**Padding Oracle on Downgraded Legacy Encryption Attack**

**Background**

The padding oracle on downgraded legacy encryption (POODLE) attack, is an attack on SSL version 3.0 that was recently discovered by Bodo Moller, Thai Duong, and Krzysztof Kotowicz who are members of the Google online security team. (Moller, 2014) Because SSL is widely adopted, any vulnerability discovered within the protocol has the potential for wide spread and lasting effects. (McCarthy) (Alashwali) (Qi, 2008) The POODLE attack exploits a critical design flaw in the SSL protocol, *SSL authenticates the message before encrypting*. (Atighetchi, 2013) This means that any data that comes after the message authentication code (MAC) such as padding, will not be authenticated and can be freely modified by an attacker. (Bright, 2014) This issue is exploitable by an attacker when a block encryption cipher in cipher block chaining (CBC) mode is negotiated during the client and server hello handshake negation phase of SSL. (SSL 3.0) If a block encryption cipher in CBC mode is chosen then the layout of the encrypted message in the SSL application data packet would be as follows:

**Figure 1: SSL Record Structure**

Where the number of pad bytes is equal to the number bytes in the last block that are not taken up by the MAC. If the MAC fills up the entire last block then an additional block of padding is added to the message creating the following structure:

**Figure 2: SSL Record Structure (Full Padding Block)**

According to the SSL version 3 request for comments (RFC) document, the only byte of the padding that is checked is the last byte (which in the case of a full padding block would be 7). (Zetter 2014) If an attacker can force the ciphertext to contain an entire block of padding by adding one byte to each page (plaintext) request until a new block is added to the ciphertext, then the attacker can modify an entire block of the message without affecting the MAC authentication of the plaintext. If the attacker replaces the padding block with any block in the ciphertext, they can use the result of the server’s decryption response to decrypt one byte of plaintext from the ciphertext message. (The Poodle, 2014) This can be shown by analyzing the way CBC mode decryption works:

**Figure 3: CBC Mode Decryption Block Diagram**

When the remote server decrypts the last block of the ciphertext the decrypted last block is XORed with the previous block, then the pad value (last byte of resulting plaintext) is used to find the MAC and verify the integrity of the data. Because the SSL version three specification does not define values of the padding bytes other than the last byte of padding, the remote server is unable to check them. Therefore if the last block of the plaintext consists of all padding then the only requirement for a properly formed last block of the message is the last byte of the plaintext must be 7. By replacing the padding block with any block in the ciphertext, the attacker has a 1/256 chance of the last byte resulting in a 7 which will cause the record to be accepted. If the record was accepted then due to the properties of XOR the attacker can conclude that:

**P[7] = 7 ⊕ Cn­1[7] ⊕ Ci­1[7]**

Where P[7] is the last byte of plaintext from the target block, Cn­1[7]is the last byte of ciphertext from the second to last block and Ci­1[7]is the last byte of ciphertext from the block before the target block that was chosen to replace the padding block. To decrypt the next byte the attacker makes a new request to the remote server with a byte added before the target block and a byte removed after the target block until all bytes of the ciphertext have been decrypted. (Moller, 2014) In order to demonstrate this process in more detail a step by step example of the POODLE attack can be found below.

**Example**

In order to demonstrate the interworking’s of the POODLE attack, I have provided a detailed step by step example of the attack. For this example I will demonstrate the poodle attack under the assumption that the attacker can make new page requests, has an active man-in-the-middle capability, and the client and server have negotiated the use of the following cipher-suite:

**Cipher-suite:** SSL\_DH\_RSA\_WITH\_3DES\_EDE\_CBC\_SHA

This cipher-suite is a commonly used SSL version three cipher selection and meets the retirement of the POODLE attack: the systematic block encryption algorithm must be operating in CBC mode. Suppose a user wanted encrypt a secret message (“POODLE!!”) and there is some information before and after the message that the attacker has the ability to change. A resulting plaintext could be as follows:

