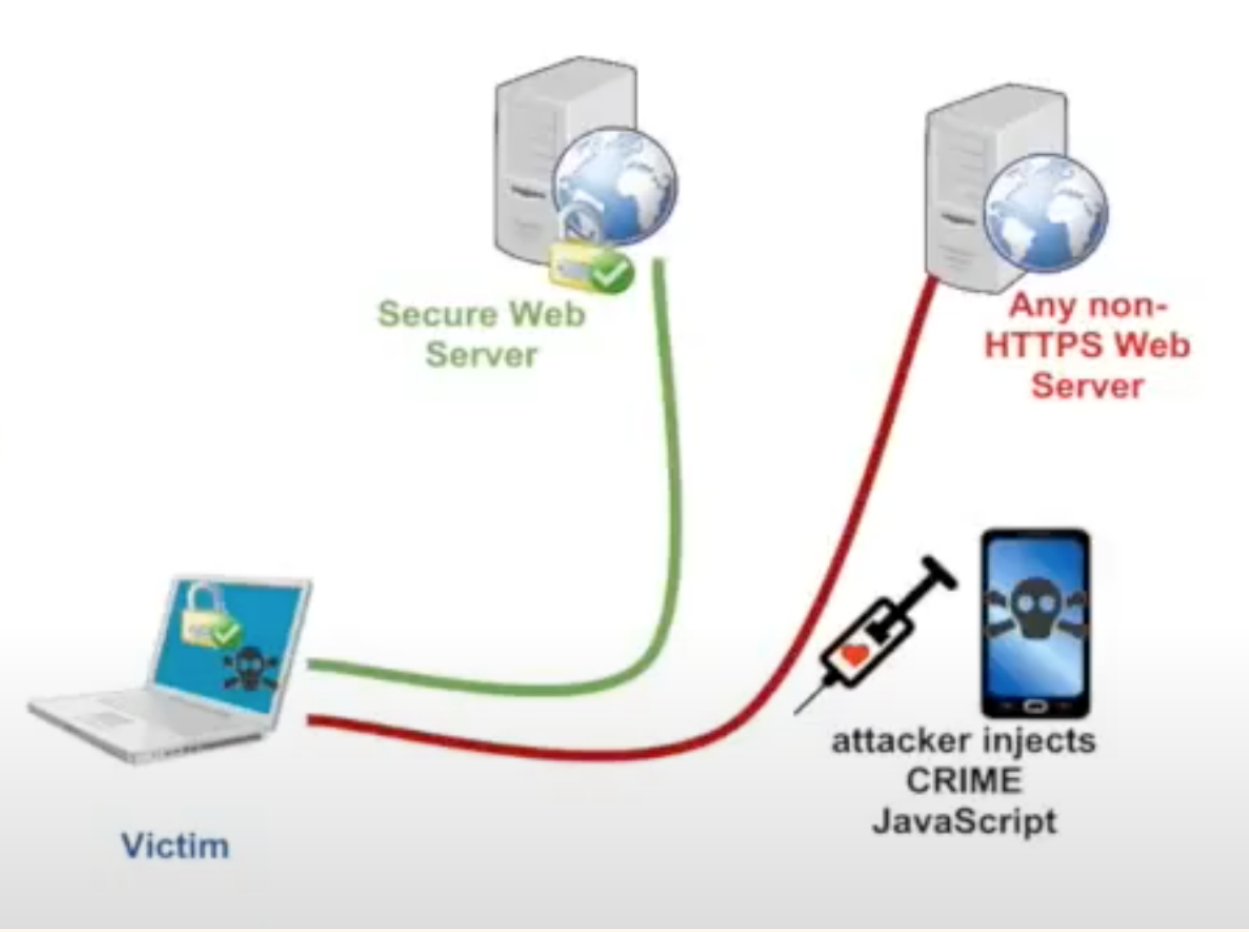
REPORT - ASSIGNMENT 3
COL759 (CRYPTOGRAPHY AND COMPUTER SECURITY)

