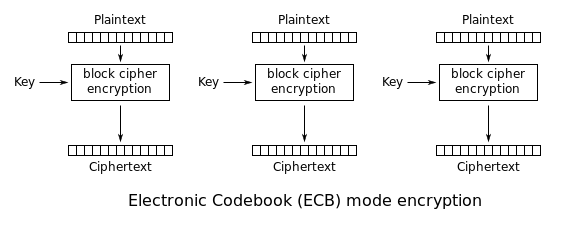
Chào các bạn, sau đây là 1 vài cách tấn công AES mode ECB và CBC trong CTF mà mình đã từng “ăn hành” và lấy nó làm kinh nghiệm “xương máu” đến giờ.

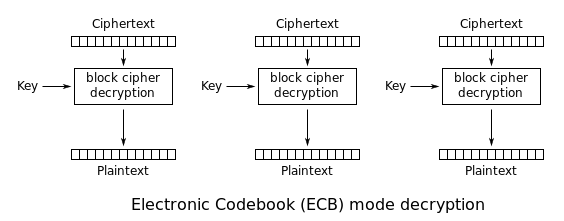


**Tổng quan về AES mode CBC và ECB**

1. **AES – ECB**



Quá trình mã hóa



Quá trình giải mã

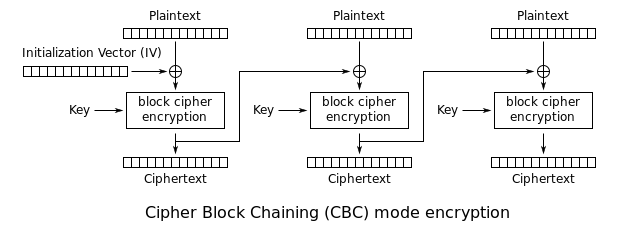
Yêu cầu để mã hóa AES mode ECB:

* Key phải có độ dài chia hết cho 32
* Plaintext phải có độ dài chia hết cho 16
* Mỗi block có độ dài = 16
* Nếu bạn dùng openssl để mã hóa thì cơ chế của PKCS1 sẽ tự động thêm các padding ở plaintext sao cho độ dài plaintext phải chia hết cho 16.

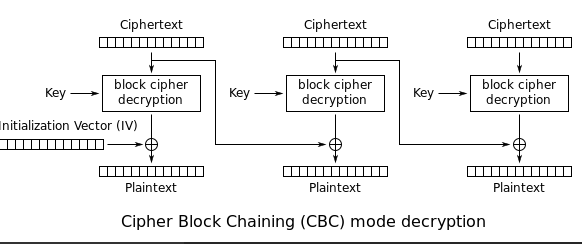
Có thể tóm tắt quá trình mã hóa của AES ECB như sau với plaintext được chia thành các block có size = 16

Ta có quá trình giải mã như sau

1. **AES CBC mode**



Quá trình mã hóa



Quá trình giải mã

Yêu cầu để mã hóa AES mode ECB:

* Key phải có độ dài chia hết cho 32
* Plaintext phải có độ dài chia hết cho 16
* Mỗi block có độ dài = 16
* Độ dài của vector IV = 16
* Nếu bạn dùng openssl để mã hóa thì cơ chế của PKCS1 sẽ tự động thêm các padding ở plaintext sao cho độ dài plaintext phải chia hết cho 16.

Ta có ^ E(Pi , k )như sau

hìddi PKCS1 sẽ tự động thêm các padding ở plaintext sao cho độ dài plaintext phải chia hết cho 16 quá trình mã hóa như sau với plaintext được chia thành các block có size = 16

…..