**Plaintext**: “AAAAAAAAPOODLE!!BBBBBBBB”

Before the message is encrypted it would first be broken into several blocks as shown below:

**Figure 4: Plaintext Broken Into Blocks for Encryption**

The plaintext would then be encrypted using the triple DES algorithm in CBC mode. A block diagram of CBC encryption can be found below:

**Figure 5: CBC Mode Encryption Block Diagram**

Because the above message will be encrypted the attacker has no idea how many of the last block of ciphertext bytes consist of padding bytes. To confirm that the last block is entirely made up of padding bytes the attacker forces the client to send a new request that has an additional byte of data added to the beginning of the message until the size of the ciphertext increases by the size of one block. An example of this process can be found below:

**Figure 6: Plaintext Blocks with a Full Block of Padding**

By adding four ‘A’s to the beginning of the plaintext, the attacker is able to confirm that entire last block of ciphertext is padding. Since all the data would be encrypted the attacker does not know the contents of the secret message (“POODLE!!”), but will know that the secret message is located in block 1 and 2. Therefore if the following ciphertext resulted from the encryption of the previous plaintext.

**Figure 7: Blocks of Ciphertext**

The attacker would use the following steps to exploit the POODLE vulnerability and decrypt one byte of plaintext:

1. Force the client to send the encrypted chosen plaintext (see above) to the remote server.
2. Intercept the ciphertext message and replace the contents of the padding block with the contents of target block, then forward the modified request to the remote server.
3. Get the server’s response
4. If the modified ciphertext did not produce an encryption alert error the attacker can decrypt one byte of the plaintext by calculating: P[7] = 7 ⊕ Cn­1[7] ⊕ Ci­1[7].
5. Else repeat the process from step one.

A practical example of this process has been demonstrated in the following diagrams below:

**The attacker captures the ciphertext and replaces the contents of the padding block with the contents of target block, then forwards the modified message to the remote server:**

**Figure 8: Attacker Replaces the Padding Block with the Target Ciphertext Block**

**The server decrypts the ciphertext using CBC mode of decryption:**

**Figure 3: CBC Mode Decryption Block Diagram**

**If the server returns a successful decryption message (aka the last byte of the last block was decrypted to a 0x07), the attacker can then infer that:**

0x44 (‘D’) = 0x07 ⊕ 0x39 (‘9’) ⊕ 0x7A (‘z’)

**The attacker has now successfully decrypted one byte of the ciphertext without the key, violating the security principle of cryptography.**

**Figure 9: Attacker Decrypts One Byte of Ciphertext without the Key**

Once the last byte of the target block has been decrypted the attacker forces the client to add a byte before the target block and remove a byte after the target block to begin decryption of the next byte. This process continues until all bytes of the secret have been decrypted.

**Figure 10: Attacker Shifts the Plaintext to Decrypt the Next Byte**

While this example used a very simple message format, the same concepts apply to the exploitation of a HTTP post request. An attacker would be able to control the URI data and the POST data that is sent to the web server, but would not be able to read the cookie. Therefore the poodle attack can provide a mechanism to decrypt unknown cookie values and allow an attacker to hijack an authenticated session.

**Figure 11: HTTPS POST Request**

**Implementation**

For my implementation of the POODLE proof of concept attack program, I choose to replicate a real world scenario. In my scenario an attacker has infected a host on the local network and is using that host as a man-in-the-middle server. Because my proof of concept is was focused on how to make poodle work successfully I set up static routes on the client machine and my router to force traffic to and from the client machine to go through the man-in-the-middle server. The man-in-the-middle server also has the ability to force the client to change the URI and POST data of arbitrary https POST requests to the website: poodle.unonullify.com. An example diagram of the architecture used to test and demonstrate the POODLE proof of concept can be found below:

**Figure 12: Architecture Used to Test and Demonstrate the POODLE Proof of Concept**

As the diagram shows my proof of concept for the poodle attack consisted of three components:

1. **Client Machine (192.168.210.70)** – This computer uses the httpsRequest.py program to make https POST requests to the poodle.unonullify.com remote web server as directed by the man-in-the-middle server. The only data that the client machine changes in the plaintext is the URI data and POST data, this simulates a real world attack where an attacker is able to execute malicious javascipt on a client machine to make arbitrary request on a web page that the client has visited.
2. **Man-in-Middle Server (192.168.210.117)** – This computer uses the poodleMiM.py program to configure the box to perform the POODLE attack. The poodleMiM.py program creates a new block of ciphertext (all padding). Then intercepts, mangles, and repacks SSL records that are being sent to the web server by the client. Finally the poodleMiM.py program checks if decryption was successful on the remote webserver, and if it was successful the program will decrypt the target byte and move to the next byte to decrypt in the target block.
3. **Remote Web Server (107.155.87.147)** – This is the remote web server that serves up the poodle.unomaha.com web page and handles all https traffic. The web server is an apache web server running on Ubuntu 14.04 LTS and has not been configured in any special way other than the fact it is using a self-signed cert.

While the actual details of how the httpsRequest.py and poodleMiM.py programs work are covered in the code comments, screenshots of the programs help options can be found below.

**Figure 13: Screenshot of httpsRequest.py Program Help Options**

**Figure 14: Screenshot of poodleMiM.py Program Help Options**

The output from running both programs to demonstrate the POODLE attack proof of concept on triple DES CBC mode be found below:

**Figure 15: Poodle Proof of Concept in Action (part 1 of 2)**

**Figure 16: Poodle Proof of Concept in Action (part 2 of 2)**

**Findings**

My main finding for this project was the amount of assumptions required for the POODLE attack to be successful:

* The attacker must be an active man-in-the-middle or able to mangle packets traveling between the client and webserver.
* The attacker must be able to force the client software to change parts of the plaintext.
* The attacker must downgrade the communication between the client and webserver to SSL version three.

Even when all of these assumptions are true the amount of time required to decrypt a session cookie can take a fairly long time. While the amount of time required is far less than an exhaustive search for the encryption key, it would still take an attacker about an hour to decrypt enough blocks to steal a session cookie. A table of average decryption times for three popular block encryption algorithms can be found below.

**Figure 17: Table of Average Decryption Times for Three Popular Block Encryption Algorithms**

**Future Work**

My project has presented a real world example of how a malicious attacker may implement and leverage the POODLE attack. Some future work that could be added to this project, however, is further research could be done to examine all current implementations of TLS to verify that they have implemented the padding protections that the TLS specification requires. If any of these implementations did not follow the specifications, and therefore do not verify every byte of the padding they may be vulnerable to the POODLE vulnerability as well. (Chelmo, 2014) This research would help determine the risk of exploitation while using https over a public network such as the internet.

**Conclusion**

In conclusion the padding oracle on downgraded legacy encryption (POODLE) attack violates the security principle of cryptography because it allows an attacker to decrypt plaintext bytes without needing the key. If used properly this attack could violate the confidentially of a SSL session with a website by allowing a remote attacker to steal a session cookie and hijack a user’s session within about an hour. While the practical risk of this attack is fairly low it is important for all users to disable SSL version 3 in the web browser in order to mitigate the risk of being affected by the POODLE attack.

