Introduction to Cryptography

Lab 2 Assignment: Block Ciphers and Chosen-Plaintext Attacks

Overview

This lab will involve various attacks on block ciphers. While the cipher used will be AES, this should not affect your decryption code in any way. Successful attacks rarely target the core cipher; rather, you will be attacking the *mode* of encryption used. Particularly, for this project you will be mounting attacks on the ECB mode of encryption.

Each of these challenges will involve a service on the network, giving out ciphertexts and returning responses. Your code will need to interact with this service. In every case, the end goal will be the retrieval of a flag. This will be accomplished in different ways, depending on the challenge.

Learning to program using sockets is not the goal of this class; you can take Network Security or CNO for that. Thus, I will provide example socket code for any languages I know are being used. If you have a language that does not support network activity, I will be excited to work with you to arrive at a solution.

Assignment

Part 1: ECB Detection

You will write a function to detect which mode an encryption oracle is using.

The oracle for this project is accessible at a URL. It takes input from a GET parameter. It will pad that input on the left and right by 5-10 randomly-chosen bytes, and will then encrypt the result with a randomly chosen key and IV. The oracle will use ECB one half of the time and CBC the other half of the time. This result will be sent back as a result.

At this point, your code should determine which of the two modes is being used, and navigate to a different url, passing the parameter "`ECB`" or "`CBC`", based on your guess.

You will need to guess correctly 100 times in a row, and then the server will send the flag. On the first incorrect response, the score will reset.

```
Server URL: http://10.10.200.42:5001
New Challenge: /new_challenge?data=DEADBEEF0123
Solve Challenge: /solve_challenge?guess=ECB
```

Part 2: ECB Forgery

Encrypting a ciphertext under a secret key does not provide any level of integrity verification on its own. You will attack the ciphertext integrity of a system that uses ECB.

The cryptosystem is a web service[1]. Data is kept on the client, and kept encrypted.

The format for the encrypted data is `email=foobar@example.com&uid=10&role=user`.

You should modify this encrypted data to change your role from `user` to `admin`. You are free to create as many profiles as you wish, but you will not be allowed to include the characters `&` or `=` in your email address.

The system uses a single AES key, and encrypts in ECB mode. Hint for the challenge – *copy-and-paste*.

```
Server URL: http://10.10.200.42:5002
New Profile: /new_profile?email=foobar@gmail.com
Solve Challenge: /view_profile?encrypted_profile=ABCDEF
View Flag: /get_flag?encrypted_profile=ABCDEF
```

Part 3: ECB Decryption

[1] kludgy raw TCP service also available if parsing HTTP is too complex

You will use your knowledge of ECB to attack a cryptosystem by decrypting secret plaintext. This process roughly models various attacks on protocols such as HTTP and SSH.

The oracle for this project will compute the following function:

```
AES-256-ECB(your-string || secret-data || padding, secret-key)
```

You can decrypt `secret-data` with repeated calls to the oracle. Here's how:

1. Feed progressively longer strings to the oracle (start with "A", then "AA", and so on). Discover the block size of the oracle. You already know that AES has a 16-byte block, but do it anyway.
2. Detect that the oracle is performing ECB. Again, you already know this, but do it anyway.
3. Craft an input that is exactly one byte short of a full block. Think about what the oracle will put in that last byte.
4. Make a mapping of every possible last byte by feeding different strings to the oracle; for instance, "AAAAAAAA", "AAAAAAAB", "AAAAAAAC", remembering the first block returned from every invocation.
5. Match the output of the one-byte-short input to one of the entries in your dictionary. You have discovered the first byte of unknown-string.
6. Repeat for the next byte.

```
Server IP: 10.10.200.42
Port: 5003
```

Rules

Please do not exploit any non-cryptographic vulnerabilities you may discover on any machines. You should not need to attempt[2] to gain any level of access to the machine beyond a TCP connection, nor should you attempt to mount a Denial of Service attack on any host on the network.

Grading

Labs are due Thursday, September 17 at 11:59pm. You will need to turn in all recovered flags and any supporting code.

[2] I hate telling people they are not allowed to hack my services. If you want to try, please go ahead. If you find something, just tell me, and don't use it to cheat on the lab.