Ta có quá trình giải mã như sau

**Các dạng bài toán thường gặp**

1. **AES ECB cho ta key hoặc AES CBC cho cả key và IV**

**Vậy vấn đề của ta bây giờ chỉ là code để giải mã :D Thư viện pycrypto hỗ trợ cho chúng ta việc này**

**1.1 Quá trình mã hóa và giải mã với AES ECB**

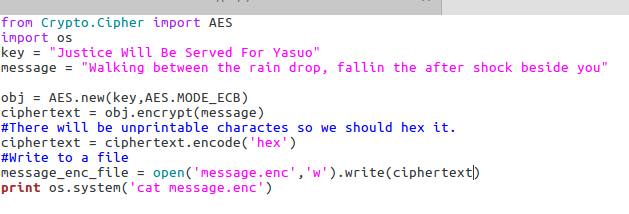
- Giả sử ta có **message = “Walking between the rain drop, fallin the after shock beside you”**

-> Có độ dài là 64 thỏa điều kiện **len(message) % 16 = 0**

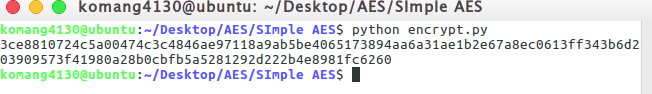
- Tiếp theo ta có **key = “Justice Will Be Served For Yasuo”**

-> Có độ dài key là 32, thỏa điều kiện **len(key) % 32 == 0**

- Ta tiến hành mã hóa message với code đơn giản:



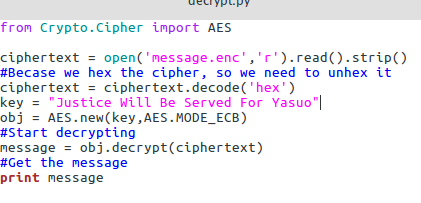
Output



Ta đã mã hóa và ghi vào file thành công :D

Giờ ta sẽ tiến hành trả lời cho câu hỏi nếu như ta nhận được 1 file hoặc 1 Message đã bị mã hóa bởi AES ECB mode và key , làm sao ta có thể phục hồi được Plaintext và Message ban đầu :D

Câu trả lời rất đơn giản :D code thôi



Output:



Ta đã phục hồi được message bị mã hóa :D

**1.2 Quá trình mã hóa và giải mã với AES CBC**

Ta tiếp tục sử dụng **message** và **key như trên**

Giả sử bạn có n message muốn mã hóa, và chúng có cùng block 1 -> attacker có thể lấy được thông tin từ block 1 bằng cách brute force chẳng hạn :D, mặc dù rất khó để tìm lại, nhưng đó cũng là 1 điều không tốt, nên người ta đã sử dụng thêm 1 vector IV để tránh trường hợp trên

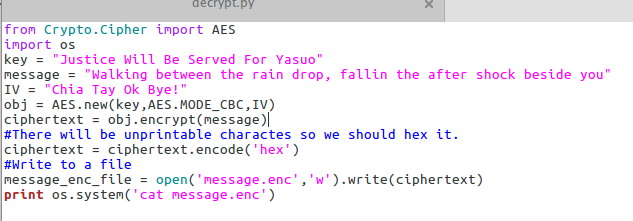
**message = “Walking between the rain drop, fallin the after shock beside you”**

**key = “Justice Will Be Served For Yasuo”**

Ta chọn **IV = “Chia Tay Ok Bye!”**

Thỏa điều kiện **len(IV) % 16 = 0**

Tiến hành mã hóa bình thường :D Code thôi

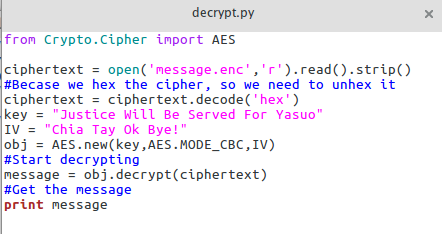


Output

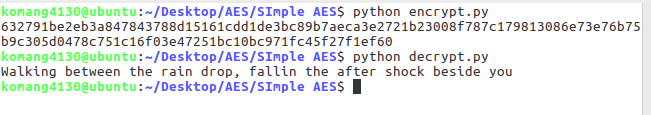


Ok mã hóa thành công :D

Giờ thì giải mã thôi :D Code



Output



Tuyệt vời

Vậy là mình đã giới thiệu xong dạng dễ nhất :D

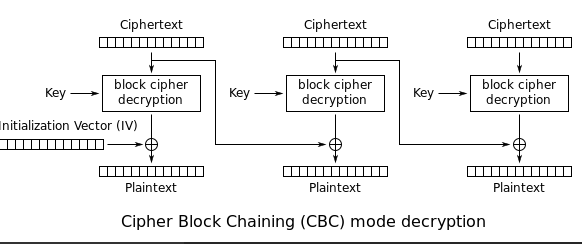
Bây giờ ta đi sâu vào vấn đề hơn 1 chút, giả sử với **AES CBC** đề chỉ cho ta **IV** thì như thế nào?