**References**

Alashwali, E. (n.d.). Cryptographic Vulnerabilities in Real-Life Web Servers. The 3rd International Conference on Communications and Information Technology. Retrieved from http://ieeexplore.ieee.org.leo.lib.unomaha.edu/stamp/stamp.jsp?tp=&arnumber=6579513

Atighetchi, M., Soule, N., Pal, P., & Loyall, J. (2013, October 1). Safe Configuration of TLS Connections. Retrieved April 15, 2015, from http://ieeexplore.ieee.org.leo.lib.unomaha.edu/stamp/stamp.jsp?tp=&arnumber=6682755

Bright, P. (2014, October 14). SSL broken, again, in POODLE attack. Retrieved April 29, 2015, from http://arstechnica.com/security/2014/10/14/ssl-broken-again-in-poodle-attack/

Chelmo, B. (2014, October 15). The SSL 3.0 Vulnerability – POODLE Bug (AKA POODLEbleed). Retrieved April 29, 2015, from http://www.symantec.com/connect/blogs/ssl-30-vulnerability-poodle-bug-aka-poodlebleed

McCarthy, C.; Zincir-Heywood, A.N. "An investigation on identifying SSL traffic", Computational Intelligence for Security and Defense Applications (CISDA), 2011 IEEE Symposium on, On page(s): 115 – 122

Moller, B., Duong, T., & Kotowicz, K. (2014, September 1). This POODLE Bites: Exploiting The SSL 3.0 Fallback. Retrieved February 10, 2015, from https://www.openssl.org/~bodo/ssl-poodle.pdf

Qi, F., Tang, Z., & Wang, G. (2008, November 1). Attacks vs. Countermeasures of SSL Protected Trust Model. Retrieved April 15, 2015, from http://ieeexplore.ieee.org.leo.lib.unomaha.edu/stamp/stamp.jsp?tp=&arnumber=4709278

SSL 3.0 Protocol Vulnerability and POODLE Attack. (2014, October 17). Retrieved February 10, 2015, from https://www.us-cert.gov/ncas/alerts/TA14-290A

The POODLE Brief. (2014, October 20). Retrieved February 10, 2015, from http://www.tenable.com/sites/drupal.dmz.tenablesecurity.com/files/sc-report-files/The POODLE Brief.pdf

Zetter, K. (2014, October 14). There Is a New Security Vulnerability Named POODLE, and It Is Not Cute | WIRED. Retrieved April 29, 2015, from http://www.wired.com/2014/10/poodle-explained/

