Server based DoS vulnerabilities in SSL/TLS Protocols

Master Thesis

Sukalp Bhople
0754744

Supervisors:

dr. ir. L.A.M. (Berry) Schoenmakers
Faculteit Wiskunde & Informatica
L.A.M.Schoenmakers@tue.nl
Technische Universiteit Eindhoven

prof. dr. Eric R. Verheul
Institute for Computing & Information Sciences
eric.verheul@cs.ru.nl
Radboud University Nijmegen

AUGUST 2012
When availability is an important security requirement for the organizations then (D)DOS ((Distributed) Denial of Service) attack is one of the important attacks that needs to be addressed.

The SSL protocols are widely used to secure e-commerce and other sensitive online transactions. The SSL protocol imposes a significant overhead on the web server. However, it does not create a significant overhead on the client-side. This creates an opportunity for an attacker to execute the computational DoS attacks on the SSL servers. In addition, the SSL protocols are complex in nature.

In this thesis, we perform a number of experiments to analyse the DoS attack possibilities on the SSL protocol. To do so, we study the SSL protocols to come up with a number of SSL functionalities those are likely to be the weak-link and can be exploited to execute the DoS attacks. We also review the Openssl implementation to investigate the presence of DoS attack vulnerabilities in the implementation. Our experimental results show that the client authentication can create a significant computational overhead on the server side. The compression utility does not introduce any buffer overflow condition. The cryptographic operations involved in the SSL protocols are expensive and these operations are unbalanced. The server has to perform more expensive operations than the client especially when RSA key exchange method is used.

We also perform the comparative experiments in which, we measure the impact of the three different DoS attack strategies on the SSL server performance. We found that the SSL renegotiation based DoS attack outperforms the other two DoS attack strategies. We describe the SSL renegotiation feature and working of the thc-ssl-dos tool which is based on the SSL renegotiation feature. The results of these tests also indicate that the SSL protocol is processor intensive. Therefore, adding more CPUs can help in alleviating the impact of the DoS attack. Since, SSL renegotiation based DoS attack only works when the renegotiation is supported, it is easy to mitigate such attack just by disabling the renegotiation.

Our results suggest that there is no simple fix to completely mitigate the DoS attacks. However, the likelihood of the DoS attacks can be reduced by employing enough CPU power, hardware accelerators, memory and network bandwidth. The SSL server needs be configured carefully according to the business requirements.

**Keywords:** DOS, (D)DOS, Denial-of-Service, SSL, TLS, SSL performance, DOS trends, DoS strategy comparison, SSL computational vulnerabilities.
### Contents

1 Introduction
   1.1 Context ................................................. 1
   1.2 Problem Description ................................. 2
   1.3 Research Goal ....................................... 3
   1.4 Research Approach .................................. 3
   1.5 Contribution ........................................ 4
   1.6 Outline ............................................. 5

2 Denial of Service Attack
   2.1 Introduction ......................................... 7
   2.2 Distributed Denial of Service attacks ............. 8
   2.3 Trends .................................................. 8
      2.3.1 Emerging trends .................................. 10
      2.3.2 DoS activities .................................. 12
   2.4 SSL protocol: DoS Attack Threat Model .......... 13
   2.5 Summary ............................................. 14

3 SSL / TLS protocols
   3.1 Introduction ......................................... 15
   3.2 SSL Protocol: Higher Layer ......................... 15
      3.2.1 The SSL Handshake protocol .................. 16
      3.2.2 The SSL Change Cipher Spec (CCS) protocol . 21
      3.2.3 The SSL Alert protocol ....................... 21
      3.2.4 The Application Data protocol ............... 22
   3.3 SSL Protocol: Lower Layer ......................... 22
      3.3.1 The SSL record protocol processing ............ 22
      3.3.2 SSL connection state ........................... 22
      3.3.3 Compression ..................................... 23
      3.3.4 MAC calculation .................................. 23
   3.4 SSL Features ........................................ 24
      3.4.1 Session resumption .............................. 24
      3.4.2 SSL renegotiation ................................ 25
      3.4.3 SSL resumption versus SSL renegotiation . . 26
   3.5 Model of Operation: SSL & TCP ..................... 26
      3.5.1 TCP ............................................. 26
3.5.2 SSL session over TCP .................................. 26
3.6 Summary ............................................. 29

4 SSL Renegotiation and Computational DoS 31
4.1 SSL Renegotiation ........................................ 31
4.1.1 Definition ........................................ 31
4.1.2 Renegotiation application scenarios ............... 32
4.1.3 Authentication Gap in SSL renegotiation .......... 33
4.1.4 Secure renegotiation ................................ 33
4.2 The SSL renegotiation based DoS attack tool: thc-ssl-dos .......... 34
4.2.1 Background ...................................... 34
4.2.2 SSL renegotiation based computational vulnerability ...... 35
4.2.3 Work-flow of thc-ssl-dos tool ..................... 35
4.2.4 Impact ........................................ 37
4.3 Mitigation techniques and their effectiveness ............. 38
4.3.1 Server control .................................. 38
4.3.2 Disable renegotiation .............................. 38
4.3.3 Heuristic based SSL Handshake Rate Limitation ...... 38
4.4 SSL Renegotiation support in Apache web server .......... 40
4.4.1 Test for renegotiation ............................ 40
4.5 Summary ............................................. 41

5 DoS Attack Analysis: SSL Record Processing 43
5.1 Record Processing ..................................... 43
5.1.1 Replay/Re-order attacks .......................... 43
5.1.2 Missing record attack .............................. 45
5.1.3 Incomplete session establishment attacks .......... 46
5.1.4 False data handling attack ......................... 47
5.2 Compression in SSL .................................. 48
5.2.1 Background ...................................... 48
5.2.2 Utility of Compression in SSL ..................... 48
5.2.3 HTTP (Level) Compression and SSL (Level) Compression .................. 49
5.2.4 Compression Burst Handling ...................... 50
5.2.5 Computational Overhead due to Compression ...... 53
5.3 Summary ............................................. 54

6 DoS Attack Analysis: Cryptographic Operations in SSL 55
6.1 Cryptographic Operations .............................. 55
6.1.1 Introduction .................................... 55
6.1.2 Cost of cryptographic operations .................. 55
6.2 Distribution of work overhead ......................... 58
6.2.1 Client-Vs-Server: Computational Differences ...... 58
6.2.2 Set-up & Tool .................................. 58
6.2.3 Result & Analysis ................................ 59
6.3 Summary ............................................. 60

7 DoS Attack Analysis: Client Authentication using RSA Certificates 61
7.1 Introduction .......................................... 61
7.2 Certificate Chain Construction ......................... 62
7.2.1 Background ..................................... 62
7.2.2 Process certificate chain construction .............. 62
### CONTENTS

7.2.3 Certificate Loop ........................................... 63
7.3 RSA Certificate Verification (Signature Verification) ................. 65
   7.3.1 Role of size of exponents ................................ 65
   7.3.2 Chain of certificates ...................................... 66
   7.3.3 Long chain of certificates with large public exponent .......... 67
7.4 Summary ..................................................... 74

8 Comparative Analysis: DOS Attack Strategies on SSL 75
   8.1 Methodology .................................................. 75
      8.1.1 Platform ................................................ 76
   8.2 DoS tools and attack strategies ................................ 76
      8.2.1 the-ssl-dos tool ........................................ 76
      8.2.2 ssqueeze tool .......................................... 76
      8.2.3 BFC (Brute Force with Crypto) tool ..................... 78
   8.3 Experiments and Analysis .................................... 79
      8.3.1 Performance benchmarking ................................ 79
      8.3.2 Micro-benchmarking ...................................... 84
   8.4 Summary ..................................................... 87

9 Miscellaneous 89
   9.1 Related Work ................................................ 89
      9.1.1 Reverse SSL ............................................. 89
      9.1.2 Improving secure server performance by re-balancing SSL/TLS handshakes ................................................ 92
      9.1.3 Performance analysis of SSL web servers .................... 94
      9.1.4 Using client puzzles to protect SSL ....................... 94
   9.2 Random Discussions: DoS attacks on SSL 95
      9.2.1 Session Tickets .......................................... 95
      9.2.2 DoS attacks with respect to different SSL/TLS protocol versions ............ 96
   9.3 Summary ..................................................... 97

10 Summary and Conclusion 99
   10.1 Attack Vectors .............................................. 99
      10.1.1 Client authentication using RSA certificates ............... 99
      10.1.2 Record processing ....................................... 102
      10.1.3 Compression in SSL ..................................... 102
      10.1.4 Cryptographic operations ................................ 103
      10.1.5 SSL renegotiation ....................................... 104
   10.2 Attack Strategies ............................................ 104
      10.2.1 ssqueeze tool .......................................... 104
      10.2.2 Brute Force with Cryptography .......................... 105
      10.2.3 the-ssl-dos ............................................ 105
      10.2.4 Final remarks .......................................... 105
   10.3 DoS Attack Mitigation Techniques ................................ 107
      10.3.1 Mitigation by monitoring ................................ 107
      10.3.2 Mitigation by load balancing and abbreviated handshakes .... 108
      10.3.3 Mitigation by strategic changes ........................ 108
      10.3.4 Mitigation by configurational changes .................... 109
   10.4 SSL Server Performance Improvement Techniques .................. 109
      10.4.1 Hardware accelerators and session cache ............... 109
List of Figures

2.1 Attack Type Distribution (First Quarter of 2012) (source: [2]) .......................... 10
2.2 Largest Bandwidth Consumption Attacks Reported (source: [3]) ....................... 11

3.1 The SSL with its (sub)layers and (sub)protocols (Source: [4]) .......................... 16
3.2 Overview of Handshake Protocol ........................................................................ 17
3.3 The SSL Record Protocol Processing (source: [5]) ............................................ 23
3.4 SSL Record .......................................................................................................... 24
3.5 Session Resumption (Abbreviated SSL Handshake Protocol) ......................... 25
3.6 SSL Session Flow over TCP Connection ............................................................ 27

4.1 An Empty Renegotiation Extension Format ...................................................... 34
4.2 Wireshark Screen-shot: SSL renegotiation packet is flagged with 'PUSH' ......... 39

5.1 Record boundary division: A scenario where a part of Record 1 is not sent ........ 45
5.2 Method Tree (Function Hierarchy) for decompression in SSL ......................... 52

6.1 Performance Comparison of Cryptographic Operations ................................. 57
6.2 Digital Signature Performance Comparison ..................................................... 57
6.3 Comparison of computational power needed by server and client to complete 1000 SSL handshakes ......................................................... 59

7.1 A Basic Certificate Chain .................................................................................. 62
7.2 The set of malicious certificates that forms a loop during certificate chain construction ................................................................. 63
7.3 Client Certificate Verification: Comparison with respect to public exponent .... 72
7.4 Client Certificate Verification: Comparison with respect to length of certificate chain ............................................................ 73

8.1 Web server throughput under different categories of attacks ......................... 80
8.2 Total request time under different categories of attacks ................................. 81
8.3 The Connection Time: brute force with crypto (BFC) Vs brute force without crypto (sslsqueeze) with respect to normal connections ........................................... 82
8.4 Total Wait Time: brute force with crypto (BFC) Vs brute force without crypto (sslsqueeze) with respect to normal connections ........................................... 83
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.1</td>
<td>Reverse SSL Handshake: (a) with client authentication and (b) without client authentication (source: [8]).</td>
<td>90</td>
</tr>
<tr>
<td>B.1</td>
<td>Wireshark Screen-shot: Compression Support in Chrome Browser</td>
<td>122</td>
</tr>
<tr>
<td>B.2</td>
<td>The basic constraint &quot;path length constraint&quot; is set to none.</td>
<td>123</td>
</tr>
</tbody>
</table>
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Session ID and its significance</td>
<td>18</td>
</tr>
<tr>
<td>3.2</td>
<td>Sample cipher suites and their meaning</td>
<td>19</td>
</tr>
<tr>
<td>5.1</td>
<td>Terminology used in Listing 5.1</td>
<td>51</td>
</tr>
<tr>
<td>5.2</td>
<td>Terminology used in Listing 5.2 and in Figure 5.2</td>
<td>53</td>
</tr>
<tr>
<td>6.1</td>
<td>A simple performance comparison of different cryptographic operations in SSL protocol</td>
<td>56</td>
</tr>
<tr>
<td>6.2</td>
<td>Public Key Cryptographic Operations in SSL during handshake</td>
<td>58</td>
</tr>
<tr>
<td>7.1</td>
<td>Terminology used in Listing 7.1</td>
<td>65</td>
</tr>
<tr>
<td>7.2</td>
<td>Terminology used in Listing 7.2</td>
<td>66</td>
</tr>
<tr>
<td>7.3</td>
<td>Terminology used in Listing 7.3 and 7.4</td>
<td>71</td>
</tr>
<tr>
<td>7.4</td>
<td>Summary of certificate chain experiment</td>
<td>71</td>
</tr>
<tr>
<td>7.5</td>
<td>Comparison of different public key exponent usage with respect to time required by the server to complete 1000 handshakes</td>
<td>72</td>
</tr>
<tr>
<td>7.6</td>
<td>Comparison of different size of the certificate chain usage with respect to time required by the server to complete 1000 handshakes</td>
<td>73</td>
</tr>
<tr>
<td>8.1</td>
<td>System information of SSL server</td>
<td>76</td>
</tr>
<tr>
<td>8.2</td>
<td>Comparison of web server performance under different attack scenarios</td>
<td>84</td>
</tr>
<tr>
<td>8.3</td>
<td>Micro-benchmarking for Attacker’s machine</td>
<td>85</td>
</tr>
<tr>
<td>8.4</td>
<td>Micro-benchmarking for Victim’s machine</td>
<td>86</td>
</tr>
<tr>
<td>9.1</td>
<td>SSL Version History</td>
<td>96</td>
</tr>
<tr>
<td>10.1</td>
<td>Certificate verification experiment: Summary of risk analysis of the client authentication based DoS attacks</td>
<td>101</td>
</tr>
<tr>
<td>10.2</td>
<td>Comparison of time required by the server &amp; the client to perform 1000 handshakes for different RSA private key sizes</td>
<td>103</td>
</tr>
<tr>
<td>10.3</td>
<td>Applicability of DoS attack strategies on different versions of the SSL protocols</td>
<td>106</td>
</tr>
<tr>
<td>10.4</td>
<td>Overview of DoS strategies comparison Experiment: Impact on the target machine and client machine</td>
<td>106</td>
</tr>
<tr>
<td>10.5</td>
<td>Applicability of DoS attack strategies on different versions of the SSL protocols</td>
<td>107</td>
</tr>
<tr>
<td>Listing</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>4.1</td>
<td>iptable Entry: Heuristic based Rate Limiting SSL handshakes (source: [7])</td>
<td>39</td>
</tr>
<tr>
<td>4.2</td>
<td>Secure Renegotiation Support in Apache Web Server</td>
<td>40</td>
</tr>
<tr>
<td>5.1</td>
<td>Compression Boundary Check</td>
<td>50</td>
</tr>
<tr>
<td>5.2</td>
<td>Decompression Method call</td>
<td>52</td>
</tr>
<tr>
<td>7.1</td>
<td>Certificate Chain Building in OpenSSL</td>
<td>63</td>
</tr>
<tr>
<td>7.2</td>
<td>Constraint on the public exponent (e) in OpenSSL</td>
<td>65</td>
</tr>
<tr>
<td>7.3</td>
<td>The client configuration to load the SSL context with cipher suite and certi-</td>
<td>69</td>
</tr>
<tr>
<td></td>
<td>cates and corresponding private key</td>
<td></td>
</tr>
<tr>
<td>7.4</td>
<td>A server configuration to request the client certificate during the SSL hand-</td>
<td>70</td>
</tr>
<tr>
<td>B.1</td>
<td>Heuristic based Rate Limiting SSL handshakes iptable Entry (source: [7])</td>
<td>121</td>
</tr>
<tr>
<td>B.2</td>
<td>Openssl s_client usage (source: [8])</td>
<td>123</td>
</tr>
</tbody>
</table>
CHAPTER 1

Introduction

1.1 Context

In the IT world today, most of the business’s processes depend upon their IT infrastructure and online presence. Performance and availability being the key factors in daily operations, an outage of the online service could cause hazardous effect on the core business of the organization.

Denial of Service (DoS) attack is one of the main threats to the (continuous) availability of the organizational resources. This attack is an attempt to make the services or the network resources unavailable to its intended users. The severity of this kind of attack is greater when this attack is conducted from the different number of places at the same time. This variety of DoS is called Distributed Denial of Service (DDoS or (D)DoS) attack.

In one variety of DoS and/or (D)DoS attack methodology, an attacker produces a huge amount of network traffic targeted at the victim server. This can cause anything from minor slowdown to complete shutdown of the service provided by the server. This is because the server cannot handle the sudden increase in demand of its service and has to partially or completely shut down its online services (resources). This makes its service unavailable to its legitimate users. In upcoming trends of DoS attacks, an attacker creates limited number of connections with the server and maintains them for a longer time. This eventually can cause depletion of the server’s resources such as a connection pool, processing power, memory. This alters the server’s behavior causing denial of service situation for legitimate users of the server. This variety of DoS attack does not require a huge amount of traffic.

The motivation for DoS attacks can be numerous. Number of surveys and news portals available on the internet have reported that the protests against controversial activities, operations [9, 10], government actions [11, 12] or mere vandalism [13, 14] is the popular reasons for organizations to be the victim of DoS attack. One of the incidents reported at [15] gives an evidence of usage of DDoS for money extortion. There is one more situation where a website ends up denied without any deliberate attempt from the attacker. Such situation may arise due to increased web traffic on a web portal that is totally unprepared for a huge amount of web traffic. One such incident is reported at [16]. It describes the time when Michal Jackson was dead in 2009, people all around the world visited the web portal reporting the news to verify the fact. The web site was not configured to handle so many requests concurrently. This resulted in a slowdown of the website.

In recent past, hackers are found to be using advanced techniques to construct DoS attacks
on websites. Automated tools such as Slowloris, LOIC (Low Orbit Ion Cannon), #RefRef are being used in wild. Hacker organizations such as Anonymous are openly taking responsibilities of DoS attacks. Recent News articles [11, 14] indicate that large financial organization such as MasterCard is also a victim of such attacks. The (D)DOS attacks on Amazon, CNN, eBay, Yahoo [9] and VISA, MasterCard, Winklers [11, 14] are well acknowledged.

The financial organizations handle one of the most critical and sensitive transactions everyday. The portals for online banking are supposed to provide service to its customers 24x7. Any outage in its service could cause great financial loss to financial organizations. Along with these financial institutions, e-commerce websites that handle sensitive customer information such as usernames, passwords, credit card numbers are protected using SSL/TLS protocols. The four main objectives of SSL/TLS protocols are cryptographic security, interoperability, extensibility and relative efficiency. It ensures that the data sent over the internet is secure.

Communication between a client (generally a user of the online service) and a server (for instance, website of a bank) that uses SSL protocol begins with SSL handshake. The SSL handshake uses public key cryptography to (optionally) authenticate each other (in general server is the one who gets authenticated) and to agree on a cipher and a corresponding shared secret key. This secret key is used to protect the application layer data and communication. The SSL protocols were developed to protect the sensitive data communicated over the internet. The possibilities of DoS attacks were not considered while developing these protocols. This is because these attacks are usually not easy to mitigate or completely protect against.

Due to increased complexity in the SSL protocols, there is always a possibility to exploit some feature of the protocol to construct the DoS attack. It is therefore, equally important to continuously revise and carefully implement the SSL protocol. Finally, it is necessary to investigate the possibility of DoS attacks on widely used SSL protocols and its implementations.

1.2 Problem Description

The SSL protocol is being used extensively to protect the sensitive information traveling across the internet. The SSL refers to a secure layer that consists of the number of sub-protocols. The SSL runs transparently on the top of the transport layer protocols such as TCP and just below the other higher level protocols such as HTTP (Hyper Text Transfer Protocol). The SSL protocol is easy to use but equally hard to configure safely because of its complex nature.

The SSL handshake protocol deals with the most expensive cryptographic operations. It is not balanced in terms of the distribution of the work load among the server and the client. The SSL handshake is computationally expensive for the server than it is for the client. The clients can request the server to perform computationally expensive operations without doing any work themselves. Therefore, an attacker can deploy (D)DOS attacks by generating too many handshake requests to exhaust the computational resource of the target server.

In spite of doubling CPU speed for every 18 month, the performance of the SSL server remains an important issue. This is due to the requirement of serving increasingly high number of clients at the same time [6].

---

1SSL: Secure Socket layer, TLS: Transport Security layer. More about these protocols in upcoming chapters.
2In this thesis, the word ‘SSL’ refers to both SSL and TLS protocols unless the difference between them needs to be addressed.
3The details of the SSL Handshake protocol are described later in this thesis. For now, only basic functionality of the SSL handshake is given.
Recently, one of the hacker groups known as 'The Hacker’s Choice' released a proof-of-concept tool over the internet [1]. This tool exploits the computational imbalance between the server and the client using a feature in the SSL protocol. This feature is the SSL renegotiation. This feature enables the client or the server to send a number of handshake requests to the other communication end in a single connection.

Therefore, SSL protocol can be the easy target of the DoS attack due to its computational imbalance and complexity.