SSL/TLS Vulnerability - CVE-2012-4929

Compression Ratio Info-leak Made Easy (CRIME)

—

By Vinayak Rastogi - 2016CS10345



PROTOCOL EXPLANATION

About SSL and TSL

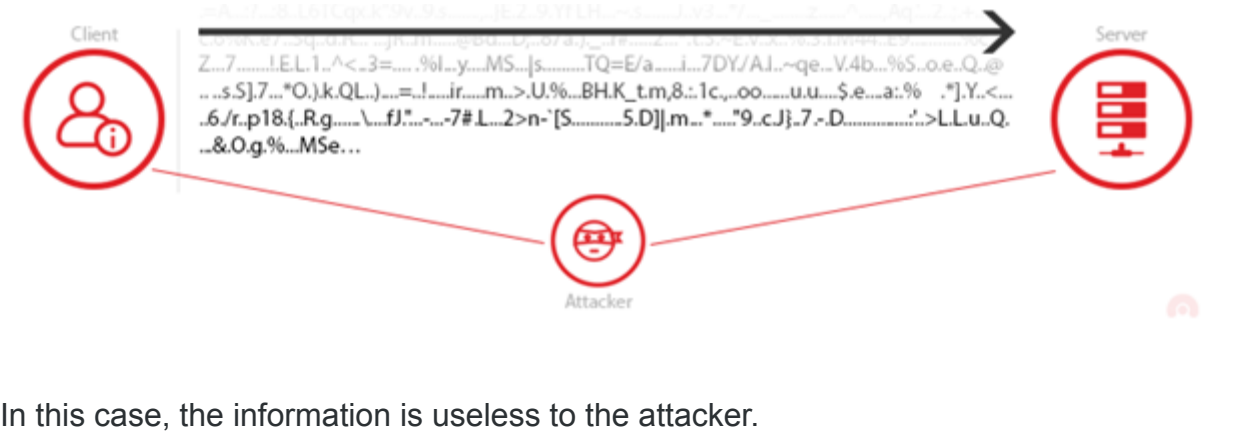
Secure Sockets Layer (SSL) and Transport Layer Security (TLS) are cryptographic security protocols. They are used to make sure that network communication is secure. Their main goals are to provide data integrity and communication privacy. The SSL protocol was the first protocol designed for this purpose and TLS is its successor. SSL is now considered obsolete and insecure (even its latest version), so modern browsers such as Chrome or Firefox use TLS instead.

SSL and TLS are commonly used by web browsers to protect connections between web applications and web servers. Many other TCP-based protocols use TLS/SSL as well, including email (SMTP/POP3), instant messaging (XMPP), FTP, VoIP, VPN, and others. Typically, when a service uses a secure connection the letter S is appended to the protocol name, for example, HTTPS, SMTPS, FTPS, SIPS. In most cases, SSL/TLS implementations are based on the OpenSSL library.

Privacy and Integrity

SSL/TLS protocols allow the connection between two mediums (client-server) to be encrypted. Encryption lets you make sure that no third party is able to read the data or tamper with it. Unencrypted communication can expose sensitive data such as user names, passwords, credit card numbers, and more. If we use an unencrypted connection and a third party intercepts our connection with the server, they can see all information exchanged in plain text. For example, if we access the website administration panel without SSL, and an attacker is sniffing local network traffic, they see the following information.

The cookie that we use to authenticate on our website is sent in plain text and anyone who intercepts the connection can see it. The attacker can use this information to log into our website administration panel. From then on, the attacker's options expand dramatically. However, if we access our website using SSL/TLS, the attacker who is sniffing traffic sees something quite different.



