

hatstack / writeups

Watch1

Star0

Fork0

<> Code

Issues 0

Pull requests 0

Projects 0

Pulse

Graphs

Branch: master writeups / defcamp2015 / crypto200 /

Create new file


Find file

History

 bef0re Final fix

Latest commit 638bde1 on Oct 5 2015

..

 README.md

Final fix

2 years ago

 README.md

# No Crypto (Crypto 200)

The folowing plaintext has been encrypted using an unknown key, with AES-128 CBC:

Original: Pass: sup3r31337. Don't loose it!

Encrypted: 4f3a0e1791e8c8e5fefe93f50df4d8061fee884bcc5ea90503b6ac1422bda2b2b7e6a975bfc555f44f7dbcc30aa1fd5e

IV: 19a9d10c3b155b55982a54439cb05dce

How would you modify it so that it now decrypts to: "Pass: notAs3cre7. Don't loose it!"

This challenge does not have a specific flag format.

I don't have prior knowledge about AES-128 CBC. Wikipedia explains a little:

[https://en.wikipedia.org/wiki/Block\\_cipher\\_mode\\_of\\_operation#Cipher\\_Block\\_Chaining\\_.28CBC.29](https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation#Cipher_Block_Chaining_.28CBC.29)

AES is a block cypher. That means that for encryption the input is first split in blocks, and then each block is encrypted individually. CBC is a way of encrypting blocks so that a change in one block also changes other blocks.

It's AES-128, so it uses blocks of 128 bits = 16 bytes. These are our blocks:

block 1: Pass: sup3r31337  
block 2: . Don't loose it  
block 3: !

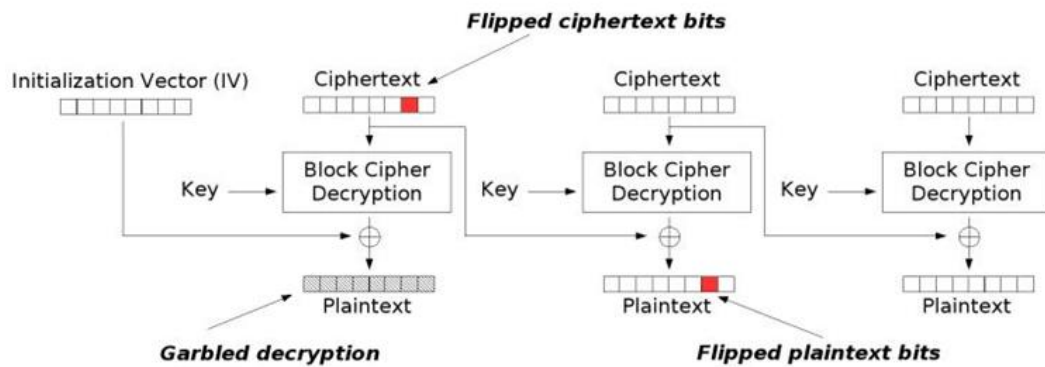
Block 3 gets appended with \x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00e The last byte is the amount of bytes the decrypter should ignore.

I searched the web with my favorite search engine for known attacks on CBC and it came up with these results:

0. <http://www.jakoblell.com/blog/2013/12/22/practical-malleability-attack-against-cbc-encrypted-luks-partitions/#toc-2>
1. <http://resources.infosecinstitute.com/cbc-byte-flipping-attack-101-approach/>

The idea of these attacks is: By XORing the ciphertext of block N with x, we corrupt block N and XOR the plaintext of block N+1 with x.

This image from the resources.infosecinstitute.com link is pretty neat:



### Modification attack on CBC

The attacks work because the plaintext gets XOR'ed with the previous ciphertext after going through AES, and we can control all of the ciphertext. As a side effect, we mess up the original block completely.

We can't afford to mess up a block however. We can maybe afford to mess up some bits from the last block, because the appended bytes should be ignored. Also, the attacks don't allow modification of the first block because modifying the previous ciphertext is required.

If only we could modify the IV...

Oh wait! We don't necessarily need to modify the ciphertext. The question is: "How would you modify it", so maybe patching the IV will do:

```
#!/usr/bin/python

# python normally only xor's integers. This function is for xor-ing strings
def xor(xs, ys):
    return "".join(chr(ord(x) ^ ord(y)) for x, y in zip(xs, ys))

iv_hex = "19a9d10c3b155b55982a54439cb05dce"
iv = iv_hex.decode("hex")

block_a_original = "Pass: sup3r31337"
block_a_target = "Pass: notAs3cre7"

# Get the difference between the original and target to know what bits to flip:
block_a_patch = xor(block_a_original, block_a_target)

# flip the bits and print the flag!
print xor(iv, block_a_patch).encode("hex")
```

**Congratulations!** Challenge solved. You received 200 points.

