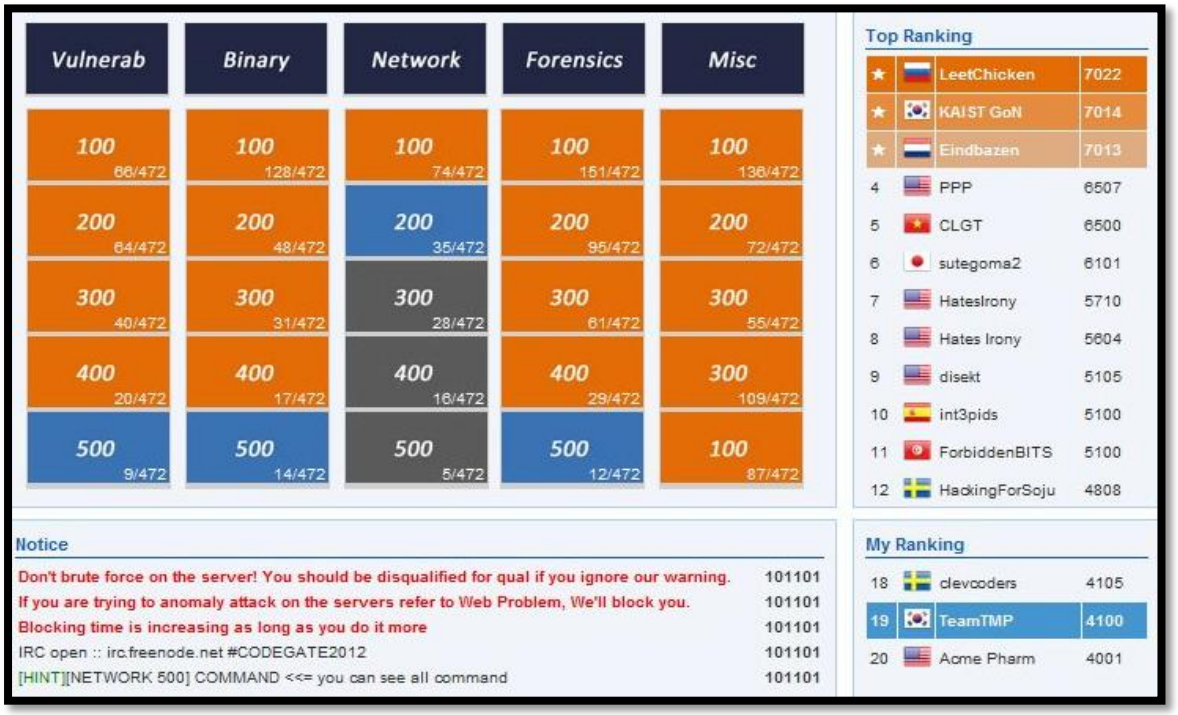
CODEGATE 2012 YUT Quals Write-up

Written by TeamTMP(19th)

2012. 03. 07

Result



Vulnerab	Binary	Network	Forensics	Misc
100 66/472	100 128/472	100 74/472	100 151/472	100 136/472
200 64/472	200 48/472	200 35/472	200 95/472	200 72/472
300 40/472	300 31/472	300 28/472	300 61/472	300 55/472
400 20/472	400 17/472	400 16/472	400 29/472	300 109/472
500 9/472	500 14/472	500 5/472	500 12/472	100 87/472

Top Ranking

★		LeetChicken	7022
★		KAIST GoN	7014
★		Eindbazen	7013
4		PPP	6507
5		CLGT	6500
6		sutegoma2	6101
7		HatesIrony	5710
8		Hates Irony	5604
9		disekt	5105
10		int3pids	5100
11		ForbiddenBITS	5100
12		HackingForSoju	4808

Notice

Don't brute force on the server! You should be disqualified for qual if you ignore our warning.	101101
If you are trying to anomaly attack on the servers refer to Web Problem, We'll block you.	101101
Blocking time is increasing as long as you do it more	101101
IRC open :: irc.freenode.net #CODEGATE2012	101101
[HINT][NETWORK 500] COMMAND <<= you can see all command	101101

My Ranking

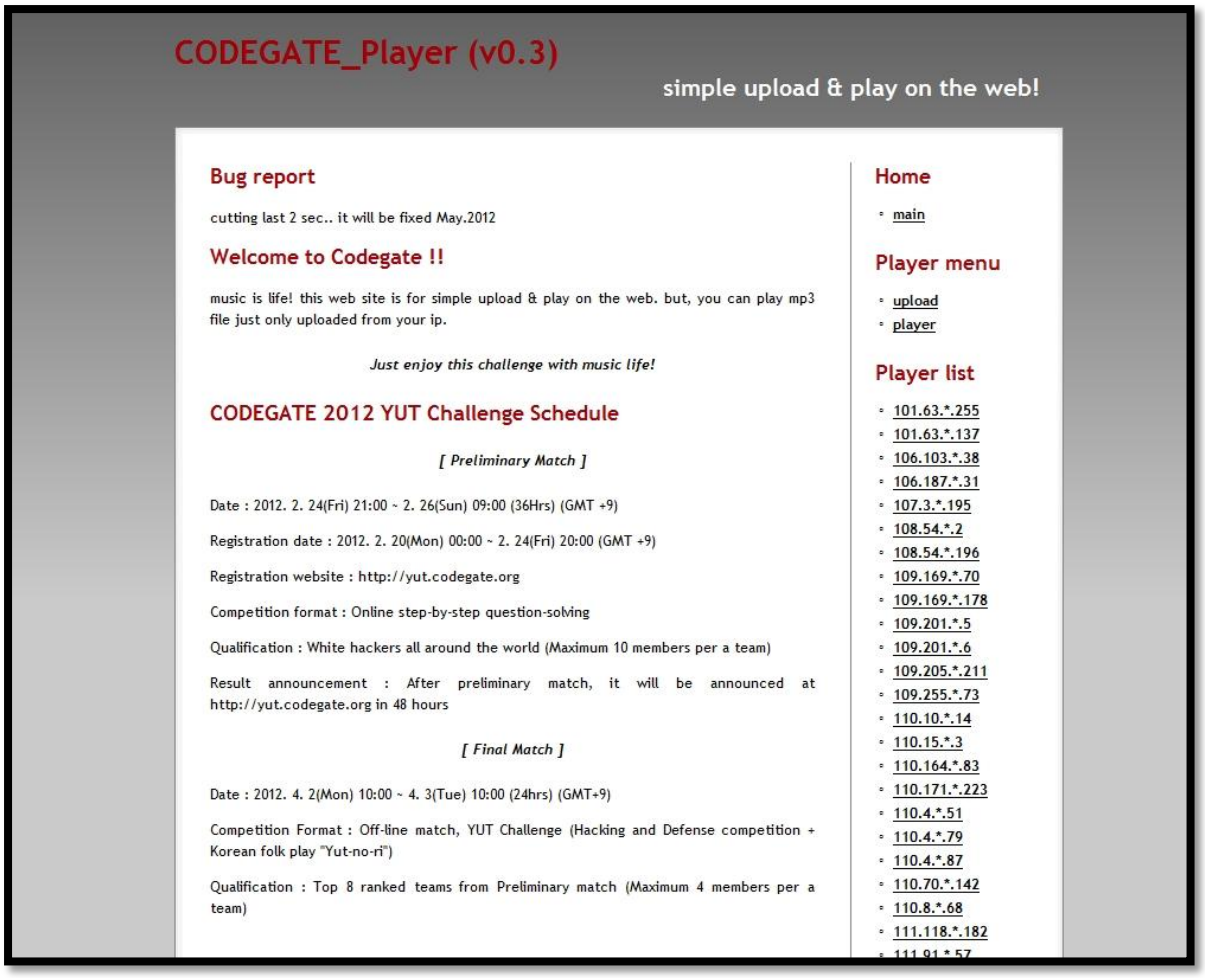
18		clevcoders	4105
19		TeamTMP	4100
20		Acme Pharm	4001

★		LeetChicken	7022
★		KAIST GoN	7014
★		Eindbazen	7013
4		PPP	6507
5		CLGT	6500
6		sutegoma2	6101
7		HatesIrony	5710
8		Pwning-Yeti (Hates Irony)	5604
9		disekt	5105
10		int3pids	5100
11		ForbiddenBITS	5100
12		HackingForSoju	4808
13		machoman	4802
14		LSE	4701
15		BlackDragon	4700
16		ㅇㅇㅇㅓㅊㅊ	4615
17		C.o.P	4406
18		clevcoders	4105
19		TeamTMP	4100

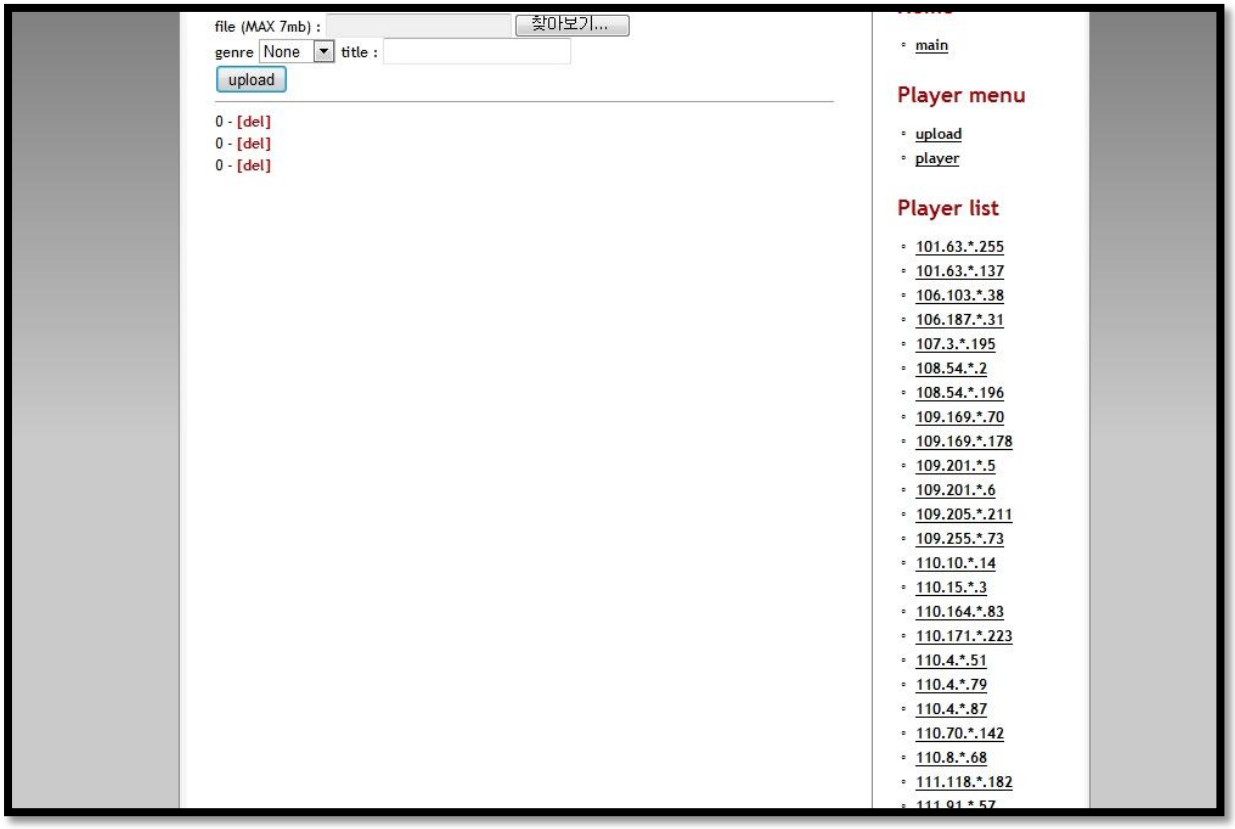
Table of Contents

1. Vuln 100 [Solver - Hexinic] 4P
2. Vuln 200 [Solver – Hexinic & Electrop] 6P
3. Vuln 300 [Solver - Sangwon] 8P
4. Vuln 400 [Solver - Hexinic] 9P
5. Binary 100 [Solver - Gogil] 11P
6. Binary 200 [Solver - Gogil] 14P
7. Binary 300 [Solver – Gogil & Sangwon] 17P
8. Binary 400 [Solver - Gogil] 21P
9. Network 100 [Solver - UknowY] 23P
10. Forensics 100 [Solver – Rav3n] 27P
11. Forensics 200 [Solver - extr] 29P
12. Forensics 300 [Solver – extr] 30P
13. Forensics 400 [Solver - extr] 31P
14. Misc 100 [Solver – Electrop] 33P
15. Misc 200 [Solver – m4st3rm1nd] 34P
16. Misc 300 [Solver – JNVB] 36P
17. Misc 300 [Solver – Hexinic & JNVB] 37P
18. Misc 100 [Solver – JNVB] 39P

Vulnerab 100 [Solver – Hexinic]



Codegate Player가 있습니다 메뉴 바에 업로드 부분을 살펴보면



보시다시피 mp3 파일 업로드 가능한 필드가 잇습니다

한번 업로드를 해서 Proxy 툴로 잡아서 SQL Injection 할 부분을 보니 Genre 필드에 취약한 부분을 알아냈습니다. [POC : (select 1)]

이렇게 하여 대충 쿼리문을 짐작하여 insert into (file,genre,title) values ('file내용',genre,'title 내용'); 이라고 한 후에

블라인드 SQL Injection이나 Insert 조작으로 테이블과 컬럼을 이용해서

genre 에 1,1),((select file from upload_mp3_127_0_0_1),1,0x494d494e544845534f4c5645)#

이런 식으로 쿼리 를 넣어주면 admin이 현재 듣는 음악을 들을 수 있습니다!

