LulzSec	2810	[16] 03:42:19
5	extr	2200	[16] 09:41:27
6	Xero	2200	[16] 09:44:48
7	Hello	2100	[16] 07:41:52
8	두루몽솔	2100	[16] 10:30:04
9	nagi	1910	[16] 05:07:46
10	fuck	1700	[15] 23:48:35
11	Gogil	1510	[16] 04:42:44
12	pepper	1400	[16] 03:06:47
13	B10SM4N	1200	[16] 10:22:32
14	나는_자연인이다	700	[16] 01:37:34
15	ffaass	500	[16] 01:42:00
16	NellP	400	[16] 07:23:08
17	SecurityFirst	300	[15] 15:42:36
18	Loup_	20	[15] 09:24:29
19	freedom	20	[15] 12:23:29

다음은 챌린지 보드판이다.

다음번에는 더욱 많은 문제를 풀어서 보드판을 파란색으로 가득 채우고 싶다.

[A] Web - 100 P OTHER LulzSec	[B] Crypto - 100 P OTHER nagi	[C] Trivial - 100 P CLEAR 인간남케총마	[D] Trivial - 100 P CLEAR SecuRex0	[E] Analysis - 100 P CLEAR Gogit
[F] Crypto - 200 P CLEAR SecuRex0	[G] Forensic - 200 P CLEAR 인간남케총마	[H] Trivial - 400 P H	—	[J] Crypto - 300 P OTHER 인간남케총마
[K] Analysis - 300 P OTHER 인간남케총마	[L] Forensic - 300 P OTHER 인간남케총마	—	[N] Forensic - 200 P CLEAR 인간남케총마	[O] Forensic - 300 P OTHER 인간남케총마
—	[Q] Analysis - 400 P Q	[R] Trivial - 200 P CLEAR 인간남케총마	[S] Web - 300 P OTHER 인간남케총마	[T] Analysis - 200 P CLEAR 인간남케총마
[U] Forensic - 300 P OTHER 인간남케총마	[V] Trivial - 300 P CLEAR 인간남케총마	[W] Analysis - 300 P CLEAR 인간남케총마	—	[Y] Trivial - 300 P CLEAR 인간남케총마
[Z] System - 400 P OTHER 인간남케총마	—	[Alt] Analysis - 400 P Alt	[Del] Analysis - 500 P Del	[Win] System - 400 P OTHER Anonymous