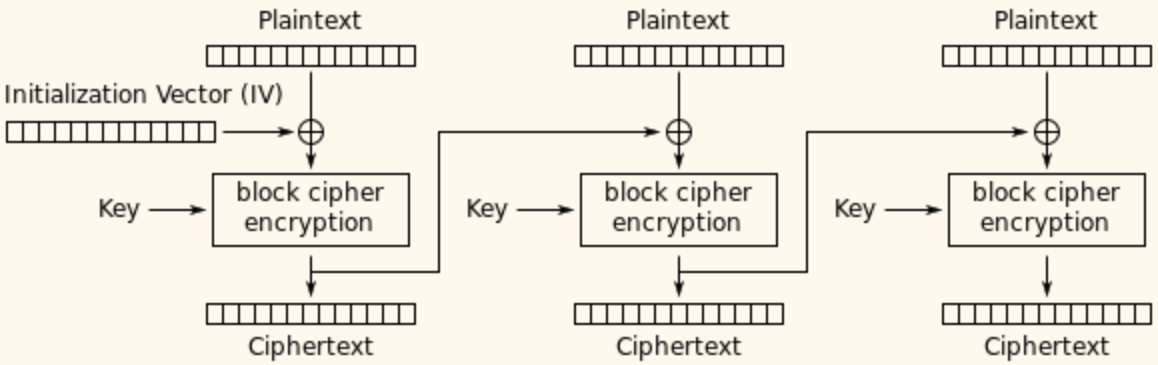
In this case, the information is useless to the attacker.

Identification

SSL/TLS protocols use public-key cryptography. Except for encryption, this technology is also used to authenticate communicating parties. This means, that one or both parties know exactly who they are communicating with. This is crucial for such applications as online transactions because must be sure that we are transferring money to the person or company who are who they claim to be.

When a secure connection is established, the server sends its SSL/TSL certificate to the client. The certificate is then checked by the client against a trusted Certificate Authority, validating the server's identity. Such a certificate cannot be falsified, so the client may be one hundred percent sure that they are communicating with the right server.

CBC Mode encryption



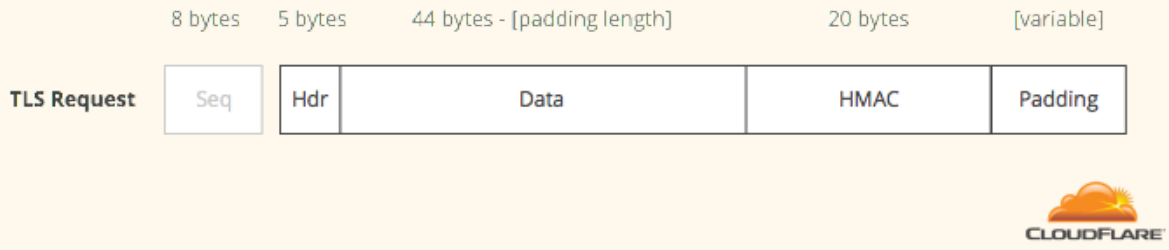
Cipher Block Chaining (CBC) mode encryption

Secure Socket Layer with cipher-block chaining (CBC) mode ciphers work as the diagram shows above.

Cipher Block Chaining (CBC) mode is a block mode of DES that XORs the previous encrypted block of ciphertext to the next block of plaintext to be encrypted. The first encrypted block is an initialization vector that contains random data. This “chaining” destroys patterns

The plaintext is divided into blocks of 16 and then XORed with a chosen Initialization vector, which is using the block cypher encryption is encrypted, resulting in a ciphertext that then repeatedly goes through the same process again and again.

This has certain vulnerabilities.



Also, TLS used to use compression for reducing bandwidth of the traffic, which turned out to be a very risky vulnerability which could lead to hacker hijacking networks by sniffing packets and cookies. We’ll discuss one such vulnerability below, by the name of CRIME.

VULNERABILITY EXPLANATION

About CRIME

Compression Ratio Info-leak Made Easy (**CRIME**) is a security exploit against secret web cookies over connections using the HTTPS and SPDY protocols that also use data compression. When used to recover the content of secret authentication cookies, it allows an attacker to perform session hijacking

It decrypts HTTPS traffic to steal cookies and hijack sessions.