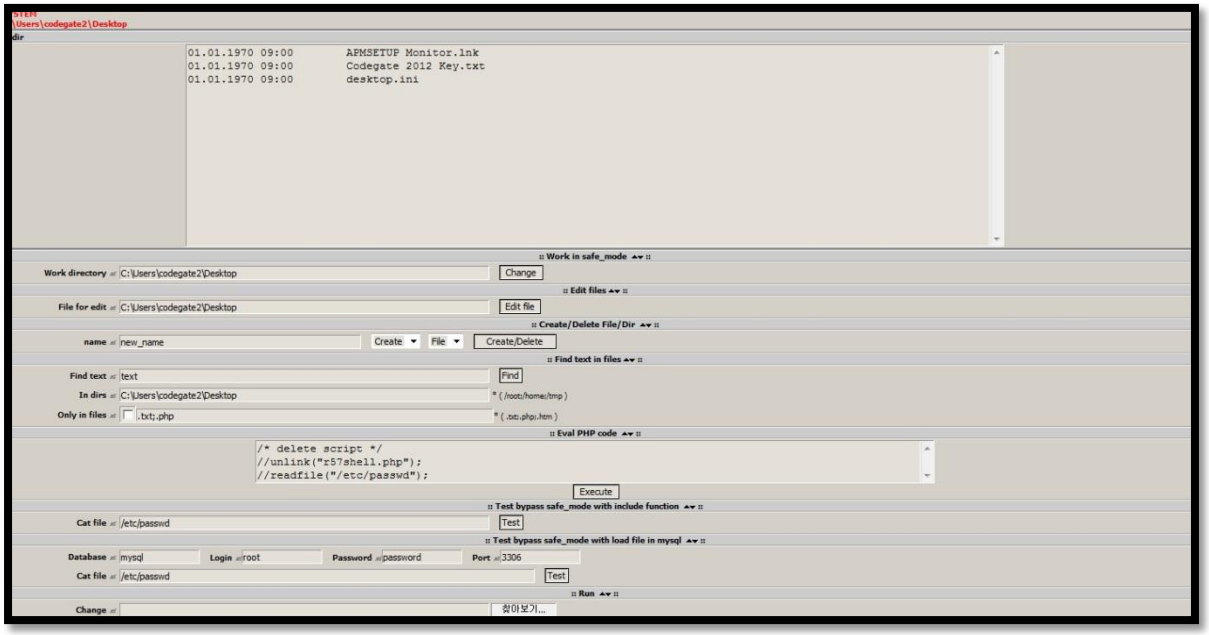
KEY : UPL04D4NDP14Y

Vulnerab 200 [Solver – Hexinic & Electrop]

이미지 파일을 Uploading 할 수 있는 페이지가 있습니다.

파일명을 jpg 이외 값을 가지게 되면 올릴 수 있게 하는 취약점을 이용해서 .jpg.php로 확장자를 변신시켜서 쉘을 올렸습니다.

쉘을 올린 후 그 페이지를 들어가서 디렉터리 있는 것을 뒤져 봅니다.



디렉터리를 뒤져보니 codegate2 라는 계정 바탕화면에 키 파일이 있는 것을 읽으면 답일 것 같습니다. [C:₩Users₩codegate2₩Desktop₩Codegate 2012 Key.txt]

먼저 C 드라이브에 APM_Setup이 있는 것을 보고 초기 페이지라는 허점을 이용해

MYSQL 비밀번호를 따서 쿼리문 select load_file("C:₩₩Users₩codegate2₩Desktop₩Codegate 2012 Key.txt"); 을 주면

<? /* Good Job ! Key is 16b7a4c5162d4dee6a0a6286cd475dfb */ ?> 이라 뜹니다.

KEY : 16b7a4c5162d4dee6a0a6286cd475dfb

Vulnerab 300 [Solver - Sangwon]

공개 보류

Vulnerab 400 [Solver - Hexinic]

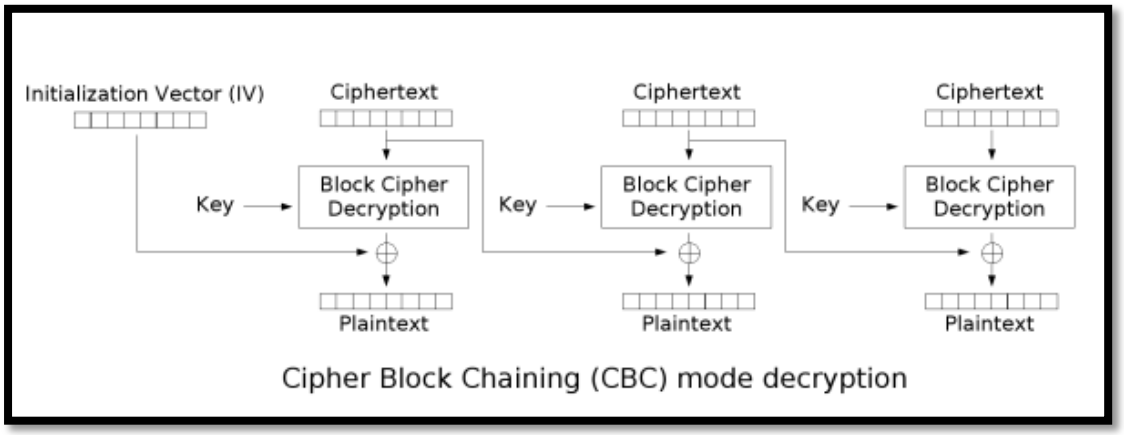
홈페이지를 뒤져보니 ctf 파일을 하나 줍니다

그파일을 읽어보니

gAI7UDFFYhs=LUFaO2IncC0=

이라는 값을 보게 됩니다.블록 사이즈가 8바이트로 base64 인코딩된 두 개의 값을 조작 해가면서 에러를 만들어보면 Invalid IV , Invalid Padding ,Invalid Class 이라는 것을 보고

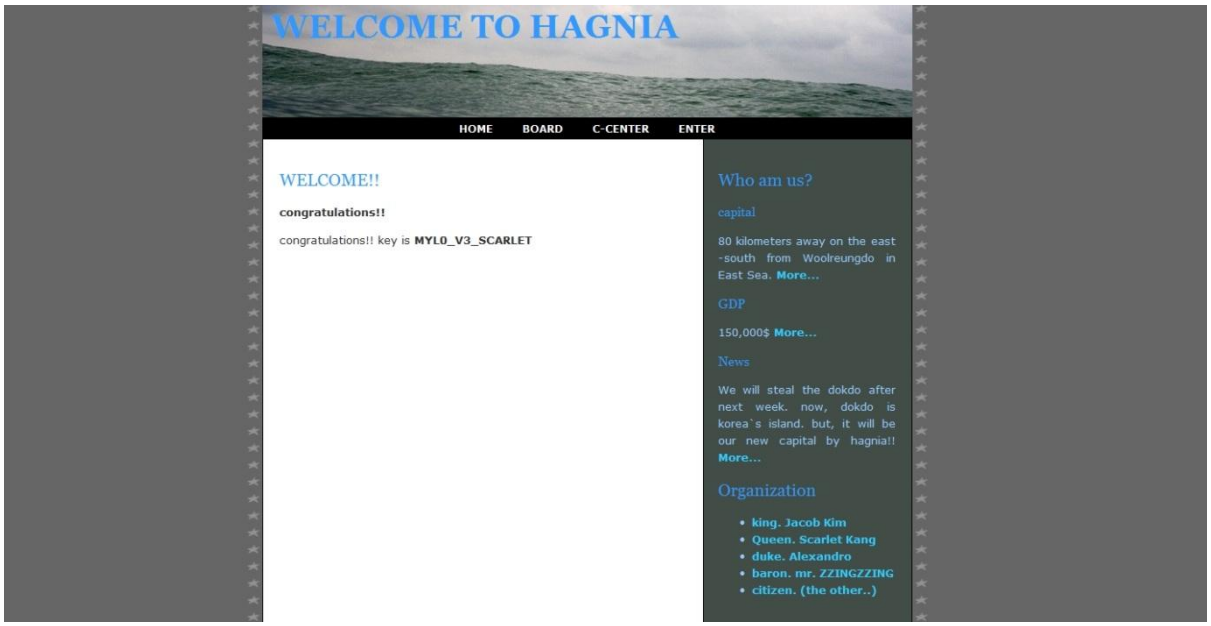
Intermediary Value 과 CBC 모드 에 citizen ,king 등 클래스가 있을 것이라 가정합니다



Block 암호화 방식을 사용하기 때문에 가장 마지막 블록부터 시작 합니다.

그러므로 플레인 값을 nezitic 이라 하고 gnik으로 8바이트 채운 값과 Intermediary Value 값은 일정하게 gAI7UDFFYhs= 을 넣어주고

hex화 한 후 xor 한 후 base64 인코딩 해준 후 LUFaO2IncC0= 을 chipher로 설정하여 계산하고 파일 업로딩 하면 king 으로 로그인이 됩니다.



보드 부분을 보면 키 값이 나와 있습니다.

KEY : MYL0_V3_SCARLET

Binary 100 [Solver - Gogil]

첨부파일 압축을 풀면 bin100.ex_ 파일이 있다.
실행 파일이므로 디버깅해보면 sxeF47B.tmp 같은 랜덤한 파일을 생성하고 실행하는 것을 알 수 있다.
bin100.exe 에서 생성한 sxeF47B.tmp 파일은 윈도우 PE 실행파일이므로 이 파일을 분석하면 된다.

PEiD 로 확인해본 결과 C# 으로 작성되어 있으므로 .Net Reflector 로 디컴파일한다.

먼저 프로그램 시작점 Program() 함수에서는 str_cypher 에 BM3aZTvv5iQAhK95EFLuz4pta 를 넣는다.
Main() 함수를 살펴보면 처음 보는 함수들로 가득하다.
필자의 시스템에서는 파일이 오류가 나며 정상작동하지 않아 테스트 할 수 없었다.

아마 음성을 출력하는 부분으로 생각된다.
Main() 에서 핵심은 recognizer_SpeechRecognized 함수를 Event Handler 로 등록하는 부분이다.

recognizer_SpeechRecognized() 함수를 살펴보면

```
private static void recognizer_SpeechRecognized(object sender, SpeechRecognizedEventArgs e)
{
    if ((e.get_Result().get_Text() == "Nothing") && (current == RecognitionState.None))
    {
        current = RecognitionState.Question;
        SpeechSynthesizer synthesizer = new SpeechSynthesizer();
        synthesizer.SetOutputToDefaultAudioDevice();
        synthesizer.Speak("Are you sure?");
        str_input = e.get_Result().get_Text();
    }
    else if (current == RecognitionState.Question)
    {
        current = RecognitionState.None;
        if (e.get_Result().get_Text() == "Yes")
        {
            str_cypher = str_cypher + "QA1f1/EqAOZktIz1RrwMPunDlqwww==";
            DoHibernation();
        }
    }
}
```

e.get_Result().get_Text() 로 가져온 값이 "Nothing" 인 경우 음성 출력을 하고 str_input 에 Nothing 을 넣는다.