Ta có thể tìm lại **key** từ **IV** được hay không ?

**Trả lời: Không nhé, vẫn phải brute tìm key hoặc dựa vào các gợi ý để tìm key.**

Đầu tiên bạn cần phải nhớ rằng, known-plaintext attack không thể tấn công AES ở hầu hết các mode ( Mode AES-CTR nếu sử dụng reused counter thì có thể bị phục hồi bằng cách known- plaintext attack , hoặc reused IV có thể làm leak được message ).

Với AES CBC quá trình giải mã như sau:



Thấy rằng: với

Giả sử

Ta vẫn phải tìm Key , chứ k thể dựa vào IV và Cipher để tìm key.

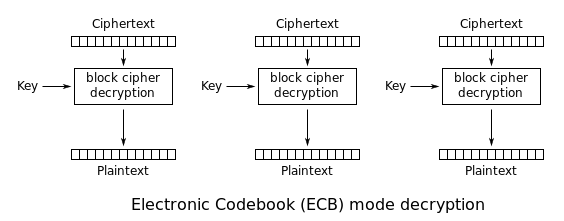
Lúc này để brute thì ta tìm mọi trường hợp của k (thông thường là 32 bytes )

1. **AES CBC hoặc AES ECB không cho key và IV**

**2.1 AES ECB không cho key**

Nếu không cho key, nghĩa là ta chỉ nhận được ciphertext

Vậy làm sao để giải mã với ciphertext, nhìn lại vào quá trình giải mã của AES ECB



Công thức tổng quát :

Quen quá luôn, khác gì trường hợp trên :D

Cách làm là BRUTE hoặc dựa theo hint đề bài cho để tìm key giải mã :D

**2.2 AES CBC không cho key và cho hoặc không cho IV**

**Trả lời: Kéo lên mục ở trước đã trả lời cho vấn đề này :D phần cuối mục 1.2**

**2.2.1 AES CBC không cho key và cho hoặc không cho IV với dạng: padding + input + message**

Trong đó :

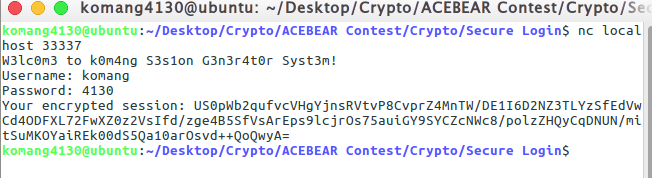
* Padding: có độ dài 0 <= len(padding) <= 16

Riêng với trường hợp này, ta vẫn có thể tìm được message bằng cách brute force message vì ta có thể control được input và đảm bảo dữ liệu theo đúng ý

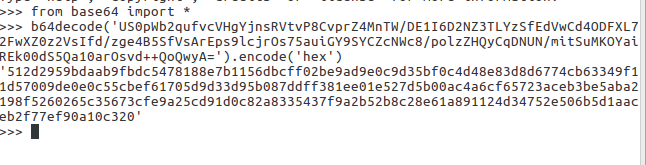
Mình sẽ demo với bài Secure Login trong đợt ACE BEAR CTF vừa rồi :D Do server đã tắt nhưng may mắn mình là người ra đề nên mình có sẵn file đề trong máy :D và dĩ nhiên brute trong local thì tốc độ rất toẹt vời và lí tưởng.

Ok bắt đầu nào

Kết nối đến server



Ta thấy server sẽ yêu cầu ta nhập vào Username và Password. Sau đó sẽ trả về cho ta 1 chuỗi base64 gọi là “Encrypted session” , thử decode và chuyển sang dạng hex đọc xem nào



Hmm, làm sao giờ nhỉ, nhìn hint đề nào

Ok , dĩ nhiên md5 là mã hóa 1 chiều, ta không thể giải mã được :D cơ mà ta biết được độ dài của md5 là 32 bytes hex -> len(k) = 32

Tiếp theo Server sẽ mã hóa các phần sau đây

Phần 1 = ]

Mình biết độ dài của k là 32 rồi, k[:32/3] = k[0:10]

Là sẽ tiến hành lấy 10 chữ cái đầu của k.

