DOTA: Laboratory #2 (Crypto)

Perform the laboratory individually. Get comfortable using the tools. Note that if you want, you can install ALL these programs on your own machine as well to try them.

1 Lab 2, part 1: WSE

Today we have a case study of the Windows Scripting Encoder. Even simple encryption schemes still find use today! In particular, we will investigate a slightly simplified version of the Windows Scripting Encoder, provided by Microsoft a few years ago to "encrypt" the program code of programs running on webservers. The scripting engine itself could decode and execute the program code, but it looks encrypted to anyone else. The motivation behind the Scripting Encoder was to prevent an attacker who illegally downloaded these programs from gaining any information about how the program works. Quite often programs written from the web also contain passwords for database servers hidden among the program code. The Scripting Encoder's aim was to hide these from an attacker. In this part of the laboratory, your goal is to investigate how it works and to recover a password!

Firstly, open a terminal window in Unix. Now download the wse encryption program - you can get it by using wget, a useful tool for retreiving files:

```
t0XXXXXXX@os103:~$ wget https://hugh.comp.nus.edu.sg/DOTA/lab2/wse
t0XXXXXXX@os103:~$ chmod +x wse                     # now make it executable
t0XXXXXXX@os103:~$ ls -l wse                        # look at the attributes of the file
t0XXXXXXX@os103:~$ file wse                         # and it's type
```

The `chmod` tells the system that this is a program (an executable file). Create some test data files to be encrypted. Each file can have a small amount of text in it (sample passwords in, say, `sample1.txt`, `sample2.txt`...). Use `wse` to encrypt them:

```
t0XXXXXXX@os103:~$ wse < sample1.txt
```

Alternatively, you can just enter in strings directly from the console window "`wse`". The password for the DOTA grading administrator has been found by harriet-the-hacker in a wse-encrypted file at:

```
https://hugh.comp.nus.edu.sg/DOTA/lab2/DOTA2024Lab2Password.encoded
```

Use `wget` to get the file, and then look at it by typing “`cat DOTA2024Lab2Password.encoded`”. It is a wse-encrypted version of the administrator’s password. Really. If you can discover the original administrator password, you can input your grade through the web page, as before. Your goal is to figure out how the encoding works so you can learn the password!

During the lecture, we discussed a few possible attacks against encryption systems. Think about the chosen ciphertext and chosen plaintext attacks. Perhaps one of them would be helpful to you here. Now it’s time to investigate! You can try to encrypt as many messages as you want by running the “`wse`” program.

Once you find the administrator password - please do not share it with others in the class. Access the DOTA grading website, and enter your username, password, and the administrator’s password, along with whatever you want to give yourself as a mark:
`https://hugh.comp.nus.edu.sg/DOTA/lab2/gradeslab2-1.php`
You can give yourself whatever mark you believe you will in the future deserve.

2 Lab 2, part 2: Encryption and decryption using different ciphers

The learning objective of this lab is for students to get familiar with the concepts in the secret-key encryption. After finishing the lab, students should be able to gain a first-hand experience on encryption algorithms, encryption modes, paddings, and initial vector (IV). Moreover, students will be able to use tools and write programs to encrypt/decrypt messages.

In this task, we will play with various encryption algorithms and modes. You can use the following `openssl enc` command to encrypt/decrypt a file. To see the manuals, you can type `man openssl` and `man enc`. In the following command, replace the `ciphertype` with a specific cipher type, such as `-aes-128-cbc`, `-aes-128-cfb`, `-bf-cbc`, etc.

```
openssl enc ciphertype -e  -in plain.txt -out cipher.bin\
    -K 00112233445566778889aabbccddeeff -iv 0102030405060708
```

What size of key is being used above? In this task, you should try at least 3 different ciphers and three different modes. You can find the meaning of the command-line options and all the supported cipher types by typing `"man openssl-enc"`. We include some common options for the `openssl enc` command in the following:

```
-in <file>     input file
-out <file>    output file
-e             encrypt
-d             decrypt
-K/-iv         key/iv in hex is the next argume nt
```

Once you are familiar with this use of the `openssl` program, retrieve an encrypted file from the server:

```
wget https://hugh.comp.nus.edu.sg/DOTA/lab2/Lab2-2.bin
```

This is an encrypted message, using the exact same key and IV given above. Decrypt it - it is a secret English phrase you can use to give yourself your heart's desire!

Access the DOTA grading website (you know the drill by now):
`https://hugh.comp.nus.edu.sg/DOTA/lab2/gradeslab2-2.php`
You can give yourself whatever mark you believe you will in the future deserve.