또는 e.get_Result().get_Text() 가 "Yes" 이고 이벤트 핸들러의 모드가 일치하면

str_cypher 에 QA1f1/EqAOZktIz1RrwMPunDlqwww== 를 덧붙이고 DoHibernation() 함수를 호출한다.

```
private static void DoHibernation()
{
    DateTime now = DateTime.Now;
    if (now.Hour == 8)
    {
        str_input = str_input + "!";
        if (str_input.Length == now.Hour)
        {
            WATCrypt crypt = new WATCrypt(str_input);
            Console.WriteLine(crypt.Decrypt(str_cypher));
        }
    }
    else
    {
        Console.WriteLine("This isn't the time yet.");
    }
}
```

DoHibernation() 함수는 str_input 과 str_cypher 를 이용해 답을 출력한다.
str_input 은 "Nothing" 에서 "!" 를 붙이므로 Nothing! 이 된다.

str_cypher 는 BM3aZTvv5iQAhK95EFLuz4ptaQA1f1/EqAOZktIz1RrwMPunDlqwww== 인 상태에서 Decrypt 로 넘겨진다.

끝으로 WATCrypt 클래스의 생성자와 Decrypt() 함수만 분석하면 된다.
WATCrypt 생성자에서는 인자로 받은 문자열을 SKey 변수에 바이트 배열로 변환하여 저장한다.

Decrypt() 함수는 인자로 받은 문자열을 Base64 로 디코딩하고 SKey 로 DES 복호화를 진행한다.

아래와 같이 코딩하여 실행하면 답이 출력된다.

```
static void Main(string[] args)
{
      DESCryptoServiceProvider provider = new DESCryptoServiceProvider();
      byte[] Skey = new byte[8];
      Skey = Encoding.ASCII.GetBytes("Nothing!");

      provider.Key = Skey;
      provider.IV = Skey;
      MemoryStream stream = new MemoryStream();
      CryptoStream stream2 =
new CryptoStream(stream, provider.CreateDecryptor(), CryptoStreamMode.Write);

      byte[] buffer =
Convert.FromBase64String("BM3aZTvv5iQAhK95EFLuz4ptaQA1f1/EqAOZktIz1RrwMPunDlqwww==");
      stream2.Write(buffer, 0, buffer.Length);
      stream2.FlushFinalBlock();
      Console.WriteLine(Encoding.UTF8.GetString(stream.GetBuffer()));
}
```

KEY : Nuno! Congratulations on your wedding!

Binary 200 [Solver - Gogil]

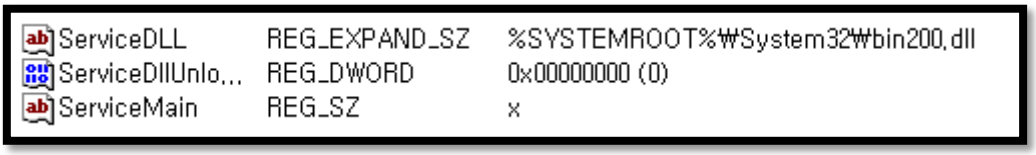
첨부파일은 DLL 이므로 올리디버거로 디버깅한다.

문제를 풀기 전에 주의할 점이 그냥 실행할 경우 윈도우가 맛가버리므로 가상컴퓨터 Windows XP 에서 분석하도록 한다.

지금 필자의 데탑은 부팅도 안돼서 윈도우 복원중이다 --ㅋ

살펴보면 CreateService 등의 함수로 서비스에 bin200 이라는 이름으로 등록한다.

레지스트리 HKEY_LOCAL_MACHINE₩SYSTEM₩CurrentControlSet₩Services₩bin200₩Parameters 를 보면



system32 폴더에 존재하는 bin200.dll 의 x 함수가 서비스 메인이라는 것을 알 수 있다.

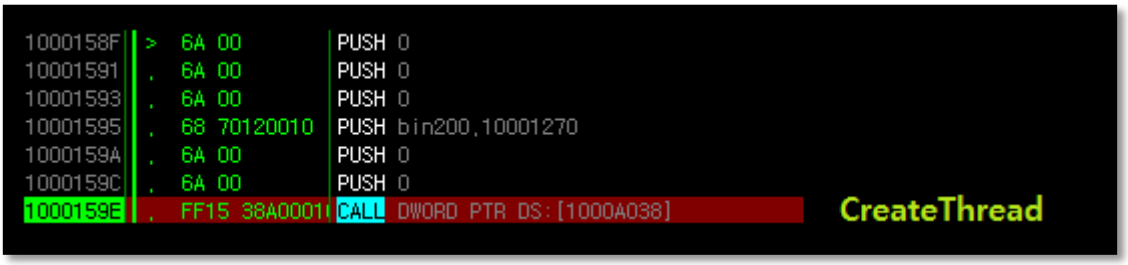
100014E0 이곳이 Export 된 X 함수이므로 이곳을 분석하면 된다.

DLL 을 system32 폴더로 이동시키고, x 함수가 호출되는 시점을 잡기 어려우므로 EIP 를 강제 이동시켜 분석하겠다.

100014E0 x 함수에서는 RegisterServiceCtrlHandler(), SetServicesStatus() 등을 호출하는데 강제 이동했으므로 실패한다.

함수 호출이 성공한것처럼 수정하여 계속 진행하겠다.

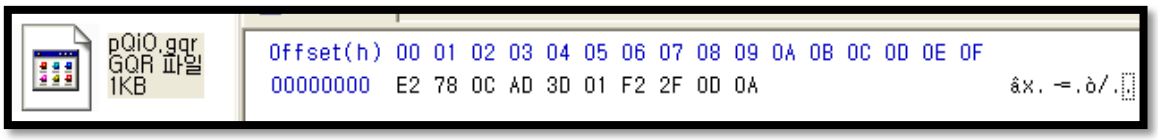
바로 아래에 CreateThread 를 호출한다. ThreadFunction 은 10001270 이다.



그리고 GetTempPath 로 Temp 폴더 경로를 얻고, 랜덤한 xxxx.xxx 문자열을 만들어서 파일명으로 사용한다.

몇몇 알 수 없는 함수를 거치고 ZwQueryInformationProcess 로 안티디버깅 확인 후

위에서 만든 Temp 폴더의 랜덤한 파일에 이상한 값을 1 초 간격으로 채운다.



그런데 몇 번 테스트를 해보니 파일의 내용이 가끔가다 달라진다.

저 내용이 어떻게 만들어 지는지 추적을 해봐야 한다.

WriteFile 을 할 때 Hardware Break 포인트를 걸어보면 10001B40 과 10001BC0 함수에서 저 값들을 수 많은 연산을 거쳐 만든다는 것을 알 수 있다.

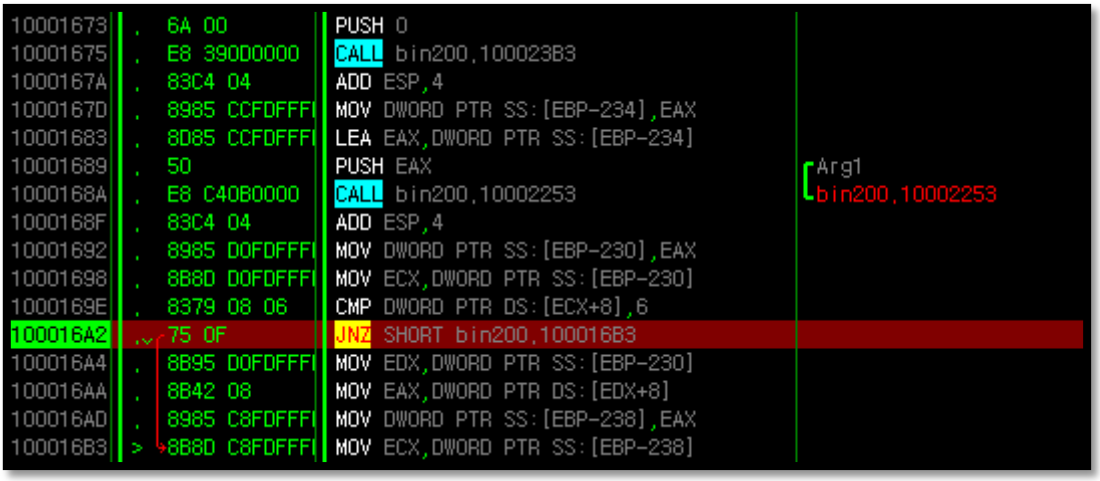
10001B40 함수에서 1000B440 영역에서 값을 가져와서 연산하고 있다.

1000B440 ~ 1000B44F 를 참조한 영역들을 확인해보면 어떠한 조건에 충족하는 경우 1Byte 씩 값을 채우고 있다.

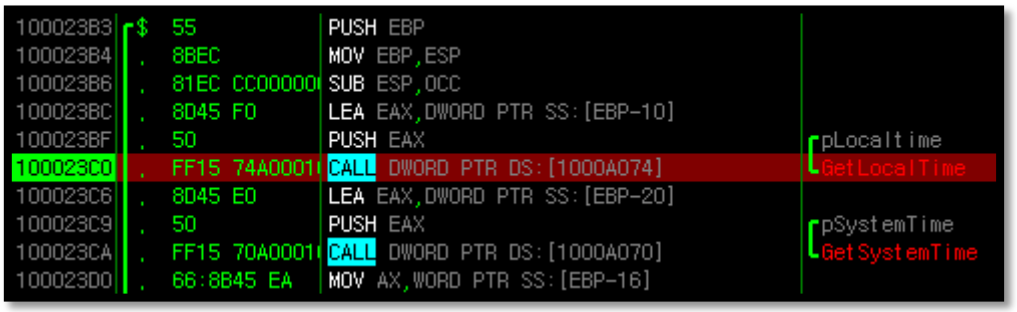
1000B440 -> 100010D0 함수의 반환값 : 0x1E
1000B441 -> LoadLibrary("ntdll.dll")이 성공한 경우 : 0xA0
1000B442 -> GetProcAddress("NtQuery...")이 성공한 경우 : 0xF5
.....
1000B44F -> CreateThread(.. 10001270 ..)이 성공한 경우 : 0xDE

등등 하나 하나 맞춰가면 된다.
그리고 한 가지 더, 시간을 체크하는 부분이 존재한다.

x 함수에서 파일에 내용을 쓰기 전에 무엇인가와 6 을 비교한다.



따라가보면 100023B3 에서 현재 시간을 구하는 것을 알 수 있다.



1000B440 의 배열과 시간을 6 시로 맞추면 답이 출력된다.

KEY : &I%W=K)l

Binary 300 [Solver – Gogil & Sangwon]

첨부파일 압축을 풀면 zombie.exe 와 dRcw.ziq 파일이 주어진다.

UPack 으로 패킹되어 있으므로 OEP 004023DF 를 구하고 언패킹을 한다.

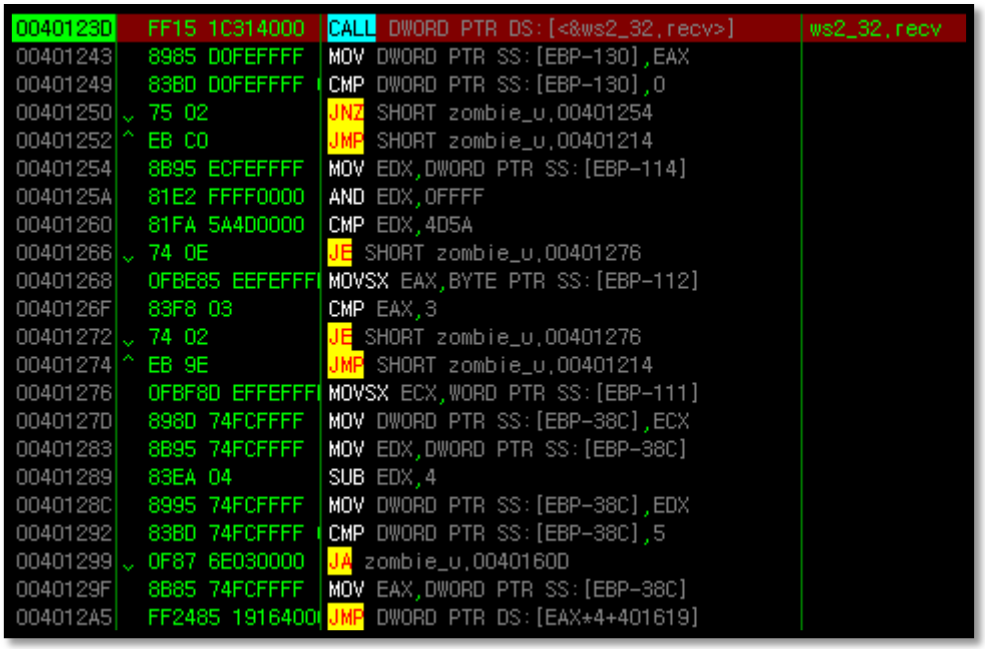
00401000 메인에서 Command Argc 가 2 개인가 확인한다.

커맨드 인자들은 inet_addr, ntohs 에서 사용하는것으로 봐서 이 둘은 IP 와 Port 이다.

zombie.exe 는 전달된 IP 와 Port 로 접속하여 0x12345678 을 send 하고,

recv 값이 0x12345679 가 아니라면 프로그램이 종료된다.