1.3 Research Goal

The goal of this research is to investigate the server based (D)DOS vulnerabilities in the SSL/TLS protocols and provide countermeasures and their effectiveness for the identified DoS attack vectors.

The intension is to assess the different features in SSL/TLS protocols with respect to possibility of the DoS attack. Whenever SSL is used to protect the communication and data transmission, it is also very important to carefully implement it. The SSL specification provides implementation guidelines for only some part of the protocol. Therefore, it is important to verify whether the implementation is in accordance with the specification.

The goal of this research can be given as follows:

- Provide a general overview of the DoS attacks and explain the importance of the investigation of DoS attacks on application layer protocols such as SSL.
- Study the SSL protocol and its implementation to provide causes of DoS attacks on the SSL protocol. On the basis of the findings, perform some experiment to measure the impact of the identified attack vectors.
- Study the SSL renegotiation and its severity with regard to the DoS attacks.
- Investigate the most common DoS strategies on the SSL protocols and measure their impact on the SSL server. Finally, provide the mitigation techniques and their effectiveness.
- To investigate the efforts-impact of the DoS attacks on the SSL infrastructure.
- Assess the available state of the art literature with the help of the findings from this research and conclude this research.

1.4 Research Approach

This study starts with the SSL specification review and implementation review. The specification review is done to understand the working of the SSL protocol and identify the DoS attack vectors in the protocol. This is followed by the implementation review.

There are number of implementation softwares such as OpenSSL [8], CrytpLib etc. An OpenSSL implementation is selected to be reviewed. It is OpenSSL source and written in C programming language. The OpenSSL is platform independent. The latest statistics at [17] suggests that the OpenSSL is a popular and significantly used open source implementation of the SSL/TLS protocols. The implementation review will focus on identifying the DoS related vulnerabilities.

In order to investigate DoS vulnerability in the SSL protocols some strategies are identified. An overview of these strategies is given below.
• Check the robustness of the SSL protocol when certain steps in the SSL protocol are not followed or changed in order to create DoS anomaly.

• Analyze the possibility of implementation of the existing DoS strategies on the SSL protocols to produce DoS attack.

• Search for common implementation mistakes in the OpenSSL that can cause the DoS on the SSL protocols.

• SSL runs transparently between the higher level protocol such HTTP and the transport layer protocols such as TCP (Transmission Control Protocol). TCP is widely used over the internet and is the de facto standard of the communication over the internet. Therefore it is reasonable to consider the DoS attacks that are possible on the TCP protocol and consider impact on the SSL infrastructures.

• Prepare the client-server environment to test the effectiveness of the traditional DoS attacks against the some new approaches.

1.5 Contribution

In this research, we investigated the possibilities of the DoS attacks on the SSL protocols. To do so, we constructed the DoS attack threat model for the SSL protocols.

In this thesis, we perform the series of experiments on various aspects of the SSL protocols to find the possibilities of the DoS attack. We have divided the experimental investigations into three categories. These categories are referred as attack vectors. These attack vectors are described below.

SSL record processing We perform a number of experiments to check if the malicious record construction can introduce any vulnerability that can be exploited to execute the successful DoS attack on the SSL servers. We show that there is no possibility to construct DoS attack using this vector. However, we stress that since the SSL works on the top of the TCP, the SSL servers are vulnerable to TCP based DoS attacks. However, such attacks do not affect the core functionality of the SSL protocol.

Cryptographic operation We demonstrate that the cryptographic operations in SSL are expensive. In addition, these are not balanced. That is, the server needs to work more when compared to the client. This is a well-known fact; however, we make an attempt to provide practical results with the help of a proof-of-concept tool. This tool is originally developed by Vincent Bernat [7]. We have modified this tool to provide the appropriate results in this thesis.

Client authentication using RSA certificates We perform a number of tests in this category of experiment. We make use of different sizes of public exponent and certificate chain length to demonstrate the impact of the client authentication on the SSL server. We found that, the OpenSSL [8] allows a large public exponent to be used in the signature verification step of the client certificate verification process.

In addition, we describe the SSL renegotiation process and its impact on the SSL protocols. We also describe the working of the thc-ssl-dos tool [1]. Finally, we decide upon the three DoS attack strategies to test their impact on the SSL servers. These strategies are described below.

• The traditional brute force DoS attack that follows standard SSL protocol and uses cryptography. A proof-of-concept tool is built as a part of this research.
• The SSL renegotiation based computational DoS attack. The thc-ssl-dos tool [11] is used to test its impact.

• The traditional brute force attack that does not use cryptography but follows the protocol with the help of canned protocol messages. The sslsqueeze tool [18] is used to test its impact.

We compare the impact of each tool on the same SSL web server. We have found that, the SSL renegotiation based DoS attack has the greatest impact on the SSL servers. Surprisingly, the sslsqueeze has the lowest impact on the SSL server. We analyse the result of this comparative experiment in respective chapter.

We give the analysis of some of the techniques that focus on improving the SSL server’s performance using the results obtained through the experiments performed in this research. On the basis of all the results obtained in this research, we suggest a number of important DoS mitigation techniques and discuss some more SSL server improvement techniques.

1.6 Outline

The remainder of this thesis is organized as follows. Chapter 2 provides an overview of the DoS attacks. It explains the basic concepts of DoS attacks as a primer to understand the number of issues discussed in this thesis. It also describes the DoS attack threat model for the SSL protocols. Chapter 3 provides the fundamental information about the SSL protocol with respect to its (sub)layers and (sub)protocols. In addition, it describes the SSL operations over the TCP (Transmission Control Protocol). Chapter 4 is dedicated to the SSL renegotiation feature of the SSL protocol. This chapter introduces the SSL renegotiation based DoS vulnerability. This information is accompanied with the description of the thc-ssl-dos tool.

Rest of the chapters are dedicated to the experimental analysis of the SSL protocol with respect to the DoS attacks. Chapter 5 considers the number of test cases to verify the applicability of the DoS attacks due to the way SSL records are constructed and handled in the SSL protocol. Further, this chapter explains the test case that focuses on the compression utility in the SSL protocol. Chapter 6 investigates the cryptographic operations involved in the SSL protocol and provides the evidence of the cryptographic overhead and its unbalance nature, which contributes to the susceptibility of the SSL protocols to DoS attacks. Chapter 7 describes the experimentation performed on the client authentication process of the SSL handshake protocol. Chapter 8 deals with the comparison of the effectiveness of the three different DoS attack strategies on the SSL protocol. Chapter 9 reviews the literature that focuses on the server performance improvement in the SSL protocol. In addition, it talks on the SSL session tickets and the effectiveness of the DoS attack on the different versions of the SSL protocols.

Finally, Chapter 10 summarizes the work done in this thesis followed by the brief discussion of the issues considered in this research. It also provides mitigation techniques for DoS attacks on the SSL/TLS web servers and two methods that can be used to improve the server’s performance. At the end, the future scope is given followed by the conclusion of the thesis.
Denial of Service Attack

In this chapter, an overview of the Denial of Service attacks is given. This chapter aims at providing the fundamental and basic principles that are necessary to understand the issues addressed in this thesis. More specifically, this chapter provides all the basic information about the DoS attacks and gives a summary of the types, methodologies and trends in the DoS attacks. This chapter starts with the introduction of the DoS attacks in Section 2.1. Section 2.2 talks about the (D)DoS attacks. Section 2.3 explains the trends in the DoS attacks and discusses the emerging trends of the DoS attacks. Finally, a brief description of the possibility of the DoS attacks on the SSL protocol is given in Section 2.4.

2.1 Introduction

Denial of Service (DOS) attack is an attempt to make an online resource or service unavailable to its intended (legitimate) users. A resource can be anything from a personal computer, router, network infrastructure, web services or any other possible entity on the network, intranet or internet. The DoS attacks are undoubtedly a very serious problem on the internet and have manifested its hazardous impact frequently on the internet architectures.

The DoS attacks are becoming progressively common and easy to execute day after day. The attackers are building more sophisticated tools and approaches to produce the DoS attacks. Just few years before, it used to require a great deal of technical knowledge and a large number of resources to launch the successful DoS attack. Today, anyone with an adequate computer skills and decent internet connectivity can launch DoS attack with the push of a button. Certainly, this is one of the biggest threat to any businesses those are dependent on availability of its services on the internet.

These attacks can be classified into 2 broad categories. The first category is a simple case where an attacker launches the attack from single computer. The other category includes the DoS attacks that involve the participation of more than one computer to execute the DoS attacks. This kind of attack is called (D)DOS (Distributed Denial of Service) attack; because the attack is conducted from number of different (distributed) locations. The distributed machines could consist of compromised computers (known as zombies), or machines belonging to multiple people who are perpetrating the attack as a result of concerted efforts. Intuitively, the effect of (D)DOS attack is likely to be more destructive as compared with the simple DOS attack described earlier.
A *Denial of Service* attack can be rendered in number of ways. Some of these are gathered and mentioned below.

- Consumption of computational resources. These may include network bandwidth, disk-space, CPU cycles, entropy source.
- Disruption of configuration information. For instance, routing information [19].
- Disruption of state information, such as unsolicited resetting of TCP (Transport Control Protocol) sessions [19].
- Disruption of physical network components [20].
- Obstructing communication media between the intended users and the victim server so as to disrupt the communication between two of them [21].

### 2.2 Distributed Denial of Service attacks

A Distributed Denial of Service (DDoS) attack is a superior kind of general Denial of service attack. The DDoS attack differs from the DoS in a way it is executed. The DDoS is said to occur when multiple systems participate in the DoS attack. The impact of DDoS is expected to be larger than DoS attack due to obvious reasons. Now, these multiple systems or machines participate in the DDoS attack either voluntarily or because they are previously compromised by the attacker. These machines can be compromised by the different methods such as installing malware when a user visits some insecure website.

There are several advantages of DDoS type of attack to the attacker. This kind of attack can purportedly generate more traffic than a single machine attack. The attack conducted from multiple machines is not easy to detect and/or block as these machines are likely to be distributed over the multiple network areas (Wide range of IP addresses). Blocking any of the attack source network can cause blocking the legitimate user or customers of the service. The effectiveness of a mitigation technique in which a web server employs the larger bandwidth is limited since an attacker can circumvent this countermeasure by using more resource to generate enough traffic to create the DoS attack.

DDoS attacks can be further classified into number of different types according to methodology used to create Denial of Service condition. For instance, Reflected/ Spoofed attack. This is the attack in which intermediate machines are used in such a way that these machines will help the attacker to create DoS attack on the victim’s infrastructure. ICMP (Internet Control Message Protocol) Echo Request attacks are the type of Reflected attacks.

In addition, a web portal can be a victim of (D)DoS attack if there are more number of visitors to the website (concurrent requests to use its services or resources) than it can handle.

### 2.3 Trends

This section describes the types and trends in the DoS attacks.

There are several types of DoS attacks. Some of the most common attack types are explained in brief.

**TCP Floods** In this type of DoS attack, an attacker often sends a huge number of TCP SYN (synchronization) packets with the forged source IP address to the target server. The SYN packets are nothing but the first message of the 3-way TCP handshake to request
2.3 Trends

the connection to the server\textsuperscript{1}. On the receipt of this request (SYN Packet), target server respond with SYN-ACK packet and waits for the respons. The response is the final message (ACK) in the 3-way TCP handshake. However, because the source address inside the SYN packet is forged, the response ACK never arrives at the target server. This way, the server is kept engaged in the half-opened connection with the client that does not exists and therefore, the servers resources needed to serve to this connection are locked till the time the TCP connection is terminated by the server due to the connection timer expiration (TCP connection time-out). The enough number of such half-opened connections depletes the server’s resources and/or capacity to respond to the new connection requests. This creates Denial of Service condition for the legitimate users of the service provided by the server. This attack is described at \textsuperscript{23}.

**ICMP Floods** This is a variant of DoS attack where bandwidth of the victim server’s network is consumed so that legitimate packets cannot reach to the victim server. A generic method to launch such attack is to send number of ICMP (Internet Control Message Protocol) echo requests to a network device which broadcasts these requests to all the hosts on that particular IP network. On the receipt of such ICMP echo request most of the host reply to the given source address (through echo request) with the echo reply \textsuperscript{24}. An attacker makes sure that the source address of echo request packets are forged to be the address of the victim server. In this way bandwidth at the victim’s network can be consumed depending upon the number of hosts replying to these ping requests and the bandwidth available to the victim server.

**UDP Floods** In this variant of DoS attack, an attacker sends a large number of UDP packets (with the forged source IP address) to random ports of the victim server. This will cause the victim server to check if any application listens on that port. If no application is listening on that port it will reply with ICMP Destination Unreachable packet. With sufficient amount such incoming UDP packets, the victim server will get entangled with sending too many ICMP packets. This could result in the server to be unreachable by other legitimate clients.

**GET Floods** This DoS attack generates a huge load on the server by sending large number of HTTP (Hyper Text Transfer Protocol) GET requests. These requests are designed in such a way that server requires more time to process each request. For instance, requesting large image on the web page or requesting some data from the database of the server that requires complex database query operations. This attack is usually launched with the number of bots (Distributed DoS type of attack) to produce large number of such requests. This attack is different from other traditional DoS attacks in a way that it focuses on consuming server’s CPU power instead of server’s network bandwidth.

**POST Floods** POST Flood attack is similar to GET flood DoS attack. The main difference is that the request method is POST method instead of GET method. The idea of depleting server’s resources remains the same.

An overview of the types of DoS attacks reported in the first quarter of the year 2012 is given in Figure \textsuperscript{2.1}. This distribution of attack types is taken from the attack report (for the 1st quarter of the year 2012) released by Prolexic\textsuperscript{2}. This distribution also indicates that the infrastructure layer (Layer 3-4) (The OSI protocol stack \textsuperscript{3}) are mostly used to execute the

\textsuperscript{1}The explanation of the 3-way TCP handshake can be referred from \textsuperscript{22}.

\textsuperscript{2}Prolexic is a security service provider company.

\textsuperscript{3}OSI: Open System Interconnection
Denial of Service Attack

DoS attacks. The attack methodologies such as SYN Flood, ICMP Flood and UDP Floods are mostly used. Among application level (Layer 7) attacks GET Floods, POST Floods are the next most observed attacks. It should be noted that this statistics is only an indicative distribution of DoS attack types obtained from the survey.

Figure 2.1: Attack Type Distribution (First Quarter of 2012) (source: [2]).

Figure 2.2 is taken from Annual Security Report of Arbor Network Inc. 2012. The information is collected from the survey that focused on the trends in the DoS attacks. This is a bar chart that shows the highest bandwidth consumption attack recorded in the span of year 2002 till year 2011. The Y-axis denotes the bandwidth consumption in GBPS. The X-axis denotes the years from 2002 to 2011.

In the year 2010, the highest DoS attack traffic produced was 100 GBPS. Even if in the year 2011, the decrease in the amount of network traffic from 100 GBPS to 60 GBPS does not necessarily represent a reduction in the overall DoS attack risk. There could be numerous reasons for this reduction in bandwidth. Perhaps, this could be an indication of the change in the trends of the DoS attack. The probable change in the trends is discussed in the next section.

2.3.1 Emerging trends

According to the recent network security surveys, the current trends in the DoS attacks are somewhat different than they were just few years before. For instance, most of the DoS attacks were based on bandwidth depletion attacks. In other words, most of the attacks were found to be executed at the layer 3 and 4 of the protocol stack. Few examples of such attacks are TCP flood and UDP flood attacks. These attacks are generated by exploiting the flaws in the network protocols (TCP, UDP, ICMP etc).

The scenario is different now. New attack types are emerging. These attack types do not require large number of resources and can be easily executed with basic understanding of the system. These attacks are significantly dependent on the complexities and flaws in application layer protocols rather than lower layer protocols. These types of attacks are immune to the existing DoS mitigation techniques such as packet level filtering. In packet level filtering mechanism, firewalls and routers are updated with definitions (signatures) that

GBPS: Giga Bit Per Second
describes ill packets that are likely to cause DoS attack. Application level DoS attack traffic looks normal at packet level since they are likely to follow the protocol. The server under such type of DoS attack has greater impact. In addition, the execution of the attack does not need large number of resource like in the case of layer 3 and 4 DoS attacks. This is because, these attack concentrate on the resources available to the service and not the resources available to the whole system. For instance, one of this kind attack focus on creating new connections and maintaining them for longer time. The server usually has limited connection pool through which it can accept the number of connection requests from the client. As number of such long duration connections with the server increases, a time comes when the server can no more server new legitimate connections. This leads to DoS condition.

On the lines of trends described in this section, some of the application level DoS attacks are identified that have gained special attention due to their greater impact with lesser resource investment at the attacker side. These attack methodologies does not focus on flooding the network bandwidth but on depletion of the server’s resources in a way that the attack keeps the server engaged in malicious connections and makes the server deny services to the legitimate users. These attacks specially focus on slowing down the connection.

**Slow Loris**

A Slow loris is HTTP DoS tool developed by RSNAKE [25]. The Slow loris exploits one of the old vulnerability in the HTTP protocol and the way few web servers handle this vulnerability. The web servers such as Apache will start engaging its resources in the connection as soon as the first byte of the request is received. The Slow loris sends the partial HTTP request and never completes it. It continues sending subsequent parts of headers at regular intervals to keep the sockets from closing. For example, Apache web server has a default timeout of 300 seconds. This is the time limit the Apache web server waits for a reply from end entity. But slow loris continues to follow this timeout to continue without connection getting closed. That is, it sends a byte value of the header with the time interval just less than the "time-out" value. In this way a web server gets quickly tied up with the attack’s connections [25].

This attack requires minimum bandwidth. This attack is not a flood attack (E.g. TCP SYN flood). This attack consumes all the sockets that a web server can create for new connection. Once all the sockets are consumed by slow loris traffic, no legitimate user can access the website hosted on the server. As long as slow loris is consuming server’s sockets, the server-side becomes irresponsive. As soon as the attack stops (and the sockets become
available), the server becomes available for other legitimate users.

**Slow Post**

Slow POST is another variant of slow attacks. This attack is similar to the Slow Loris attack. It focuses on consuming server CPU cycles and memory instead of network bandwidth. The HTTP POST DoS attack uses POST requests instead of GET as in slow loris. A POST request includes a message body along with a URL. This message body is usually used to specify information for the action being performed [26]. The length of this message body is specified in POST headers through field "Content Length".

The HTTP POST DoS attack starts with sending a complete header to the server unlike in the slow loris attack where a HTTP header is not sent completely. This header specifies a large value for content length. E.g. 1000 bytes. After completing the header, the message body is sent very slowly. The web server waits for the message body to arrive completely as specified by the "content length" field in the HTTP POST header. Enough number of such connections depletes all the server’s resources that are engaged in handling such slow connections with large message body. Therefore, the web server can no more handle the new connection requests from the legitimate users.

HTTP POST requests are commonly used for login, uploading photos/videos, sending webmail/attachments, submitting feedbacks etc [26].

**Slow Read**

Slow read is the HTTP DoS attack similar to slow loris and slow post strategies. This technique produces DoS attack on the HTTP web servers by draining concurrent connection pool as well as by consuming significant memory and CPU cycles on the server side. This attack sends a complete request to the server unlike slow loris. However, when server replies to the request, the response is read very slowly by the attacker [27]. This is possible due to a feature in the TCP protocol. In every SYN packet that contains the request to the server, the client advertises the incoming buffer window. The server has to send the data limited by the buffer window size. Therefore, the attack advertises very small incoming buffer capacity causing slowing down the connection. Such slow connection can consume significant amount of CPU cycles and memory at server side. As well as such concurrent (multiple) requests disables server capacity to serve to any new connection ultimately denying the service/access to the legitimate users.

### 2.3.2 DoS activities

Internet has become a medium to communicate and share. This is not an exception for the DoS attacks and related activities. There are number of tools that are being developed and distributed actively. Easy to use tools play an important role in attacking the target. Anonymous, a underground hacker community is said to have used tools such as slow-loris, HOIC (High Orbit Ion cannon) for DoS attack [11] [28]. These tools are very much sophisticated. Anyone with sufficient computer skills can use these tools and launch the attack with a push of a button. Due to the wide network of such communities huge number of users can launch the attack simultaneously on a same or different target resulting in Distributed Denial of Service ((D)DoS) attack.

The number of DoS tools share similarities in terms of architecture, impact etc. These tools are developed and modified continually. Number of recent tool are the result of little modification of the older tools adapting the model of changing internet architectures to be attacked.
2.4 SSL protocol: DoS Attack Threat Model

The SSL/TLS protocols are used widely to protect the network communication over the internet. These protocols are described in Chapter 3. The SSL protocols involve the number of cryptographic operations. The public key cryptography is used to perform authentication and key exchange process. The key negotiated through the key exchange process is processed to generate the symmetric key that is used to protect the communication between the server and the client. As one can imagine, this operation is an integral part of the SSL protocol and is something that cannot be avoided.

Unfortunately, process of key exchange and authentication, especially when the RSA method is used, are the most expensive operations on the server to perform. The client, however, can get away with this operation with a little investment of the memory and computational resources. Therefore, clients can easily open a number of new SSL connections to request a server to perform such computationally expensive operations without doing any significant work themselves. This leads to the fact that an attacker can deploy (D)DOS attacks by generating too many requests (probably from multiple sources) to exhaust the computational resource of the target server. In spite of doubling CPU speed for every 18 months, SSL performance of server machines remains an important issue. This is due to the requirement of serving increasingly high number of clients at the same time [6].