3 Lab 2, part 3: Two writeup tasks

3.1 Writeup task 1: Encryption Mode – ECB vs. CBC

The file `https://hugh.comp.nus.edu.sg/DOTA/lab2/pic_original.bmp` contains a simple picture. We would like to encrypt this picture, so people without the encryption keys cannot know what is in the picture. Please encrypt the file using the ECB (Electronic Code Book) and CBC (Cipher Block Chaining) modes, and then do the following:

1. Let us treat the encrypted picture as a picture, and use a picture viewing software to display it. However, For the `.bmp` file, the first 54 bytes contain the header information about the picture, we have to set it correctly, so the encrypted file can be treated as a legitimate `.bmp` file. We will replace the header of the encrypted picture with that of the original picture. You can use a hex editor tool (e.g. `ghex` or `Bless`) to directly modify binary files.
2. Display the encrypted picture using any picture viewing software. Can you derive any useful information about the original picture from the encrypted picture?

3.2 Writeup task 2: Encryption Mode – Corrupted Cipher Text

To understand the properties of various encryption modes, we would like to do the following exercise:

1. Create a text file that is at least 64 bytes long.
2. Encrypt the file using the AES-128 cipher.
3. Unfortunately, a single bit of the 30th byte in the encrypted file got corrupted. You can achieve this corruption using a hex editor.
4. Decrypt the corrupted file (encrypted) using the correct key and IV.

Please answer the following questions:

1. How much information can you recover by decrypting a corrupted file, if the encryption mode is ECB, CBC, CFB, or OFB, respectively?
2. Please explain why.
3. What are the implication of these differences?

3.3 Submission of your writeup tasks

Your task: write a short (less than 8000 characters) lab report to describe what you have done and what you have observed, in particular - for the "Writeup Tasks" 3.1 and 3.2. For task 3.11, you should show your results, and explain what you are seeing. For task 3.2, you should answer the three questions in as concise a way as you can.

Once you have your plan ready, both you and your lab partner should access the DOTA grading website to enter in your plan:

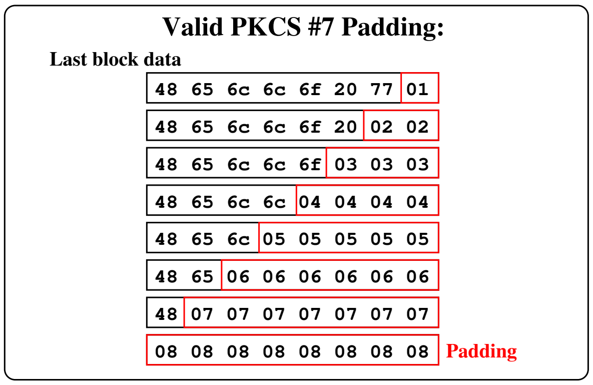
`https://hugh.comp.nus.edu.sg/DOTA/lab2/gradeslab2-3.php`

You will only see your own plan, not everyone else's. The laboratory will remain online and open until Saturday midnight.

4 Lab 2, part 4: Padding oracle - for extra credit!

4.1 The scenario...

The file `https://hugh.comp.nus.edu.sg/DOTA/Files/oracle` is a linux program that is a padding oracle for an AES-CBC encrypted message. Download it and make it executable as before. What "padding oracle" means, is that if you give the program an encrypted message, it will immediately tell you if the padding is correct. The type of padding used in this system is byte-wise:



Valid PKCS #7 Padding:

Last block data

```
48 65 6c 6c 6f 20 77 01
48 65 6c 6c 6f 20 02 02
48 65 6c 6c 6f 03 03 03
48 65 6c 6c 04 04 04 04
48 65 6c 05 05 05 05 05
48 65 06 06 06 06 06 06
48 07 07 07 07 07 07 07
08 08 08 08 08 08 08 08
```

Padding

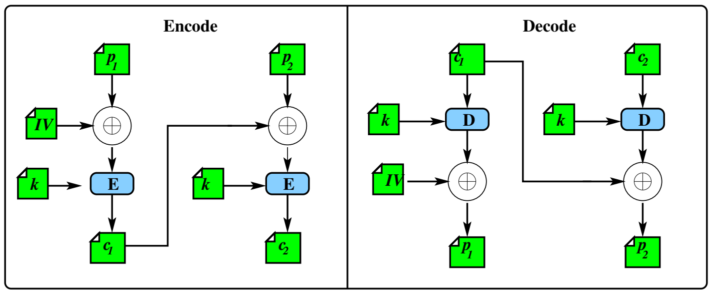
The following 32 byte hex value is the 16 byte IV, and a 16 byte encrypted password. If you pass this value to the oracle program, it will return immediately with no errors:

```
CA1DB62557DDD1AD98953B7C8EB43F85 8B2B9DCD0A6633A612E415CE64B42CF1
```

