DEFCON 18 CTF
문제풀이

김은수 hahah
KAIST GoN / Beistlab
hahah@kaist.ac.kr
twitter.com/Hah4h

Contents

- DEFCON CTF?
- Binary l33tness 100
- Binary l33tness 200
- Binary l33tness 300
- Binary l33tness 400
- Binary l33tness 500
- Pwtent pwnables 100
- Pwtent pwnables 400
- 후기

DEFCON CTF?

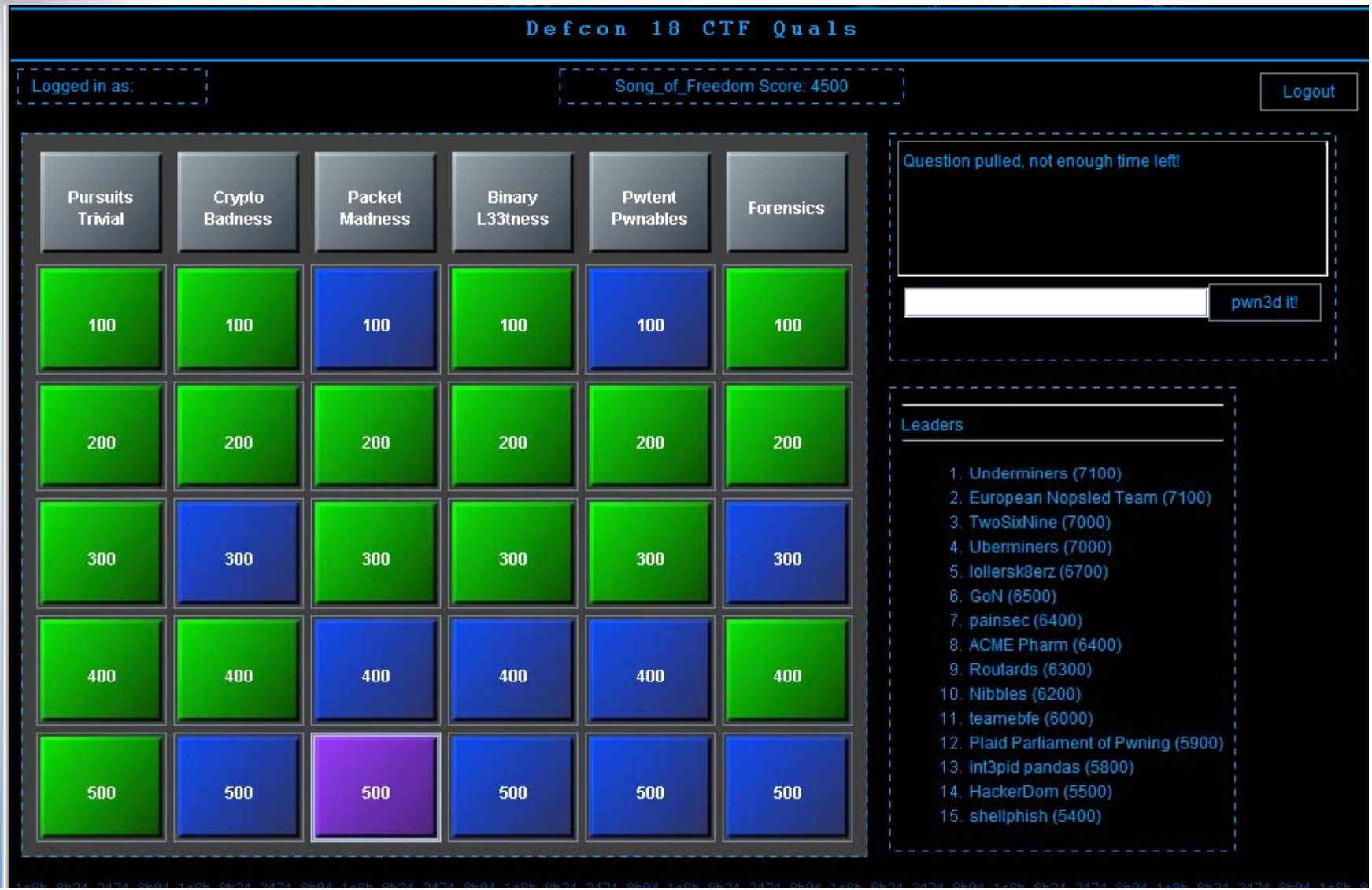
- DEFCON
  - 매년 라스베가스에서 개최되는 Hacking Conference
  - 올해로 18번째
- Capture The Flag
  - 상대방의 시스템을 공격하여 Flag를 빼오고, 동시에 자신의 시스템을 방어하는 방식의 대회
  - 작년부터 Diutinus Defense Technologies Corp(ddtek)에서 문제 출제

DEFCON CTF?