CRIME is a Chosen Plaintext Attack

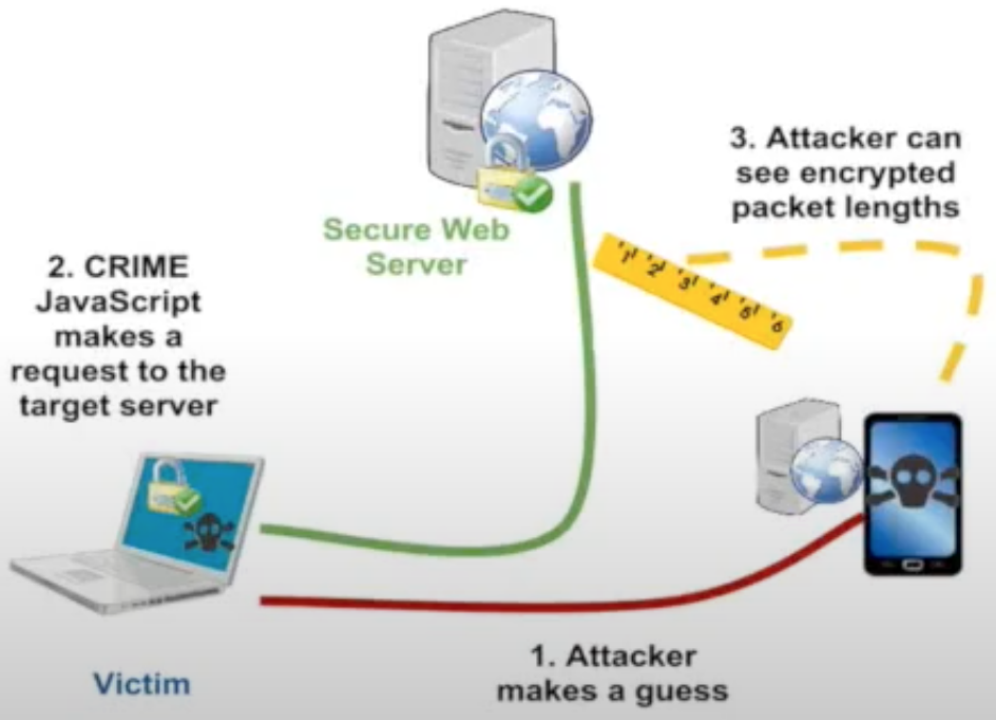
- "Compression" used in this context reduces the number of bytes contained in a data stream by removing redundant bits. When you do this, there's a side effect of the compression, as in it **leaks** clues about the encrypted content and provides a side-channel to those with the ability to monitor data.
    - That ***LEAK*** is the **Data Length,** and it's really difficult if not impossible to hide the length of the data.
- By modifying the clear-text payload hundreds or thousands of times and watching how each one interacts with the encrypted data, attackers can deduce its contents.
- An encrypted message is combined with attacker-controlled JavaScript that, letter by letter, performs a brute-force attack on the secret key. When the bad guy guesses the letter X as the first character of the cookie secret, the encrypted message will appear differently than an encrypted message that uses W or Y.
- Once the first character is correctly guessed, the attack repeats the process again on the next character in the key until the remainder of the secret is deduced. The use of JavaScript isn't necessary but does make the brute-force attack

Attack Flow

Basically, the attacker is running script in Evil.com. He forces the browser to open requests to Bank.com by, for example, adding <img> tags with src pointing to Bank.com, each of those requests contains data from mixed sources.

In these requests, attacker data and data produced by the browser is compressed and mixed together. Those requests can include the path, which the attacker controls, the browsers headers, which are public, and the cookie, which should be secret.