Tiếp theo sẽ lấy 10 chữ đầu ấy + user + pwd + flag

Với user và pwd ta có thể kiểm soát được

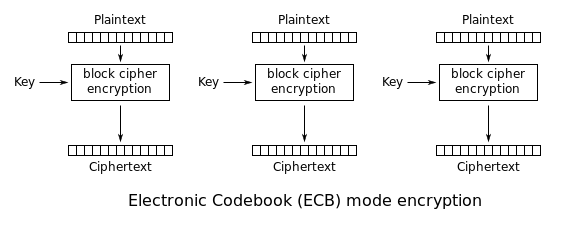
Vậy giờ ta đặt phần mã hóa là message = k[0:10] + input + input + flag

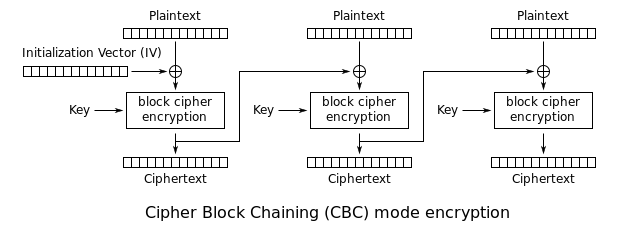
Tiếp theo, ta cần phải xác định xem đây là mã hóa AES CBC hay ECB hay là mode nào đó

Cơ mà mình k cần quan tâm :D

Nhìn vào cách mã hóa của 2 mode đó nào

AES ECB



Và AES CBC

Các bạn thấy rằng với ECB: nghĩa là các block được mã hóa độc lập với nhau

Ta thấy rằng block 1 được mã hóa với E( P1, k ) và block 2 được mã hóa với E( P2,k )

Thấy rõ rằng cả 2 đều không liên quan gì nhau. (1)

Tiếp theo với AES CBC: Yeah, kết quả C1 xor với P2 sau đó tiến hành quá trình mã hóa Ek để ra C2 (2)

(1)(2) để ý block 1 cũng chẳng quan trọng đúng k? giờ ta chỉ việc làm sao cho block 1 đủ 16 bytes. Và tiếp tục đến block 2, ta sẽ điền thêm 15 bytes, vậy lúc này chuyện gì xảy ra?, giả sử ta nhập sao cho đủ block 1 có 16 bytes và tiếp tục thêm 15 bytes ở block 2 nghĩa là 6 + 15 = 21 bytes.

Thì lúc này, quá trình mã của của ta sẽ như thế này với :

P1 P2 P3…..

Vậy giờ ta chỉ cần tìm 1 kí tự nào đó thỏa mã quá trình mã hóa P2 trả về đúng kết quả bằng ciphertext[16:32]

Tiếp tục, ta giảm 1 chữ A và thay thế bằng chữ cái ta tìm được, ta lai tìm được thêm 1 kí tự nữa

P1 P2 P3….

Ta tiếp tục brute tương tự và tìm được kí tự tiếp theo, và cứ như vậy, ta phục hồi được toàn bộ flag.

Cách tấn công này có thể break được cả AES mode CBC và ECB :D

Ok thử viết code làm thử xem sao

Và ở bài này mình biết có nhiều bạn gặp dạng này như cá gặp nước, bỏ thằng cái code brute ngay và luôn :D nên mình đã thêm 1 vài filter đó là độ dài của Username và Password được nhập vào không được quá 43 mỗi loại.