- CTF 예선
  - 본선과는 달리 CTF 방식이 아닌 6개의 분야에 각각 5개의 문제 풀이
  - 100점에서 500점까지의 난이도
  - 공개된 문제를 처음 푼 팀이 다음 문제를 선택

DEFCON CTF?

- 문제 Board



DEFCON CTF?

- 문제 분야
  - PURSUITS TRIVIAL
    - 넌센스, 게임, 음악분석 등 다양한 분야
  - CRYPTO BADNESS
    - 암호 분석/크랙
  - PACKET MADNESS
    - 주어진 패킷 캡쳐 파일 분석

DEFCON CTF?

- 문제 분야
  - BINARY L33TNESS
    - 바이너리 리버싱
  - PWTENT PWNABLES
    - 리모트 취약점 공격
  - FORENSICS
    - 디스크 이미지 파일 등에서 키 값 찾기

Binary l33tness 100

- Find the key
- Linux x86-32
- 실행 결과

```
gon@qwerty:~/defcon18/b100$ ./b100.bin
Please supply a key

pentest@pubuntu:~/ctf/binary100$ ./b100 aaaa
I hope you got it! Good luck
&0iE.m5 J
]{>=5tC!B
```

Binary l33tness 100

- 분석

```
public start
proc near
xor     ebp, ebp
pop     esi
mov     ecx, esp
and     esp, 0FFFFFFF0h
push    eax
push    esp
push    edx
push    offset sub_8048A20
push    offset sub_8048A30
push    ecx
push    esi
push    offset sub_80485B4
call    __libc_start_main
hlt
endp
```

```
int __cdecl sub_80485B4(int a1, int a2)
{
  unsigned int v3; // eax@4
  char v4; // [sp+1Eh] [bp-102h]@4

  if ( a1 != 2 )
  {
    fwrite("Please supply a key", 1u, 0x13u, stderr);
    exit(1);
  }
  ++dword_8049ECC;
  fwrite("I hope you got it! Good luck\n", 1u, 0x1Du, stderr);
  v3 = strlen(*(const char **)(a2 + 4));
  sub_8048698(&v4, *(_DWORD *)(a2 + 4), v3);
  sub_804886A((int)&v4, (int)byte_8049E60, 34);
  fprintf(stderr, "%s\n", byte_8049C60);
  return 0;
}
```

- sub_8048698, sub_804886A 함수
  - RC4 알고리즘!

Binary l33tness 100

- sub_8048698 함수
  - 256byte의 table을 만드는 RC4 함수!

```
  result = memset(s, 0, 0x100u);
  if ( a3 )
  {
    for ( i = 0; i <= 0xFF; ++i )
      *((_BYTE *)s + i) = i;
    for ( j = 0; j <= 0xFF; ++j )
    {
      *((_BYTE *)s + 257) += *((_BYTE *)s + j) + *(_BYTE *)(a2 + j % a3);
      v4 = *((_BYTE *)s + *((_BYTE *)s + 257));
      *((_BYTE *)s + *((_BYTE *)s + 257)) = *((_BYTE *)s + j);
      *((_BYTE *)s + j) = v4;
    }
    result = s;
    *((_BYTE *)s + 257) = 0;
  }
  return result;
}
```

Binary l33tness 100

- sub_804886A 함수
  - RC4 인코딩 함수

```
for ( i = 0; ; ++i )
{
  result = i < a3;
  if ( i >= a3 )
    break;
  *(_BYTE *)a2++ ^= sub_80487AA(a1);
}
return result;
```

Binary l33tness 100

- 입력한 문자열을 키 값으로 RC4 알고리즘을 사용, 결과를 출력
  - 키 값을 모르면 풀 수 없다!
  - Brute force?
  - Guessing?

No !

Binary l33tness 100

- constructor에서 atexit함수를 이용하여 sub_80488e0 함수를 끝날 때 실행하도록 설정

```
int __cdecl sub_80489B5(int a1, int a2)
{
  int result; // eax@3

  if ( a1 == 1 )
  {
    if ( a2 == 65535 )
    {
      sub_80488AC((int)&unk_8049EE0);
      result = __cxa_atexit(sub_80488E0, &unk_8049EE0, &unk_8048AE0);
    }
  }
  return result;
}
```

```
v3 = 'uces';
v4 = 'r';
v5 = 'e';
v6 = 'k';
v7 = 'e';
v8 = 'y';
sub_8048698(&s, (int)&v3, 9u);
sub_804886A((int)&s, (int)&unk_8049E84, 30);
result = fprintf(stderr, "%s\n", &unk_8049E84);
```

"securekey"를 키로 어떤 데이터를 복호화, 출력

Binary l33tness 100

- 해당 데이터를 복호화

- The real key is: bob's yer mom?