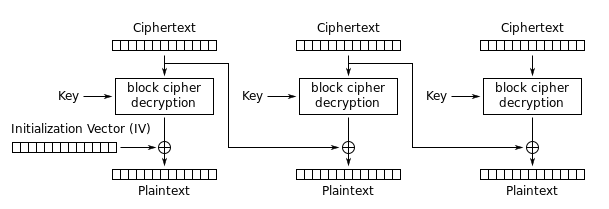
**Figure 1**



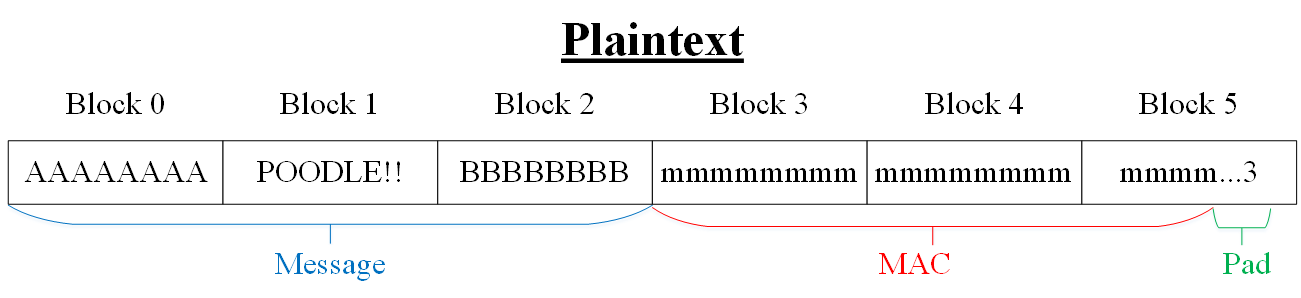
**Figure 2**



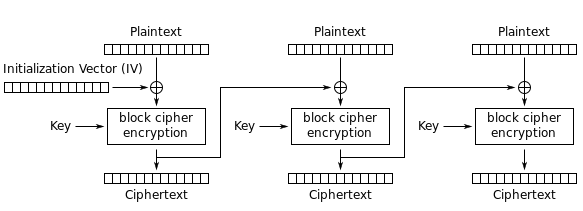
**Figure 3**



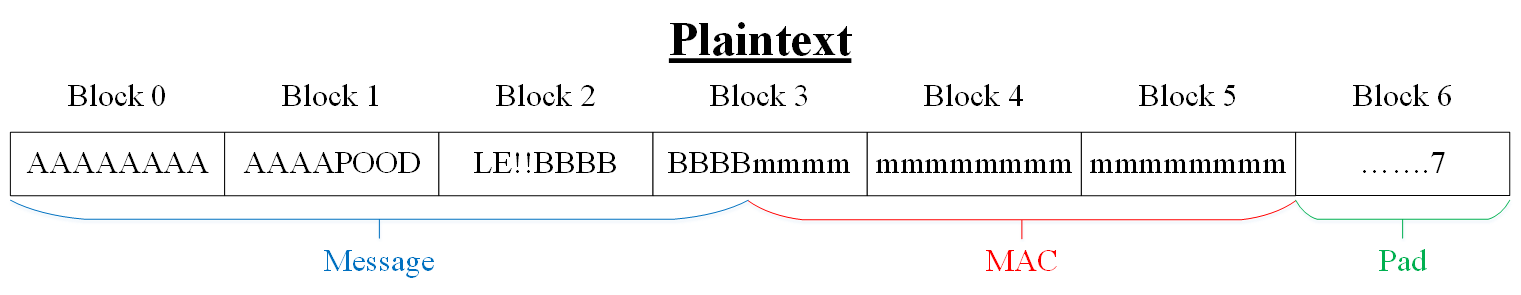
**Figure 4**



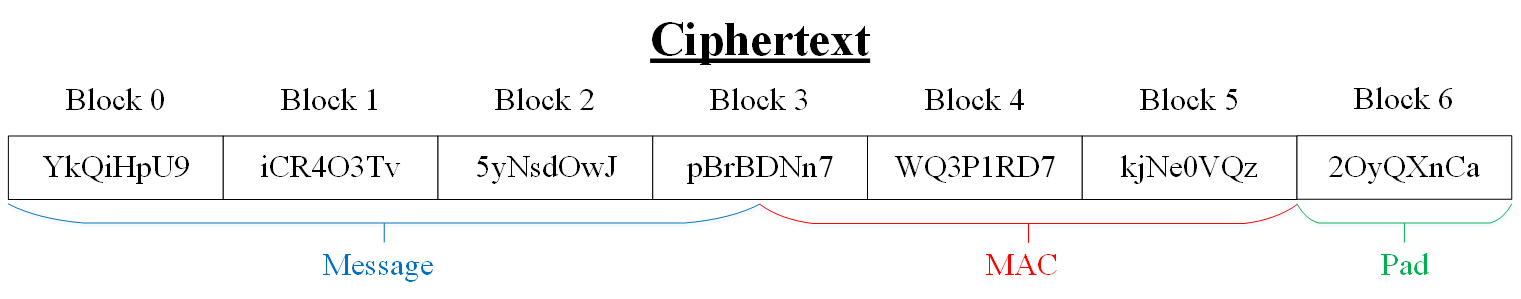
**Figure 5**



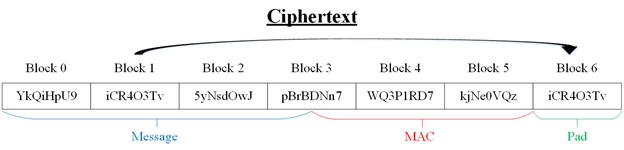