Thử làm ví dụ tìm thử kí tự đầu tiên nào :D yêu cầu các bạn phải biết lập trình mạng nhé, các bạn có thể dùng socket hoặc pwntools

Ok demo tấn công thử để tìm kí tự đầu tiên của flag nào

Trước tiên ta cần phải kết nối đến Server, mình sẽ sử dụng socket

Các bạn khởi tạo kết nối đến server với 1 vài lệnh sau

Tiếp theo ta xây dựng 1 hàm đọc đến 1 đoạn nào đó để hỗ trợ việc làm ăn dễ hơn :D

Vì ta sẽ gửi request lên server liên tục :D nên để tiện hơn ta sẽ xây dựng 1 hàm để gửi

Okay, giờ ta sẽ tạo charset là 1 bộ kĩ tự để test, thay vì brute 256 kí tự ASCII, ta biêt chắc rằng flag sẽ chỉ bao gồm các kí tự trong range(33,128) hay string.printable là các kí tự có thể đọc được.

Ok vậy xong phần setup cho việc gửi đến Server, tiếp theo là phần luận lí

Giờ brute tìm chữ cái đầu flag[0] nào :D

n = 21

flag = ‘’

real = b64decode( payload( n \* ‘A’ ) ) [16:32]

for char in string.printable:

brute = b64decode( payload( n\*’A’ + flag + char ) [16:32]

if brute == real:

flag+= brute

print flag

break

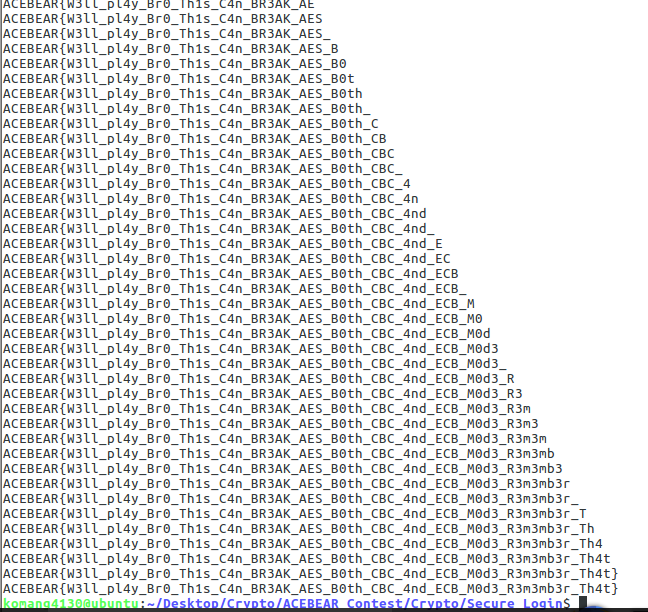
Run code và ta thấy kết quả trả về là chữ A trong chữ ACEBEAR{ ….. } :D tuyệt vời nhỉ, đúng đúng rồi đó, đây là full code của mình, các bạn có thể tham khảo và file Server để các bạn có thể test

Về cách sử dụng thì các bạn chỉ cần chạy python server.py để tiến hành chạy Server.

[Server.py](https://gist.github.com/anonymous/3d99136213036b07d60c0eb64ae8491e)

[Solve.py](https://gist.github.com/anonymous/13e0060a51c8985bc5591127c050232f)

Okay thử chạy file solve.py nào



Đúng là [Magic](https://giphy.com/gifs/reactiongifs-ujUdrdpX7Ok5W)

**2.2.2 AES CBC cho dạng message + padding thì ta có thể tìm được message hay không?**

**Trả lời: ta vẫn có thể tìm được message, đây là dạng Padding Oracle**

Trước khi nói về dạng Padding Oracle, các bạn cần hiểu pad là gì?

Để đơn giản nhất để nói về pad.

Pad có charset = ‘\x01 -> \x10’ ở PKCS7 tương ứng với 1 đến 16

Giả sử bạn có chữ “test” thì với PKCS7, sẽ tự động thêm 16 – len(“test”) = 12 = 0xC \*12 pad

Vậy lúc này khi bạn encode hex thì bạn sẽ nhận được đoạn message sau khi thêm pad là:



Đó là cơ chế làm việc của PKCS7 nếu bạn nào mã hóa aes với openssl thì để ý khi message có len(32), bạn mã hóa thì khi check header sẽ có thêm 16 bytes nữa để kết thúc ở đoạn [32:48], những padding ấy người ta gọi là **Padding Oracle**