Binary l33tness 100

- 또 다른 풀이~

```
gon@qwerty:~/defcon18/b100$ ./b100.bin
Please supply a key
```

NULL key를 입력

```
gon@qwerty:~/defcon18/b100$ ./b100.bin ""
I hope you got it! Good luck
M.+Q
The real key is: bob's yer mom
```

으..으어??

Binary l33tness 200

- what treasure did pirates get?
- file 명령어 결과
  - ELF 32-bit LSB shared object, Intel 80386, version 1 (SYSV), dynamically linked, stripped
- 32bit Linux에서 실행
  - Segmetation 오류!!?!?

Binary l33tness 200

- Strings를 이용하여 보면...
  - GCC: (GNU) 2.95.3-haiku-090629
  - 일반 linux가 아니라 haiku OS !!

- http://www.haiku-os.org/에서 vmware이미지를 받아서 실행 !

Binary l33tness 200

- 실행하면 pre_init에서 crash 발생

```
public pre_init
proc near               ; CODE XREF: frame_exit+44↑p
mov     eax, 1Dh
int     80h             ; LINUX - sys_pause
leave
retn
endp
```

- Haiku os에 맞는 interrupt로 패치

```
public pre_init
proc near               ; CODE XREF: frame_exit+44↑p
mov     eax, 1Dh
int     63h             ; reserved for user interrupt
leave
retn
endp
```

Binary l33tness 200

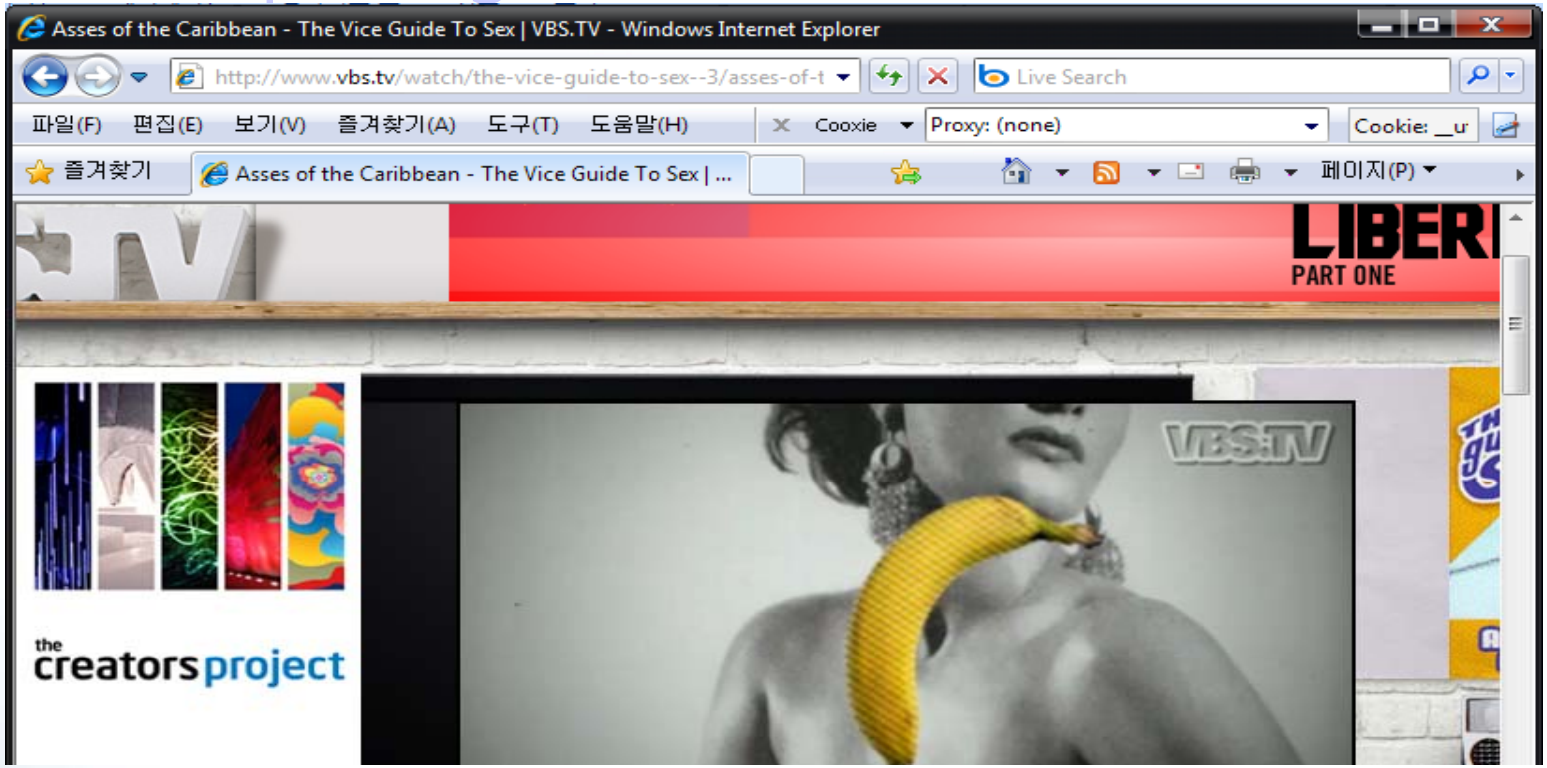
- main 함수 앞부분

```
v4 = getpid();
v5 = v4;
v6 = v4 + (v4 < 0 ? 0x3F : 0);
LOBYTE(v6) = v6 & 0xC0;
v3 = v5 - v6;
fdopen(v5 - v6, "w");
```