그리고 300 밀리초마다 recv 로 데이터를 받아와서 데이터를 확인한다.



00401260 - 데이터의 앞 2Byte 는 0x5A 0x4D 이여야 한다.
0040126F - 데이터의 3 번 째는 0x03 이여야 한다.
00401292 - 데이터의 4 번 째는 0x00 ~ 0x09 이어야 작아야 한다.

위의 조건들을 만족하면 데이터의 4 번째 값을 가지고 switch 한다.

각각 분석을 해보면 아래와 같이 명령을 내릴 때 사용된다는 것을 알 수 있다.

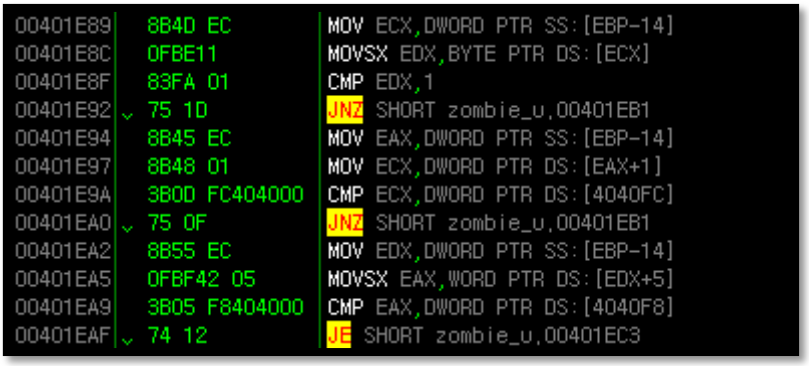
0x5A 0x4D 0x03 0x05 - Mac Address 를 전송
0x5A 0x4D 0x03 0x06 - 레지스트리 SOFTWARE₩...CurrentVersion₩₩Uninstall 를 읽어서 모든 값 출력
0x5A 0x4D 0x03 0x07 - Temp 폴더에 랜덤한 이름의 파일을 생성하고 받아온 데이터로 채워 넣음
0x5A 0x4D 0x03 0x08 - 생성한 파일을 읽어 들여서 DDOS 공격
0x5A 0x4D 0x03 0x09 - 자기 자신을 삭제하는 bat 파일을 만들고 종료

문제에서 dRcw.ziq 파일이 주어졌으므로 0x08 명령을 내려 파일을 읽도록 하면 된다.

0x08 명령어를 받으면 00401DE0 함수를 호출한다.

0x07 명령을 내린적 없어 파일 이름이 비어있으므로 dRcw.ziq 를 넣는다.

fread 로 파일 전체를 읽어들여서 파일 내용을 비교한다.



00401E8F - 데이터의 1 번째 바이트는 0x01 이여야 한다.
00401E9A - 데이터의 2~4 번째 4Byte 는 커맨드로 받은 IP 와 일치해야 한다.
00401EA9 - 데이터의 5~6 번째 2Byte 는 커맨드로 받은 Port 와 일치해야 한다.

dRcw.ziq 파일에는 명령을 내리는 컴퓨터의 IP 와 Port 가 적혀있는데 커맨드 인자로 받은 값과 일치하는가 확인한다.

IP 는 166.253.42.74 이고 Port 는 1793 이다.

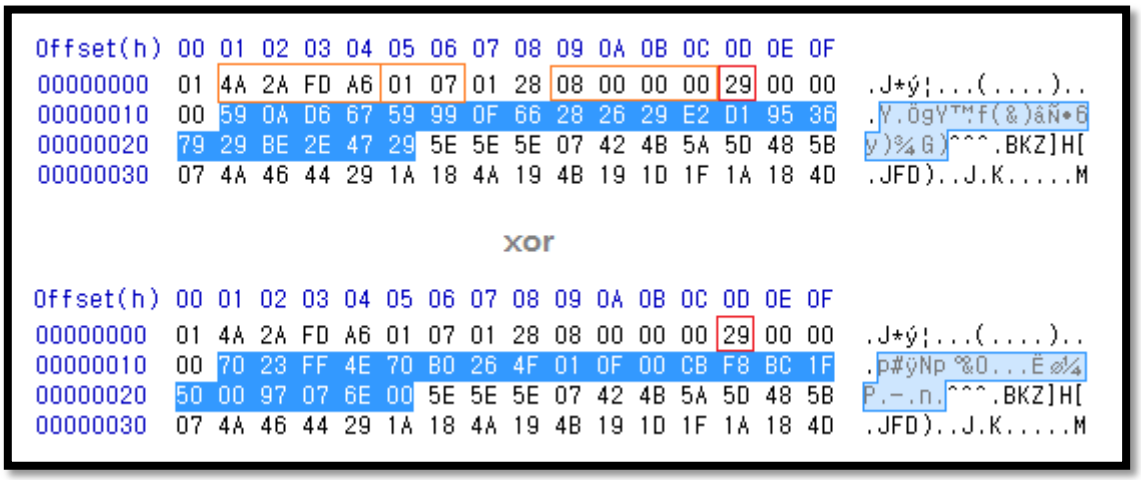
IP 를 필자는 자기 자신에게 테스트 중이므로 비교를 무시했다.

그리고 파일 데이터의 9 번째 값 만큼 Loop 를 돈다.

루프 내부에 00401F28 에서 004022B0 함수를 호출하는데, 004022B0 함수가 파일의 내용을 복호화한다.

이 함수는 3 개의 인자를 취한다.

첫 번째 인자가 복호화를 진행할 메모리 주소, 두 번째 인자는 복호화할 크기, 세 번째 인자는 xor 에 사용되는 값이다.



위와 같은 xor decrypt 를 8 번 반복하게 된다.

그리고 ThreadFunction 을 00401B20 으로 CreateThread 를 생성한다.

00401B20 을 살펴보면 아까와 마찬가지로 복호화를 또 진행하고

COleDateTime 클래스의 함수들을 이용해 현재 시간과 위에서 복호화한 부분을 비교한다.

그런데 Thread 호출 전에 복호화를 이미 했는데, 또 xor 하므로 암호화가 된다.

그래서 Thread 함수 내부에 있는 복호화 부분은 호출하지 않도록 하겠다.

시간 비교를 무시하도록 수정하고 계속 진행해보면 또 다시 복호화 부분이 있다.

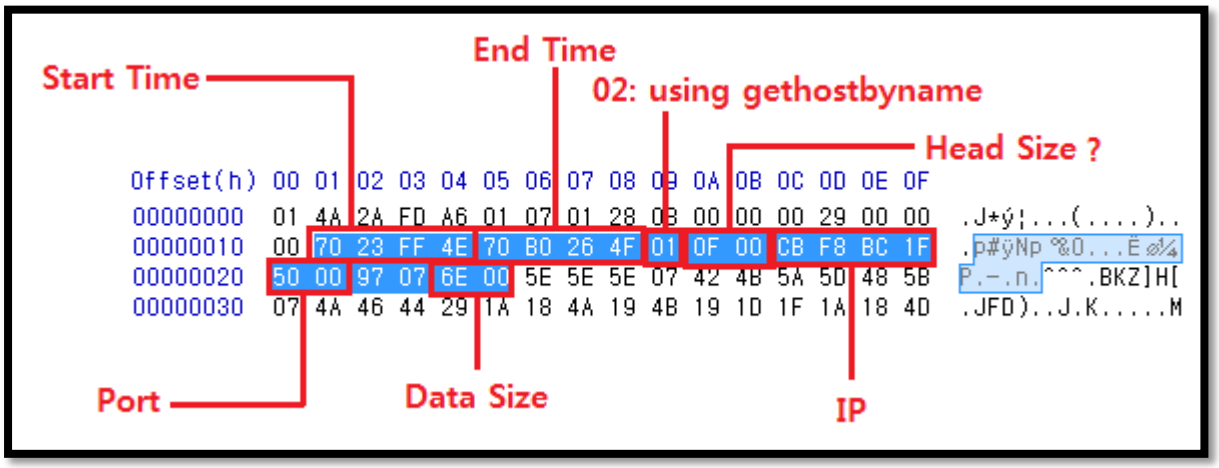
이번에는 아까와 같은 부분이 아니라 그 다음 데이터 영역을 복호화 한다.

www.kbstar.com.31c0b04631db31c9cd80e ... 이러한 내용이 나온다.

그리고 문제에서 요구한 Port 를 사용하는 부분이 있다.

```
00401C7C   8B4D E4          MOV ECX,DWORD PTR SS:[EBP-1C]
00401C7F   66:8B51 13       MOV DX,WORD PTR DS:[ECX+13]
00401C83   52               PUSH EDX
00401C84   FF15 08314000    CALL DWORD PTR DS:[<&ws2_32.htons>]   ws2_32.ntohs
```

저 부분을 살펴보면 위에서 맨 처음에 복호화한 곳의 16 번째 데이트를 가져와서 사용하는 것을 알 수 있다.
여러 부분에서 분석을 해보면 저 헤더가 사용되는 곳을 파악할 수 있다.



위와 같이 구성되어 있으며 이렇게 8개가 반복적으로 있다.
사용되는 Port를 모두 모아보면 0050, 01BB, 0015, 0017, 0037, 006E, 0D3D, 05F1 이고 모두 더하면 10진수로 5642이다.
문제에서 요구했던 포트 번호는 모두 구했고, 공격 횟수만 구하면 되겠다.
Unix Time 0x4EFF2370 ~ 0x4F26B070 을 변환하면 2012-01-01 ~ 2012-01-31 이 된다.

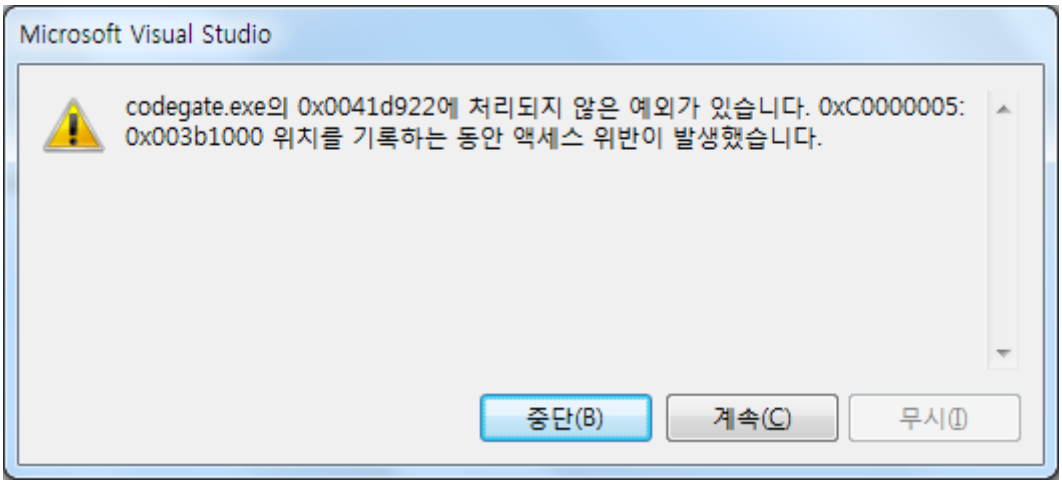
총 2592000초 동안 공격을 하게 되고, 공격 간격은 10초 이므로 259200번 공격한다고 여겼다.
나머지 영역도 모두 구하면 259200*2+241920*6 이 공격 횟수인줄 알았지만, 답 인증이 되지 않았다.
여러가지 작업을 해보다가 팀원분이 인증에 성공했는데, 공격 횟수가 아닌 공격한 사이트 개수였다.
attack_count -> attack_site_count ...

총 8개의 사이트를 공격하게 되므로 답은 5642*8 = 45136 이다.

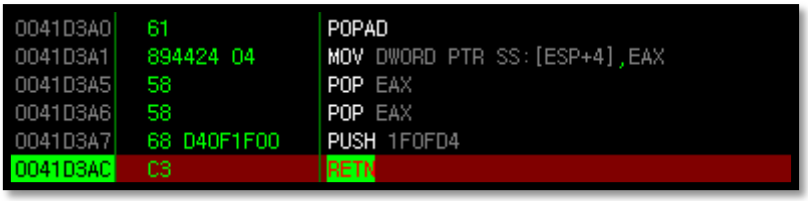
KEY : 45136

Binary 400 [Solver - Gogil]

실행해보면 에디트와 버튼이 Disable 되어있고, 키보드로 입력을 해보면 비정상적으로 종료된다.
크래쉬된 지점을 확인하면 0041D922 에서 메모리 참조 오류가 발생했다고 알 수 있다.