The SSL protocol has one feature called SSL Renegotiation. This feature is easily exploitable to execute the computational DoS attack on the SSL web servers. The original description can be found at [1].

Other than these cryptographic operations, the SSL also has requirements in terms of proper memory management, data marshaling (such as copying, fragmenting, etc.), SSL session cache management. All these operations involve the risk of exploitation to build a DoS attack.

In order to protect the network infrastructures such as web servers those uses SSL to protect network communication, it has become very important to investigate the possibility of the DoS attacks on the SSL protocols. There is always a possibility to exploit some features in this protocol to construct the destructive DoS attack.

Therefore, a study that describes the broad picture of the DoS possibility in the SSL protocols is required. The following list summarizes the DoS attack threat model of the SSL protocol.

- **Flood of connection requests** The SSL server can be flooded with a number of gratuitous requests to consume the bandwidth available to the SSL server. This is the traditional type of DoS attack.

- **DoS attack threats inherited from TCP and HTTP protocols** The slow attacks described in this chapter can be executed on the SSL protocol too.

- **The SSL Server configuration** The server needs to configure balancing the requirements of the service it is providing. For instance, the server should be equipped with adequate processing power to handle estimated amount of concurrent connections. If the users of the server are likely to re-visit the server multiple times in a short time, the server should be equipped with the appropriate cache memory in order to enable the abbreviated SSL handshakes.

---

5 The SSL renegotiation and corresponding DoS attack are investigated in Chapter 4.
6 The estimated amount could be derived from history of server usage hits.
7 Abbreviated handshakes are described in Chapter 3.
The SSL server features The SSL renegotiation can be exploited to execute computational DoS attack.

2.5 Summary

This chapter provided the fundamental and basic concepts that are necessary for dealing with the DoS attacks on the SSL protocols. More specifically, the DoS attacks are explained and their popular types are described. A note on the emerging trends in the DoS attacks is given to understand the possibility of the DoS attacks on the application layer protocols and its effectiveness compared to traditional flood DoS attacks.

In the later part of this chapter, three types of DoS attacks are given those work on the strategy of slowing down the connection (between the attacker traffic and the server). Finally, a DoS attack threat model is drawn for the SSL protocols. This chapter gave the basic understanding of the DoS attack in general followed by the DoS attack threat on the SSL protocols. In rest of this thesis, SSL protocol is explained and investigated for the possibilities of executing the DoS attack on it.
This chapter provides describes the SSL protocols and provides all the information related to the SSL protocols that is required to understand DoS attack vectors and strategies described in this thesis. This chapter starts with the explanation of the logical location of the SSL in the protocol stack. The SSL protocol can be seen as a layered protocol. It is explained in Section 3.1. Section 3.2 explains the higher layer of the SSL. Section 3.3 explains the SSL record protocol that is responsible for encapsulating the higher layer protocol data and its protection during transmission.

Section 3.4 explains the two of the important SSL features and continues with their differentiation. Section 3.5 provides the details of the SSL protocol with respect to its working on the TCP protocol. Finally, this chapter is summarized in Section 3.6.

3.1 Introduction

Secure Socket Layer/ Transport Layer Security protocols 1 provide a secure channel between two machines 2 over the intranet/internet. It facilitates the authentication of the communicating parties to each other and confidentiality and integrity of the communication between them. Any protocol that can be run over transport layer protocol such as the TCP (Transmission Control protocol) can be run over the SSL with minimal modification 2.

The term SSL can be referred to a layer 2 that is best viewed as an intermediate layer between the transport and the application layer 4. The placement of the SSL layer is shown in Figure 3.1.

3.2 SSL Protocol: Higher Layer

This section describes all the sub-protocols from the Higher Layer as indicated in Figure 3.1. The higher layer SSL protocol comprises of four protocols. These protocols are listed below and are described in this section.

1For convenience, the term ‘SSL’ is used to refer to both the SSL and the TLS protocols in rest of the thesis; unless some protocol version-specific feature needs to be addressed. In fact TLS protocols are predecessors of SSL protocols and there are no fundamental differences in them.

2Please note that the SSL stands for Secure Socket LAYER.
3.2.1 The SSL Handshake protocol

The SSL protocol starts with the Handshake Protocol. The cryptographic parameters of the session state are produced by the SSL Handshake protocol.

Overview

The Handshake Protocol allows communicating parties (referred as client and server) to authenticate each other and negotiate a cipher suite and a compression method used for the communication. The cipher suite and derived cryptographic keys are further used to protect the application level data in terms of the authenticity, integrity and confidentiality.

The Handshake protocol commence by exchanging Hello messages to agree upon the SSL protocol version to be used, authentication mechanism, key exchange algorithm to generate shared secret and encryption algorithm to protect application level data. The Hello messages also optionally decide upon compression method. This is followed by authentication phase in which the server and the client optionally authenticate each other. This is done using exchanging certificates signed by some trusted entity. If the certificate provided by the server does not provide necessary cryptographic information to allow client and server to agree on a pre-master secret then a separate message called Server key exchange is sent by the server to the client so that the client can transfer the pre-master secret to the server by appropriate secure method. This pre-master secret is further used to generate a master secret and other secret material called key block required to communicate secretly.

Before actually starting an encrypted communication protected under just negotiated algorithms and secrets, the client and the server exchanges one special message called Finished.
Figure 3.2: Overview of Handshake Protocol
message. The successful verification of theFinished message at each side of the communication confirms that the same security parameters have been generated and the handshake communication is not tampered in any way.

An overview of the Handshake protocol messages with its variations (optional client authentication and server key exchange messages) is shown in figure 3.2. The SSL handshake messages are described in the following sections.

**Hello request**

This message is sent by the server. This message is NOT shown in Figure 3.2. According to protocol specification [31, 30, 32], the server can send this message at any time to the client. This is an empty message to request the client to begin the handshake protocol. On the receipt of this message client may begin with the handshake protocol by sending the client hello message. This message is not included in Handshake hashes used and maintained throughout the Handshake Protocol [30].

This message typically allows the server to request renegotiation of a session [30]. The SSL protocol specifications [31, 30, 32] does not specify any other application of hello request message than Session Renegotiation.

**Client hello**

Client initiates handshake by sending a client hello message. The client hello message consist of following attributes.

**Protocol version** This is a highest protocol version supported by the client or protocol version with which the client wishes to initiate the negotiation (Handshake) with the server.

**Client Random** This is a structure consisting of 4I byte current unix GMT time-stamp and 28 bytes long number generated by a secure random number generator. This structure is unique to each session between the server and the client.

**Session ID** This is a variable length byte sequence (up to 32 bytes as defined in RFC 4346 [30]) which identifies a session. The significance of the session ID in the client hello message is described in Table 3.1.

<table>
<thead>
<tr>
<th>Content of Session ID</th>
<th>Meaning (through client’s perspective)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Empty</td>
<td>No session ID is available from previous communication or To generate new security parameters with server</td>
</tr>
<tr>
<td>Value from earlier connection</td>
<td>Session resumption</td>
</tr>
<tr>
<td>Value from this connection</td>
<td>Update only random values</td>
</tr>
<tr>
<td>Value from another active connection</td>
<td>Establish several simultaneous connections without repeating full handshake.</td>
</tr>
</tbody>
</table>

Table 3.1: Session ID and its significance

**Cipher Suites** This is a list of set of algorithms supported by the client in its preferred order. The cipher suite defines the authentication method, the key exchange algorithm,
the cipher and the hashing algorithm. The illustrative samples of the cipher suites are given in Table 3.2.

<table>
<thead>
<tr>
<th>Cipher suites</th>
<th>Authentication</th>
<th>Key Establishment</th>
<th>Encryption</th>
<th>Digest</th>
</tr>
</thead>
<tbody>
<tr>
<td>TLS_NULL_WITH_NULL_NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>TLS_DHE_DSS_WITH_AES_256_CBC_SHA</td>
<td>DSS</td>
<td>DHE</td>
<td>AES_256_CBC</td>
<td>SHA-1</td>
</tr>
<tr>
<td>TLS_DHE_RSA_WITH_AES_256_CBC_SHA</td>
<td>RSA</td>
<td>DHE</td>
<td>AES_256_CBC</td>
<td>SHA-1</td>
</tr>
<tr>
<td>TLS_RSA_WITH_RC4_128_SHA</td>
<td>RSA</td>
<td>RSA</td>
<td>RC4_128</td>
<td>SHA-1</td>
</tr>
<tr>
<td>TLS_DH_anon_WITH_RC4_128_MD5</td>
<td>NULL</td>
<td>DH</td>
<td>RC4_128</td>
<td>MD5</td>
</tr>
</tbody>
</table>

*a* No Authentication  
*b* No Authentication

Table 3.2: Sample cipher suites and their meaning

**Compression Methods** This is a list of compression methods supported by the client. If the session is a *resumed session*, this field contains the same compression method negotiated in the last session. This vector must also contain the CompressionMethod.null. Therefore, the server that does not support any compression method as listed in the client hello message can choose for the CompressionMethod.null method.

**Server hello**

The **Server Hello** message is a reply to the client hello message sent by the client to initiate the handshake. On the receipt of the client hello message, the server checks for its willingness to negotiate with the client supplied protocol version. It generates the server random in the similar way as that of the client. This is continued with the selection of the cipher suite and the compression method from the list provided by the client through the client hello message.

**Session ID** The session ID identifies a particular session between the client and the server. The server is responsible for generating the session ID. If the session ID field from the client hello message is not empty then the server will search for the session ID into its session cache, if the session ID is found and the server is willing to resume the session identified by the matched value then the session ID field in the server hello will contain the same value as that of the session ID from the client hello message. In any other case, the server will generate the new session ID.

**Session ID generation** The SSL protocol specification does not comment on how the session ID should be generated. It only provides a guideline that the session ID should not contain any confidential data since it is transferred in plain text. The length of the session ID is 16 byte for SSL v2 [33], SSLv3.0 [31] and TLSv1.0 [30] uses variable length Session ID ranging from 1 byte to 32 bytes.

The OpenSSL [8] generates unique session ID from pseudo random number generator. The session ID generator can be optionally configured for using additional information like a unique host id, a unique process number, and a unique sequence number into the Session ID [34].
Server certificate

This message consist of either a single server certificate or a chain of certificates starting with a server’s certificate. These certificate are such that each certificate except the last CA (Certificate Authority) certificate certify the one preceding it. This message is sent when the negotiated authentication method is not anonymous (i.e. NULL in first row and last row of table 3.2).

Server key exchange

This message is sent followed by the server hello message if anonymous negotiation is in use or followed by the server certificate message if it does not contain enough information for the client to exchange the pre-master secret with the server. This is an optional message.

This message has two variants. The structure of final server key exchange message consists of either of the following key exchange parameters followed by the signature of the parameters.

RSA  This variant of Server Key Exchange consist of RSA key exchange algorithm. It constitutes of RSA modulus and exponent.

Diffie-Hellman  This variant of Server Key Exchange consist of Diffie-Hellman parameters (prime modulus and public value).

Certificate request

This message is an optional message. This message is sent to request the client certificate and to indicate that client needs to be authenticated to the server. The certificate request message consists of the list of certificate types and certificate authorities accepted by the server.

Server hello done

This message is sent by the server to indicate that the server has completed sending server hello and associated handshake messages and the client can proceed with the further steps in the handshake protocol.

Client certificate

This message is an optional message in the handshake protocol and it is sent by the client. This message is only sent whenever client authentication is requested by the server through certificate request message.

Client key exchange

This message is sent by the client. The client obtains the required server’s parameters from either the server certificate or the server key exchange message. These parameters are used to encrypt the pre-master secret. The encrypted pre-master secret is sent as the client key exchange message. The pre-master secret is the 48 byte structure produced by the client. This structure consists of the following two attributes: The latest protocol version supported by the client (2 bytes) and the random data (46 bytes) produced using a cryptographically secure pseudo random number generator.
Case RSA: RSA Encrypted premaster secret  The client key exchange message consists of the premaster secret encrypted with server’s public key obtained from the server certificate or the RSA parameters obtained from server key exchange message.

Case Diffie-Hellman: Client Diffie-Hellman Public value  The client key exchange message consist of the client’s DH (Diffie-Hellman) public value. This message could also be empty whenever the client authentication is requested and the client can produce the certificates with matching public key as specified in the server certificate or the server key exchange message.

The pre-master secret is further processed to obtain the master secret and other key materials required to protect the application layer data.

Certificate verify
This is an optional message and is sent only when the client’s certificate with a signing capability is sent. This message provides the explicit verification of the possession of the certificate by the client. The content of this message is a digest of all the previous handshake messages. The algorithm used for the digest depends upon the signature algorithm (i.e. anonymous, RSA, or DH) used in current connection.

Finished messages
This message is sent immediately after the change cipher spec message. This message consist of the digest of all the messages exchanged so far in the handshake protocol. This is the first encrypted message in the current active connection. The digest does not consist of Alert protocol messages and Change Cipher Spec protocol messages exchanged so far. Since, these are not considered as the SSL Handshake protocol.

The finished messages are sent by both the client and the server. These finished messages are different since the digest included in the finished message sent by the server contains the finished message sent by the client. This scenario reverses in an abbreviated handshake protocol (see Section 3.4.1), where the server is the one who sends its finished message first and then the client sends its finished message.

3.2.2 The SSL Change Cipher Spec (CCS) protocol
This protocol enables the client and the server to notify the other end in the communication that future messages in the communication will be protected under just negotiated cryptographic ciphers and parameters. Under this protocol, a single message of byte value 1 is exchanged. This message is sent by both the client and the server.

3.2.3 The SSL Alert protocol
The SSL alert protocol enables the communicating parties to signal the potential problems. The messages under this protocol are 2 bytes long. The first field (byte) denotes the level of the alert. There are two levels for the alert: the warning alert and the fatal alert. The fatal alert always results in the connection termination from the alert issuing end. The second field (byte) in the alert message provides the description the alert, for instance, unexpected _message. The description is denoted by pre-defined (standard) numeric values For instance, the unexpected _message is denoted by value 10.

The change cipher spec message is a part of the Change Cipher Spec protocol which itself not a part of the Handshake protocol. Therefore the Change Cipher Spec message is not described here.
3.2.4 The Application Data protocol

This protocol is responsible for carrying the higher layer such as HTTP (Hyper Text Transfer Protocol) data and submitting it to the SSL record protocol at the SSL lower layer (see Figure 3.1). The functionality of the SSL record protocol is described in Section 3.3.

3.3 SSL Protocol: Lower Layer

This section describes the SSL sub-protocol called The SSL Record Protocol from the (SSL) Lower Layer as indicated in Figure 3.1.

3.3.1 The SSL record protocol processing

The record protocol is an encapsulation protocol. The record protocol is responsible for fragmenting the data that into the chunks of $2^{14}$ bytes or less. The multiple handshake messages and the application layer data can be combined together into a single plaintext fragments provided the data inside the fragments are of same type. If a single message is too long to fit into the plaintext fragment boundaries then such message may be fragmented across several records. The SSL specification does not provide a guideline on how to determine the message boundaries. The boundaries of the messages could be one-to-one with application data write calls.

Once the plaintext fragments are ready, these fragments are optionally compressed by the compression method agreed through handshake protocol. The length of the compressed fragment should be less than $2^{14} + 1024$. The MAC (Message Authentication Code) is calculated and concatenated at the end of the (compressed) data fragment. This data is applied with the padding in the case of the block cipher mode to match the size of the block in the cipher. This whole content is then encrypted. The length of this encrypted fragment is allowed to be $2^{14} + 2048$ at the max. This encrypted block is then pre-appended with SSL record header. An overview of the record protocol processing is given in Figure 3.3. The final product of this process is called SSL record. Generally, such processing (MAC calculation, encryption, padding) of the data fragment is not applicable for the handshake messages that are transmitted before the change cipher spec protocol message because a connection state is initialized after sending the change cipher spec protocol message. More information about the connection state is given in Section 3.3.2.

The SSL record is divided into two parts. The record header and the record body. Figure 3.4 shows the pictorial view of the SSL record. The record header consists of the record type (Handshake protocol, change cipher spec protocol, alert protocol and application data protocol), version of the SSL/TLS protocol that is being used, the length of the record payload i.e. the length of the record body (in bytes). The record body consists of the protocol specific data. The record body is formatted as described in Figure 3.3. The plaintext (optionally compressed text) is appended with the padding data so as to match the size of the message data with the block size of the cipher. The padding consist of the set of bytes. The last byte of the padding data indicates the size of the padding and rest of the padding data consist of the bytes of value equal to the pad length. The padding is not required when no encryption is performed or the stream cipher is used to encrypt the data.

3.3.2 SSL connection state

The SSL connection state is an operating environment of the SSL Record protocol. There are always two connection states maintained at server and client side. They are called Current...
3.3 SSL Protocol: Lower Layer

Figure 3.3: The SSL Record Protocol Processing (source: [5])

State and Pending State. It (Connection state) specifies a compression algorithm, encryption algorithm and a MAC algorithms.

The Change Cipher Spec protocol is responsible for changing the Pending State to the Current State. The original pending state is initialized to an empty state. The initial Current State always specifies that no encryption, compression, or MAC will be used [30].

3.3.3 Compression

The TLS [30] protocol includes the compression structure which could accommodate 256 compression methods. The RFC 3749 [35] provides the updated definition for this structure. The values for the compression methods are divided into three categories. These categories are "the IETF Standards Track protocols" category, "the non-Standards Track methods" category and the 'private use' category. In addition, [35] gives the identifier for the DEFLATE compression method.

The TLS [30] and predecessors of this protocol should support identifier 0 for no compression (compressionmethod.null) and identifier 1 for deflate compression method. The compressionmethod.null is very important since all the applications that uses SSL protocol to secure their communication are not expected to be able to support compression. The DEFLATE compression method and encoding format is described in the RFC 1951 [36].

3.3.4 MAC calculation

The SSL record consist of the MAC data to preserve the integrity of the record payload. The calculation of the MAC consist of the following fields.

The sequence number The sequence number is the number obtained from sequential counter that is maintained separately for read and write state at each side of the communica-
Figure 3.4: SSL Record

This counter is initialized to zero once change cipher spec message is received or sent. The maximum value of this counter is $2^{64} - 1$. Once this maximum value is reached, this counter is again re-initialized back to zero. This counter is incremented each time the SSL record is sent or received.

**The record type** This is the type of the record. This is the same value as mentioned in the first byte of the record header. This field can include record types such as change cipher spec, alert, handshake or application data.

**The SSL version** The version number of the SSL protocol that is being used in the SSL communication.

**The record data length** The length of the plaintext data if no compression is used or the length of the compressed data if compression is negotiated to be not null.

**The record data** This is the record payload itself.

### 3.4 SSL Features

#### 3.4.1 Session resumption

The SSL protocol has a feature called **Session Resumption**. This feature is added because the client (user) is likely to connect to the same server (web site) several times (within a short duration). Therefore, it is reasonable to bypass the heavy cryptographic operations for the same client to reduce the cryptographic overhead on the server. The session resumption makes use of a technique called **session caching**. The SSL consist of a number of expensive operations such as RSA private key operations. Modern web servers make use of hardware accelerators to offload the heavy cryptographic operations reducing the significant overhead on the core part of the servers. However, it has been proved that the session caching is still effective in the presence of hardware accelerators [37]. This is because the session caching helps to avoid extra network traffic and requires some other lesser expensive operations to be performed. The operations such as digest calculation, master secret computation account for moderate overhead on the SSL protocols. In spite of these small overheads, session cache is assured to help web server performance [38]. The cryptographic parameters of the
SSL connection such as cipher, message digest algorithm, pre-master secret and compression method choice are stored in SSL session cache.

Therefore, session resumption that utilizes session caching is a feature designed as an optimization feature in the SSL protocols. The change in the normal SSL handshake protocol when Session Resumption is in use can be observed from Figure 3.5. [37] described practical experiments that evaluated the performance of different units, processes and operations belonging to the SSL protocol. This experiment demonstrated that the session resumption improves the server’s throughput by the factor of 2.2 to 2.7. However, reducing protocol messages (Network overhead) in the session has negligible impact on the performance (1 to 3%)

![Figure 3.5: Session Resumption (Abbreviated SSL Handshake Protocol)](image)

**Process of session resumption**

Whenever client includes the *Session ID* from previous session in the *client hello* message, it indicates the client’s willingness to resume the session identified by that particular session ID. This *session ID*, if found by the server in its session cache and if the server is willing to resume the session indicated by that *session ID*, the server directly proceeds to sending the *change cipher spec* and the *finished* message.

### 3.4.2 SSL renegotiation

SSL renegotiation is the process where either the client or the server can initiate a new handshake during and within the existing SSL connection. Unlike the session resumption, the SSL renegotiation is a full handshake process. Hence, all the cryptographic operations are performed during the SSL renegotiation. The SSL renegotiation is further explained in Chapter [4].
3.4.3 SSL resumption versus SSL renegotiation

The SSL resumption is designed as an optimization feature since during the session resumption, only the new client random and server random are exchanged and these are used to renew the key material. The cipher suite negotiated in the session that is being resumed is maintained in the resumed session. The session resumption process or abbreviated handshake process bypasses all the heavy public key operations that server has to do otherwise. In addition, authentication process is not performed in the resumed session. A server cannot initiate to resume any session with the client. The client is the one who can resume the session. The server can always reject a session resumption request from the client.