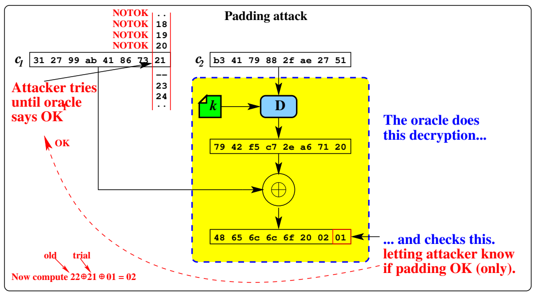
However if you change any byte, you may get an error message, that the padding is incorrect:

```
$ ./oracle
    CA1DB62557DDD1AD98953B7C8EB43F85 8B2B9DCD0A6633A612E415CE64B42CF1
$ ./oracle CA1DB62557DDD1AD98953B7C8EB43F85 8B2B9DCD0A6633A612E415CE64B42CF1
$ ./oracle CA1DB62557DDD1AD98953B7C8EB43F85 8B2B9DCD0A6633A612E415CE64B42CF0
00000000:error:1C800064:Provider routines:ossl_cipher_unpadblock:bad decrypt::0:
$
```

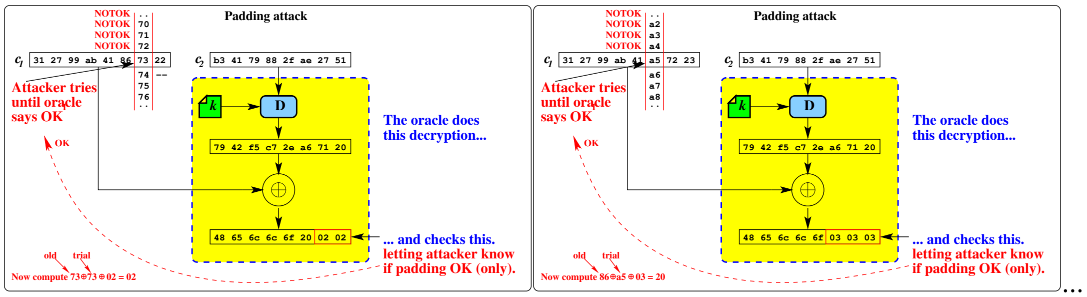
With CBC, when the first block is encrypted, the IV is XORED with the message and then the result is encrypted. To decrypt this you reverse the order, so you decrypt the ciphertext and then XOR the result with the IV. When the second block is encrypted, the first ciphertext block is used to XOR with the second block in place of the IV, and so on.



We can make use of this, because the decryption oracle will XOR the decrypted message and then check the padding. By changing the last byte of the IV, we can change the last byte of the final decrypted message, and so we can set it to (say) 0x01 - which is a correct padding:


Padding attack
NOTOK
NOTOK
NOTOK
NOTOK
18
19
20
21
23
24
c_1 31 27 99 ab 41 86 73
c_2 b3 41 79 88 2f ae 27 51
Attacker tries until oracle says OK
OK
k
D
79 42 f5 c7 2e a6 71 20
48 65 6c 6c 6f 20 02 01
The oracle does this decryption...
... and checks this.
letting attacker know if padding OK (only).
old
trial
Now compute 22⊕21⊕01 = 02

After this, we know the final byte is 0x01, and can deduce the final byte of the decrypted text is 0x20 (0x21 XOR 0x01), and can now set the final byte of the decrypted message to 02, and try the next byte. Of course in this case, the REAL values are 0x02, and 0x02, so we carry on setting the last values to 0x030x03... This process reveals all the bytes of the decrypted message, and since we know the original IV, we can find the plaintext:


Padding attack
NOTOK
70
71
72
74
75
76
c_1 31 27 99 ab 41 86 73 22
c_2 b3 41 79 88 2f ae 27 51
Attacker tries until oracle says OK
OK
k
D
79 42 f5 c7 2e a6 71 20
48 65 6c 6c 6f 20 02 02
The oracle does this decryption...
... and checks this.
letting attacker know if padding OK (only).
old
trial
Now compute 73⊕73⊕02 = 02
Padding attack
NOTOK
a2
a3
a4
a6
a7
a8
c_1 31 27 99 ab 41 a5 72 23
c_2 b3 41 79 88 2f ae 27 51
Attacker tries until oracle says OK
OK
k
D
79 42 f5 c7 2e a6 71 20
48 65 6c 6c 6f 03 03 03
The oracle does this decryption...
... and checks this.
letting attacker know if padding OK (only).
old
trial
Now compute 86⊕a5⊕03 = 20
...

Your task in this lab is to answer the following questions:

1. How many padding bytes are there?
2. What is the last character of the password?
3. What is the second-to-last character of the password?
4. What is the secret password?

Once you have your answers ready, access the DOTA grading website to enter in your plan:
`https://hugh.comp.nus.edu.sg/DOTA/lab2/gradeslab2-4.php`
You will only see your own answer, not everyone else's. The laboratory will remain online and open until Saturday midnight.