The problem is that compression combines all those sources together, the attacker can sniff the packets and get the size of the requests that are sent. By changing the path, he could attempt to minimize the request size, i.e., when the file name matches the cookie.



Defense:

- BROWSER SIDE: Upgrade browsers to the latest version
- SERVER SIDE: Disable Compression

ATTACKER's POV

Pre-requisites

Following are the assumptions before we move on the attacker's perspective:

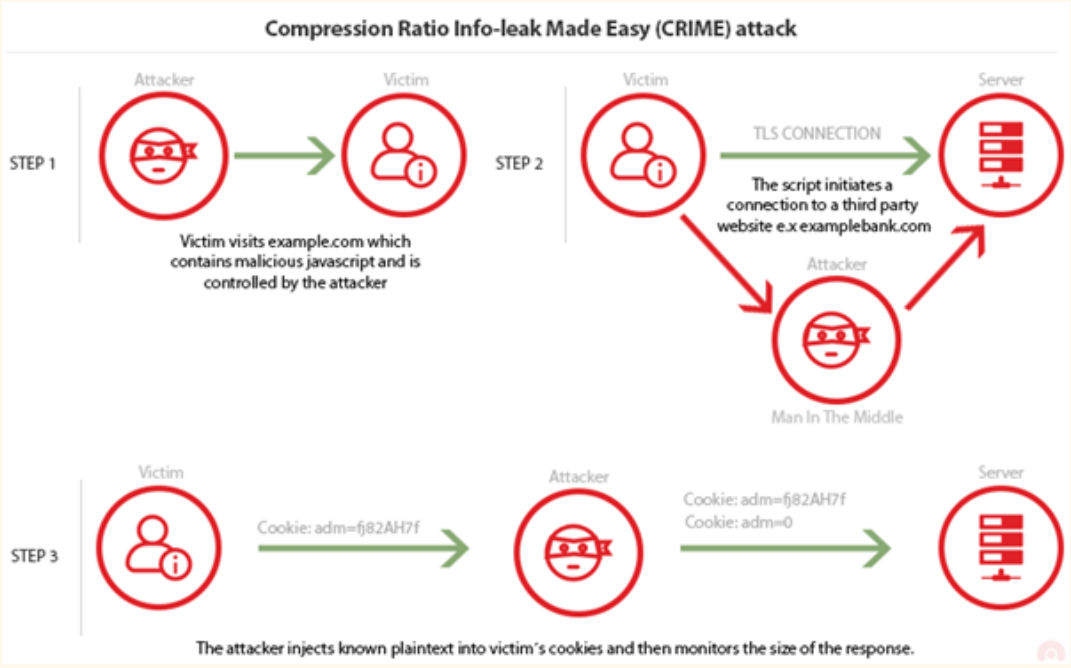
- The Attacker can sniff the network traffic
- The victim (referred toa s the recipient in some cases) visits the malicious website (we'll refer to it as evil.com)
- Both the Browser and server support any version of TLS Compression or SPDY

What's happening really?

The situation is that the attacker wants to get the victim's cookie. It is known by them that the website that they want to visit, that contains personal and confidential information, (example email.com or bank.com) creates a cookie for some session by the name of adm let's say.