The SSL renegotiation can be carried out for several reasons such as to change the cipher suite, to authenticate each other (client and server), generate a completely new set of key material etc. In the SSL renegotiation, all compulsory steps in normal handshake are carried out. This means, heavy cryptographic operations are also performed. The SSL renegotiation can be initiated by either communicating party. The SSL renegotiation process is carried out in the current session state. This means, the renegotiation messages are exchanged in encrypted mode unless the parent (immediately previous) negotiated session used the NULL cipher.

3.5 Model of Operation: SSL & TCP

This section describes the model of operation of the SSL protocols over the TCP to protect the communication.

The TCP/IP (Transmission Control Protocol/Internet Protocol) regulates the transmission of data over the internet. Application layer protocols such as HTTP, FTP (File Transfer Protocol), runs on top of the TCP protocol in a way that they use the TCP protocol to transfer application level data such as web pages, emails over the internet.

The SSL protocol runs above the TCP/IP protocol but below the other higher level protocols such as HTTP. The SSL protocols use TCP protocol to exchange the data on the behalf of higher level protocols. The SSL protocol enables the data exchange between SSL enabled entities to be encrypted and hence it could be protected from other unwanted entities. Additionally, the SSL protocol allows the entities to authenticate each other before starting the actual exchange of the higher level application data.

3.5.1 TCP

When an application wants to send the data across the internet it uses some transport layer protocol such as TCP to do so. The TCP packages this data into suitable sizes and delivers the packets to the IP and let IP handle the transmission across the internet. The TCP is responsible for reliable delivery of the data packets and guarantees that the bytes are received by the receiving application in the correct order. Therefore, TCP provides a communication service at an intermediate level between an application program and internet protocol. The relative placements of the TCP, the SSL protocol and the application layer (application protocol) can be referred from Figure 3.1.

3.5.2 SSL session over TCP

The SSL offers an abstraction of secure sockets on the top of TCP/IP sockets.

This abstraction enables the applications to simply replace the regular read/write calls over the TCP/IP sockets with SSL_read/SSL_write calls over the corresponding secure
sockets \cite{37}. The operation of the SSL protocol over TCP is explained in Figure 3.6. It is explained by dividing the message sequences into 4 phases as follows.

**Initialization**

- The server creates a new SSL _CTX object. This is a framework to establish SSL connections. This object is configured with SSL parameters such as the connection method (E.g. SSLv3 _server _method), the cipher suits, server and CA certificates, compression method etc.

- A socket is created.

- A new SSL object is created for the SSL _CTX object using SSL_new(SSL _CTX *ctx) API.

- The identical (client related) steps are carried out at the client side when it wants to connect to the server thats is specified by the application program such as the web browser.
Connection: Abstraction of secure sockets

- The socket identifier (created initially) is used to accept an incoming connection request on the server side using the accept() system call. The client initiates the TCP connection issuing connect() system call. It performs the TCP handshake (SYN, SYN/ACK, ACK) with the server.

- The SSL_set_fd(SSL *ssl, int fd) API is used to connect the SSL object (ssl) with a socket (fd) that is previously used to connect or accept at client or server side respectively. This operation automatically creates an interface between the SSL object and the established socket identified by fd.

- The SSL server issues the SSL_accept(SSL *ssl) API to accept the SSL connections. On the client side, the contrary function SSL_connect(SSL *ssl) API is used to connect to the specified server.

- The SSL_connect() at the client side and the SSL_accept() at the server side are responsible for executing the SSL handshake protocol and establish the SSL connection.

- These function automatically uses the read() and write() system calls to exchange SSL handshake data.

Application layer data exchange

- Once the SSL handshake is completed, the application layer data is exchanged. This exchange is protected under the just negotiated protection mechanism through SSL handshake.

- The application layer data is read and written using SSL_read() and SSL_write() APIs respectively.

Connection termination

- When either the client or the server wants to end the connection, they issue a SSL_shutdown(SSL *ssl) API. This causes client and server to send alert message notifying the termination of the SSL session. As a result, the client or the server notifies the other entity about the termination of the connection by sending the alert message. The alert message is described later in this chapter.

- After exchanging and confirming about the termination, the TCP connection is closed in a traditional way by sending FIN and ACK (of FIN) message.

- The client and the server deallocates the memory reserved for the SSL connection and closes the socket created for this connection.

- The server can opt to continue to listen on the listener socket that is just disengaged.

In this way, the SSL session over the TCP is carried out.

---

7 The arguments of the accept() system call are eliminated for brevity.
8 The arguments of the connect() system call are eliminated for brevity.
3.6 Summary

In this chapter, we described the SSL with respect to its (sub)layers and (sub)protocols. This chapter summarized the model of working of the SSL protocol above the TCP protocol. The SSL protocol is easy to use but needs to be carefully implemented due to its complex nature. With the information provided in this chapter, we are ready to dive in the main issues discussed further in this thesis.
This chapter focuses on the SSL renegotiation feature in the SSL protocol. Further, the DoS vulnerability due to the SSL renegotiation feature is explained. This leads to the discussion of an SSL (D)DoS tool called thc-ssl-dos [1]. The thc-ssl-dos is one of the recent (D)DoS tool developed by a group called The Hacker’s Choice [1]. This chapter explains the theory behind the working of this tool. A detailed analysis of this tool is provided on the basis of a detailed review and a number of tests performed. Finally, the mitigation techniques and their effectiveness are provided for the SSL renegotiation based DoS attack. Chapter 8 explains the experimental results of impact of this tool on the web server.

Section 4.4 describes one experiment that investigates the secure renegotiation support in the Apache web server. Finally, Section 4.5 summarize this chapter.

4.1 SSL Renegotiation

4.1.1 Definition

Session renegotiation is one of the SSL protocol feature where the client or the server can initiate a new session within a current active session.

Client initiated renegotiation Client can send the client hello message anytime during a current session to initiate the renegotiation for a new session. The server decides if it wants to negotiate a new session. If the server does not want renegotiate, it just ignores this message and renegotiation is not executed.

Server initiated renegotiation In an active connection, the server sends the hello request message to the client. On the receipt of this message the client can decide if the renegotiation should be continued. If affirmative, then the client sends a client hello and the renegotiation is carried out (the usual full SSL handshake protocol is executed).

If either of them (client or server implementation) does not wish to renegotiate then they can simply ignore the renegotiation request or send no renegotiation alert to the renegotiation requesting end. An end connection should ignore the renegotiation request if it is already under the negotiation phase [30].

1Although, the name of this group includes the word Hacker, this group is NOT an underground hacking club that performs illegal activities.
After the client sends client hello message, the handshake protocol is carried out. Each renegotiation carried out results in a new SSL session. This new session is identified by negotiated session ID in hello messages and data will be protected using the new session state. The only difference between a normal negotiation and a renegotiation is that the process of renegotiation is carried out (usually) in an encrypted mode unless the current session state was negotiated under NULL cipher.

### 4.1.2 Renegotiation application scenarios

This subsection discusses a few scenarios where a renegotiation appears to be an apparent choice in order to continue current connection state or to keep already negotiated connection alive.

#### Client authentication

A server may have been configured to require the client authentication for some resources (protected resources) and no authentication for some other (common) resources. An environment where client authentication is made compulsory by default for every client is not desirable because this is likely to exclude clients who does not need to access privileged resources but common resources. The SSL session is negotiated before any application data is actually received by the server. Therefore, there is no way the server can distinguish between the clients who intend to access protected resources and the clients who does not. Therefore, one refined approach would be to allow clients to connect to the server by default without client authentication and require authentication if the client requests the protected resource. This process could be achieved using the SSL feature called the SSL renegotiation. Therefore, whenever a client requests for a protected resource that requires the client authentication, the server can send the hello request message to the client to request renegotiation. The client can be authenticated in this renegotiation.

The SSL specification does not provide any guidelines for managing renegotiation. However, intuitively, whenever a server requires a client authentication and sends the hello request to request a new negotiation, the server has to stop serving the requests for the resources from client until the client gets authenticated in the renegotiation. Once the client is authenticated, the server can process the queued (awaiting) requests. Otherwise, the SSL connection is closed or at least (ideally) requests for protected resource is rejected.

#### Server authentication

Due to the probability that the server’s certificate can be expired in a certain time, the client may wish to renegotiate to confirm the server identity.

#### Cipher suite upgrading

If a particular set of data exchange requires higher security than already given by the existing cipher suite, the server or client can request renegotiation to upgrade the existing cipher suite.

One more application of SSL renegotiation feature is Server Gated Cryptography. In Server Gated Cryptography (SGC), the client can use stronger cryptography once the server authenticates itself with extension indicating that the certificate is a SGC-capable. This application of renegotiation was however useful when United States Export regulations were active. It was the time when not all products were allowed to use stronger cryptography.

---

2 For further understanding of Server Gated Cryptography please refer to and many other materials available on the internet
4.1 SSL Renegotiation

Replenishment of keying material

Whenever the SSL connection is likely to be long lived (for instance, a VPN (Virtual Private Network) type of connections) where there is also a possibility of exchange of large amount of data, key material should be renewed or generated on equal intervals. New key material is required to avoid few attack, for instance, CBC IV collision attack where IVs generated from key material are likely to be repeated after certain amount of period. Both the client and the server could ask for renegotiation for this purpose.

Replenishment is also necessary whenever there is a probability of compromise of the session key.

4.1.3 Authentication Gap in SSL renegotiation

The SSL Renegotiation is nothing but the establishment of new cryptographic parameters. The process of renegotiation is conducted in already protected (encrypted) session. When renegotiation is completed, the new session is started with the newly negotiated cryptographic parameters (new key material etc.). Therefore, there is NO GAP in the encryption of the data. However, there is no scheme to authenticate or identify that the client who is renegotiating is the one who had initiated original negotiation at some time in the past. An attacker can intercept the transport layer communication of the victim client and prefix his own communication with the server. This is referred as Authentication Gap.

There are several ways in which the authentication gap can be exploited. Following section explains the basic attack scenario caused due to this vulnerability.

Authentication gap: A Basic Attack Scenario

The attacker initiates the SSL handshake with the server. After the establishment of the session, the attacker requests some/any resources at the server in order to wait for the client (victim of the attack) to start the communication with the same server. When the victim starts the SSL handshake with the server (in plain text), the attacker intercepts this handshake traffic and forwards it to the server under the attacker’s encrypted connection. The attacker also forwards the respective handshake reply from the server to the (victim) client.

Therefore, the server treats this handshake traffic as the SSL renegotiation traffic from the attacker. Once the handshake is completed, the client (victim) communicates with the server over the newly established (protected) connection. Since, this connection is established over new security parameters that are obviously unknown to the attacker, an attacker cannot read this traffic. However, in this situation, the server has failed to differentiate the source of the handshake traffic.

4.1.4 Secure renegotiation

The SSL renegotiation feature in the SSL protocols introduces the vulnerability in which the attacker can maliciously inject (prefix) its plaintext to the client’s (HTTP) requests. This is possible because there is no binding between the original session and the renegotiated session. The RFC 5746 introduces an extension called RenegotiationInfo to provide the solution to attack described in previous section. The RenegotiationInfo extension has attribute called renegotiated_connection. The RenegotiationInfo extension allows the communicating parties to maintain and communicate securely with the SSL Renegotiation. To use this extension, both the client and the server needs to store three additional values (in session cache) for each connection state. These values are as follows:

---

3Discussion of such attacks is not in the scope of this research paper.
A **secure renegotiation flag** This flag indicates whether secure renegotiation is in use or not. The server set this flag if it finds an empty RenegotiationInfo extension in the client hello during the initial handshake. Similarly, the client set this flag if the server hello contains an empty RenegotiationInfo extension during initial handshake.

A **client verify data** This is the data from the finished message sent by the client on the immediately previous handshake.

A **server verify data** This is the data from the finished message sent by the server on the immediate previous handshake.

Whenever the client initiates the handshake, it includes an **empty RenegotiationInfo** extension in the client hello message. This indicates that the client supports the **secure renegotiation**. If the server also supports the secure renegotiation, it includes the same empty extension in the server hello to indicate the support for the secure renegotiation at the server side.

When the client initiates the renegotiation, the RenegotiationInfo extension contains the client verify data in the extension. Similarly, whenever the server initiates renegotiation, it includes the same information (from the client’s RenegotiationInfo extension) in the server hello along with the server verify data (client verify + server verify).

The structure of the renegotiation_info extension is shown in Figure 4.1. The first two bytes represent the extension type. Next two bytes indicate the length of the extension. In Figure 4.1 the length is given as 1. The length of this extension will be 1 in initial Client Hello and server hello. When the length of this extension is 1, then it is an empty renegotiation extension that merely indicates the support for the secure renegotiation.

```
0 1 2 3 4
0xFF 0x01 0x00 0x01 0x00
```

Figure 4.1: An Empty Renegotiation Extension Format

In this way binding between immediately previous session is created so that either of the client or the server can renegotiate securely.

### 4.2 The SSL renegotiation based DoS attack tool: thc-ssl-dos

The SSL renegotiation feature (with and without RenegotiationInfo extension support) introduces a different threat to the SSL protocols. This threat is DoS attack. This section explains the DoS threat due to the SSL renegotiation feature along with the proof-of-concept tool called **thc-ssl-dos** that exploits the SSL renegotiation feature to execute the DoS attacks on the SSL protocols.

#### 4.2.1 Background

The SSL handshake protocol deals with the expensive operations when compared with the SSL application data protocol. The cryptographic operations performed during the SSL handshake protocol are not symmetric between the server and the client. During the SSL handshake protocol, the server has to perform heavier operations when compared with the client. The cryptographic operations performed during the transmission of the application layer data (the SSL application data protocol) are symmetric between the server and the client. The cryptographic operations in the SSL protocol are described later in Chapter 6.
The traditional computational DoS attack focuses on initiating a large number of SSL handshake protocol with the server. The idea behind such attack is to force the SSL server to perform a large number of cryptographically expensive operations without performing any significant work at the attacker’s side. This attack intends to deplete the server’s computational resources to disable the server from serving the other legitimate users. The impact of such attack can be easily scaled up using multiple resources that can further open multiple SSL connections with the server.

4.2.2 SSL renegotiation based computational vulnerability

The attack strategy mentioned in this section is a general category of DoS attack where the attacker simply initiates a large number of SSL handshake requests to coerce the SSL server into performing unexpectedly large number of expensive operations. The SSL protocol includes one important feature called SSL renegotiation. The SSL renegotiation is already explained in this chapter. The SSL renegotiation is the process of starting a new handshake in an existing SSL session. The handshake conducted during the renegotiation is protected under existing session state. This means that the renegotiation handshake protocol messages are (mostly) encrypted unless the existing session uses NULL cipher. The normal SSL handshake is conducted in the plain text. This is the only difference between the usual handshake and the handshake carried out during the SSL renegotiation. Intuitively, the SSL renegotiation handshake is as expensive as usual SSL handshake.

In traditional DoS attack as mentioned in this section, the attacker opens a numerous SSL connections with the server to computationally exhaust the server. However, with the help of renegotiation feature in the SSL protocol, the attacker does not have to open many connections to increase the computational overhead on the server. Since, the SSL handshake is the culprit behind the computational overhead, the attacker can simply open a single connection and renegotiate multiple times under the same SSL connection. At the end, motivation behind the attack is to computationally exhaust the SSL server.

4.2.3 Work-flow of thc-ssl-dos tool

The thc-ssl-dos tool originally creates 400 peers. The maximum number of allowed peers are 999. However, this maximum number can be easily changed by tweaking the source code. Obviously, the effectiveness of this number depends on several factors such as availability of the bandwidth, the CPU power and OS (Operating System) etc.

The attack starts with the creation of the sockets. The sockets are created in the first step. The TCP connection is established using the file descriptors returned by these sockets. Once the TCP connection is established, OpenSSL APIs (Application Programming Interface) are used to handle SSL-related operations. These operations include the SSL connection establishment and the SSL renegotiation operations under the established SSL connection. The OpenSSL APIs that are used in the thc-ssl-dos tool are described below.

int SSL_connect(SSL *ssl) This API initiates the SSL handshake with the server. The SSL connection is identified by the SSL object passed to this function.

int SSL_renegotiate(SSL *ssl) This API sets the flag for the SSL renegotiation. This API does not performs the SSL renegotiation operation. It only sets the flag that signals the appropriate API (explained below) that the SSL renegotiation can be performed.

int SSL_do_handshake(SSL *ssl) This API performs an the actual SSL handshake. This API is responsible for completion of the entire SSL renegotiation.

\footnote{The number of peers denotes the number of distinct connections with the target server}
int SSL__free(SSL *ssl) This API decrements the reference count of the SSL object and
removes the respective SSL structure from the cache memory [8].

THC-SSL-DOS tool makes use of a non-blocking I/O. Therefore, one function does not
block the functionality of the other function. This implies that the every function used in
this tool has to return immediately in spite of its success or failure. It helps in switching
control from one function to another and it is performance efficient. The non-blocking I/O
increases the scalability of the attack.

The attack starts with creating a single peer to connect to the target server. Peers are
dynamically expanded in the program. Whenever a TCP connection is successfully completed
for any peer, a new peer is initialized and introduced in the attack. A new peer is introduced
in the attack till the time maximum peer limit is not crossed. Once the new peer is introduced
in the attack, socket is created for this newly created peer. This socket is used to establish
the new TCP connection with the server. On successful establishment of the TCP connection,
the SSL connection is established using the standard OpenSSL API SSL__connect(SSL *ssl). Once the SSL connection is established, the server expects the application layer data protected
under the negotiated cipher suite. Since the tool is created to attack the server, instead of
sending any application layer data, it sends the renegotiation request to the server.

If server sends any data to the peer created through this tool, the data is read from the
server and ignored. After every 50 renegotiations, a dummy data is written on the server.
The number 50 has no significance. The dummy data is nothing but a byte of value zero.
The act of writing something (dummy data) is required to handle a possibility where the
server may close the connection from the client (peer = client) if no (application layer) data
is sent in the connection for longer time.

The peer is disconnected if one of the following condition occurs.

- If there is a Time Out and a TCP connection cannot be established for a socket. The
default time out is set to 10 seconds in this tool.
- If a TCP connection cannot be completed successfully.
- If the server does not support the SSL protocol. This condition is not specific to any
  single peer. If this condition occurs then thc-ssl-dos tool terminates immediately.
- If the SSL connection fails for some reasons other than the support for the SSL. Some-
times the SSL connection cannot be completed if underlying BIO (I/O abstraction) [8]
has not completed its operation. An appropriate error condition is handled and appro-
priate actions are taken to satisfy the requirements to complete the SSL connection.
- If the SSL renegotiation fails for some reason.
- If SSL renegotiation fails for some unknown reasons then it is concluded that server
does not support SSL renegotiation feature. This condition is not specific to any single
peer. When this condition occurs, the tool terminates with "SSL renegotiation is not
supported" error message. This tool cannot work if renegotiation is not supported at
server side.

The peer disconnection process consist of series of operations. Whenever a peer is to be
disconnected, the API function SSL__free(SSL *ssl) is called. If the SSL__free(SSL *ssl) is
called without calling SSL__shutdown(SSL *ssl) then the session is removed from the session
cache. Therefore, whenever a new connection with the same SSL structure is established then
the SSL session will not be resumed and complete handshake will take place. After calling
SSL__free(SSL *ssl), the file descriptor of the socket is closed.

[A BIO is an I/O abstraction, it hides number of the underlying I/O details from an application [8].]
4.2.4 Impact

Several sources, including the analysis by Eric Rescorla at [42] highlights that the way to measure the impact of computational DoS is by the ratio of the work the attacker has to do, and the server has to do. Greater the ratio more is the impact. In other words, the attack on the server is more effective if the server is made computationally overloaded with lesser computational investment from the attacker’s side. To understand the difference in the traditional computational DoS and renegotiation based DoS, Eric Rescorla has provided a simple theoretical analysis of the computational DoS attack at [42].

The traditional computational DoS is subject to sending three packets to complete the TCP handshake for each new SSL connection. Renegotiation based computational DoS requires to perform TCP handshake (three packets) only once at the starting of the attack.

The traditional DoS attack can be constructed in two way. Either perform all the client side cryptography and overload the server with the large number of SSL connection requests or send the canned messages (constructed in advanced). Technically, if the aim of the attack is merely to overload the server with the heavy cryptographic operations, then the attacker has to send only the client hello message and the client key exchange message. This is because the main reason, the server might get computationally overloaded is due to the decryption of the client key exchange message sent by the attacker. In addition, if the client key exchange message is canned and does not contain correct data inside, the server is going to return the fatal error and close the connection after decryption is finished. Therefore, theoretically, sending handshake protocol messages until the client key exchange messages should be sufficient.

It is very important to parse the server hello message before sending the client key exchange message. This is not considered at [42]. The OpenSSL SSL-server implementation does not expect the client key exchange message as long as (at least) server hello is completely sent. Therefore, it is important to parse the server’s response after sending the client hello message to confirm if the server has sent the server hello message, and it is safe to send the client key exchange message.