0041D922 에 Break Point 를 설정하고 키보드 입력을 해보면 걸리고, 이 함수 밖으로 나가보면 익숙한 코드가 펼쳐진다.



작년 codegate 2011 Binary 500 점과 똑같은 방식의 VM 코드 가상화이다.

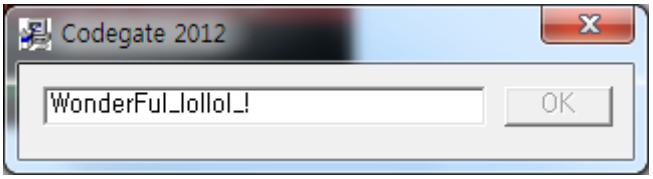
0041D3AC 에서 복호화가 된곳으로 점프하여 한 줄씩 코드가 실행된다.

계속해서 한 줄씩 살펴보면 입력한 키보드 값과 어떠한 값을 비교한다.

비교 구문을 일치하게 수정하면 Edit 에 한 글자가 추가되어 있다.

이런식으로 계속해서 수정하다보면 Full_of_Wonder 이라고 문자가 완성되고, OK 버튼이 활성화된다.

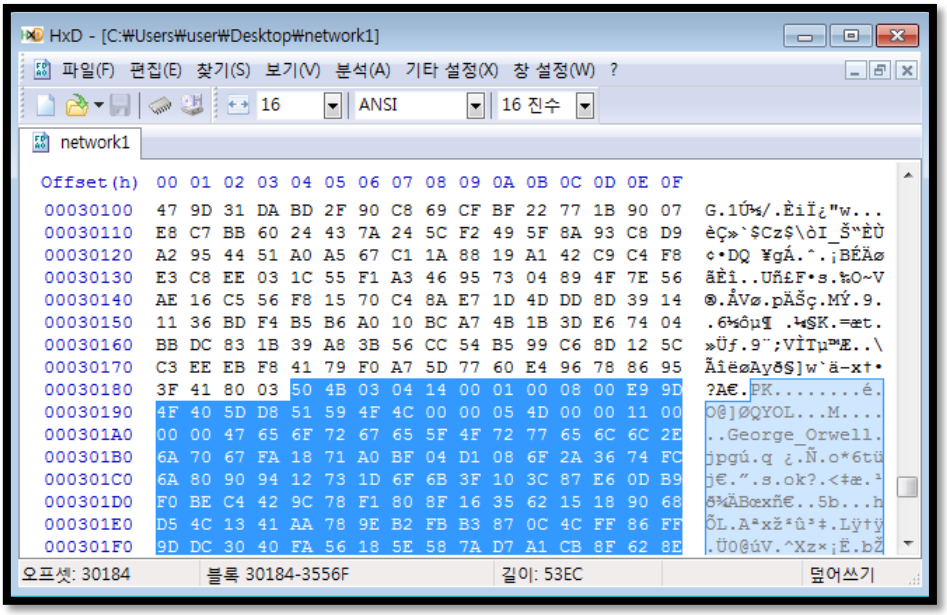
버튼을 클릭하면 답이 출력된다.



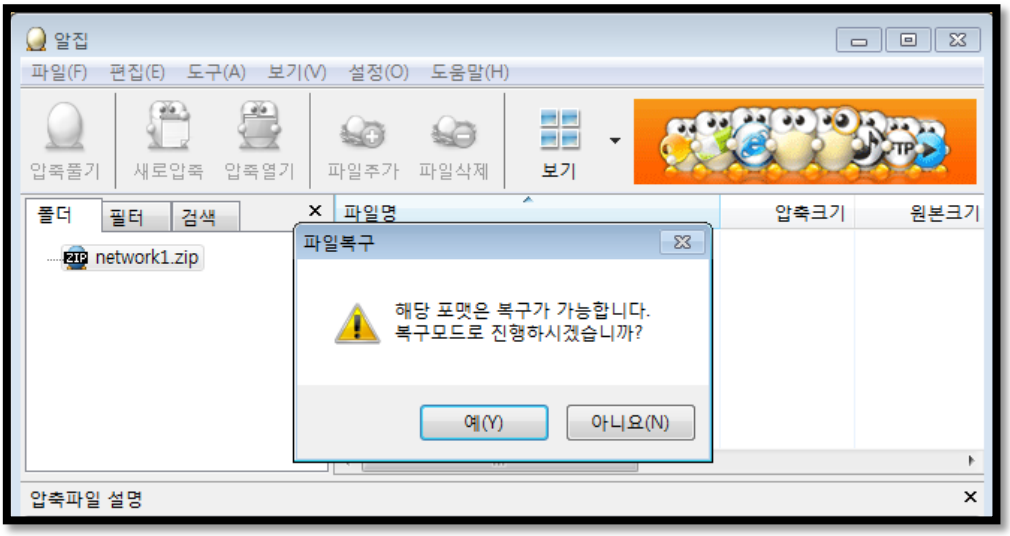
KEY : WonderFul_lollol_!

Network 100 [Solver - UknowY]

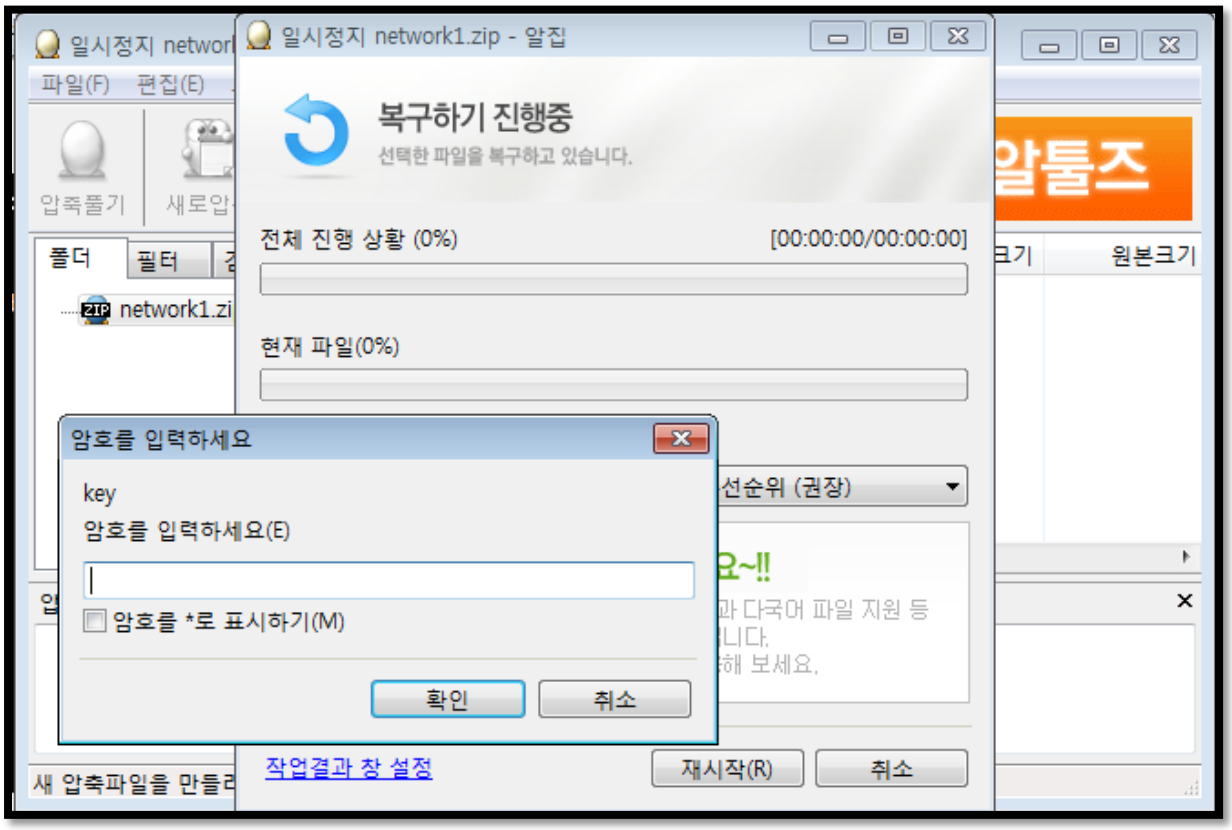
문제를 받아보니 손상된 pcap 파일이 나왔다. HxD 에디터로 이리저리 살펴보다 무언가 다른 타입의 파일이 속해 있지 않을까 검색을 해보았다. 그렇게 삽질을 하다 결국 50 4B 03 04로 시작하는 zip 파일 헤더를 찾았다.



50 4B 03 04로 시작하는 부분부터 끝까지 긁어와서 zip 파일을 추출했다.



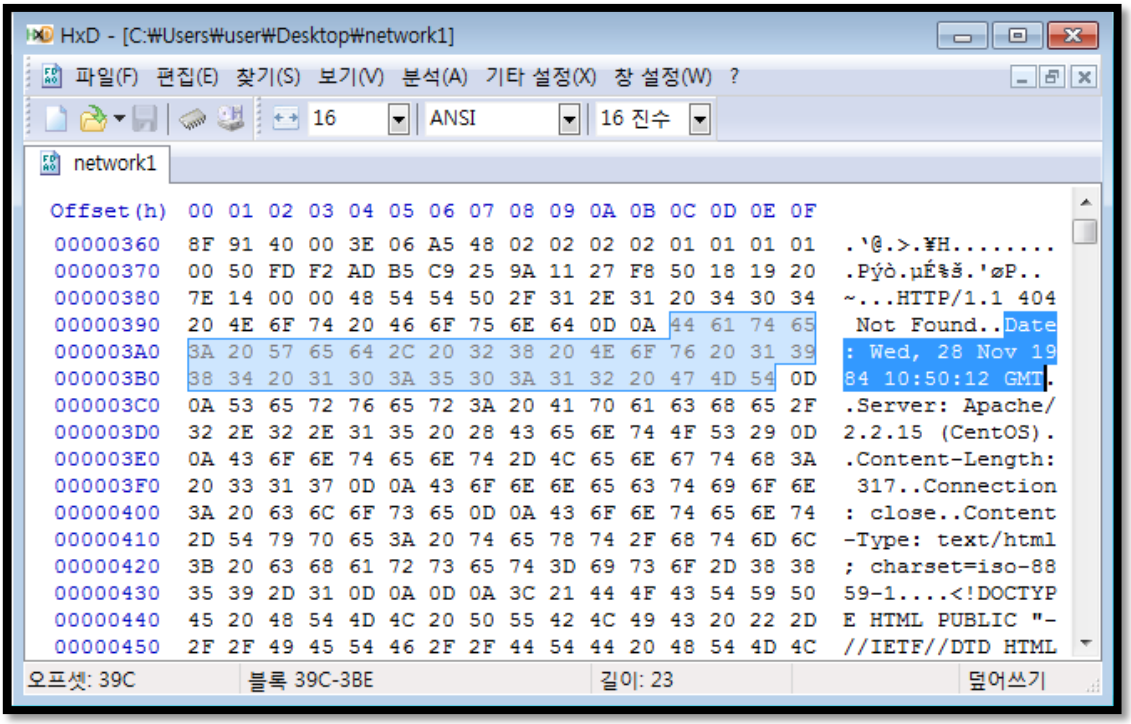
Zip 파일을 열었더니 지원하지 않는 포맷이라며 복구를 할 수 있다고 한다.



예를 눌렀더니 key를 입력하란다.

여기서 조금 삽질을 했는데, 힌트를 보니 "The password of disclosure document is very weakness and based on Time, can be found easily." 이라며 시간에 기초를 해둔다고 했다.

