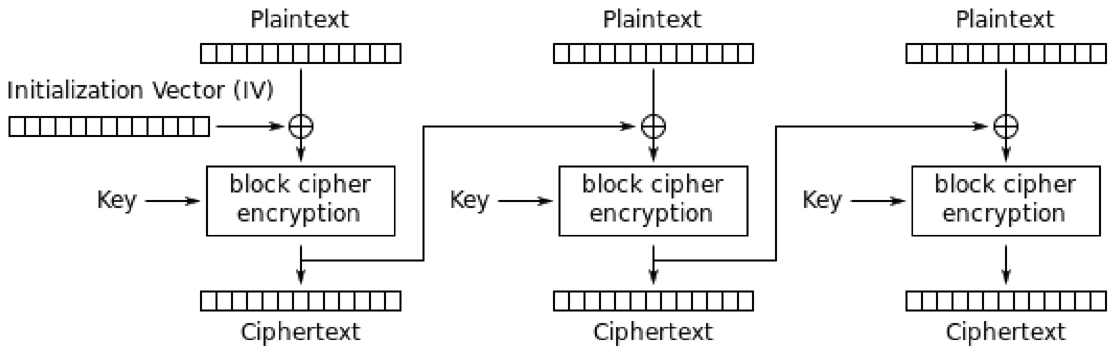
Web 400 (CryptoMatv2)

by HoLyVieR

First let’s look at what “AES 128 CBC” is. “AES 128” is an encryption algorithm that can encode 128 bits (16 ASCII char) of information at the time. The algorithm takes 2 parameters; a block (128 bits) and a key (any size). Since we want to encode more than 128 bits this is where “CBC” comes in. “CBC” is an encryption mode, it dictates how to use the algorithm to encrypt more than 1 block. This image from Wikipedia explains CBC well:



Cipher Block Chaining (CBC) mode encryption

The circle with a plus inside are XOR operation. The IV is a special value that is fed into the algorithm. In the case of the CryptoMatv2 application, this value is unknown (we will get a bit later).

Just a few more definition before we get to the meat. For this writeup I will use the following notation.

AES_ENCRYPT(data, key, iv) : Encryption with AES CBC.
AES_DECRYPT(cyphertext, key, iv) : Decryption with AES CBC.

Let’s look at how the application uses AES 128 CBC, in details. When it encrypts the message it does the following:

AES_ENCRYPT(text, key, constant_iv)

Here “text” and “key” are both input that you define in the form to send a message. “constant_iv” is an unknown value, but we know that it’s constant for all the messages.

The first step is to find the “constant_iv” (IV) used in the application. To do this we need to encrypt a message (M) with a key (K). This will give an output (C). If we follow the CBC diagram we can see that “IV XOR M” was fed in AES with the key “K” and it gave the output “C”. We decrypt M assuming an IV consisting of only null byte was used. The result of the operation will be “IV XOR M”. From that point, since we know M, we can do “IV XOR M XOR M” which will give “IV”. In term of formula it gives the following:

```
C = AES_ENCRYPT(M, K, IV) // This operation is done on the server
D = AES_DECRYPT(M, K, NULL_BYTES x 16) // Same value as “M XOR IV”
IV = D XOR M
```

Now we have all the information needed to do anything with the encryption used in the application.

The second part is about exploiting the SQL injection in the search page. What the search page essentially do is the following:

```
$sql = "SELECT * FROM message WHERE text = \"" . aes_encrypt($_GET["text"], $_GET["key"]) . "\" ORDER BY id ASC LIMIT ?, 0";
```

To exploit the vulnerability, we need to forge value so that when encrypted it injects the SQL we want. This step is rather easy, but a bit conter-intuitive. We start by creating a string “S” that we want to inject. Then we pad S with space so that it’s length is a multiple of 16. Then we use the decryption function of AES on S. This gives a value that when encrypted the result is the string we want. In term of formula it gives the following:

```
T = "\" UNION SELECT …  -- "
T = PAD_WITH_SPACE(T)
Q = AES_DECRYPT(T, "test", constant_iv)
```

Q is the value that we pass in the query along with ‘test’ as the key. And when the server receives the value, it does this:

```
D = AES_ENCRYPT(Q)
SQL = "…" + D + "…" // Equals "…" + T + "…"
```

At this point, you can extract everything from the database. The first step is to find how much column the query you are trying to append data from has. You basically just need to try them one by one until the mysql error about column mismatch doesn’t show up.

The second step is to query information_schema.tables to find all the table in the database.

```
T = "\" UNION SELECT 1, 2, 3, 4, 5, TABLE_NAME, 7, 8 FROM information_schema.tables WHERE 0 = ? -- "
```

Then you check the information of the table you want to extract data

```
T = "\" UNION SELECT 1, 2, 3, 4, 5, COLUMN_NAME, 7, 8 FROM information_schema.columns WHERE name = 'message' and 0 = ? -- "
```

Then you extract the message that dog has received or sent.

```
T = "\" UNION SELECT HEX(text), 2, 3, 4, 5, HEX(`key`), 7, 8 FROM message WHERE to_user_id = 1 or from_user_id = 1 and 0 = ? -- "
```

The key used to encrypt the message are stored in the database, you just have to use them to decrypt the message. They key is in the first few messages.