The renegotiation based DoS attack needs to follow the complete SSL handshake protocol. Because, in the renegotiation based attack, the attacker needs to maintain the SSL connection with the server. The renegotiation is possible only if the SSL connection exists. The SSL protocol is very sensitive to the inconsistency in the protocol messages. Therefore, if the server expects a specific message, and if it does not receive that particular message, the server terminates the SSL connection issuing the "fatal error" alert. It does not request for the re-transmission of the message. For example, if the size of the encrypted client key exchange messages is not equal to what server expects, it closes the connection. Consequently, the renegotiation based computational DoS attack needs to follow SSL protocol strictly. This adds the extra cryptographic operations and hence the overhead at the attacker’s side. This overhead is not present while conducting traditional DoS attack described in this section.

However, there are number of advantages to the SSL renegotiation based DoS attacks. The need of the separate SSL connections is dramatically reduced. Only one TCP connection needs to be established (three packets) for performing more than one handshake with the server. This increases the effect of the attack with respect to the single machine. The traditional attack is limited by the number of concurrent sockets that can be created and the available bandwidth to scale the attack. As the SSL renegotiation based attack can create the same computational overhead on the server with the lesser sockets, on a given system this attack is likely to perform better than the traditional DoS attack. Chapter 8 performs a number of DoS-attack strategy-comparison experiments to confirm these theoretical evaluations.
4.3 Mitigation techniques and their effectiveness

This section provides some mitigation techniques for the SSL renegotiation based DoS attack. Further, the effectiveness of these techniques is also discussed.

4.3.1 Server control

The server might have been configured to serve single connection with a single thread. Such threads might have been allowed to use a specific amount of the CPU and the memory. Such configuration is not uncommon \[42\]. In such configuration, the renegotiation based computational DoS can overload the single thread pertaining to the renegotiating connection and making it slower. This cannot affect the other threads serving to the legitimate users. Hence, such threading mechanism cannot be completely affected by the renegotiation based computational DoS. On the other hand, the traditional computational DoS attack can effectively attack such servers because it can create a large number of connections with the server and can produce the computational overhead. The server can deny the legitimate users due to the probability of the lack of the threads to serve the connections.

Under the traditional computational (D)DoS attack, it is very hard to block the sources of such attack as the attacking machines could be distributed over a wide area of the network. Blocking some part of the network sources introduces the risk of blocking the legitimate users. Whereas, in the case of renegotiation based DoS attack, the attacker machines are likely to be limited in numbers (since they are likely to produce the same effect with lesser machines). However, the machines can also be distributed over the large network area making it equally hard to block such attacking resources like the traditional distributed attacking mechanism.

The renegotiation is carried out in the cipher text mode. Therefore, it is not easy to detect DoS attacks based on the renegotiation. Traditional IDS/IPS systems, including anti-DOS mechanisms and firewall configuration cannot easily detect these attacks. In the renegotiation based attack, the packets are encrypted and hence it is not easy to understand the contents inside the packets. On the similar lines, IDS/IPS cannot be really configured to detect such attack unless and until the server is already overloaded.

4.3.2 Disable renegotiation

The simplest way to mitigate renegotiation based DoS attack is to disable the renegotiation feature on the server side. Certainly, this is not the best way to mitigate any attack (The authentication gap and the Computational DoS attack described earlier in this chapter). The SSL renegotiation is one of the important features of the SSL protocol. The applications of the SSL renegotiation are already mentioned in this chapter. The administrator of the server should consider the requirement of his/her server and take appropriate action about disabling the SSL renegotiation. SSL renegotiation can be initiated both by the clients as well as the servers. The better choice would be to disable the client initiated renegotiation. However, this will not be the best thing to do since some of the application where the client initiated renegotiation is required cannot work. Finally, as previously said, the choice of disabling the SSL renegotiation depends on the application requirements.

4.3.3 Heuristic based SSL Handshake Rate Limitation

The SSL renegotiation is carried out in the encrypted mode. Therefore, it is usually not possible to detect such packets with the help of usual packet filtering mechanism. However, it is possible to infer the existence of such packets by monitoring the format of the encrypted SSL messages.
4.3 Mitigation techniques and their effectiveness

Figure 4.2: Wireshark Screen-shot: SSL renegotiation packet is flagged with 'PUSH'.

By looking at the packet, it cannot be decided if the packet is encrypted or not. Therefore, some kind of structural aspect of the SSL handshake needs to be considered. Following list gives some structural peculiarities of the renegotiation message.

- If the handshake protocol message type is unknown, then such packet can be considered as the SSL renegotiation message. Since if the record type is displayed to be the handshake protocol message, it is confirmed that the record payload contains the handshake protocol message. Next step is to observe the handshake message type. There are 10 types of handshake messages. If the handshake message type is none from this standard list of handshake types, then such record can be considered as the SSL renegotiation message.

- The renegotiation messages are usually encapsulated in a TCP packet flagged with "PUSH" [7].

Vincent bernat [7] has written a simple iptable entry for the packet filtering detecting a possible encrypted SSL renegotiation message following the logic given above.

Figure 4.2 shows the output from packet capturing software wireshark. Observe that the Encrypted handshake message has TCP PUSH flag set. When TCP payload is observed, the first five bytes denote that this record consist of SSL handshake message (Handshake message identifier: 16). The TLS protocol version 1 is used (03 01). These three bytes are followed by two other bytes denoting the length of the TCP payload (00 e0). This confirms that this is a handshake message. Please note that the SSL record header is not encrypted. But rest of the record data is encrypted. The next byte after the length bytes of the record header denotes the handshake type. The handshake type is found to be 51. The number 51 is not a known handshake type. Therefore, heuristically it can be assumed that this packet is an encrypted packet. In addition, this packet consist of the record header of valid handshake type. Therefore, it can be concluded that this TCP packet consist of encrypted SSL renegotiation message.

The iptable entry written by Vincent Bernat [7] is given in Listing 4.1.

```
1 payload="0 >> 22 & 0x3C @ 12 >> 26 & 0x3C @"
2 $IPTABLES --A LIMIT_RENEGOCIATION \
```
SSL Renegotiation and Computational DoS

This packet filter works if there is no TCP fragmentation. The payload on Line 6 is defined to access the TCP payload inside the IP protocol. The protocol selected is TCP protocol and only packet with PUSH flag set is selected. Line 6 is important entry in this iptable entry. This line is responsible for checking the record type residing inside TCP payload. It is checked whether the record has valid header with correct indication of the SSL protocol version. (Either SSLv3.0, TLSv1.0, TLSv1.1 or TLSv1.2). This is followed by the checking if 6th byte (Record header is 5 byte long and 6th byte (first byte of handshake message header) consist of handshake type) is not a valid handshake message type. If these conditions are satisfied then the message inside the encrypted TCP packet is assumed to be the SSL renegotiation message. Further hashlimit match extension is used to limit the rate of these packets. The entries in Listing 4.1 are explained in the Appendix B in detail.

In this way, the renegotiation requests can be kept limited using a simple heuristic technique to mitigate the computation DoS using the SSL renegotiation. However, this technique is likely to produce number of false positive result. Therefore, this technique cannot become the de-facto standard to completely mitigate the renegotiation based computational DoS.

4.4 SSL Renegotiation support in Apache web server

The Apache web server is tested for its support for the secure renegotiation to highlight the behavior of the Apache web server. The Apache web server is tested to investigate how it handles the renegotiation to protect the server from renegotiation based computational DoS attack.

4.4.1 Test for renegotiation

The OpenSSL has its own functionality to test whether the SSL renegotiation is enabled. The OpenSSL application called s_client can be used to emulate the SSL client. This client can be used to test the support for the SSL renegotiation. Listing 4.2 shows the output of this application when connected to the server. On Line 1 the OpenSSL s_client is executed with -connect option to connect to the server identified by the IP address 192.168.15.125. This machine runs the Apache/2.2.14 server. The standard port for the SSL services is 443. For brevity, the output of the s_client application is abbreviated.
4.5 Summary

This chapter introduced the SSL renegotiation feature. There are several applications where the SSL renegotiation appears to be suitable to be enabled while using the SSL protocols. However, SSL renegotiation does not necessarily add much value to the SSL protocols. Later, this chapter introduced two attacks based on the SSL renegotiation.

The first one is the attack, where an attacker can prefix his plaintext in the secure communication between the client and server. This vulnerability is called authentication gap. This vulnerability exists because the way SSL renegotiation is conducted in the SSL communication. This vulnerability is tackled by using the secure renegotiation instead of usual renegotiation. The secure renegotiation is specified in RFC 5746 [41].

The second attack is the computational DoS attack. This attack is applicable even when the secure renegotiation is implemented. The SSL based computational DoS attack focus on initiating a large number of SSL handshake within single SSL connection. The mitigation


```
New, TLSv1/SSLv3, Cipher is DHE-RSA-AES256-SHA
Server public key is 2048 bit
Secure Renegotiation IS supported
Compression: zlib compression
Expansion: zlib compression
SSL-Session:
  [...] 
R
RENEGOTIATING
3077585640: error:1409E0E5:SSL routines:SSL3_WRITE_BYTES:ssl handshake failure:s3_pkt.c:591:
```

Line 15 indicates that Secure Renegotiation is supported on the server side. The SSL renegotiation is triggered by sending letter 'R' on an empty line after the SSL connection is established. Line 22 indicates that the renegotiation is in progress. In spite of being advertised on Line 15 that secure renegotiation is supported, Line 23 reports the failure message. With this message the SSL connection is terminated immediately. It should be noted that the version of OpenSSL used is up to date.\textsuperscript{6}

The similar experiment is performed replacing the Apache web server with the emulated SSL server implemented using s_server command (application) of the OpenSSL, triggering renegotiation by sending R on an empty line does trigger successful renegotiation.

Similar tests are also performed on older version of Apache (Apache 1.13) server and the renegotiation was successfully triggered using the same commands. Therefore, it can be observed that recent versions of Apache just forbid the renegotiation in spite of advertising the otherwise. However, this is not the complete truth. The latest version of the Apache does support the secure renegotiation. However, only the server can initiate the renegotiation. That is, only server initiated renegotiation is allowed.

This way, the latest versions of Apache web servers can mitigates the renegotiation based DoS attack.

\textsuperscript{6} The experimental set-up uses OpenSSL 1.0.0e 6 Sep 2011 version
techniques and their effectiveness are explained in this chapter. At the end, the way latest versions of the Apache web servers handle the SSL renegotiation, is described with the help of practical results.

This chapter mainly introduced the SSL renegotiation based DoS attack and explained the working of the thc-ssl-dos DoS tool. The comparative experiments are performed in Chapter 8 using this tool to evaluate the theoretical analysis made in this chapter.
CHAPTER 5

DoS Attack Analysis: SSL Record Processing

This chapter considers the number of test cases to verify the applicability of the DoS attacks due to the way SSL data is handled at the Record layer. This chapter is the result of the detailed review of the SSL protocol and its implementation OpenSSL.

In the first section, the processing of the SSL record is considered to examine if it is properly done and it does not induce any vulnerabilities that could lead to DoS attacks. Some experiments are inspired from the vulnerabilities present in TCP protocol that cause DoS attacks.

Section 5.2 talks about the compression mechanism in the SSL protocol. The compression mechanism in the SSL protocol is not widely used. There are several known issues in using compression in general. These are discussed in this section. However, it is very important to analyse the compression mechanism in the SSL to produce the guidelines for the future use of the compression and explain the probable overhead that the compression might introduce. In order to understand the compression during the record layer processing, a detailed source code review of the OpenSSL implementation is done and the detailed analysis is provided along with the explanation of sample code to describe the compression mechanism in the SSL protocol.

Finally, this chapter is concluded with some final remarks in Section 5.3.

5.1 Record Processing

This section considers the strengths of the SSL record processing in resistance to the DoS attack. This section examines the SSL record processing for the common DoS vulnerabilities those are still a threat for the other related protocols such as TCP. Finally, an evaluation is provided whether the DoS attacks identified and described in this section are applicable to SSL protocols. If these attacks are applicable to the SSL protocol then to what extent these attacks are applicable is also described.

5.1.1 Replay/Re-order attacks

Experiment 1. Replay/Re-order SSL records while sending it to the SSL server.
Explanation

This experiment focuses on testing the behavior of the SSL server when repeated or re-ordered records are sent to it. The repeated SSL record might create confusion for the SSL server and consume extra CPU cycles and memory to correctly process or discard them.

Protection provided by TCP protocol

The TCP protocol provides the protection against the replay and re-ordering attacks by including the field called "sequence number" inside each TCP packet. The receiving end of the TCP packet discards all the TCP packets which have the same sequence number as those of the previously received TCP packet in the current time window. The same sequence number also signifies the correct order of the TCP packets. The damaged packets are detected by verifying the correctness of the checksum included in the TCP packets.

The SSL protocol runs on the top of the TCP protocol. The sequential delivery of data to the SSL layer is the responsibility of the TCP layer. Therefore, SSL assumes that all the packets are in correct order.

Analysis

When an attacker acts as a client, he can construct the legitimate TCP packets containing the duplicate SSL data (TCP payload) and send it to the server. This will circumvent the replay protection by the TCP protocol. However, the server will discard such packets due to the way record is created. Consider a scenario where the attacker replays the same record and not the TCP packet. Please note that, in this scenario, the TCP packet will have distinct "sequence numbers" and hence these TCP packets will be treated as distinct TCP packets. Hence, the protection mechanism at TCP protocol layer will not discard these packets and these packets will be submitted to the record layer for further processing. However, as already explained in Chapter 3, the connection state of the record layer maintains the sequential counter to predict the "sequence number" for each new record received.

Therefore, when an attacker crafts two SSL records with the same sequence number, the MAC verification will fail at the server side. For instance, assume that record y is duplicate of the record x with sequence number 2. When the server receives record x, the sequence number counter at the server gives value 2. The MAC is calculated and it is verified with the MAC data included in the record. The same procedure is followed for record y. This time, the sequence number counter at the server side gives value 3. Where as, the record y being the copy of the record x includes the MAC data with sequence number 2. Therefore, the MAC data calculated at the server side and the MAC data included in record y does not match. This results in the server issuing the "Bad MAC" alert. This alert is always fatal alert and leads to the termination of the SSL connection.

In the similar way, the records that are reordered results in the termination of the connection.

When the data inside the record is kept same and the sequence numbers are calculated correctly, the SSL server will always discard such repeated data if received during the SSL handshake phase. This is because, the SSL server expects a specific data that has been specified by the protocol. For example, if the server receives the Client Hello message, the SSL server expects the client key exchange message if it is regular handshake or the Client Certificate message if the client’s certificate has been requested.

If the attacker constructs the SSL records with repeated data (SSL plaintext) but with sequential and correct sequence numbers, these records bypass all the protection mechanisms discuses so far in this section. Such repeated records is sent to the services running at the
5.1 Record Processing

higher application layer. The decision of discarding or accepting such data is left to the application layer services. Such repeated data cannot significantly affect the performance of the SSL server since the attacker cannot create such SSL records in advanced due to the fact that symmetric shared key is not available to the attacker before the SSL session establishment and the calculation of the sequence numbers cannot guessed in advanced. Therefore, with such attack, the attacker will have to invest the same amount of resources as that of the server. The server is always the one who has greater resources than that of the client.

Therefore, the attacks that involve the replaying or re-ordering SSL records cannot produce DoS attack condition.

5.1.2 Missing record attack

Experiment 2. Do not send some part of the SSL records as described in Figure 5.1.

Explanation

Consider the scenario presented in Figure 5.1 where Record 1 and Record 2 are divided into 2 parts each to accommodate inside the TCP packet as shown. However, the attacker does not pack a part of the Record 1 inside the TCP fragment. The part of record that is not sent is highlighted with the light blue color.

There is a possibility that SSL server will recognize the missing part of the record and request the client to re-send the record again. This might result in degradation of the server performance due to the fact that server will have to send a "missing record" retransmission request (1 TCP packet). On the receipt of such request, the attacker does send the required part of the SSL record (that was intentionally not sent) in order to continue with the SSL session. This will also result in slowing down the SSL connection.

![Figure 5.1: Record boundary division: A scenario where a part of Record 1 is not sent.](image)

Analysis

Let us assume that there is no problem in TCP packet construction and transmission. All the three TCP packets are received by the server in correct order. When the server starts reading the record data inside the packet, the server cannot assemble the SSL record correctly according to the information specified in the record header that was received through the first TCP packet as shown in Figure 5.1. This results in termination of the SSL connection.

This is confirmed by reviewing the SSL implementation OpenSSL. Whenever the server cannot find the complete record whose length is obtained from the record header, it issues the "record decode error". This error is always fatal and hence results in termination of SSL connection.

Therefore, the attack where an attacker sends record with some part of it missing, cannot degrade server’s performance to produce the DoS attack condition.
5.1.3 Incomplete session establishment attacks

**Experiment 3.** Do not complete the SSL handshake protocol in order to keep the SSL server engaged in the SSL connection establishment phase (SSL handshake phase) for longer time.

**Explanation**

This experiment focuses on the behavior of the SSL server when the SSL handshake is not completed in order to engage the SSL server in the SSL connection establishment phase for longer time. This experiment is considered due to the fact that such type of attack is possible on TCP protocols and produces the hazardous effects on the HTTP servers. This attack can be called TCP SYN flood attack. It is described in Section 5.1.3.

**Description: TCP SYN flood attack**

The TCP SYN flood attack is a type of protocol exploit DoS attack in which the attacker exploits the weakness of the TCP handshake process that is needed to establish the TCP connection.

In TCP handshake process, the client sends a SYN packet (TCP connection request) to the server. The server replies with the with the SYN+ACK packet. This reply is sent by the server to inform the client that the server has received its SYN packet and it can send next packet that is required to complete the TCP handshake. When the client receives the SYN+ACK packet, it sends the ACK packet to complete the TCP handshake.

In TCP SYN flood attack, the attacker sends the SYN packet and does not send the ACK packet when SYN+ACK packet from the server is received. As a result, the server waits for the attacker to complete the TCP handshake process. But the attacker’s ACK packet never arrives at the server side [23]. This consumes the server’s resources for a longer time. This is because the server keeps waiting for the ACK packet from the client. Increasing number of such half-opened connections will bind the server’s resources until no new connections can be made. Since, no new connections can be made, it will produce the DoS condition for the legitimate users who tries try to connect to the server.

In one of the variable of this attack, the attacker sends the TCP SYN packet with spoofed source IP address. With this the server replies with SYN+ACK to the spoofed IP address. The IP address that receives the SYN+ACK from the server does not respond to such packets since it has never sent the SYN packet to the target server. Therefore, server will wait until the TCP connection timer expires at the server. This is the time for which the server will wait for the connection to be completed.

**Analysis**

The SSL handshake protocol is started once the TCP connection is established. Therefore, there will not be any situation where an attacker can open TCP handshake and does not complete it to produce any SSL based DoS attack. However, the basic TCP SYN flood attack is still possible on the SSL servers since the SSL runs on the top of the TCP protocol. The basic SSL functionality is not affected by such attack since this attack focuses on not completing the TCP handshake. Therefore, the SSL handshake does not even start to consume any SSL specific resources.

On the similar lines of the TCP SYN flood, if an attacker sends the Client Hello and does not send the client key exchange message (client key exchange message is the next the client is supposed to send after Client Hello message) then the SSL connection terminates.

---

1If client authentication is used, the client sends its certificate before the client key exchange message
immediately. Generally, the SSL servers are configured to handle heavier cryptographic operations and large number of connections. Therefore, such an investment of the resources only for handling a single Client Hello message does not significantly affect the server’s performance.

5.1.4 False data handling attack

Experiment 4. Send the unexpected SSL messages to the server.

Explanation

This experiment focuses on testing the SSL server’s behavior when unexpected SSL message is sent to the server. This experiment is considered in this thesis due to the fact that whenever the unexpected TCP packet is received by the server (or the client) such that it acknowledges the TCP packet that has not been sent yet, then the server (or the client) has to follow certain protocol that leads to a DoS attack condition. This attack is originally described in [21]. The mechanism behind this attack is given in Section 5.1.4.

Description: TCP Storm attack

The work at [21] describes one of the DoS attack on the TCP protocols due to the flaw in the TCP specification. This flaw has arisen due to the way TCP handles the unexpected packet that contains a wrong sequence number inside the TCP packets. In particular, according to TCP RFC[22], whenever a TCP connection is established and following condition occurs:

A packet is received with an ACK field that acknowledges data that has not been sent yet. That is, the packet containing the ACK field with the number greater than the sequence number in SYN packet sent earlier.

The receiver should send the last sent packet again and drop all the incoming packets in the current segment.

In TCP Storm attack, the attacker sends the TCP packet (as described above) in already established TCP connection to both the server and the client on behalf of each other. The packets sent by the attacker have the appropriate "sequence" and "acknowledgement" numbers. It is assumed that the attacker can obtain these numbers in advanced by monitoring the network traffic for adequate time period. On the receipt of such packets, each side of the connections send the ACK packet for the received TCP packet. When other side receives such packet with ACK field greater than the SEQ number in the last sent packet (since, in fact, these packets are sent by the attacker and not by opposite connection end), each side follows the TCP protocol: They discard the ACK packet received and send the ACK packet that was sent last time.