여기서 다시 원래의 손상된 pcap 파일을 살펴 보았다.
맨 위에서 차근차근 훑어보니 coge.hackthepacket.com 이라는 곳에 request를 보내고, 404 Not Found라는 reponse를 받은 부분이 있었다. 바로 여기에 어떤 날짜 값이 적혀있었는데,

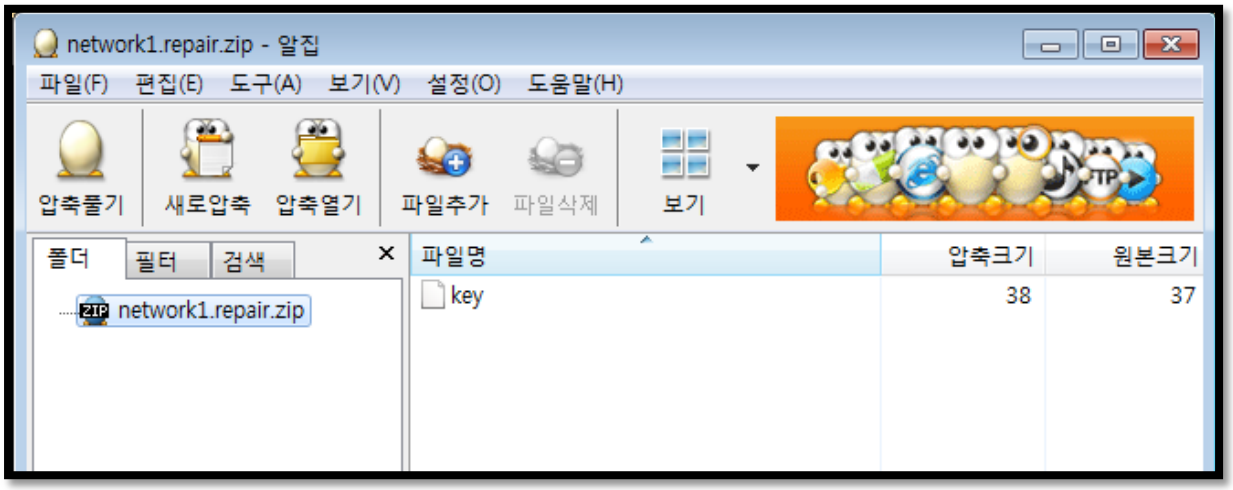


이 부분을 풀어써서 "Key = 0x20120224 (if date format is 2012/02/24 00:01:01)" 라는 형식에 맞춰 0x19841128 이라 대입해 봤다.

안 된다.

다시 그냥 19841128이라 대입해 봤다.

풀렸다.



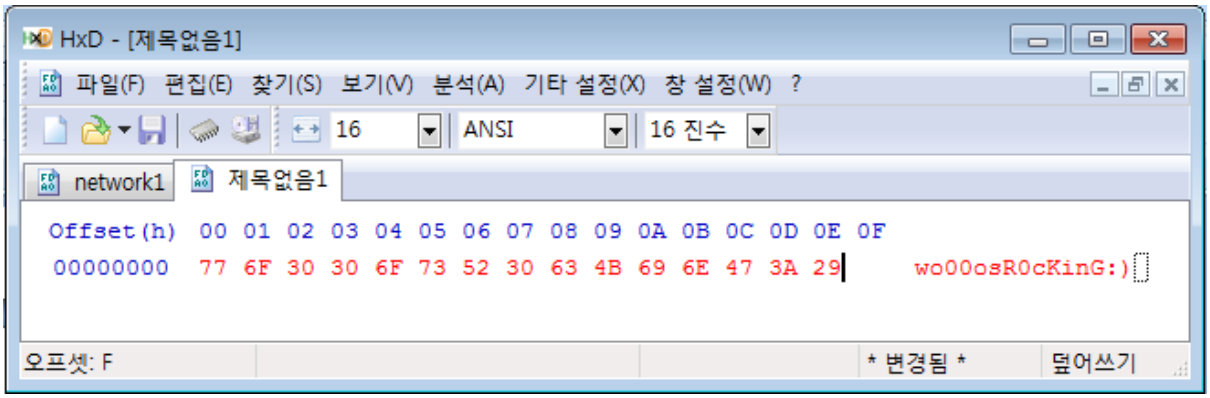
Key 라는 이름의 파일이 나왔고 읽어보니 be7790a9f6e79752d1f9e55a79a33f421cf68라는 값이 써 있었다.

힌트를 다시 보니 Msg값이 Key로 인해 Crypto = C(M) = Msg * Key 식으로 인코딩 되어 있었고, Msg 값을 구하려면 Crypto(=be7790a9f6e79752d1f9e55a79a33f421cf68) 값을 Key(=0x19841128)로 나누어야 한다.

파이썬을 이용하여 나누었다.

```
>>> print hex(0xbe7790a9f6e79752d1f9e55a79a33f421cf68 / 0x19841128)

0x776f30306f735230634b696e473a29L
```

이 0x776f30306f735230634b696e473a29 값을 HxD 에디터에 넣었더니 어떤 값이 나왔다.

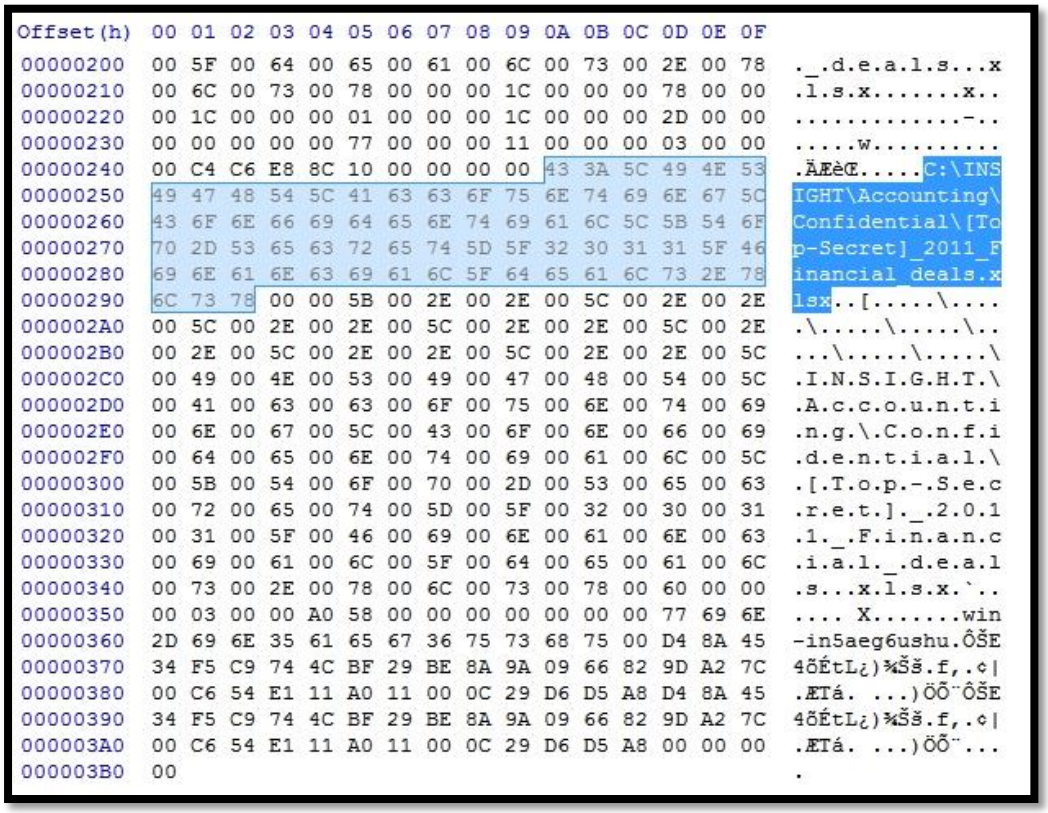


KEY : wo00osR0cKinG:)

Forensics 100 [Solver – Rav3n]

먼저 문제 내용을 보면 엑셀 파일을 빼냈다고 했다.

위에 이미지처럼 파일을 검색해본 결과 의심되는 파일을 찾아 Hex에디터로 열어본 결과



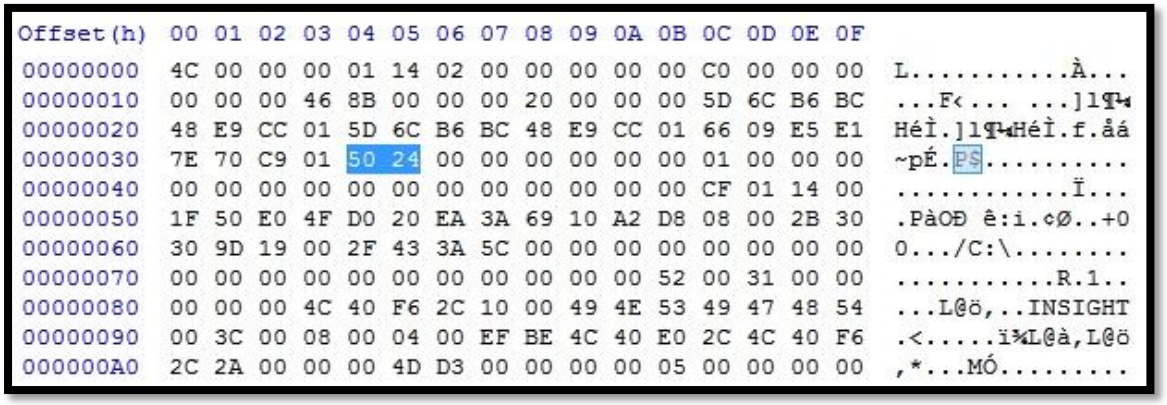
Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	
00000200	00	5F	00	64	00	65	00	61	00	6C	00	73	00	2E	00	78	._.d.e.a.l.s...x
00000210	00	6C	00	73	00	78	00	00	00	1C	00	00	00	78	00	00	.l.s.x.......x..
00000220	00	1C	00	00	00	01	00	00	00	1C	00	00	00	2D	00	00	.............-..
00000230	00	00	00	00	00	77	00	00	00	11	00	00	00	03	00	00	.....w..........
00000240	00	C4	C6	E8	8C	10	00	00	00	00	43	3A	5C	49	4E	53	.ÄÆè.....C:\INS
00000250	49	47	48	54	5C	41	63	63	6F	75	6E	74	69	6E	67	5C	IGHT\Accounting\
00000260	43	6F	6E	66	69	64	65	6E	74	69	61	6C	5C	5B	54	6F	Confidential\[To
00000270	70	2D	53	65	63	72	65	74	5D	5F	32	30	31	31	5F	46	p-Secret]_2011_F
00000280	69	6E	61	6E	63	69	61	6C	5F	64	65	61	6C	73	2E	78	inancial_deals.x
00000290	6C	73	78	00	00	5B	00	2E	00	2E	00	5C	00	2E	00	2E	lsx..[.....\....
000002A0	00	5C	00	2E	00	2E	00	5C	00	2E	00	2E	00	5C	00	2E	.\.....\.....\..
000002B0	00	2E	00	5C	00	2E	00	2E	00	5C	00	2E	00	2E	00	5C	...\.....\.....\
000002C0	00	49	00	4E	00	53	00	49	00	47	00	48	00	54	00	5C	.I.N.S.I.G.H.T.\
000002D0	00	41	00	63	00	63	00	6F	00	75	00	6E	00	74	00	69	.A.c.c.o.u.n.t.i
000002E0	00	6E	00	67	00	5C	00	43	00	6F	00	6E	00	66	00	69	.n.g.\.C.o.n.f.i
000002F0	00	64	00	65	00	6E	00	74	00	69	00	61	00	6C	00	5C	.d.e.n.t.i.a.l.\
00000300	00	5B	00	54	00	6F	00	70	00	2D	00	53	00	65	00	63	.[.T.o.p.-.S.e.c
00000310	00	72	00	65	00	74	00	5D	00	5F	00	32	00	30	00	31	.r.e.t.]._.2.0.1
00000320	00	31	00	5F	00	46	00	69	00	6E	00	61	00	6E	00	63	.1._.F.i.n.a.n.c
00000330	00	69	00	61	00	6C	00	5F	00	64	00	65	00	61	00	6C	.i.a.l._.d.e.a.l
00000340	00	73	00	2E	00	78	00	6C	00	73	00	78	00	60	00	00	.s...x.l.s.x.`..
00000350	00	03	00	00	A0	58	00	00	00	00	00	00	00	77	69	6E	.... X.......win
00000360	2D	69	6E	35	61	65	67	36	75	73	68	75	00	D4	8A	45	-in5aeg6ushu.ÔŠE
00000370	34	F5	C9	74	4C	BF	29	BE	8A	9A	09	66	82	9D	A2	7C	4õÉtL¿)¾Šš.f,.¢|
00000380	00	C6	54	E1	11	A0	11	00	0C	29	D6	D5	A8	D4	8A	45	.ÆTá. ...)ÖÕ¨ÔŠE
00000390	34	F5	C9	74	4C	BF	29	BE	8A	9A	09	66	82	9D	A2	7C	4õÉtL¿)¾Šš.f,.¢|
000003A0	00	C6	54	E1	11	A0	11	00	0C	29	D6	D5	A8	00	00	00	.ÆTá. ...)ÖÕ¨...
000003B0	00																.