- pid가 어떤 조건을 만족하지 않으면 fdopen에서 에러
  - 패치 또는 pid의 조건이 맞을때까지 반복실행!

Binary l33tness 200

- 실행결과
  - http://is.gd/bUBRD 를 출력



키는 동영상의 제목인 Asses of the Caribbean

Binary l33tness 300

- There's an answer in here somew here.

- file 명령어 결과
  - ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked (uses shared libs), for GNU/Linux 2.6.15, not stripped

Binary l33tness 300

- 64bit에서 실행결과

  Encrypted key
  7d343be755ff63e0a5908f625c203573
  Hash for offset 0:
  4a977fd18d9eada7b5a8cb9009311b77
  Xor value is
  37a34436d861ce47103844f255112e04
  Hash for offset 17:
  5afa2306bc15a62e345137b6157a1d7d
  Xor value is
  6d596730647468692469734406b3379
  Final decrypted key value
  6d596730647468692469734406b3379

Binary l33tness 300

- ascii 코드 !!!

```
6D59 6730 6474 6869 2469 7344 406B 3379|mYg0dthi$isD@k3y
```

- mYg0dthi$isD@k3y

Binary l33tness 400

- Running at blitz.ddtek.biz:5566
- file 명령 결과
  - ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), for GNU/Linux 2.6.18, dynamically linked (uses shared libs), stripped
- Java에 관련된 string들
  - J2ME's KVM (K Virtual Machine)

Binary l33tness 400

- KVM을 수정한 것으로 추정
  - 원래 KVM은 class 파일을 인자로 받지만 bin400은 그렇지 않다
  - 내장된 class를 찾거나 VM 분석 !
- 분석
  - 어…어렵다..!!

Binary l33tness 400

- 실행하면 아무런 출력이 없음 !
  - netstat으로 확인하면 5566 포트가 열려있음
  - 서버 프로그램 !

- 접속해보면..

```
hahah@hahah-desktop:~/defcon18$ nc localhost 5566
Please submit your key for verification:
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
Hmm, not quite right.
```

Binary l33tness 400

- 분석
  - main 함수 : 80517F0
  - main 함수를 따라가다가 string 중 “Illegal bytecode %ld”를 쓰는 함수를 발견
  - sub_804E000 : VM 핵심 함수 !
  - bytecode interpreter !

```
.text:0804E018    movzx   eax, byte ptr [esi]
.text:0804E01B    cmp     al, 0DFh          ; switch 224 cases
.text:0804E01D    jbe     short loc_804E048
```

Binary l33tness 400

```
movzx   eax, byte ptr [esi]
cmp     al, 0DFh          ; switch 224 cases
jbe     short loc_804E048
```

- switch가 시작되는 0804E01B
  - eax : opcode
  - ebx : parameter pointer
  - esi : 해당 opcode가 저장된 주소

Binary l33tness 400

- gdb를 이용하여 bytecode 추출
  - attach 하여 switch에서 원하는 데이터를 알아낸다

```
(gdb) b *0x804E01B
(gdb) commands 1
> silent
> printf "%08x %x %x %x₩n", $esi, $eax, *$ebx, *($ebx+4)
> continue
> end
```

Binary l33tness 400

- Bytecode 분석
  - 오랜 시간이 걸린다
  - IF_ICMPEQ 같은 비교 opcode 위주로 분석
  - 문자열을 사용하므로 ARRAYLENGTH opcode를 통해 많은 데이터를 처리