Therefore, both the end again receives the packet with ACK number greater than the their SEQ number. Therefore, they keep on following the TCP protocol as specified in [22] by sending last sent packet again and discard packet with greater ACK field.

This loop continues as both the connection ends keep receiving packets with the ACK number larger than their sequence number (of the last sent TCP packet). This process is stopped only when both these packets are dropped (or lost) in the connection or when either end reaches a timeout and ends the connection by RST (RST message is a TCP packet used to notify the connection termination).

The results in the experiment obtained from [21] suggests that sending only 2 packets, each at one side of the connection end can cause an amplification factor of over 261,000 times the original size.
Analysis

This attack is a type of DoS attack that is limited to the client-server pair. That is, the DoS attack is limited to the client under consideration and the services provided by the server will not be available to the client under consideration only. Such attack cannot directly affect the other legitimate clients connecting to the server unless a large amount of server’s resources are engaged in this single connection which very much unlikely.

It has been claimed in [21] that, this attack is also applicable to SSL servers. However, the attack should be limited when SSL is used. The attack can affect the SSL servers because the SSL works on the top of the TCP layer and the SSL depends on the TCP for delivery of packets in the correct order. However, core functionality of the SSL protocol is not affected by this attack.

5.2 Compression in SSL

This section investigates the compression utility available in SSL protocols for possible causes which can lead to the server’s performance degradation and possibilities of DoS condition. Section 5.2.1 provides the background for compression explaining how and when compression works better. The possible options of the compression method in SSL and how compression can improve or degrade the server’s performance is explained in Section 5.2.2. Section 5.2.3 provides the difference between the HTTP level compression and SSL level compression. Finally, this leads to the compression test case that is explained in Section 5.2.4.

5.2.1 Background

One of the main factor in reducing the latency of the website is the amount of data that needs to be transferred in a communication over the network. There are other factors that also affects the response time of the service provided on the web server. Some of the important factors could be listed as network bandwidth, physical memory and CPU power. However, compression can play important role in increasing the data transmission throughput.

Compression is the method to reduce the length of the data so as to save the space in the storage devices. When transmitted, compressed data can reduce the bandwidth requirement. Compression ideally should reduce the length of the data and must be lossless. That is, compression and decompression (decoding compressed data to original format) should not loose any data. However, sometimes, compression do not satisfy these basic requirements. For example, sometimes compression instead of reducing the length of the data increases it or do not make any significant change in size. This is because, the compression works by identifying and eliminating redundant patterns in the data. This is one of the reason compression works better (i.e. gives significant reduction factor) when data has repeated patterns.

In SSL communication, the data to be transmitted is in encrypted format. Encrypted data is often random in nature. Therefore, compressing the encrypted data cannot guarantee significant reduction in data size. Therefore, compression in SSL is done before the data is encrypted. This is already explained in Chapter 3.

5.2.2 Utility of Compression in SSL

Compression is not mandatory in the SSL protocols. The compression utility cannot be used at any random time in the client-server communication. Compression method is negotiated in the SSL handshake along with cipher suites. It is mandatory to include compression-Method.null method (NO compression) in the Client Hello message during SSL handshake.
This is required for compatibility where the server does not support the compression can select compressionMethod.null method [30].

Currently, there are two compression methods defined for SSL other than compressionMethod.null (No compression). RFC 3749 describes DEFLATE compression method and RFC 3943 describes Lempel-Ziv-Stac (LZS) compression method. The OpenSSL has only implemented DEFLATE compression method.

DEFLATE compression method as well as LZS compression method requires separate compression history to be maintained at both the connection end [35, 43]. The compression history contains an extra information needed to decompress the data at the other communication end. Maintaining history achieves a better compression ratio as the compression history needs to be shared only once per session and can be used through out the session.

LZS compression history contains the last 2KB of plaintext of the TLS session [43]. A byte-long header is prepended with each compressed fragment indicating the details of compression of the particular fragment. The details of the encoding can be found in [43].

In practice, only Chrome browser with version 6 and above supports SSL compression. To check the support for SSL compression in web-browsers few simple tests were conducted on several browsers. The results are observed using the Wireshark network protocol analyzer. The example screen shot is included in Appendix B showing support for the DEFLATE compression method in the Chrome browser.

Compression is a tradeoff between network bandwidth and CPU cycles. Compressed data (ideally) requires lesser network bandwidth. However, the process of compression consumes extra CPU cycles as well as requires an extra memory to maintain the compression state through-out the SSL session [2]. According to [44], dynamic contents utilizes 10 – 20% CPU cycles with reduction factor of 70 – 80%.

A website may consist of different types of contents including plaintext data, data from web-forms, images (for instance, JPEG images), already compressed contents such as .gzip files etc. As already discussed in this section compression ideally should reduce the size of the data. However, compression of images (.JPEG), already compressed data (.gzip, .tgz) does not give significant reduction factor in compression [44]. This might lead to utilization of CPU cycles with no gain in terms of reduction in bandwidth requirements.

5.2.3 HTTP (Level) Compression and SSL (Level) Compression

There could be some confusion between HTTP compression and SSL compression. Often websites provides the facility to compress the data. This is referred as HTTP level (or HTTP compression) compression. The HTTP compression requires web browser client side support. Almost all the browser supports the HTTP compression and almost all the performance aware websites uses compression. To enable the communication in compression mode, the server needs to be notified that the client (the browser) accepts compressed encoding. This is done by using HTTP field Accept-Encoding = ”gzip” in the HTTP request. In this encoding, the gzip compression method is declared as being supported.

In the SSL protocol, the compression method is negotiated during the SSL handshake (SSL connection establishment). In SSL compression, the compression is done prior to encryption. Enabling the http compression and the SSL compression simultaneously is not quite useful. the SSL level compression will not give the higher compression factor since the data will be already compressed at HTTP level.

2This memory requirement may vary for different compression methods.

3ASP pages are used as dynamic contents.
5.2.4 Compression Burst Handling

Considering the fact that SSL does have an option to use compression at SSL level, it becomes necessary to investigate the implementation of the compression mechanism at SSL level. Compression can introduce some memory problems and requires extra CPU cycles as seen from [44]. Therefore, in protocols such as the SSL where the memory usage and CPU usage is always critical, it is important to assess the impact of the compression in the SSL protocols.

**Experiment 5.** Send a maliciously crafted SSL record such that when it is in compressed format, it follows the boundary limit imposed by the SSL specification, but when decompressed, it expands into a large amount of data that is not allowed by the SSL specification and/or the SSL implementation cannot handle it.

**Explanation**

According to [30], compression must be lossless. The compression ideally should reduce the length of the plain text fragment. However, sometimes length of the compressed data increases instead of decreasing. In such scenario, the length of the compressed data fragment should not be greater than the original plaintext length. The plain text is divided into fragments of size $2^{14}$. According to SSL specification, the length of compressed fragment should not be greater than $2^{14} + 1024$ bytes (Worst-case scenario). At the other (connection) end, the length of the decompressed fragment is expected to be less than or equal to $2^{14} + 1024$ bytes. If compressed data decompresses into more than $2^{14}$ then the SSL implementation should issue a fatal decompression error. An adversary might take an advantage of incorrect implementation of this condition in the SSL protocol. The adversary can craft a data in such a way that the compressed length is less than or equal to $2^{14} + 1024$ bytes but it decompresses into the length much greater than $2^{14}$ bytes. Such conditions might create situations such as buffer overflow if the implementation softwares are not prepared to handle the larger data than expected. It can also results in software to crash.

In order to verify if the OpenSSL does implement the (de)compression as expected and does not introduce any memory related problems, the OpenSSL source code is reviewed. Following section describes the OpenSSL source code related to compression and investigates the experiment described in this section.

(De)compression in OpenSSL

Listing 5.1 gives the snippet from OpenSSL where the length of compressed record and decompressed record (after de-compression) is checked. This snippet is a part of the function that is used to get a new input record. The terminologies used in Listing 5.1 are summarized in Table 5.1.

```
if (s->expand != NULL)
{
    if ((r->length > (16384 + 1024) + extra))
    {
        al=SSL_AD_RECORD_OVERFLOW;
        SSLerr(SSL_F_SSL3_GET_RECORD,SSL_R_COMPRESSED_LENGTH_TOO_LONG);
        goto f_err;
    }
    if (!ssl3_do_uncompress(s))
    {
        al=SSL_AD_DECOMPRESSION_FAILURE;
        SSLerr(SSL_F_SSL3_GET_RECORD,SSL_R_BAD_DECOMPRESSION);
        goto f_err;
    }
```

5.2 Compression in SSL

Listing 5.1: Compression Boundary Check

Line 1 checks if the (de)compression method is NULL. If this method is not NULL then
the data in the record is expected to be a compressed data. Line 3 checks if length of the
record (compressed data fragment) is less than 16384 + extra. Here, variable extra can
take values either 0 or 16384. This is because some versions of Internet Explorer sends SSLv3
records of size greater than 16384 bytes [29, 45]. If SSL_OP_MICROSOFT_BIG_SSLV3
_BUFFER is defined (enabled) in implementation then it allows packets with greater sizes.
Finally, it is correctly checked in OpenSSL that the compressed data is not greater than
2^{14} + 1024 bytes or greater than 2^{14} + 1024 + 2^{14} bytes when large records are expected.

<table>
<thead>
<tr>
<th>Name</th>
<th>Significance</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>SSL Structure (SSL *s)</td>
</tr>
<tr>
<td>s-&gt;expand</td>
<td>(de)compression structure (containing details of the (de)compression method)</td>
</tr>
<tr>
<td></td>
<td>inside SSL *s structure</td>
</tr>
<tr>
<td>rr</td>
<td>SSL Record structure</td>
</tr>
<tr>
<td>rr-&gt;length</td>
<td>Length of block (data payload) inside rr</td>
</tr>
<tr>
<td>extra</td>
<td>extra = 0 or extra = 16384</td>
</tr>
<tr>
<td>al, SSLerr,</td>
<td>Error handling</td>
</tr>
<tr>
<td>goto f_err</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.1: Terminology used in Listing 5.1

ssl3_do_uncompress(s) decompresses the record. It is shown on Line 9. Failing to
uncompress correctly results in DECOMPRESSION_FAILURE fatal error. After decom-
pression, on Line 17 the length of the plaintext (now, rr->length indicates the length of the
decompressed data fragment) is checked to be less than 2^{14} + extra. If the length of the
expanded record message is found to be greater than 2^{14} + extra then RECORD_OVERFLOW
alert is issued. This alert is always fatal and results in SSL connection closure. This
is how any anomaly that could result in buffer overflow or excess memory consumption is
avoided.

It is important to note that OpenSSL uses zlib library to perform the compression
and decompression operations. There are no recent buffer overflow attacks known on zlib
library.

Figure 5.2 shows the hierarchy of function calls for expansion (decompression) of the
compressed data fragment. The first method from Figure 5.2 int ssl3_do_uncompress (SSL * ssl)
is called at Line 9 as shown in Listing 5.1. This function then calls the function
int COMP_expand_block (ssl->expand, rr->comp, 16384, rr->data, (int)rr->length). The first argument in
this function ssl->expand denotes the decompression method to be used. This is the method that is negotiated during the SSL handshake
and stored in the SSL structure. The expand is nothing but the respective compression

---

4 The rr structure is updated dynamically.
5 At the time of performing this research.
DoS Attack Analysis: SSL Record Processing

Figure 5.2: Method Tree (Function Hierarchy) for decompression in SSL

structure. Depending upon the compression structure next function is called. In SSL, stateful compression is preferred and in OpenSSL, by default the stateful compression function is called. Therefore, the third function in the hierarchy is \texttt{static int zlib\_stateful\_expand\_block (COMP\_CTX * ctx, unsigned char * out, unsigned int olen, unsigned char * in, unsigned int ilen)}.

Listing 5.2 gives the excerpt from OpenSSL source code that initializes the zlib state (lines 6 to 10) that is used to decompress the data fragments. Lines \texttt{[13 to 16]} initializes the variables in zlib state. Line \texttt{[18]} calls the \texttt{inflate} method that finally does the process of decompression. The \texttt{inflate} method has been provided with input buffer and its length (compressed data fragment and its length) and output buffer and its length (space for decompressed data fragment and its expected length). The inflate method is a standard zlib library implementation and decompresses as much data as possible and stops when input buffer becomes empty or the output buffer becomes full.

Listing 5.2: Decompression Method call

```
static int zlib\_stateful\_expand\_block (COMP\_CTX * ctx, unsigned char * out, unsigned int olen, unsigned char * in, unsigned int ilen)
{
    int err = Z\_OK;
    struct zlib\_state * state =
        (struct zlib\_state *)CRYPTO\_get\_ex\_data(&ctx->ex\_data,
            zlib\_stateful\_ex\_idx);
    if (state == NULL)
        return 0;
    state->istream\_next\_in = in;
    state->istream\_avail\_in = ilen;
    state->istream\_next\_out = out;
    state->istream\_avail\_out = olen;
    if (ilen > 0)
        err = inflate(&state->istream, Z\_SYNC\_FLUSH);
    if (err != Z\_OK)
        return -1;
    return olen - state->istream\_avail\_out;
}
```
5.2 Compression in SSL

The malicious data that is compressed and follows the maximum length restrictions of the compressed data fragment but decompresses into a very large data (greater than maximum plain text length allowed ($2^{14}$)) will not create any damage to the server. This is because, as soon as the data being expanded reaches the boundary restrictions, the process of expansions stops and the expanded data is returned to the calling function. The OpenSSL double checks if the data that is expanded follows the limit imposed by the SSL specifications. This double check can be seen from Listing 5.1 at Line 17.

## 5.2.5 Computational Overhead due to Compression

As already mentioned in this section that the process of compression adds the extra computational overhead on the client and server processing. If compression is used in the communication, it is required to maintain the compression history through out the session. The process of compression and decompression requires few extra CPU cycles. At times, compression can add an extra delay in processing the compressed and decompressed data according to the availability of input and output buffers.

### Practicality of attack

One can think of the attack where the client requests a server to decompress a large amount of data multiple times. This can introduce the performance degradation at server side. However, the effect of such attack cannot be significant as client will also have to compress the data before a server can decompresses it. One alternative of such attack where the client can compress the data in advanced and reply it in the new session. In this alternative, the client will have to take care of the compression history and compress the SSL plaintext according to the compression history. Therefore, in this alternative, the client by pass the extra computational overhead due to compression. The cost of process of MAC calculation, of such compressed data fragment, the cost encryption of the final SSL data are the costs.

<table>
<thead>
<tr>
<th>Name</th>
<th>Significance</th>
</tr>
</thead>
<tbody>
<tr>
<td>state</td>
<td>zlib_state structure variable used to maintain the history of compression</td>
</tr>
<tr>
<td>ctx</td>
<td>Compression structure</td>
</tr>
<tr>
<td>sslexpand</td>
<td></td>
</tr>
<tr>
<td>in</td>
<td>input buffer (compressed data fragment)</td>
</tr>
<tr>
<td>llen</td>
<td>length of input buffer</td>
</tr>
<tr>
<td>out</td>
<td>Output buffer (space for decompressed data)</td>
</tr>
<tr>
<td>comp</td>
<td></td>
</tr>
<tr>
<td>olen</td>
<td>SSL3 RT_MAX_PLAIN_LENGTH ($2^{14}$ bytes)</td>
</tr>
<tr>
<td>Z_SYNC_FLUSH</td>
<td>zlib library constant</td>
</tr>
</tbody>
</table>

Table 5.2: Terminology used in Listing 5.2 and in Figure 5.2.
that client will have to pay at run time. This is because, the MAC calculation requires the
sequence numbers those cannot be obtained in advanced and the encryption process requires
a symmetric key that can be only obtained once the SSL session is negotiated.

It should also be noted that almost no one uses the SSL level compression. In addition,
clients can only give the choice of the compression algorithm to the server. According to
the protocol, SSL clients have to include the \texttt{compressionMethod.null} (No Compression)
method for compatibility issues such that those servers which do not support any compression
method offered by the clients can choose the \texttt{compression.null} method. Therefore, it is very
difficult to take an advantage of the compression feature in SSL to surely impact the server’s
computational overhead.

5.3 Summary

This chapter addressed some tests cases those focused on investigating the DoS vulnerabilities
in SSL record processing and compression mechanism. Some of the DoS attacks that are
possible due to some vulnerabilities in TCP protocol are still possible when SSL is used.
However, these attacks does not directly impact the functionality of the SSL protocols. The
attacks such as replay, re-order attacks when analogously implemented in SSL protocol, they
do not work as they work on TCP protocols. For instance, the TCP SYN attack that aims at
opening the half-opened connections with the server. But, when the SSL handshake is started
and left incomplete, the SSL connection terminates immediately if required SSL handshake
message is not found by the server.

The compression mechanism in the SSL protocol is optional. Unlike HTTP compression,
the decision whether to use compression in a connection is made during the SSL connec-
tion establishment phase (during SSL handshake). The plaintext is compressed before it
is encrypted to achieve better compression ratio. In spite of wide usage of SSL protocols,
not many web servers and web browsers supports the SSL level compression. The OpenSSL
source code is reviewed to examine whether it follows the boundary limitations imposed by
the SSL specification correctly. It has been found that, the implementation of compression
utility in the OpenSSL does not introduce any memory faults which could lead to the system
crash or similar problems that eventually can cause DoS attack. The exact overhead that
compression in SSL can introduce needs to be measured and it is not covered under this
thesis work.
This chapter aims at providing the evidence that the SSL handshake protocol is not balanced in terms of the cryptographic overhead between the server and the client. Section 6.1 summarizes the cryptographic operations performed in the SSL protocol. It also provides the experimental analysis of the cryptographic costs incurred during the SSL protocol. This section leads to the need of understanding the cryptographic overhead distribution among client and server. Therefore, Section 6.2 provides the distribution of the cryptographic operations which needs to be performed by the client and the server. With this understanding, this section includes a experimental proof of the unbalance nature of the SSL handshake protocol.

Finally, Section 6.3 concludes this chapter with some final remarks.

6.1 Cryptographic Operations

This section introduces the cryptographic operations performed in the SSL protocol. The expensive nature of the overall SSL protocol is verified using the experimental results obtained in this section.

6.1.1 Introduction

The SSL protocol is responsible for authentication of communicating entities, negotiation of cipher suite and compression method, and finally integrity and confidentiality of the application data. To do so SSL employs number of cryptographic operations. Table 6.1 summarizes the approximate comparative analysis for the cryptographic operations utilized in the SSL protocol. The Speed is denoted by the arithmetic symbols. The + denotes the faster operations and − denotes the slower operation. The comparative speed is denoted by variable number of + or − signs. The speed column in Table 6.1 suggests the rough idea for comparing the types of the cryptographic operation and does not consider the performance (speed) differences among the different operations within the same category.

6.1.2 Cost of cryptographic operations

Experiment 6. Measure the cost of different cryptographic operations to understand their roles in the SSL being the computationally expensive protocol.

Generally only "server" is authenticated.
Table 6.1: A simple performance comparison of different cryptographic operations in SSL protocol

<table>
<thead>
<tr>
<th>Operation Type</th>
<th>Operation Example</th>
<th>Speed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Public Key encryption</td>
<td>RSA encryption</td>
<td>+</td>
</tr>
<tr>
<td>Public Key decryption</td>
<td>RSA decryption</td>
<td>-</td>
</tr>
<tr>
<td>Signature generation</td>
<td>RSA signature generation</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>DSA signature generation</td>
<td>+</td>
</tr>
<tr>
<td>Signature verification</td>
<td>RSA signature verification</td>
<td>+</td>
</tr>
<tr>
<td></td>
<td>DSA signature verification</td>
<td>-</td>
</tr>
<tr>
<td>Cryptographic hashing</td>
<td>MD5</td>
<td>+ + +</td>
</tr>
<tr>
<td></td>
<td>SHA1</td>
<td>+ + +</td>
</tr>
<tr>
<td></td>
<td>SHA256</td>
<td>+ + +</td>
</tr>
<tr>
<td>Symmetric encryption</td>
<td>RC4</td>
<td>+ +</td>
</tr>
<tr>
<td></td>
<td>AES-128 CBC</td>
<td>+ +</td>
</tr>
<tr>
<td></td>
<td>AES-256 CBC</td>
<td>+ +</td>
</tr>
<tr>
<td></td>
<td>DES CBC</td>
<td>+ +</td>
</tr>
</tbody>
</table>

Explanation

This experiment aims at providing the overview of the comparison of the cost of the different cryptographic operations involved in SSL protocol. To understand the basic (theoretical) differences in the cryptographic operations is given in Table 6.1. Considering these comparative differences further practical tests are performed to understand the performances given by these operations on the system.

Set-up & Tools

Tests are conducted on the Linux virtual machine with CPU model Intel(R) Core (TM) i5 CPU 2.53 GHz. The Operating System is Ubuntu 10.04.4. OpenSSL 1.0.1. is installed on this machine as the SSL implementation. OpenSSL provides a functionality called speed with which speed of the cryptographic operations can be measured. It should be noted that the data shown in Figure 6.1 is the result of tests performed on Linux Virtual Machine. These statistics can be different for different machines. However, this figure is indicative of the differences in their performances (speed).