전체 경로는 나오지만 lnk파일이라 원본의 파일 사이즈를 찾을 수가 없었는데

검색을 해본 결과 lnk파일 헥스값 중 원본 파일 사이즈 관련 값이 있다는걸 알았다

34 – 37	4	FileSize	링크 대상의 크기

그래서 이거에 해당된 값을 찾아 보았는데



Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	
00000000	4C	00	00	00	01	14	02	00	00	00	00	00	C0	00	00	00	L.....À...
00000010	00	00	00	46	8B	00	00	00	20	00	00	00	5D	6C	B6	BC	...F‹... ...]l¶¼
00000020	48	E9	CC	01	5D	6C	B6	BC	48	E9	CC	01	66	09	E5	E1	HéÌ.]l¶¼HéÌ.f.åá
00000030	7E	70	C9	01	50	24	00	00	00	00	00	00	01	00	00	00	~pÉ.P$....
00000040	00	00	00	00	00	00	00	00	00	00	00	00	CF	01	14	00	....Ï...
00000050	1F	50	E0	4F	D0	20	EA	3A	69	10	A2	D8	08	00	2B	30	.PàOÐ ê:i.¢Ø..+0
00000060	30	9D	19	00	2F	43	3A	5C	00	00	00	00	00	00	00	00	0.../C:\.....
00000070	00	00	00	00	00	00	00	00	00	00	00	52	00	31	00	00	.....R.1..
00000080	00	00	00	4C	40	F6	2C	10	00	49	4E	53	49	47	48	54	...L@ö,..INSIGHT
00000090	00	3C	00	08	00	04	00	EF	BE	4C	40	E0	2C	4C	40	F6	.<.....ï¾L@à,L@ö
000000A0	2C	2A	00	00	00	4D	D3	00	00	00	00	05	00	00	00	00	,*...MÓ.....

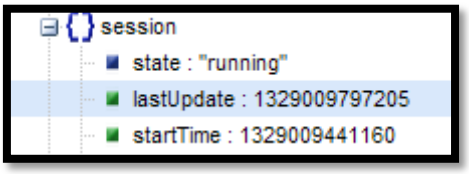
3번째 줄에서 4~7까지 이라고 했는데 6하고 7에 해당된건 아무 값이 아니므로 사이즈 값은 50 24 그리고 이걸 10진수로 바꿔야 하기 때문에 리틀 엔디안 식으로 바꿔서 10진수로 하면 파일 사이즈는 9296.

그리고 킷값 형식에 맞춰서 변형하면..

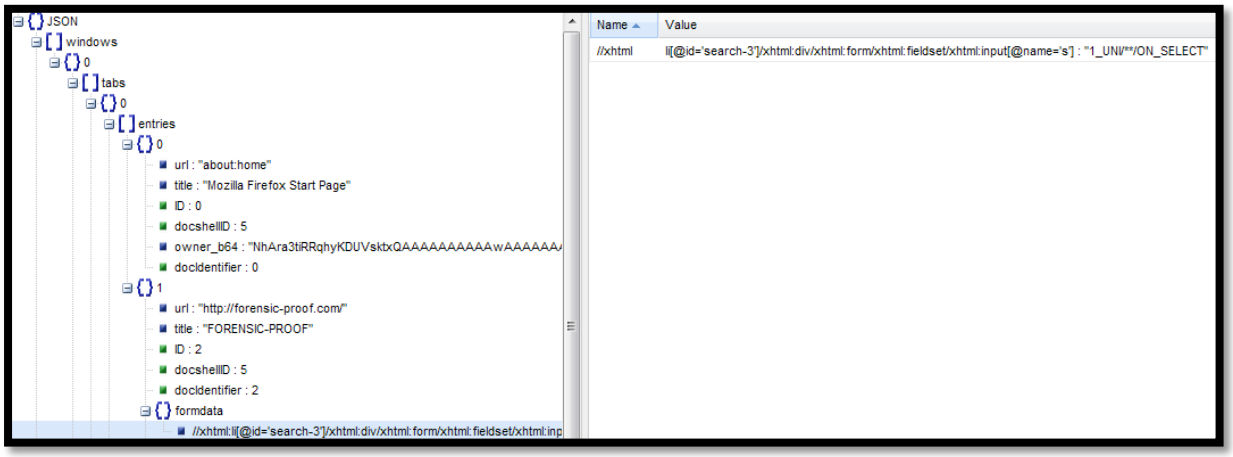
KEY : d3403b2653dbc16bbe1cfce53a417ab1

Forensics 200 [Solver - extr]

우선 브라우저가 비정상적으로 종료가 되었다고 함에서 Crash File에 초점을 두기로 했습니다.
일단 첫 분석 대상 브라우저로 FireFox(이하, FF)를 잡고 분석을 하였습니다.
FF의 경우, 세션이 비정상적으로 종료가 되는 경우를 대비하여 현재 열린 페이지를 저장하여 다음에 다시 열지 않아도 자동적으로 열리는 기능을 수행하기 위한 SessionStore.js 파일을 생성하는 특징이 존재합니다.
이 파일 내부에는 복구를 위한 URL뿐만 아니라, Page Title, Referrer, Formdata, Cookies 등의 다양한 값들이 존재하기 때문에 충분히 답이 존재한다는 가능성이 있었습니다.
SessionStore.js 파일은
Users₩proneer₩AppData₩Roaming₩Mozilla₩Firefox₩Profiles₩075lfxbt.default에 저장되어 있었습니다.



제일 먼저 마지막 세션이 끊긴 시간을 조사하여 값을 구한 후, 타임스탬프 변환을 통해 2012-02-12T10:23:17+09:00라는 시간 값을 얻게 되었습니다.

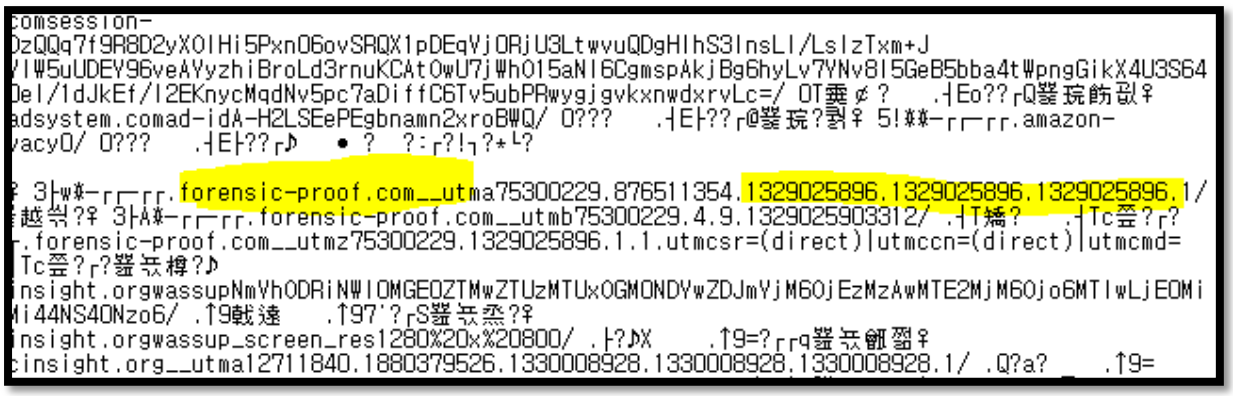


또한, Formdata를 조사하여 쿼리 값이 1_UNI/**/ON_SELECT라는 것도 알게 되었습니다.
이렇게 하여 답을 조합하면 1_UNI/**/ON_SELECT|2012-02-12T10:23:17+09:00가 답임을 알 수 있습니다.

KEY : 1_UNI//ON_SELECT|2012-02-12T10:23:17+09:00**

Forensics 300 [Solver – extr]

File Hex를 보면 URL과 Timestamp로 추정되는 동일 값이 반복됩니다.



저러한 패턴을 파악한 후, 먼저 URL을 전체적으로 한번 뽑아보았습니다.

1. tools.google.com
2. forensic-proof.com
3. forensicinsight.org
4. global.ahnlab.com
5. google.com
6. forensics.sans.org
7. test.wargame.kr
8. dailysecu.com
9. hanrss.com
10. sans.org
11. disqus.com
12. gomtv.com
13. slashdot.org

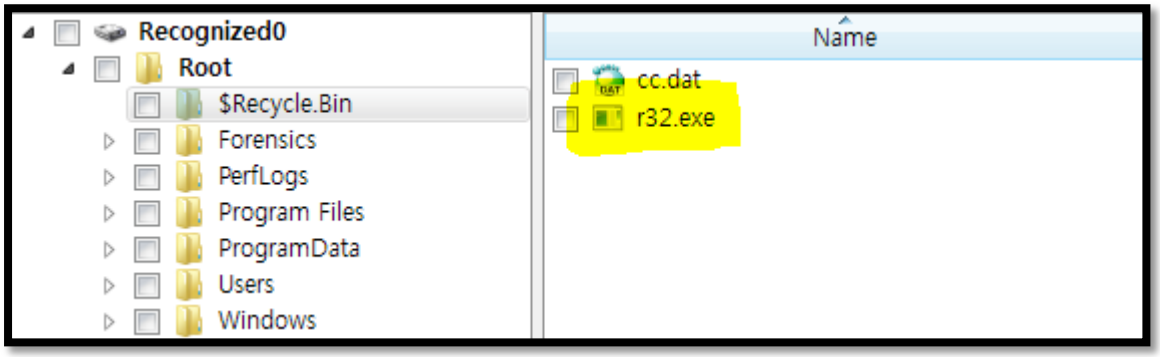
그냥 딱 봐서 test.wargame.kr이 제일 의심스러워 보이길래 test.wargame.kr이 적힌 부분 뒤에 있는 타임스탬프로 추정되는 값을 변환(2012-02-09T23:57:27)하여, Key 형식으로 변환해서 (test.wargame.kr|2012-02-09T23:57:27) 인증하니 인증이 되었습니다.

KEY : test.wargame.kr|2012-02-09T23:57:27

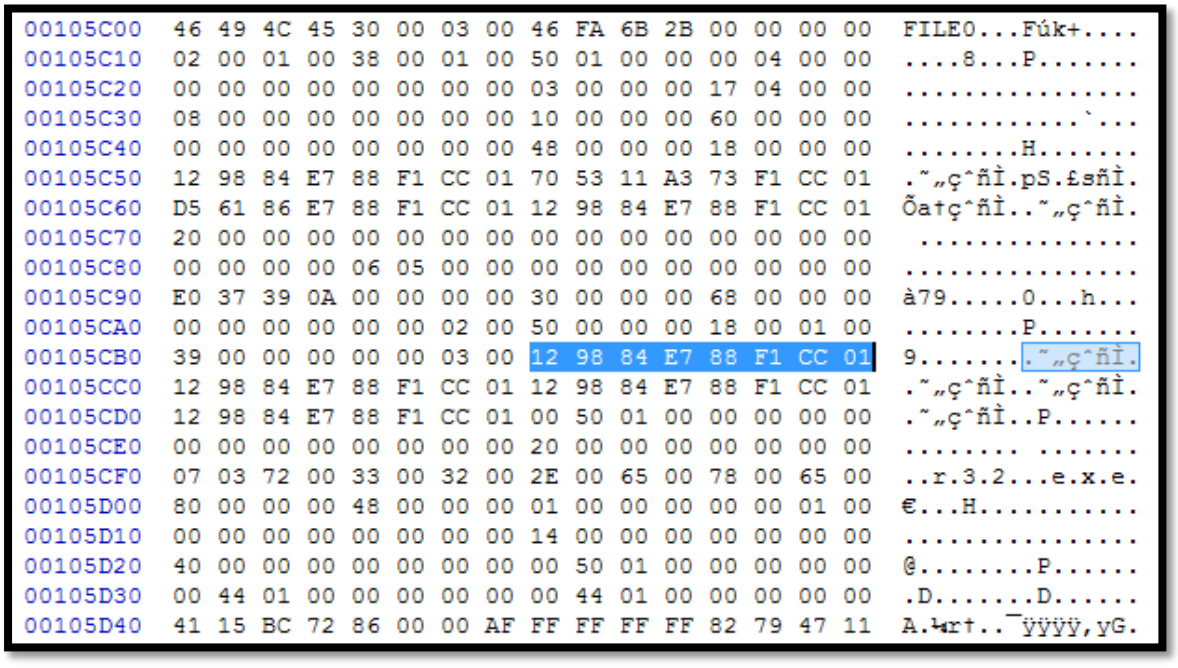
Forensics 400 [Solver - extr]

MFT영역을 분석하여 악성 파일의 생성된 시각을 목적으로 하는 문제입니다.