Binary l33tness 400

- ARRAYLENGTH에 대해 breakpoint

```
(gdb) b *0x804E01B
Breakpoint 1 at 0x804e01b
(gdb) cond 1 ($eax==0xbe) // 0xbe – ARRAYLENGTH opcode
(gdb) commands 1
>silent
>printf "%d %s₩n", *(*($ebx)+8), *($ebx)+12
>continue
>end
```

- *(*($ebx)+8) : length
- *($ebx)+12 : array 문자열

Binary l33tness 400

- Client

hahah@hahah-desktop:~/defcon18$ nc localhost 5566
Please submit your key for verification:
aaaaaaa

- gdb - server

Continuing.
16
.... (생략)
7 aaaaaaa
34 Y'Ïs4Q¦üÉ#–tÿþ$íþd7–t4oçPp•₩¡ô
22 Hmm, not quite right.

Binary l33tness 400

- Key의 길이는 34 글자 !
- IXOR opcode를 반복적으로 사용
- 256byte의 array를 사용
  - RC4 알고리즘으로 추측 !

Binary l33tness 400

```
(gdb) b *0x804E01B
Breakpoint 1 at 0x804e01b
(gdb) cond 1 ($eax==0xbe)&&(*(*($ebx)+8)!=256)
(gdb) commands 1
>silent
>printf "%d %s₩n", *(*($ebx)+8), *($ebx)+12
>x/10x (*($ebx)+12)
>continue
>end
```

- 위 코드를 이용하여 문자열을 관찰
- 길이가 256byte인 것은 rc4 key table이라 생각되어 무시

Binary l33tness 400

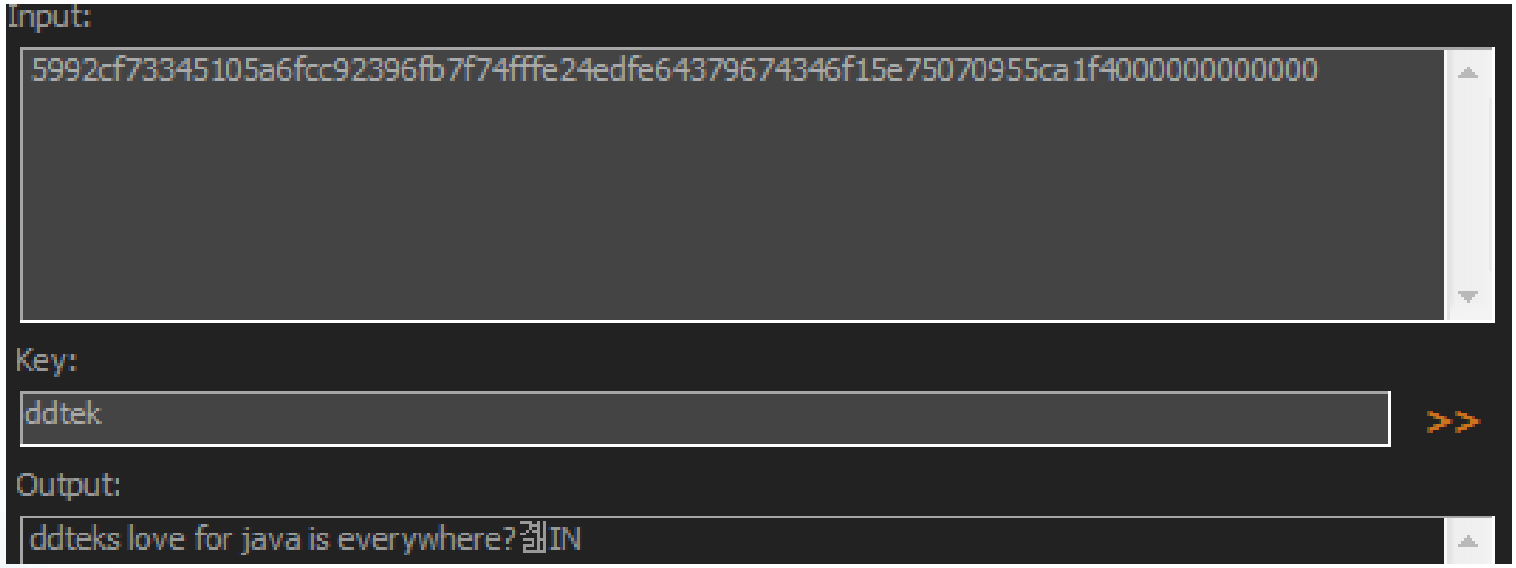
- client에서 34글자 입력

34 Y'Ïs4Q¦üÉ#–tÿþ$íþd7–t4oçPp•₩¡ô
0x808d49c: 0x73cf9259 0xa6055134 0x9623c9fc 0xff747ffb
0x808d4ac: 0xfeed24fe 0x74963764 0xe7156f34 0x5c957050
0x808d4bc: 0x0000f4a1 0x0000050c
5 ddtek
0x808d2d8: 0x65746464 0x0000006b 0x00000c0c
....