Ta biết quá trình mã hóa của AES CBC là:

Block plaintext đầu tiên sẽ xor với IV và được mã hóa với key

Block plaintext thứ I thì sẽ xor với block cipher I – 1 và được mã hóa với key

Tiếp tục quá trình giải mã như sau:

Block cipher đầu tiên sẽ được giải mã với key sau đó xor với IV để giải mã block plaintext đầu tiên

Block cipher thứ I sẽ được mã hóa với key sau đó xor với block cipher thứ I – 1 để giải mã block plaintext thứ i

**Cách tấn công**

**Gọi : block ciphertext C’ mà ta tự chọn để tấn công**

**Gọi ] : phần tử thứ k của block ciphertext**

**Gọi : block plaintext được decrypt từ**

**Gọi phần tử thứ k của block plaintext được decrypt từ**

**Gọi : là block plaintext thứ n**

Ta có:

Ta biết rằng:

Vậy

Ta đã rút gọn lại rồi đấy :D , ta có thể control được giá trị của

là một giá trị nào đó **mà Server sẽ tính toán và trả về** cho ta khi ta tấn công

là block plaintext tại vị trí thứ n

là block ciphertext tại vị trí thứ n-1 cái mà ta có thể biết

Vậy có 2 giá trị mà ta không thể kiểm soát được là và

**1 công thức có 4 biến số nhưng lại có tận 2 ẩn số thì ta không thể giải được**

**Nhưng mà thật sự là hoặc là ta không biết không?**

Để tấn công, ta sẽ sử dụng 1 điểm có thể khai thác được đó là **Padding Oracle**

Giả sử **Padding oracle** xuất hiện lần đầu ở vị trí thứ **k** .

Vậy Pn[k] ta có thể tìm được giá trị của P’2[k] = hex(16 – k)

Như ví dụ “test” như trên, padding bắt đầu xuất hiện ở vị trí thứ 4 vậy là 16 – 4 = 12 = 0xC

Vậy ta đã biết được 3 biến, ta có thể tìm được 1 ẩn hay không ?



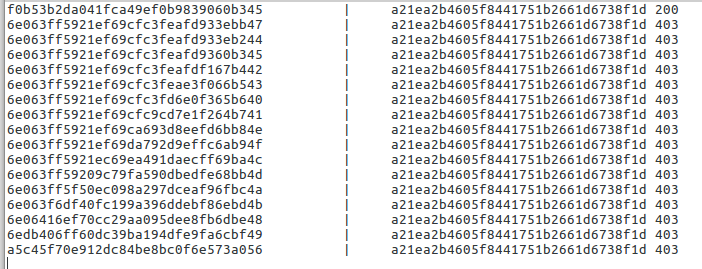
Như vậy ta có thể khôi phục lần lượt từng bytes của plaintext.

Ví dụ:

Ta phân tích 1 log sau đây

[log](https://gist.github.com/anonymous/1f8ccf9f54485ac0710733f75ad8cc72)

**Phân tích**

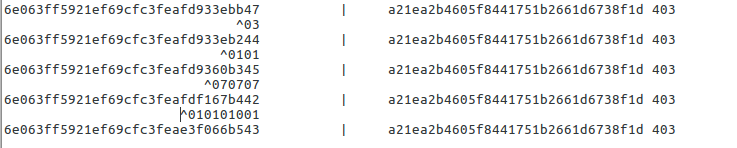


Nhớ lại công thức của ta, dạng bài này có thể ta k cần phải brute bytes by bytes

Đoạn cipher mà có response = 200 có thể chính là Cn-1

Giờ ta cần tìm P’2 và C’ là tìm được Pn :D

Và ta để ý rằng:



Ngoài ra để nhìn nhận nhanh, ta thấy ở các request 403, 2 bytes cuối xor nhau đều ra 0xf6 và bytes[k][-1] và bytes[k+1][-1] khi xor với nhau đều trả về các số trong range 0x01 -> 0x10 , chỉ duy nhất request cuối cùng là không theo nguyên tắc đó 0x49 ^ 0x56 = 0x1f -> có thể đó chính là Attacker plaintext C’