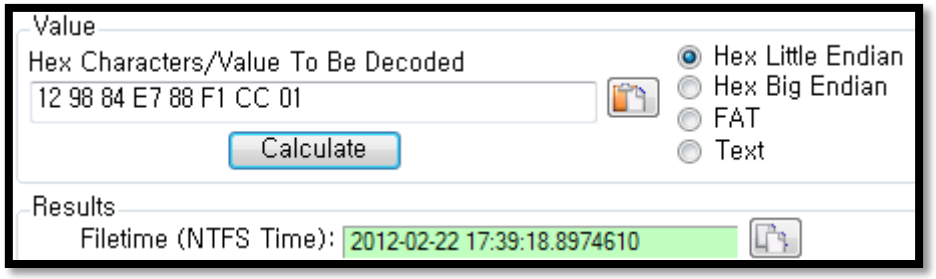
공격자가 악성 파일을 삭제했다고 하니 Recycle.bin을 먼저 접근해 보아야 할 대상으로 두었습니다.



RSTUDIO로 복구하여 $Recycle.Bin에 접근해보니 r32.exe라는 악성 파일로 추정되는 파일이 존재했습니다. 완전한 시간을 알기 위해선 file time 값을 봐야 하기 때문에 파일이름을 검색하여 생성 시간을 알 수 있었습니다.



생성 시간을 토대로 File Time 변환 프로그램을 이용해 변환한 뒤, UTC +09시로 바꿔주고 답 형식을 맞춘 뒤 인증을 하면 됩니다.



KEY : 2012-02-23T02:39:18.8974610+09:00

Misc 100 [Solver – Electrop]

자동으로 풀어주는 툴을 사용하면 된다.

http://www.purplehell.com/riddletools/applets/cryptogram.htm

입력:

Az hrb eix mcc gyam mcxgixec rokaxioaqh hrb mrqpck gyam lbamgarx oatygqh Erxtoigbqigarx Gidc hrbg gasc gr koaxd erzzcc zro i jyaqc Kr hrb ocqh rx Ockubqq ro Yrg man?

Gyc ixmjco am dccqihrbgm

출력:

IF YOU CAN SEE THIS SENTANCE ORDINARILY YOU SOLVED THIS ZUISTION RIGHTLY CONGRATULATION TAKE YOUT TIME TO DRINK COFFEE FOR A WHILE DO YOU RELY ON REDJULL OR HOT SIX? THE ANSWER IS KEELAYOUTS

KEY : KEELAYOUTS

Misc 200 [Solver – m4st3rm1nd]

문제를 보고 바로 crypto++ 라이브러리를 사용하여 위의 키를 브루트포싱으로 찾아내는 프로그램을 짰다.

```
//written by m4st3rm1nd@TMP

#include <iostream>
#include <vector>
#include <string>
#include <cryptopp/des.h>
#include <cryptopp/modes.h>

using namespace std;
using namespace CryptoPP;

vector<pair<const char*,int>> chars{{"QWERTYUIOP",10},{"ASDFGHJKL",9},{"ZXCVBNM",7}};

char translate(int n1,int n2){

        if(n2==0)n2=10;
        if(n2>chars[n1-1].second)return '₩x00';

        return chars[n1-1].first[n2-1];
}

string plain="₩xB6₩xB2₩xB6₩xAC₩xAC₩xA6₩xB0₩xAA";
string cipher="₩x05₩xD9₩x12₩xE7₩xCC₩xD9₩xBB₩xCA";

void dotest(const vector<int>& k, vector<int>& cur_keys, size_t xxx){

        size_t i;
        int sx;
        if(xxx==cur_keys.size()){

                //test for stds
                char key[cur_keys.size()+1];
                key[xxx]='₩x00';

                for(i=0;i<cur_keys.size();i++){

                        key[i]=translate(cur_keys[i],k[i]);
                }

                ECB_Mode<DES>::Encryption asdf;
                asdf.SetKey((byte*)key,cur_keys.size());
                char out[plain.size()];
                asdf.ProcessData((byte*)out,(byte*)plain.data(),plain.size());

                if(memcmp(out,cipher.data(),cipher.size())==0){
```

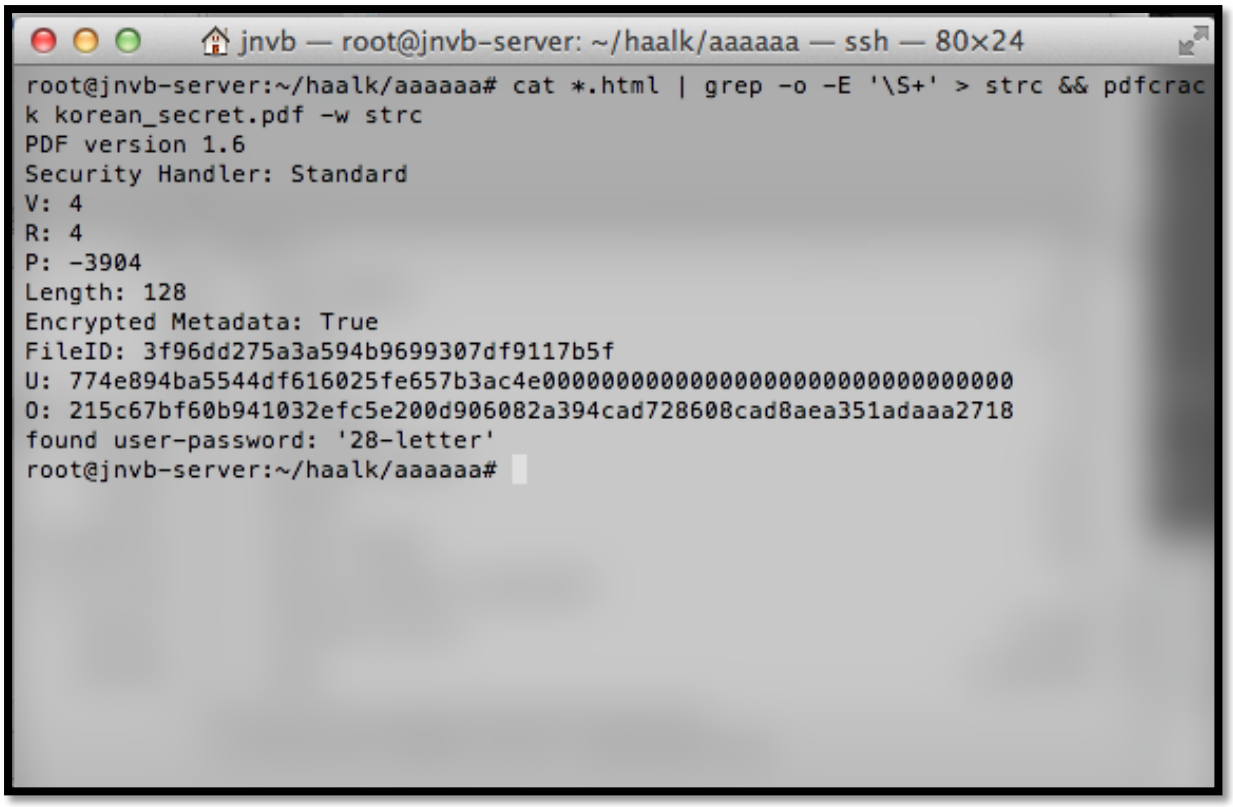
```
                        cout << key << endl;
                        for(i=0;i<cur_keys.size();i++){

                                cout << cur_keys[i] << k[i] << " ";
                        }

                        cout << endl;
                        cout << endl;
                }
                //cout << key << endl;
                return;
        }

        for(sx=1;sx<4;sx++){

                cur_keys[xxx]=sx;
                dotest(k,cur_keys,xxx+1);
        }
}

ECB_Mode<DES>::Encryption asdf;
int main(){

        vector<int> k{8,9,9,4,3,5,9,5};
        vector<int> asdf(k.size());
        vector<char> key(8);

        dotest(key,0);
        //cout << plain << endl;
}
```

나온 키 값은 ILOVEBOB, hex값은 494C4F5645424F42.

₩xFF와 xor하여 나온값은 B6B3B0A9BABDB0BD.

KEY : B6B3B0A9BABDB0BD

Misc 300 [Solver – JNVB]



html파일을 다 읽어서 정규식으로 공백을 제외한 문자를 전부 추출해서 strc파일로 저장합니다.

그 다음에 pdfcrack 을 사용하여 strc 파일로 브루트포싱 기법으로 pdf의 비밀번호를 찾습니다.

그 후, 문제에 나와있는 대로 28-letter를 MD5해서 대문자로 바꾸면 답입니다.

KEY : 23FB0EC48DF3EACABCA9E98E8CA24CD1

Misc 300 [Solver – Hexinic & JNVB]

zip파일 하나 주어진 것을 풀어보면 여러 html 파일들을 볼 수 있습니다.

그 중에서 홈페이지 파일 이라는 폴더 안에 잇는 js 파일을 보면 맨 아랫줄에

```
eval(function(p,a,c,k,e,d){e=function(c){return c};if(!''.replace(/^/,String)){while(c--){d[c]=k[c]||c}k=[function(e){return d[e]}];e=function(){return'₩₩w+'};c=1};while(c--){if(k[c]){p=p.replace(new RegExp('₩₩b'+e(c)+'₩₩b','g'),k[c])}}return p}('127=["₩₩10₩₩8₩₩26₩₩21₩₩16₩₩25₩₩8","","₩₩21₩₩8₩₩20₩₩14₩₩22₩₩24","₩₩23₩₩27₩₩30₩₩23₩₩22₩₩10₩₩29₩₩20₩₩14","₩₩31₩₩10₩₩17₩₩28₩₩18₩₩24₩₩16₩₩10₩₩18₩₩17₩₩35₩₩8"];32 37(5){5=5[7[0]](/ /15,1);5=5[7[0]](/₩₩38/15,0);1213=5;5=7[1];19(6=0;6<13[7[2]];6++){5=13[7[3]](6,6+1)+5};1211=7[1];19(6=0;6<5[7[2]];6+=9){11+=36[7[4]](33(5[7[3]](6,6+9),2))};34(11)};',10,39,'|||||_0x272dx2|i|_0xfd3a|x65||x72|_0x272dx4|var|_0x272dx3|x67|g|x61|x6F|x43|for|x6E|x6C|x74|x73|x68|x63|x70|x75|x6D|x69|x62|x66|function|parseInt|eval|x64|String|c|t'.split('|'),0,{}))
```

이라는 코드가 있습니다. 이거는 eval(function(p,a,c,k,e,d) 이거를 보고 난독화된 코드라 생각하고 디코딩 해본 결과

```
function c(_0x272dx2) {
    _0x272dx2 = _0x272dx2['replace'](/ /g, 1);
    _0x272dx2 = _0x272dx2['replace'](/₩t/g, 0);
    var _0x272dx3 = _0x272dx2;
    _0x272dx2 = '';
    for (i = 0; i < _0x272dx3['length']; i++) {
        _0x272dx2 = _0x272dx3['substring'](i, i + 1) + _0x272dx2
    };
    var _0x272dx4 = '';
    for (i = 0; i < _0x272dx2['length']; i += 9) {
        _0x272dx4 += String['fromCharCode'](parseInt(_0x272dx2['substring'](i, i + 9), 2))
    };
    eval(_0x272dx4)
};
```

이런 식으로 나오는데 eval 함수를 document.write 로 이용해서 결과를 보면

```
if (new Date().getTime() > 1330268400000) {  var dummya = '1';  var dummyb = '1'; var dummyv = '1';  var dummyc = '1';  var dummys = '1';  var dummyae = '1';  var dummyasefa = '1';  var dummeya = '1';  var dummya = '1';  var dum3mya = '1'; var dumm54ya = '1';  var dumm3ya = '1';  var dum1mya = '1';  var p = 'YTK4YPT1YK48PTK48TK34PTYK6TDKT5P2KT73TKPY4TBTK3TT4YKT4ETK4YTP7K4T6KT30TKYP7T2KYT33TKP7TY6KTYP33TKPY7PT2YT';  p = p.replace(/T/g, '').replace(/P/g, '').replace(/Y/g, '').replace(/K/g, '%');  var authkey = unescape(p);}
```

이 코드를 해석해보면 unescape 를 이용해서 YTK4YPT1YK48PTK48TK34PTYK6TDKT5P2KT73TKPY4TBTK3TT4YKT4ETK4YTP7K4T6KT30TKYP7T2KYT33TKP7TY6KTYP33TKPY7PT2YT 를 풀라는 뜻입니다.

바로 js 에서 답 처리 하기 위해서 alert(authkey) 를 추가해서 js 를 돌려보면 경고 창으로 'AHH4mRsK4NGF0r3v3r' 이라 뜨는데 인증하면 정답!

KEY : AHH4mRsK4NGF0r3v3r

Misc 100 [Solver – JNVB]

RDCVGF_YGBNJU_TGBNM_YGBNJU_TGBNM_TGBNM_YGBNJU_TGBNM

키보드 배치를 보니

RDCVGF = G

YGBNJU = O

TGBNM = L

그래서 답은 G_O_L_O_L_L_O_L

KEY : G_O_L_O_L_L_O_L