- 위의 값이 cipher text, 아래의 ddtek 이 key로 추정

Binary l33tness 400

- 앞에서 구한 값들을 이용하여 RC4로 디코딩을 하였다.



- ddteks love for java is everywhere

Binary l33tness 500

- looking for a key of course: file

- file 명령 결과
  - ELF 64-bit MSB executable, SPARC V9, total store ordering, version 1 (SYSV), dynamically linked (uses shared libs), stripped
  - Solaris, SPARC !!
  - c500_b6427ab1a64e6836.dat 파일도..

Binary l33tness 500

- SPARC machine 이 없다!
- SPARC-ASM을 모른다 ㅠ_ㅠ

- IDA로 정적 분석 !
  - 다행히 symbol이 남아있음
  - 루틴이 복잡하지 않음

Binary l33tness 500

```
.text:0000000100000CB8                  mov     0x125, %l0
.text:0000000100000CBC                  stx     %l0, [%fp+arg_7CF]
.text:0000000100000CC0                  stx     %l3, [%fp+arg_7D7]
.text:0000000100000CC4                  mov     8, %l0
.text:0000000100000CC8                  stx     %l0, [%fp+arg_7DF]
.text:0000000100000CCC                  ldx     [%fp+arg_7CF], %o0
.text:0000000100000CD0                  call    _SUNW_C_GetMechSession
```

- GetMechSession()에 대해 검색
  - pkcs11t.h 내용 중
  - #define CKM_DES_CBC_PAD 0x00000125
- DES-CBC 암호화 !

Binary l33tness 500

- 관련 예제 소스를 찾아보면

```
/* Set the encryption mechanism to CKM_DES_CBC_PAD */
    mechanism.mechanism = CKM_DES_CBC_PAD;
    mechanism.pParameter = des_cbc_iv;
    mechanism.ulParameterLen = 8;
```

- cbc_iv 값이 필요
  - main 앞부분에서

```
.text:0000000100000C5C                  set     0x100000, %l1
.text:0000000100000C64                  sllx    %l1, 12, %l1
.text:0000000100000C68                  bset    0xFE0, %l1
.text:0000000100000C6C                  add     %fp, arg 78F, %l3

.rodata:0000000100000FE0                .quad 0xDEADBEEFBAADF00D
```

  - 0xDEADBEEFBAADF00D가 iv로 추정됨

Binary l33tness 500

- iv뿐만 아니라 key도 필요
  - k라는 이름의 전역변수가 보인다.

```
.data:0000000100101668                  .global k
.data:0000000100101668 k:               .quad 0xDD73CC7FDD7ECC7F
```

- iv와 key로 .dat파일을 디코딩

openssl enc -d -des-cbc -in c500_b6427ab1a64e6836.dat -K DD73CC7FDD7ECC7F -iv DEADBEEFBAADF00D

allthatworkanditsjustDES!?!

- allthatworkanditsjustDES!?!

Pwtent Pwnables 100

- Running on pwn16.ddtek.biz. Pass: zenata
- FreeBSD mail service(?) server
- server의 시작 시간에 대한 cookie가 stack에 존재

$ nc 127.0.0.1 49217
Password: zenata
220 youdont.own.me C-Mail service ready at Wed May 26 00:58:53 2010
UPTM
250 2:32:22

Pwtent Pwnables 100

- 프로그램 구조
  - 명령어를 입력 받고 lookup함수를 통해 알맞은 handler 함수를 선택, 실행
- 취약점
  - data() handler 함수에서 data를 입력 받을 때 BOF취약점 발생
  - 0x100 byte의 buffer에 0x200 byte를 입력 받음

Pwtent Pwnables 100

```
FILE *stream; // [sp+38h] [bp-110h]@5
char v7; // [sp+3Ch] [bp-10Ch]@1

if ( fgets(&v7, 256, stream) )
{
  v11 = lookup(&v7);
  if ( v11 < 0 )
    fwrite("502 HuhWn?", 1u, 9u, stream);
  else
    v5 = ((int (__cdecl *)(FILE *, char *))handlers[v11])(stream, &v7);
}
int __cdecl data(FILE *s, char *s1)
{
  int v3; // [sp+24h] [bp-4h]@1

  v3 = secret;
  fwrite("354 End data with <CR><LF>.<CR><LF>WrWn", 1u, 0x25u, s);
  do
    fgets(s1, 0x200u, s);
  while ( strcmp(s1, ".WrWn") && strcmp(s1, ".Wn") );
  fwrite("250 2.0.0 Ok: queued as 9BDF718A98WrWn", 1u, 0x24u, s);
  if ( v3 != secret )
    _exit(1);
  return 0;
}
```