Result & Analysis

Figure 6.1 gives the overall comparison of cryptographic operations. The X-axis denotes the number of cryptographic operations used in SSL protocols. The Y-axis indicates the number of operations which can be performed per second. Cryptographic hashing is clearly fastest among other operations listed in this bar-chart. The second fastest operations are symmetric key operations. The signature generation and verification operations are the slowest as shown found in the test.

To understand the difference between signature algorithms Figure 6.2 is compiled comparing only digital signature schemes. This figures shows that RSA signature verification operations are clearly faster than RSA signature generation. The difference in DSA signature generation and verification is less than it is in case of RSA signature scheme. In addition, DSA signature verification operations are slightly slower than DSA signature generation operations.
6.1 Cryptographic Operations

Figure 6.1: Performance Comparison of Cryptographic Operations

These operations are done with key size of 1024 bits. The difference in performance will continue to exist and grow with the increase in key sizes.

Figure 6.2: Digital Signature Performance Comparison

It is promptly observed that DSA signature generation and verification operations are quite balanced (shown in Red color bars) unlike RSA signature generation and verification (shown in Blue color bars). However, according to [17], virtually all websites make use of RSA signature certificates and not of DSA signatures. In addition, DSA cipher suites if used are used in ephemeral mode. In Ephemeral mode, DH (Diffie-Hellman) keys are generated on the fly and are signed and verified adding substantial work on both client and server sides [29]. Finally, when RSA is used as key exchange algorithm produces unbalanced work overload on the server and client side. This is explained in detail in section 6.2.
6.2 Distribution of work overhead

From experiment performed in section 6.1, it is clear that signature generation, verification (private key encryption, public key encryption) operations are the set of slower operations among other cryptographic operations used in SSL protocols. They cause significant amount CPU overhead. Table 6.2 shows the distribution of these operations among client and server in SSL handshake protocol.

It can be observed from Table 6.2 that server performs more private key operations and hence server is the one who has to work more in order to complete the handshake protocol as compared with client. Therefore, SSL handshake protocol is itself asymmetric in terms of work overhead especially when RSA is used as key exchange algorithm and authentication method which is the common practise in real world [47].

6.2.1 Client-Vs-Server: Computational Differences

Experiment 7. Measure the difference between computational overhead for the client and the server during SSL handshake protocol.

Explanation

This experiment intends to measure the cost of overall SSL connection establishment phase. This phase mainly consist of SSL handshake protocol and SSL change cipher spec protocol 2. The SSL connection establishment phase is the most expensive phase of the SSL protocol. This experiment is indicative of this fact and provides the practical evidence to it.

6.2.2 Set-up & Tool

To confirm the difference in computational overload, a small tool built by Vincent Bernat [48] is modified to measure the computational cost incurred during SSL handshake protocol. This tool is a simple C program. This program uses OpenSSL library to emulate server and client side. It measures the CPU time of a client and a server performing 1000 handshakes with various TLSv.1.0 [30] cipher suites and varying key sizes.

This tool is used to measure the difference between CPU time required by the client and the server when RSA and DH is used as key exchange algorithms. The standard procedure of server authentication is performed. The client authentication is not performed in

---

Table 6.2: Public Key Cryptographic Operations in SSL during handshake

<table>
<thead>
<tr>
<th>Operation</th>
<th>Client with Client authentication</th>
<th>Client without Client authentication</th>
<th>Server with Client authentication</th>
<th>Server without Client authentication</th>
</tr>
</thead>
<tbody>
<tr>
<td>Public key operations</td>
<td>Verify Server’s certificate(s)</td>
<td>Encrypt premaster secret with server’s public key</td>
<td>Verify client’s certificate(s)</td>
<td>No Operation</td>
</tr>
<tr>
<td>Private key operations</td>
<td>Generate CertificateVerify message</td>
<td>No Operation</td>
<td>Decrypt client’s Key exchange message to recover premaster secret</td>
<td>Decrypt client’s Key exchange message to recover premaster secret</td>
</tr>
</tbody>
</table>

---

2Although, SSL session establishment phase can consist of change cipher spec protocol, alert protocol along with the SSL handshake protocol, the SSL session establishment phase will be referred as the SSL handshake protocol unless any protocol specific feature needs to be discussed.
this measurement. The same system as described in Section 6.1.2 is used to conduct the experiment.

While generating key pairs for RSA, public exponent of value 65537 is used which is a default value for public exponent in OpenSSL. On the similar lines, while generating DH public value for the server a generator of value 2 is used. This is a default value for the generator in OpenSSL. This experiment is performed using different cipher suites available for usage in TLSv.1.0 [30]. It has been found that the different cipher suites does not make significant difference in cryptographic overhead when only SSL handshake protocol is considered. Therefore, finally, TLS _RSA _WITH _AES _256 _CBC _SHA cipher suite is used for RSA key exchange method and TLS _DHE _RSA _WITH _AES _256 _CBC _SHA cipher suite is used for DH key exchange method. In both cases RSA certificates are used for authentication. These tests are performed two times each for RSA and DH. Each time using different key sizes.

![Figure 6.3: Comparison of computational power needed by server and client to complete 1000 SSL handshakes](image)

6.2.3 Result & Analysis

Figure 6.3 shows the difference in CPU overload at client and server side. The CPU power needed by the server and client is shown in red and blue color bars respectively. The X-axis denotes the CPU time needed by the server and the client to complete the 1000 handshakes. The Y-axis denotes the key exchange algorithms and key sizes used to test the performance. The observations from Figure 6.3 are summarized as below.

- Whenever RSA is used, client needs more CPU power than server as compared with DH key exchange algorithm.

- With the increase in key size the performance overhead is increased at both client and server sides.

\[ More \ about\ client\ authentication\ and\ its\ impact\ is\ discussed\ in\ Chapter 7 \]
• When RSA with key size of 1024 is used the difference in overhead is 130%. This difference in overhead increases with the increase in RSA key size.

• This scenario of increased overhead at server side changes when DH is used as key exchange algorithm. Client needs to perform more than server when DH is used as key exchange algorithm. With DH public value of 1024 the difference in computational overhead is $-22.83\%$ For 2048 key size the overhead on client increases even more.

The difference in overhead shown in Figure 6.3 is due to how RSA and DH works. This is explained in Appendix sections A.1 and A.2 respectively. It should be noted that these are the results of the tests conducted on virtual machine and only provides the representative data as a proof to the concept that the server has to more cryptographic overhead than the client. This is the case when RSA is used and most of the browsers and web servers prioritize only RSA key exchange algorithm. In addition, the differences in overhead on the client and the server are going to vary from platform to platform. For instance, the difference in overhead in client on a normal PC and a well configured web server with additional hardware accelerators for performing cryptographic operations (RSA private key decryption) is going to differ. The difference in the client and the server overhead is most likely to be small. This experiment is performed and results are given here just to place an idea that server needs more CPU power than client. In addition, the real web servers (on the internet) will have to provide its services to not just a single machine. There will be number of client machines those will be requesting server’s resources.

6.3 Summary

This chapter provided the introduction to the cryptographic operations performed in SSL protocol. Further the cost of different cryptographic operations are compared and heavy cryptographic operations are identified. It has been verified that the SSL handshake protocol is the most expensive protocol in SSL protocol. The cryptographic overhead between the client and the server is not balanced. The server is the one who has to perform more CPU intensive operations than the client. Such unbalanced environment provides the attacker an opportunity to request the server to complete multiple handshake protocol and spend large amount of CPU power with insignificant CPU investment himself. When the attacker uses the multiple resources to spawn the multiple SSL handshake requests for the server to complete, the impact on the server performance can be increased drastically.

Finally, this chapter provided the practical proof of concept results to highlight the important cause for the SSL protocol to be the attractive basis for executing DoS attacks.
This chapter describes number of experiments regarding the client authentication process using RSA certificates in the SSL protocols.

Irrespective of the place where certificate verification is done (at the client side or the server side), there is no difference in the overall process. However, when it is done at the server side it is an extra overhead for the server along with RSA decryption. That means, whenever certificate verification is done at the server side the server has to perform the RSA signature verification (public key operation) along with the RSA decryption (private key operation) for obtaining the pre-master secret as explained in Chapter 3. This might lead to significant degradation in the server’s performance.

This chapter investigates whether the certificate verification at the server side could be an extra computational overhead leading to the degradation of the server’s performance and hence become a possible vector for producing DoS attack. To do so, various aspects of the certificate verification are considered in this chapter.

7.1 Introduction

The SSL handshake protocol optionally authenticates the client to the server. There are several applications where client authentication is required and demanded by the server. Some of these applications are already discussed in Chapter 4. Most of the times, RSA certificates are used in the process of authentication in the SSL protocol 47. The process of RSA certificate verification consist of number of steps. The main steps can be listed as follows:

- Certificate chain construction.
- Certificate constraint verification / Integrity checking.
- Certificate Signature verification.
7.2 Certificate Chain Construction

7.2.1 Background

Whenever the server requires the client authentication, the server sends the certificate request message to the client after sending its own certificate message and before sending the server hello done message in the SSL handshake protocol. Figure 3.2 in chapter 3 explains the message sequences in the SSL handshake protocol. When the client receives the certificate request from the server, the client is supposed to send its certificate before it sends the client key exchange message. The client also sends the certificate verify message to prove that the certificate sent is indeed belongs to the client. The client verify message is sent after the client key exchange message.

The server cannot possibly trust the certificate provided by the clients. There are almost uncountable clients. Trusting each client’s certificate is practically impossible. Therefore, when a server asks for a certificate a client should provide a list of certificates containing its own certificate along with a set of certificates leading to an acceptable certificate authority. The certificate provided by the client should be inherently trusted by the server to complete the client authentication. That is, this list of certificates should contain at least one certificate that is trusted by the server.

The diagrammatic representation of a basic certificate chain that can be used to validate the client certificate is shown in Figure 7.1. The first certificate (left-most) in this diagram is the client certificate. The issuer of this certificate is ICA (Intermediate CA). This certificate is originally issued by the CA. The client can be authenticated successfully if the client sends these three certificates (Client, ICA and CA) to the server and if and only if the server trusts either the CA certificate or the ICA certificate and signature included in client, ICA and CA certificates can be verified by the server.

Figure 7.1: A Basic Certificate Chain

The SSL specification [31, 30, 32, 49] requires that the list of certificates should start with the client’s own certificate followed by the issuer of the client’s certificate and in this way leading to the certificate/certificates that is/are trusted by the server.

7.2.2 Process certificate chain construction

When the server receives a certificate chain from the client, a temporary stack is created to store these certificates. Then the last certificate in the stack is retrieved. This certificate is supposed to be the client’s certificate. The issuer of the certificate is searched using the issuer name on this certificate. Once the issuer certificate is identified, a further check is done to confirm that the certificate found as an issuer of the client certificate is indeed the true issuer. The standard procedure is described in the RFC 4158 [50]. However, the OpenSSL does not employ all the check necessary to find a valid issuer of the certificate to build the certificate chain. For example, an Authority Key Identifier (AKID) should be matched with Subject Key Identifier (SKID) of the issuer in order to confirm that bearer of the SKID is

1CA:Certificate Authority
surely issuer of the AKID [50]. But OpenSSL checks this constraint only if SKID field is present in the certificate of the subject. The discussion of this issue is out of the scope this thesis.

This procedure of finding the issuer of the certificate is continued till the time it finds the self signed certificate in the chain. The final CA certificate is always self signed. This way the certificate chain is constructed in the OpenSSL.

7.2.3 Certificate Loop

Experiment 8. Send the list of client certificates in such a way that it creates a loop at the server side while constructing the valid certificate chain.

Explanation

The idea behind this experiment is to construct the malicious certificates such that the final CA certificate is never found by the server.

![Figure 7.2: The set of malicious certificates that forms a loop during certificate chain construction](image)

The client can send the certificates as shown in Figure 7.2. The client sends a set of 5 certificates. One of them belongs to the client. As observed from Figure 7.2, If an issuer of the certificates is to be retrieved recursively then the order of retrieval will be the client certificate that is always retrieved initially and found at the first place in the list provided by the client, then the issuer of the client certificate is searched. In the example given in this figure, the issuer of the client certificate is certificate with the subject name A, so briefly the certificate searching procedure can be continued as follows: Client(A), A(B), B(C), C(A), A(B), B(C), C(A), ... It should be noted that a loop has formed in the process of retrieving the issuer certificates. Therefore, chain building process will never completed successfully. This can create hazardous effects for the server in terms of the CPU cycles invested for this process.

Analysis

The OpenSSL source code snippet is given in Listing 7.1 where certificate chain is constructed. The terminology used in this listing is summarized in Table 7.1.

```plaintext
for ( ; ; ) {
   }
```

The certificate chain can also be formed with only 2 certificates. More certificates are used in this example to visualize and explain the scenario clearly.
```c
if (depth < num)
{
    break;
}
if (ctx->check_issued(ctx, x, x))
{
    break;
}
if (ctx->untrusted != NULL)
{
    xtmp = find_issuer(ctx, sktmp, x);
    if (xtmp != NULL)
    {
        if (!sk_X509_push(ctx->chain, xtmp))
        {
            /* Error Handling */
            [...]  
            x = xtmp;
            num++;
            continue;
        }
    }
    break;
}
```

As shown in Listing 7.1, the depth of the certificate chain generated (so far) is checked against the depth value (More about it later.) at the starting of the unconditional for loop at Line 3. Initially the stack of the certificates denoted by the `ctx->chain` contains only the client’s own certificate. The variable `num` contains the number of certificates in the certificate stack `ctx->chain` that is updated in the for loop. Therefore, initially (at line 3), `num` equals 1. This for loop is ended if the self signed certificate is detected. On Line 7 static int check_issued (X509_STORE_CTX *ctx, X509 *x, X509 *issuer) function is used to check if a certificate is self issued. The `ctx->untrusted` is the stack containing all the certificates sent by the client. On Line 13 an issuer of the certificate `x` is searched from `sktmp`. The `sktmp` is the duplicated stack of `ctx->untrusted`. Therefore, the stack `sktmp` also contains all the certificates sent by the client. If the issuer of the certificate `x` is found in the stack of the certificates containing client’s certificate (sktmp) then it is added into another stack `ctx->chain` (line 16). For brevity, unrelated lines from this code snippet are not shown (line 20). Once the certificate is pushed into the another stack `ctx->chain`, the variable `num` is incremented and for loop is continued. As already explained, the first check in the for loop is to check if `num > depth`. Whenever, this condition is reached, the for loop is ended.

The variable `depth` indicates the number of certificates allowed to be searched or retrieved before finding the trusted certificate. This variable can be set to any integer value. Setting this value very large cannot be very useful. With the increase in the number of the certificates in the constructed chain, the computational overhead on the server also increases. This is because, with the increased number of certificates, the server will also have to verify the signatures in each certificate. As already seen in Chapter 6, the signature verification does require considerable CPU time. The default value for this variable in the SSL configuration file of the Apache web server is set to 10. Therefore, by default, the SSL server running on

---

3Client certificate is previously pushed into the stack. This is not shown in 7.1.
### Table 7.1: Terminology used in Listing 7.1

<table>
<thead>
<tr>
<th>Name</th>
<th>Significance</th>
</tr>
</thead>
<tbody>
<tr>
<td>depth</td>
<td>A variable set initially to limit the length of certificate chain</td>
</tr>
<tr>
<td>num</td>
<td>Number of certificates in the chain stack</td>
</tr>
<tr>
<td>x</td>
<td>Certificate of type X509</td>
</tr>
<tr>
<td>ctx</td>
<td>Structure variable of type X509_STORE_CTX</td>
</tr>
<tr>
<td></td>
<td>This structure is used while verifying the certificate chains.</td>
</tr>
<tr>
<td>check_issued</td>
<td>Method to check subject-issuer relation</td>
</tr>
<tr>
<td>untrusted</td>
<td>Stack of certificates containing all the certificates sent by the client</td>
</tr>
<tr>
<td>sktmp</td>
<td>duplicate of untrusted</td>
</tr>
<tr>
<td>find_issuer</td>
<td>Function that finds the issuer of the provided certificate.</td>
</tr>
<tr>
<td>xtmp</td>
<td>Certificate of type X509</td>
</tr>
<tr>
<td>sk_X509_push</td>
<td>Push the given certificate in specified stack</td>
</tr>
</tbody>
</table>

Apache can accept certificate chain of length 10. The impact of length of certificate chain on the server’s performance is discussed in the next section.

### Conclusion

Because of the **depth** limitation, an attacker cannot successfully construct the attack described in Experiment 8. Even if the attacker is able to construct malicious certificates as shown in Figure 7.2, the certificate chain construction in the OpenSSL as shown in Figure 7.1 does not allow retrieval of more than **depth** number of certificates. If the certificate chain cannot be built successfully as in this case (see Figure 7.2), the certificate chain building fails since no trusted CA can be found then the certificate chain construction is said to be unsuccessful. Once the certificate chain construction fails at the server side, the SSL handshake protocol terminates immediately.

### 7.3 RSA Certificate Verification (Signature Verification)

#### 7.3.1 Role of size of exponents

The RSA public key operation (signature verification) is cheaper process than the RSA private key operation (signature generation) process. Due to the property of the RSA and the key generation process \(^4\), the public key exponent \((e)\) is usually (very much) smaller when compared with the private key exponent \((d)\).