Vậy ta đã có

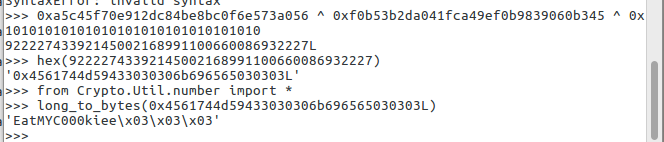
Ta cũng có

Tiếp tục tìm P’2 , ta thấy block tiếp theo của C’ là a21ea2b4605f8441751b2661d6738f1d -> cũng chỉ là **padding** , vậy nên với PKCS7, sẽ trả về cho ta 16 bytes 0x10 padding

Vậy

1 phương trình 4 biến mà ta đã biết 3, vậy ta dễ dàng tìm được biến còn lại

Decode ta nhận được message



Để hiểu rõ hơn các bạn có thể đọc tài liệu tại các trang sau đây

[Padding Oracle Attack](https://blog.skullsecurity.org/2013/padding-oracle-attacks-in-depth)

[Crypto400 CodegateCTF 2011 ( chính là bài trên )](http://mslc.ctf.su/wp/codegate-ctf-2011-crypto-400/)

**3. Bit flipping**

Đây là cách tấn công mà bạn khi biết được các yếu tố sau đây thì bạn có thể ép 1 ciphertext bất kì decrypt thành 1 message nào đó mà bạn mong muốn :D

1. Ta cần phải biết IV, và điều này cũng không đến nỗi quá khó để tìm.
2. Ciphertext
3. Ta cần phải biết được plaintext thật của cipher text đó
4. Đoạn plaintext mà ta cần giả mạo
5. :D bằng 1 cách Magic nào đó ta có thể bắt Server sử dụng 1 IV mà ta đã đánh tráo để giải mã ciphertext

Ok bắt đầu nào

Giờ giả sử 1 tình huống vi diệu

Bạn muốn đốt nhà cặp đôi nào đó :D Ví dụ cặp đôi mà bạn muốn đốt là **M** và **A**

**M: Anh yeu em**

Bạn bắt được đoạn message trên và bạn có IV trong tay cũng như cipher text của đoạn message trên là

**Message = Anh yeu em**

**IV = dayladautoilaanh**

**Ciphertext = bbc939c5d6a82b3a13638b6e7e2d2358417d8811d393f313a2c63decd833bdf7**

Bạn bắt được gói trên, và bạn sẽ chỉnh sửa lại để gửi sang cho bạn A

**Fake Message = Anh khong yeu em**

Và bạn đã hack thành công máy của bạn A , giờ bạn chỉ việc gửi đến cho bạn A IV fake để hệ thống giải mã AES của bạn A decrypt Message trên thành Fake Message.

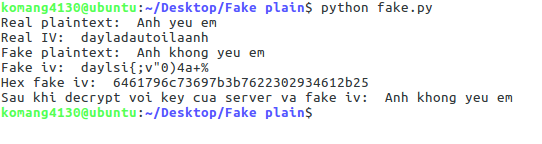
Vấn đề bây giờ là tìm IV fake

Vì quá trình giải mã của AES CBC như sau

Để thay đổi cả plaintext, ta thấy rằng chỉ cần thay đổi IV thì P1 sẽ trả về cho ta 1 kết quả hoàn toàn khác

Ta có

Ta tìm được IV fake và gửi đến cho server sử dụng IV đó để decrypt ra fake message mong muốn :D



Code thì chắc các bạn nhìn cũng tự code được rồi -\_- riêng các bạn miền Bắc thì không code là bị anh Khuyến gank nhá haha :D

Tài liệu của mình hiện tại tới đây là hết, tài liệu tiếp theo có lẽ sẽ về web vì mình đang try hard cày cái này để đi xin việc :v Cám ơn các bạn đã đọc , mọi góp ý hoặc thắc mắc các bạn hãy liên lạc với mình   
Mail: [komang4130@gmail.com](mailto:komang4130@gmail.com)

Facebook: <https://www.facebook.com/thao.lee.54584>

SĐT : md5().hexdigest = fcafa600f54a8bca1e8c2845506d7247