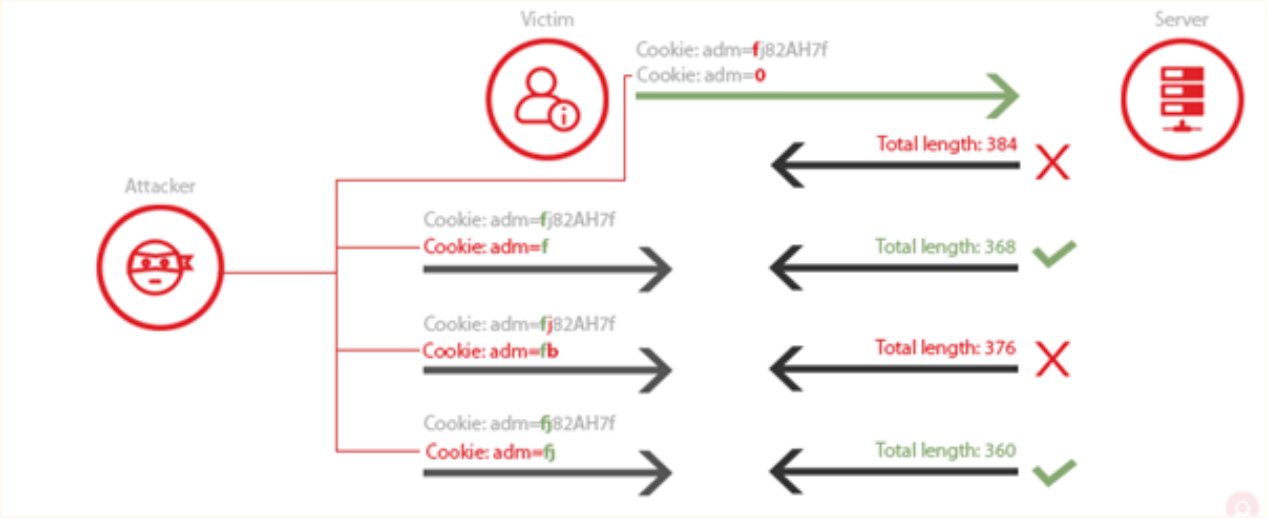
Now, DEFLATE was the compression algorithm that was introduced in SSL/TLS to reduce the bandwidth . The attacker also knows the fact that DEFLATE algorithm replaces bytes.

Hence, the attacker injects `Cookie:adm=0` into the victim's cookie. The server now only appends 0 to the compressed response because `Cookie:adm=` is only a redundancy and is repeated.



All the attacker has to do now is the brute force injection of different chars and then keep a watch on the size of the the responses.

In case the responses are shorter than the initial one, the injected character is contained in the cookie value, and so it got compressed, which confirms the presence of that particular character in the cookie in the eyes of the attacker. If the character is not present in the cookie value, the response will be longer.



Using this method an attacker can reconstruct the cookie value using the feedback that they get from the server.

An Elaborate Perspective (Optional Reading)

In a single session, the same secret/cookie is sent with every request by the browser. TLS has an optional compression feature where data can be compressed before it is encrypted. Even though TLS encrypts the content in the TLS layer, an attacker can see the length of the encrypted request passing over the wire, and this length directly depends on the plaintext data which is being compressed. Finally, an attacker can make the client generate compressed requests that contain attacker-controlled data in the same stream with secret data. The CRIME attack exploits these properties of browser-based SSL. To leverage these properties and successfully implement the CRIME attack, the following conditions must be met:

• The attacker can intercept the victim's network traffic. (e.g. the attacker shares the victim's (W)LAN or compromises victim's router)

• Victim authenticates to a website over HTTPS and negotiates TLS compression with the server.

• Victim accesses a website that runs the attackers code.

When the request is sent to the server, the attacker who is eavesdropping sees an opaque blob (as SSL/TLS encrypts the data), but the compressed plaintext's length 23 is visible. The only thing the attacker can fully control is the request path.

In order to perform the attack, the attacker must load attack code to be loaded into the victim's browser, perhaps by tricking the victim into visiting a compromised or malicious website or by injecting it into the victim's legitimate HTTP traffic when connected over an open wireless network. Once this is done, the attacker can force the victim's browser to send repetitive requests to the target HTTPS website using the following options:

• Cross-Domain requests

• Moving the payload to the query string in a GET request

• Using <img> tags (a method used by Rizzo/Duong)

The attacker can then compare every request that the attack code is sending to the server. Every request with a correct guess of the secret will be shorter than requests with incorrect guesses. Thus by comparing the content length, the attacker can obtain the values of correct guess and hence, complete secret.