Pwtent Pwnables 100

- time based cookie
  - 처음 접속 시 알려주는 현재시간과 UPTM 명령어를 쳤을 때 보여주는 경과 시간의 차이로 서버 시작시간을 알아냄
  - server time zone에 따른 차이를 추측 (-12~+12 시간차)
  - 구한 시간을 이용하여 같은 방식으로 cookie를 계산

Pwtent Pwnables 100

- 공격 방법
  - BOF를 통해 return address 변조
  - time based cookie 우회
  - read 함수 등을 이용하여 reverse shellcode를 임의의 주소에 받아서 실행하도록 공격 코드 구성

- 공격!!!!

Pwtent Pwnables 400

- 작년 PP400과 동일한 문제
- Mach-O executable, PowerPC !!
- 익숙하지 않은 assembly
  - 프로그램의 흐름 파악이 중요

Pwtent Pwnables 400

- 실행 결과
  - 4138 포트로 서버 형태로 실행

- 접속 결과

nc localhost 4138
Send me some floats (max of 16), I will tell you some stats!
1 2 3^D
The average of your 3 numbers is 2.000000
The standard deviation of your 3 numbers is 0.816497

- float 값을 입력 받음
  - 16개 이상도 입력가능

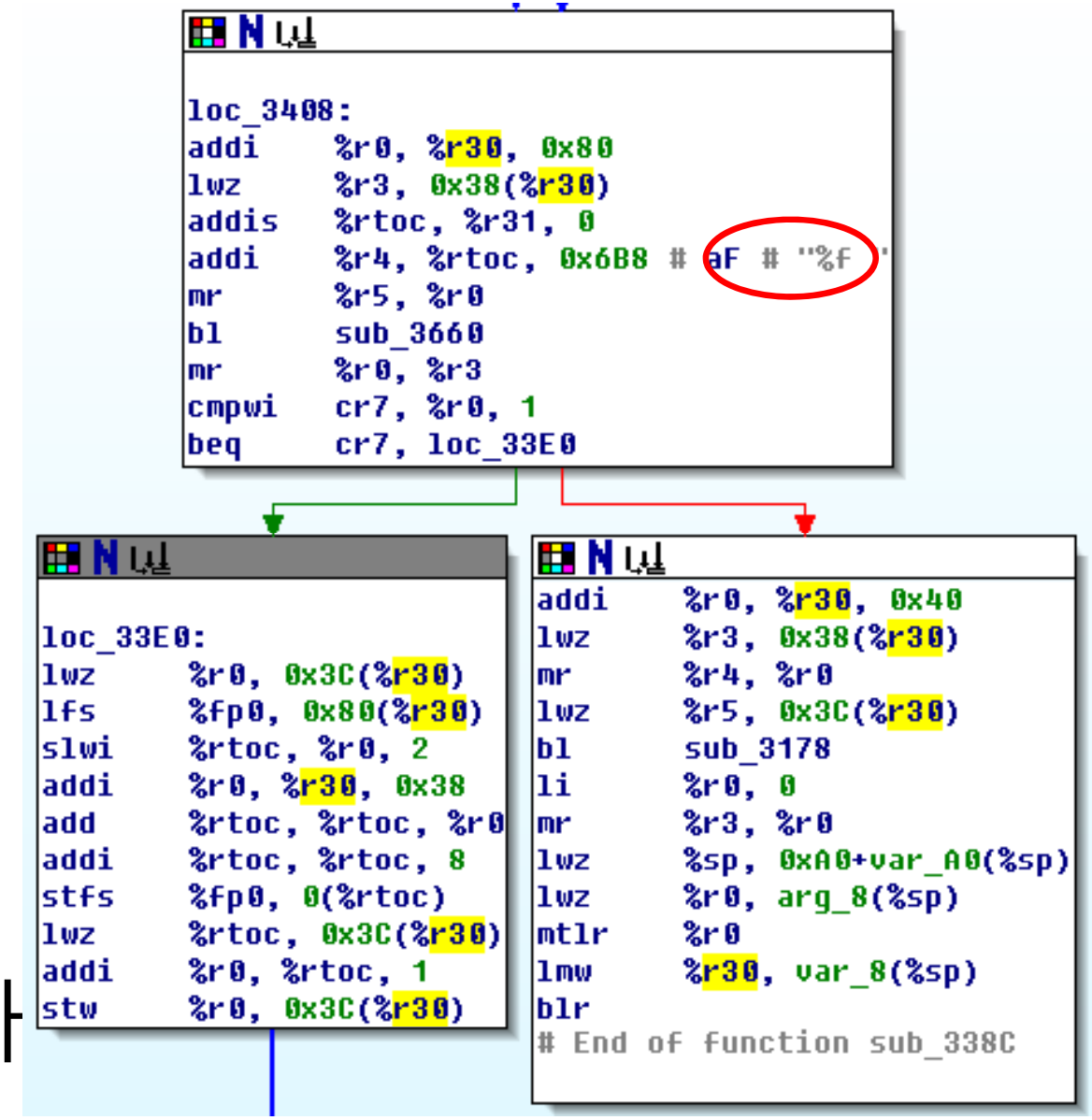
Pwtent Pwnables 400

- 분석
  - 클라이언트 handler를 찾기 위해 “Send me some floats”를 이용
  - sub_338C 함수가 handler

```
sub_338C:

.set var_A0, -0xA0
.set var_8, -8
.set arg_8,  8

mflr    %r0
stmw    %r30, var_8(%sp)
stw     %r0, arg_8(%sp)
stwu    %sp, -0xA0(%sp)
mr      %r30, %sp
bcl     20, 4*cr7+so, loc_33A4

loc_33A4:
mflr    %r31
stw     %r3, 0xB8(%r30)
li      %r0, 0
stw     %r0, 0x3C(%r30)
lwz     %r3, 0xB8(%r30) # int
addis   %rtoc, %r31, 0
addi    %r4, %rtoc, 0x674 # aRb # "rb+"
bl      _fdopen
mr      %r0, %r3
stw     %r0, 0x38(%r30)
lwz     %r3, 0x38(%r30)
addis   %rtoc, %r31, 0
addi    %r4, %rtoc, 0x678 # aSendMeSomeFloa
