CSCI3381-Cryptography

Project 3: Padding Oracle Attack on CBC-encrypted Messages

September 25, 2014

A ‘chosen-ciphertext attack’ does not at first glance appear to make much sense. If you really had ‘temporary access to the decryption mechanism’, as the textbooks write, why not just give it the ciphertext you’re interested in decrypting and be done with it? There is, however, a realistic scenario where such an attack can be effective: Imagine a server accepts enrypted communications. Of course it does not respond with the plaintext when it decrypts, but let’s suppose that when it *discovers an incorrectly-formatted plaintext, it responds with an error message,* something like ‘This transaction could not be processed.’ Thus the server deliberately leaks information about the plaintext given the ciphertext.

Now imagine an attacker, in possession of intercepted ciphertext, repeatedly alters this ciphertext and collects information about when the altered ciphertext is the encryption of a correctly-formatted message. Is it possible to use this to obtain a decryption of the original ciphertext? of the key?

Amazingly, the answer is ‘yes’ for some real-life implementations of cryptographic protocols. The attack described in the project, discovered by Serge Vaudenay, works against block cipher encryption in CBC mode using a standard method of padding plaintext. This succeeds in recovering the plaintext message, although it does not reveal the key. A later project works against RSA public-key encryption and actually recovers the key.

1 The Details

1.1 PKCS #5 Padding Scheme

If a plaintext message to be encrypted with, say, DES, consists of b bytes, and b is not a multiple of 8, the message must be padded to fill out a whole number of blocks. The obvious thing to do is to just fill out the remainder of the block with 0's. The problem with this scheme is that the recipient may be unable to tell whether a 0 byte occurring near the end of the plaintext is part of the message, or part of the padding. A better solution, described in a standards document, is to use the padding to communicate where the boundary between the message and the pad is. Here is how we can do this: If the message is 20 bytes long, then we need to add four bytes to fill out the second block. The four bytes that are added are:

```
04 04 04 04
```

that is, four copies of the number four. The recipient, upon decrypting this message, sees that the last byte of plaintext is four, and strips off the last four bytes to recover the intended plaintext message.

In the special case where the number b of plaintext bytes actually is a multiple of 8, we still need to pad the message. This time we add an entire block of copies of 8—in hex this is

```
08 08 08 08 08 08 08 08
```

1.2 Tweaking the ciphertext to extract information from the padding oracle

Our server accepts CBC-encrypted messages padded according to this scheme. Of course it does not tell us the decryptions of the ciphertexts it sees, but it checks for correct padding: In other words, if it sees that the last byte of plaintext is 03, it will make sure that there are at least three repetitions of 03 at the end of the plaintext. Otherwise, it responds with an error message. In this respect, the server acts as a 'padding oracle'.

Now suppose we have a block of ciphertext C and we're interested in finding $D_K(C)$. We create a random 8-byte block

$$R = r_7 r_6 \cdots r_0$$

and submit the two-block message $R||C$ to our padding oracle. (We may need to submit an IV as well.) Since CBC mode is being used, the padding oracle is telling us whether the block

$$R \oplus D_K(C)$$

is correctly padded. Observe that any block ending in the byte 01 is correctly padded, so more than $1/256$ choices for R will elicit a 'yes' answer. This means that we will not have to try out a lot of values. Moreover, 01 is by far the likeliest pad to give a correct padding, so with high confidence, we can guess that the low-order byte of $D_K(C)$ is

$$r_0 \oplus 01.$$

The one bit of information given away by the padding oracle has allowed us to decrypt one byte!

I'll leave you to think about how to handle the relatively rare instance where $R \oplus D_K(C)$ is correctly padded, but the low-order byte is something different, say 02.

Once we've established the low-order byte, we can build on this information to determine the next byte, then the next, etc. Each sweep requires 256 queries to the oracle in the worst case, although about half this on average. After eight such sweeps, we will have obtained the complete decryption $D_K(C)$.

The complete attack is described in Vaudenay's original paper (not hard to read the part describing the attack), which you can get at

http://www.iacr.org/cryptodb/archive/2002/EUROCRYPT/2850/2850.pdf

It may seem far-fetched, but there are real systems vulnerable to this attack. See the paper by Rizzo and Duong at

https://www.usenix.org/legacy/event/woot10/tech/full_papers/Rizzo.pdf

2 Deliverables

You will need a working version of DES in CBC mode to experiment with. I suggest you install the pycrypto package. You can use this to create your own padding oracle. The oracle should return only the single bit of information that the plaintext is correctly/incorrectly padded. Your attack code should have access only to this, not to the key or the decryption. Write a program that takes a block of ciphertext and decrypts it by using the padding oracle. Then build on this to write a program that decrypts and entire CBC-encrypted message. Your project should include your code, along with a brief report on the problem, your approach, the results you obtained, and how to run your program. I will also ask you to demonstrate a working version. I may provide you with a padding oracle whose code you cannot see and ask you to decrypt messages of my choice. (As project topics go, this is not a difficult one; once you understand the attack, the implementation can be done with no more than 100 lines of code.)