ABOUT THE CODE

There's a tiny bit of modification in the implementation of this attack as compared to the theory.

In the code, I've used a recursive algorithm to figure out the possible secret flag values.

The main golden method is the recursive find_recuriseve method. It is an entirely recursive method. Its job is to first send a request with the character that we wish to find and then is followed by multiple abstract misc characters that are not present in the initial request for sure. (example: $*@#*#:[/&"$).

Then it sends a second request with an inversion in the payload. So if for example at first it was chr(i) + "$*@#*#:[/&"$)", the next request would be "$*@#*#:[/&"$)" + chr(i).

The function then compares the two lengths. Taking the function that encrypts that uses AES CBC Cipher and compresses and encrypts the message, let's say enc()

```
if len(enc(request1)) < len(enc(request2)):
    print("found byte")
```

This means that the attacker found a byte!

The reason that two tries is recursive is because some times, more than one byte can be found because of the compression match with multiple patterns. The algorithm needs to take all the paths in the tree in order to find all the possible solutions.

And that's the beauty of it. Listing all the possible solutions.

The rest of the helper functions such as padding and encrypt work alongside the principles of CBC encoding.

As shown in the table below, the adjust_padding function makes sure that padding length will be 1 by adding random value in the GARBAGE variable (in the GET parameter since the attacker control the GET and POST data)

block1	block2	block3	block4	length
GET /GARBAGE	cookie= DATA	Cookie =quokkalight + PAD(1)		**48**
GET /GARBAGE	Cookie = a DATA	Cookie =quokkalight	PAD(16)	64
GET /GARBAGE	Cookie = b DATA	Cookie =quokkalight	PAD(16)	64
GET /GARBAGE	cookie= . DATA	cookie =quokkalight	PAD(16)	64
GET /GARBAGE	cookie=q DATA	Cookie =quokkalight + PAD(1)		**48**

If the byte matchesta a pattern, the length will be the same, if it doesn't match, the length will be different! Hence in the table above we can see that Q is present in the cookie! :)

After this, the next 2 steps in the code are:

- adjust_padding() so we can have a padding of length 1
- find_recursive() for finding the secret FLAG!

Sample Runs:

```
$ python3 CRIME-cbc-poc.py
CRIME Proof of Concept (using a recursive two_tries method) - Vinayak Rastogi (2016CS10345)

Enter your own secret cookie (WITHOUT SPACES): course_COL759
Secret TOKEN : flag={course_COL759}
Encrypted with: [33mAES-256-CBC]


flag={course_COL759}

Found 1 possibilities of secret flag
```

```
$ python3 CRIME-cbc-poc.py
CRIME Proof of Concept (using a recursive two_tries method) - Vinayak Rastogi (2016CS10345)

Enter your own secret cookie (WITHOUT SPACES): My_t34am_inCOL759_is_5h3_BesT
Secret TOKEN : flag={My_t34am_inCOL759_is_5h3_BesT}
Encrypted with: [33mAES-256-CBC]


flag={My_t34am_inCOL759_inCOL759_inCOL759_inCOL759_inCOL759_inCOL759_inCOL759_inCOL759_in
flag={My_t34am_inCOL759_is_5h3_BesT}
flag={My_t34am_is_5h3_BesT}

Found 3 possibilities of secret flag
```

REFERENCES:

- https://security.stackexchange.com/questions/19911/crime-how-to-beat-the-beast-successor/19914#19914
- https://www.nccgroup.trust/globalassets/our-research/us/whitepapers/ssl_attacks_survey.pdf
- https://github.com/mpgn/CRIME-poc
- https://www.acunetix.com/blog/articles/tls-vulnerabilities-attacks-final-part/#:~:text=The%20Secure%20Sockets%20Layer%20(SSL,flaws%20like%20every%20other%20technology.&text=They%20all%20affect%20older%20versions,found%20that%20affects%20TLS%201.3.
- https://www.youtube.com/watch?v=GzoUm_zDA_w