**Figure 6**



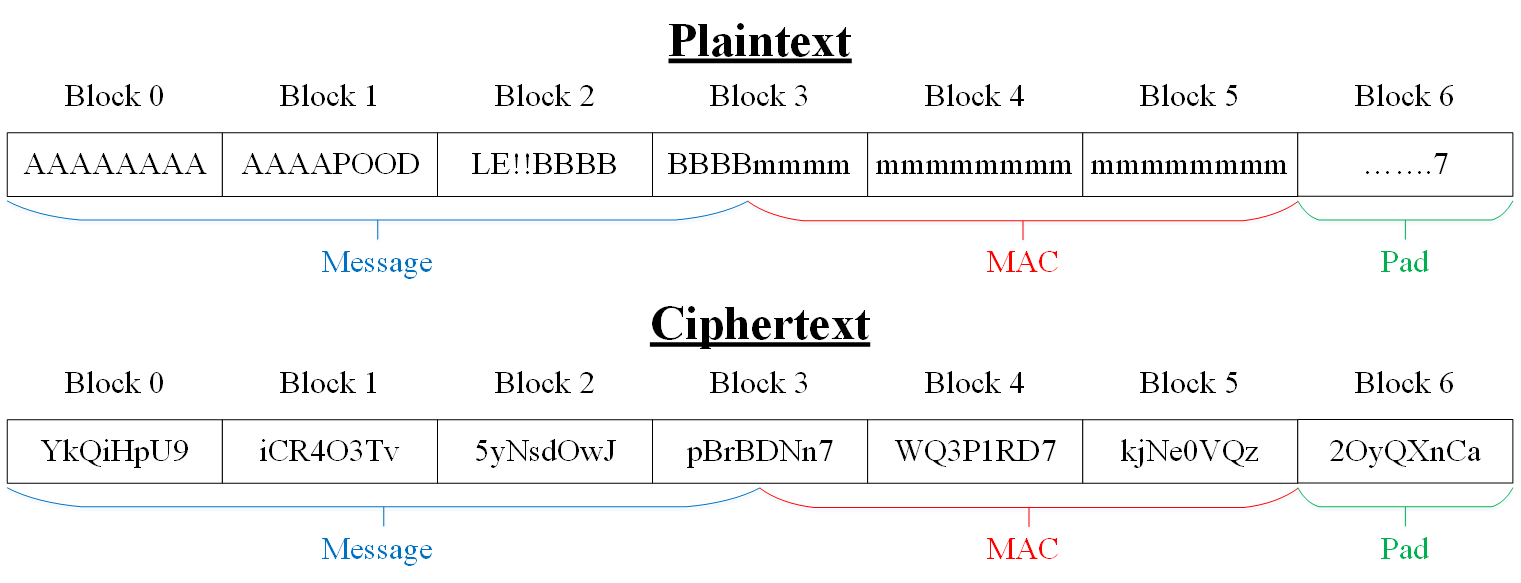
**Figure 7**



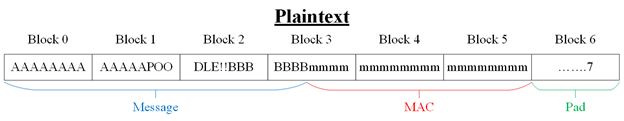
**Figure 8**



**Figure 9**

****

**Figure 10**



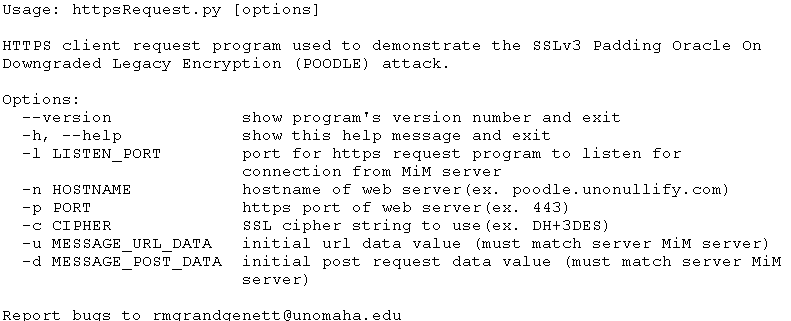
**Figure 11**

****

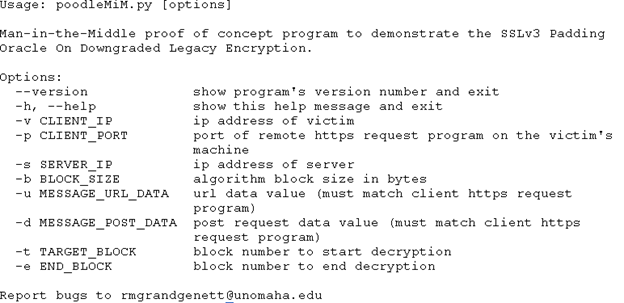