bl      sub_35B4
b       loc_3408
```

Pwtent Pwnables 400

- 입력 받는 루틴에서 "%f "
- scanf 같은 함수를 쓰는 것으로 보인다
- float 값을 반복적으로 받아 저장



Pwtent Pwnables 400

- 공격 테스트

```
$ python -c 'print " ".join(map(str, range(100))), "₩4"' | nc
    localhost 4138
```

- 0~100 까지의 숫자 입력
  - crash 발생 !!
  - 숫자들이 stack에 저장되어 가다 return address를 변조하게 됨
  - gdb를 이용하여 몇 번째에 return address가 변조 되는지 검사

Pwtent Pwnables 400

- 공격 방법
  - 쉘코드가 float 형태로 스택에 저장되도록 4바이트씩 float 값으로 바꿔서 입력
  - return address를 쉘코드의 위치로 변조(nop sleding)
- 공격 !!!

후기

- 너무 많은 함정들 ㅠ_ㅠ
  - 답을 찾아도 답인지 아닌지...
  - 불필요한 코드 & 암호들
- Binary l33tness
  - 분석보다 실행에 초점
- 운영에 대한 아쉬움
  - pp500 유출 !
  - pp400은 작년 그대로 !!

후기

- ddtek ..... -_-;;
  - I'm a sheep lover
  - video & pics...
- GoN
  - lucky
  - googled pwnable400 !!

모범생 이벤트 !!

- 다양한 OS, Architecture의 문제들이 나왔는데요..
- 오늘 나온 OS와 Architecture들을 맞춰주세요 !

모범생 이벤트 !!

- Bin100 – 1. ???, Linux
- Bin200 – x86-32, 2. ???
- Bin300 – 3. ???, Linux
- Bin400 – x86-32, Linux
- Bin500 – 4. ???, Solaris
- PP100 – x86-32, 5. ???
- PP400 – 6. ???, Mac OS X

모범생 이벤트 !!

모범생 이벤트 !!

- Bin100 – 1. x86-32, Linux
- Bin200 – x86-32, 2. haiku-OS
- Bin300 – 3. x86-64, Linux
- Bin400 – x86-32, Linux
- Bin500 – 4. SPARC 64, Solaris
- PP100 – x86-32, 5. FreeBSD
- PP400 – 6. PPC, Mac OS X

References

- http://ddtek.biz/
  - 출제진 홈페이지
- http://www.vnsecurity.net/2010/05/defcon-18-quals-writeups-collection/
  - 풀이 모음