**Figure 12**



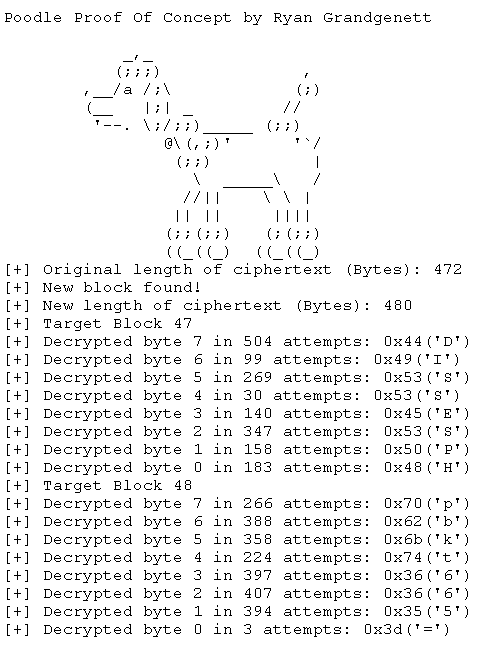
**Figure 13**



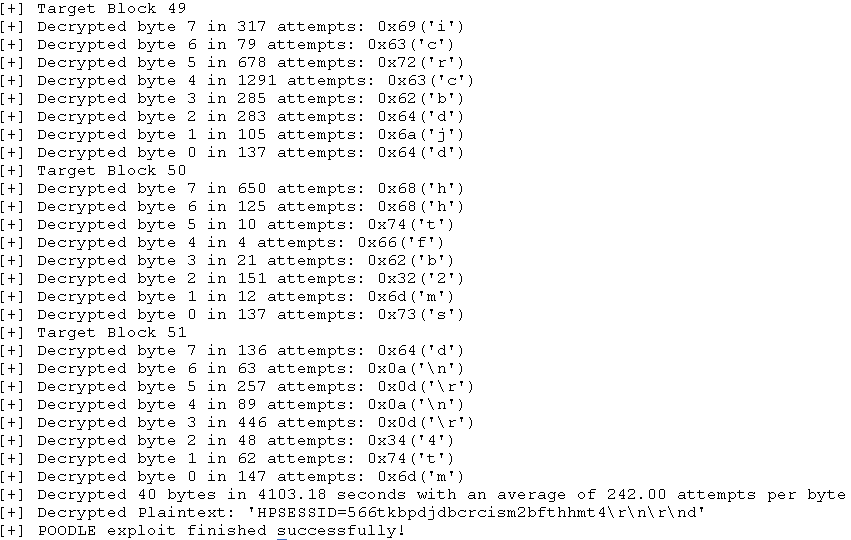
**Figure 14**



**Figure 15**



**Figure 16**



**Figure 17**

|  |  |
| --- | --- |
| **Algorithm** | **Average Decryption Time Per Block** |
| DES-CBC | 10 Minutes |
| 3DES-CBC | 13 Minutes |
| 256AES-CBC | 16 Minutes |