Certainly, if the size of the public exponent is increased, the signature verification operation should become expensive. The SSL specification does not provide any guidelines on the allowed size of the public exponent. The implementation of the SSL, the OpenSSL does not directly specify the maximum allowed size of the public exponent either. However, there are few limitation entailed on the size of the public exponent by the OpenSSL while performing the signature verification process.

```c
1  [ . . ]
2  if (BN_ucmp(rsa->n, rsa->e) <= 0)
3     { /* Error Handling and Reporting */
4       return -1;
5     }
6
\(^4\)Briefly explained in Section A.1
Listing 7.2: Constraint on the public exponent (e) in OpenSSL

Listing 7.2 depicts the code snippet from the OpenSSL source code. This code snippet is a part of the core RSA signature verification method implemented in the OpenSSL. The terminology used in this listing is summarized in Table 7.2.

<table>
<thead>
<tr>
<th>Name</th>
<th>Significance</th>
</tr>
</thead>
<tbody>
<tr>
<td>rsa → n</td>
<td>RSA modulus (n from rsa structure (defined in OpenSSL))</td>
</tr>
<tr>
<td>rsa→e</td>
<td>RSA public exponent (e from rsa structure (defined in OpenSSL))</td>
</tr>
<tr>
<td>BN_ucmp</td>
<td>A compare function. It takes two arguments of type BIGNUM structure (Defined in OpenSSL), it returns 1 if first argument is greater than the second argument</td>
</tr>
<tr>
<td>BN_num_bits</td>
<td>Returns the number of bits in the argument passed to this function</td>
</tr>
</tbody>
</table>

Table 7.2: Terminology used in Listing 7.2

Line 2 of Listing 7.2 makes sure that the length of the public exponent is less than the modulus. Whenever the modulus is greater than 3072 bits, the public exponent cannot be greater than 64 bits. This condition is ensured on the Lines 8 and 10 of Listing 7.2. Hence, the OpenSSL allows the signature verification with RSA key that have public exponent of size up to 64 bits when the RSA modulus is greater than 3072 bits and public exponent of size 3071 bits when the RSA modulus is equal to 3072 bits.

Ideally, there are number of constraints that are followed while selecting the public exponent e. One of the main constraint is that e should be prime. One more constraint (or common practise) is e should be as small as possible. The public exponent and the private exponent are inversely proportional to each other. The private exponent is not preferred to be very small. This is because the security of the RSA public key cryptography depends on the length of the private exponent. Greater private exponent ensures increased security. However, there are no constraints on creating the RSA key pair. Thats is, one can create RSA key pairs with extremely large public exponent that need not be a prime number. As a result, RSA certificates containing the signatures that are created from such RSA keys will require more time for verification when compared to verification of signatures created with comparatively smaller public exponents.

7.3.2 Chain of certificates

The certificate verification involves the process of signature verification. Once a certificate chain is built (as described in Section 7.2) and the various constraints on the certificates are verified, the server starts verifying the signature inside each certificate. This verification process involves the verification of the signature inside the certificate. The issuer of the certificate signs the certificate. The verification of the certificate signature is nothing but the RSA signature verification process when RSA certificates are used. The signature verification
process is done for each certificates provided by the client except the CA certificate that is trusted by the server. The certificate signature verification process starts from the certificate issued by the CA certificate in the chain. This process continues up to the last certificate that is the client’s certificate.

The expensive operation that the server has to perform in the SSL handshake is the RSA decryption to obtain the pre-master secret. It is already explained and described in Chapters 3 and 6. However, in spite of the RSA signature verification process being the cheaper operation than the RSA decryption operation, it can create some overhead on the server when client authentication is required. When the client certificate is required by the server, the client can send the long list of valid certificates. Longer the chain of the certificates, greater are the number of the signature verification operations that server has to perform. The default value for the maximum allowed length of the certificate chain is 10 as set in the SSL configuration file of the Apache web server. In addition, the client can also send the certificates with larger public exponent (see Section 7.3.1) which makes the signature verification operation expensive.

This leads to the need of next experiment. This experiment is described in Section 7.3.3.

### 7.3.3 Long chain of certificates with large public exponent

**Experiment 9.** Send the chain of the client certificates with maximum possible depth. Measure the performance of the server for:

1. The different public exponent values.
2. The different depth of the certificate chain.

The motivation behind this experiment is to investigate the impact of the client authentication with different client certificates on the performance given by the server.

**Experiments performed**

In this experiment, 9 set of tests are performed using 3 different public exponent values against 3 different certificate chain depths.

The RSA key pairs to generate respective certificates are generated using the OpenSSL library [8] and the RSA package from the Go library [51]. All the certificates used in this experiment are created using the utilities provided by the OpenSSL [8]. These utilities are the `req` to create leaf (the end entity or client) certificates and the `ca` to create CA certificates. These certificates are created with the RSA key pair having modulus of size 3072 bits. The maximum allowed public exponent value in the OpenSSL is 3071 bits long provided that modulus is 3072 bits long. This is already explained in Section 7.3.1 of this chapter. Since one of the intension behind this experiment is to investigate the impact of the size of public exponent on the server performance when client authentication is used in the SSL handshake, it was necessary to use the public exponent of the maximum size (i.e. 3071 bits long) to compare the impact with the other common values of the public exponents. To maintain the symmetry in the results, the size of the modulus was kept the same for all categories of tests in this experiment.

**The choice of public exponents** Three different exponents are selected for this experiment.

\[ e = 65537 \] The first one is 65537. This is standard value of the public exponent used in OpenSSL [8].
Highest 32 bit prime The second value of the public exponent is the largest 32-bit prime number (4294967291). The exponentiation operations are expensive. The motivation behind choosing this value was the fact that most of the applications expects the public exponent to be 32 bits. However, it could be confirmed during this project.

3071 bit random number The third exponent used in this experiment is a random number of size 3071 bits. The RSA key pair with public exponent of 3071 bits long is generated with the Go library.

The choice of certificate chain length Three different certificate chain length configurations are selected for this experiment.

Single certificate The first configuration contains the certificate chain depth equal to one. That is, the client sends only single certificate to the server. This certificate is such that it is trusted by the server.

Chain of 3 certificates The second configuration contains the three certificates in the certificate message sent by the client. This chain contains the certificate of the client, two intermediate CA and the final CA trusted and stored in the trust store of the server. The trust store of the server contains all the certificates that server trusts. The number 3 is selected since most of the time, the chain of the certificates contains three certificates.

Chain of 10 certificates This configuration contains the chain of 10 certificates. This configuration is not likely to exists in the real world scenario. However, this configuration is considered in this experiment due to the fact that the default value for the maximum allowed length of the certificate chain is set to 10 in the SSL configuration file of the Apache web server.

A note on basic constraints

In one of the test case of this experiment, certificate chain of length 10 is used. The scenario where the certificate chain of length 10 is being used could be very much rare. This scenario is used since, the default SSL configuration file of Apache web server allows this length. However, it is administrator’s job to decide the maximum allowed certificate length. The X.509 version 3 certificate consist of an extension called basic constraint. One of the attribute of this extension is path length. This attribute in the particular certificate indicates the number of certificates that can appear in the chain below this trusted certificate and the end entity certificate. Therefore, if any CA certificate has this attribute set to 3 then, the certificate chain starting with this certificate can no longer than depth 3. The CA certificate is considered at depth 0.

The server implementation in the OpenSSL verifies if the path length constraint is followed in the constructed chain of the certificates. This check is performed after constructing the chain from the certificates provided by the client and before the actual signature verification. Therefore, the server does not perform the certificate verification if the path length constraint is violated. That is, if a certificate is maliciously used to sign another certificate even if it not allowed to do so through path length attribute. The standard Internet X.509

---

5 Number of sources such as [2] and [52] mentions that most of the applications expects the size of the public exponent to be 32 bit at the maximum.

6 Thanks to Adam Langley [52] for providing the patch to tweak the Standard RSA package of the Go library.

7 The other attribute of the basic constraint extension is CA, this is already described at the starting of Section 7.3.2. This attribute indicates if it is allowed to sign other certificate or not.
Public Key Infrastructure Certificate (RFC 5280) [53] does not mandate to include this field in the certificate. In addition, when this field is not present or set to \texttt{NONE}, there is no limitation on the length of the certificate paths built from a client certificate (end-entity certificate) that would lead to the considered CA certificate [53]. Therefore, the configuration where this experiment uses chain of 10 certificates is not completely impossible or unacceptable. For example, few real world evidences are provided in the Appendix\texttt{E} in Section\texttt{B.4}. It is shown that the few CA does have path length constraint set to \texttt{NONE}. The CA certificates listed in Section\texttt{B.4} are taken from the list of certificates stored in the common web browser. The screen shot can be found in Figure\texttt{B.2}.

The client certificates constructed and used in this experiment does not contain the attribute \texttt{path length}. Therefore, OpenSSL does not throw any error while verifying the certificate extensions during constraint checking.

\section*{Set-up}

A simple program is written to create the server and the client threads. These threads are configured as follows:

The server and the client are configured to use TLS v1.0 protocol for the communication. The cipher suite used in this experiment is \texttt{TLS\_RSA\_WITH\_AES\_256\_CBC\_SHA}. This cipher suite uses RSA for both authentication and key exchange. The rest of this cipher suite indicates that AES cipher with key length 256 bits in CBC (Cipher Block chaining) mode is used to protect the application layer communication. The SHA is used for MAC. However, the choice of the cipher and the MAC algorithm does not affect this experiment since this experiment focuses on the client authentication and does not involve exchange of any application data protocol messages that needs symmetric encryption and calculation of MAC.

In this experiment, the client does not authenticates the server. This saves the time at the client side. The communication between the server and the client is performed using the standard OpenSSL library functions. The client thread is configured to connect to the server 1000 times. It is made sure that the server does not cache the certificate trust and avoid the client authentication next time it connects. This enables to obtain the optimum results. Therefore, each time, the client connects to the server, the client authentication is re-done.

\section*{Client configuration}

Listing\texttt{7.3} provides the snippet from the test program where the client is configured with the given cipher suite, set of certificates to be sent and private key of the client certificate. The terminologies used in this listing is summarized in Table\texttt{7.3}.

```c
if (SSL_CTX_set_cipher_list(ctx, cipherSuite) != 1) {
    /* Error handling */
}

if (SSL_CTX_use_certificate_chain_file(ctx, client_cert) <= 0) {
    /* Error handling */
}

if (SSL_CTX_use_PrivateKey_file(ctx, client_key, SSL_FILETYPE_PEM) <= 0) {
    /* Error handling */
}
```

Listing 7.3: The client configuration to load the SSL context with cipher suite and certificates.
In Listing 7.3 Line number 1 is responsible for deciding which cipher-suites to be used for the SSL communication. Variable `ciphersuite` contains the cipher suite method. Line 6 configures the client with the client certificate to be used when the server requests the client certificate. The file denoted by `client_cert` can either contain client certificate alone or the list of certificates beginning with the client’s certificate and the other certificates that providing the chain of trust. In order to use the client certificate, `SSL_CTX` needs to use the corresponding private key. Therefore, on line 11 corresponding private key is supplied to the `SSL_CTX`.

Server configuration

The server is configured to request the client certificates during the SSL handshake. This needs to be done explicitly since generally the client does not get authenticated in the SSL protocol. Listing 7.4 provides the code snippet from the server configuration that is used in the test program. Line 2 is used to configure the server implementation with the list of certificate trusted by the server. The file `ca_srvr_cert` contains the list of CA certificates that will be used to trust the certificate or the certificate chain returned by the client. In this set-up, this file contains the CA certificate that directly or indirectly issued the client certificate. Therefore, when the SSL handshake starts, the server checks this file and successfully confirms that the client certificate is trusted.

```
/* Load trusted CA. */
if (!SSL_CTX_load_verify_locations(ctx, ca_srvr_cert, NULL))
{
    /* Error handling */
}

/* Set to require peer (client) certificate verification */
SSL_CTX_set_verify(ctx, SSL_VERIFY_PEER, verify_callback);

/* Set the verification depth to 10 */
SSL_CTX_set_verify_depth(ctx, 10);
```

Listing 7.4: A server configuration to request the client certificate during the SSL handshake

Line 5 in this listing configures the server to request the client certificate during the SSL renegotiation process. The method `verify_callback` is used for it. Line 11 defines the depth of the certificate chain allowed at the server side. This is set to number 10 due to reasons already explained earlier in this chapter. The variables and constants used in this listing are summarized in Table 7.3 together with other values from listing 7.3.

Measurements

To obtain the time taken by the server thread, the clock identifier of the server is passed to the `int clock_gettime(clockid_t clk_id, struct timespec *tp);` function. The server clock identifier is passed at the place of the variable `clk_id`. The structure `timespec *tp` is updated with the CPU clock time required by the server thread.

Result & analysis

Table 7.4 summarizes the results obtained in Experiment 9. The first column denotes the public exponent used in the certificate. Each category of public exponent is divided into three sub-categories of length of certificate chain. As expected, when public exponent was equal to 65537 and only single certificate was used, the time required by the server to perform the 1000 handshakes was minimum. Whereas, the time required by the server to complete the
7.3 RSA Certificate Verification (Signature Verification)

<table>
<thead>
<tr>
<th>Name</th>
<th>Significance</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctx</td>
<td>SSL_CTX object to establish SSL enabled connections</td>
</tr>
<tr>
<td>ciphersuite</td>
<td>Variable containing ciphersuite name</td>
</tr>
<tr>
<td>client_cert</td>
<td>A file containing client certificate or chain of certificates</td>
</tr>
<tr>
<td>client_key</td>
<td>File containing private key of the client</td>
</tr>
<tr>
<td>SSL_FILETYPE_PEM</td>
<td>Indication that private key is PEM (Privacy Enhanced Mail) encoded</td>
</tr>
<tr>
<td>ca_srvr_cert</td>
<td>A file containing a list of CA certificates trusted by the server</td>
</tr>
<tr>
<td>SSL_VERIFY_PEER</td>
<td>This is a mode flag thats is used to request the client certificate.</td>
</tr>
<tr>
<td>verify_callback</td>
<td>A function that is used to control the behavior of the SSL_VERIFY_PEER flag</td>
</tr>
</tbody>
</table>

Table 7.3: Terminology used in Listing 7.3 and 7.4

<table>
<thead>
<tr>
<th>Exponent value/size</th>
<th>Length of the Certificate Chain</th>
<th>Time taken by the server to complete 1000 handshakes (in seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>13.774</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>15.005</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>19.124</td>
</tr>
<tr>
<td>65537</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>15.008</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>17.306</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>24.354</td>
</tr>
<tr>
<td>32 bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>80.249</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>285.283</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>695.206</td>
</tr>
<tr>
<td>3071 bit</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 7.4: Summary of certificate chain experiment

1000 handshakes when the client sent the chain of 10 certificates with each certificate signed with the RSA public key that had public exponent of size 3071 bits was the highest.

Figure 7.3 shows the comparative plot of the time taken by the server to complete the 1000 SSL handshakes for different public exponent verification that is divided into three categories of different certificate chain length. The percentage difference between the usage of different public exponent against different certificate chain length is given in Table 7.5.

Figure 7.4 shows the comparative plot of the time taken by the server to complete the 1000 SSL handshakes for different certificate chain length that is divided into three categories of different public exponents. The percentage difference between the usage of different certificate chain length against different public exponent is given in Table 7.6.

Comparative impact of public exponent size  From the result shown in Figure 7.4, it is observed that the server’s performance is not changed significantly for the public exponent values of 65537 and highest 32 bit prime number. However, when the public exponent value is a (random) number of size 3071 bits, there is a drastic difference in the performance given by the server. The server requires much longer time to complete 1000 SSL handshakes.
DoS Attack Analysis: Client Authentication using RSA Certificates

Figure 7.3: Client Certificate Verification: Comparison with respect to public exponent

<table>
<thead>
<tr>
<th>Certificate chain length</th>
<th>e=65537 (seconds)</th>
<th>e=Highest 32 bit prime number (seconds)</th>
<th>e=3071 bit random number (seconds)</th>
<th>Percentage change (% increase) between A &amp; B</th>
<th>B &amp; C</th>
<th>A &amp; C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single certificate</td>
<td>13.774</td>
<td>15.008</td>
<td>80.249</td>
<td>8.96</td>
<td>434.71</td>
<td>482.61</td>
</tr>
<tr>
<td>3 certificates</td>
<td>15.005</td>
<td>17.306</td>
<td>285.283</td>
<td>15.33</td>
<td>1548.46</td>
<td>1801.25</td>
</tr>
<tr>
<td>10 certificates</td>
<td>19.124</td>
<td>24.354</td>
<td>695.206</td>
<td>27.35</td>
<td>2754.59</td>
<td>3535.25</td>
</tr>
</tbody>
</table>

Table 7.5: Comparison of different public key exponent usage with respect to time required by the server to complete 1000 SSL handshakes.

Along with the client certificate verification operation, the server performs the private key operation to decrypt the client key exchange message sent by the client. The client encrypts the pre-master secret (to produce the client key exchange message) using the public key of the server and sends it to the server. In this experiment, the size of the public exponent of the public key in the server certificate is 65537 ($2^{16} + 1$) and it is kept the same throughout the experiment. Therefore, whatever difference in server’s performance is observed is due to the size of the public exponent in the client’s public key.

Whenever, the signature needs to be verified with the public key of the issuer certificate with the public key having 65537 or 4294967291 as public exponents values, it required less complex calculations (modular multiplication) when compared with the signature verification operation with the public key having 3071 bit public exponent.

This difference can be understood when the binary format of these numbers are observed. The binary representation of the 65537 is 1 0000 0000 0000 001. The binary representation of 4294967291 is 1111 1111 1111 1111 1111 1111 1111 1111. The binary representation of 3071 bit number is not given here due to its very large length. There are 2 one’s in 65537 and 31 one’s in 4294967291. Number 4294967291 contains 29 more one’s than 65537 has one’s. Whereas, number of one’s in 3071 bit number are in the range of 1000s and are very much higher in number when compared with the number of one’s in other two exponents. This difference in composition of these numbers is reflected in the result.

Therefore, the difference in the certificate verification processes for 65537 and 4294967291 public exponents cannot be significantly observed due to the higher overhead of 3071 bits public exponent operation at the server side. In addition, the private key operation (decryption of client key exchange message) overshadows the public key operation in case of the...
7.3 RSA Certificate Verification (Signature Verification)

Figure 7.4: Client Certificate Verification: Comparison with respect to length of certificate chain

<table>
<thead>
<tr>
<th>Certificate exponent length</th>
<th>Single certificate (A) (seconds)</th>
<th>3 certificates (B) (seconds)</th>
<th>10 certificates (C) (seconds)</th>
<th>Percentage change (% increase) between A &amp; B</th>
<th>B &amp; C</th>
<th>A &amp; C</th>
</tr>
</thead>
<tbody>
<tr>
<td>$e = 65537$</td>
<td>13.744</td>
<td>15.005</td>
<td>19.124</td>
<td>9.17</td>
<td>27.45</td>
<td>39.14</td>
</tr>
<tr>
<td>$e =$ Highest 32 bits prime number</td>
<td>15.008</td>
<td>17.306</td>
<td>24.354</td>
<td>15.31</td>
<td>40.73</td>
<td>62.27</td>
</tr>
<tr>
<td>$e = 3071$ bits random number</td>
<td>80.249</td>
<td>285.283</td>
<td>695.206</td>
<td>255.50</td>
<td>143.69</td>
<td>766.31</td>
</tr>
</tbody>
</table>

Table 7.6: Comparison of different size of the certificate chain usage with respect to time required by the server to complete 1000 handshakes

Other two exponent values. When 3 certificates are used, the performance given by the server decreases by 15.33% when public exponent is changed from 65537 to 4294967291. But this performance is decreased by 1801.25% when the public exponent is changed from 65537 to 3017 bits long value. The performance is further decreased when the length of the certificate chain is further increased. This change can be observed from the last column of the Table 7.6.

Comparative impact of certificate chain length The impact of the path length of the certificate chain can be observed from Figure 7.3. When the public exponent is 3071 bits long, the difference in the certificate verification process can be significantly observed for different certificate chain length.

The certificate length for public key exponent of size up to 32 bits decreases the server performance (increases the time required by the server to complete 1000 handshakes) in range of 39% to 62%. The server performance in hindered the most for different length of the certificate when the public exponent was 3071 bits long.

Summary of results

- The process of client authentication results in the decrease in the server’s performance.
• As the size of the public exponent increases significantly, the server requires more time to complete the certificate verification operation resulting in the degradation of the server’s performance.

• The impact of the length of the certificate chain is greater when the certificate has higher public exponent.

• The combination of higher public exponent and higher certificate chain plays the significant role in the degradation of the server’s performance and can be an important vector for the cause of the DoS attack on the server.

• The OpenSSL checks for the size of the public exponent size during the signature verification process. The maximum allowed size of the public exponent is 3071 bits with the maximum modulus of size 3072 bits. However, this check can be performed while constructing the certificate chain and avoid further processing if the limits on the size of the public exponent is not satisfied.

• As observed from the result, the size of the public exponent up to 32 bits does not affect the server performance significantly. Therefore, in order to prevent the possibility of the DoS attack through client certificate verification, it is important that the OpenSSL does not allow the public exponent of size greater than 32 bits.

• The increased allowed public exponent size invites the other bigger concerns regarding the confidentiality. But, discussion of this issue is out of the scope of this thesis and is not related to the DoS attacks.

• The OpenSSL checks the path length attribute of the basic constraint in the certificate provided it is present in the certificate. However, when the path length is set to NONE, it does not affect the certificate verification process.

• The wrong configuration of the certificate extension, e.g. not including the path length, can allow clients to issue certificates. Usage of Such certificates increase the possibility of DoS attacks as described in this section.

7.4 Summary

This chapter presented two experiments to investigate DoS service attacks on the SSL protocol using the client authentication process. The client authentication process is an optional process in the SSL protocol. The first experiment focused on verifying the implementation of the mechanism of the certificate path construction at the server side. This process does not create any possibility for the attacker to execute the DoS attacks on the SSL protocols. The second experiment focused on constructing DoS attack on the SSL protocol using client certificate verification process. This experiment considered the two aspects of the client certificates verification process namely the size of the public exponent of the key that is used to sign the certificate and the length of the certificate chain that can be sent by the client to provide the chain of trust to the server. The result and analysis of this experiment shows that the higher values of the public exponent and certificate chain lengths does hindered the server’s performance.

*provided CA attribute is set to TRUE.
CHAPTER 8

Comparative Analysis: DOS Attack Strategies on SSL

This chapter explains the three DoS tools each using different DoS attack strategy to assess their impact on the server. All of these strategies focus on degrading the SSL server’s performance by forcing the server to repeatedly perform computationally expensive operations.

The first part of this chapter explains the methodology that is used to assess the impact of the DoS tools. This is followed by the brief information about the working of the tools. This will be useful to understand the experimental results explained later in this chapter. Finally, this chapter explains the experiments performed to compare the impact of DoS attack tools and their respective strategies to provide in-depth analysis.

8.1 Methodology

The main objective of the experiment described in this chapter was to measure the differences in the impact of the different DoS tools based on different attacking strategies. This was not a category of experiment where one measured the performance of the SSL web server for the different server configurations. The impact of the DoS tools was measured on the given system. Three different SSL DoS tools were selected to measure their impact on the SSL server.

This experiment used a simple web server benchmarking technique. The Apache Benchmark (ab) is the tool for benchmarking Apache web server. This tool can be used to assess the performance of the Apache web server, especially in terms of how many requests per second, the Apache web server can serve. The motivation behind using this benchmarking tool was to record the performance of the web server for the different number of connections and concurrent connections. After analyzing the optimal server performance under normal conditions, i.e. when there was no attack on the server, the identical performance measurement process was repeated while different DoS tools were activated to attack the same web server.

Along with the ab benchmarking, a brief "micro-benchmarking" is done for the target system (system that was attacked) and the attacker system (System that ran the DoS tool). The micro-benchmarking consist of measurement of CPU utilization, memory usage and load averages of the system. However, these measurements involve a lot of complexities in terms of I/O operations, shared memory usage, etc.. Therefore, the output of the micro-benchmarking is not (always) precise in the experiment where applications as complex as the web servers
are involved. Nevertheless, micro-benchmarking can provide some guidelines about real time system performance. Therefore, a brief evaluation of the system has been done using micro-benchmarking. These evaluations were done in two sets of experiment. One for the victim machine and other for the attacker machine. The victim machine was tested when there was no attack and similar tests are performed when the victim’s machine was under attack. The attacker machine is also evaluated to understand the performance requirement to execute the DoS attack.

8.1.1 Platform

This experiment used three different virtual machines. The first machine hosted the SSL web server. The second machine acted as the attacker machine that ran the DoS tool. The third machine used to monitor the performance and throughput of the SSL web server. The attacker and monitor machine ran Linux Ubuntu 10.04.4 and the target machine ran Linux Ubuntu 8.04.4. The other system information is summarized in Table 8.1.

<table>
<thead>
<tr>
<th>Particulars</th>
<th>Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>OS Type</td>
<td>Linux</td>
</tr>
<tr>
<td>CPU Model</td>
<td>Intel(R) Core(TM) i5 CPU 2.53GHz</td>
</tr>
<tr>
<td>Memory</td>
<td>1024 MB</td>
</tr>
</tbody>
</table>

Table 8.1: System information of SSL server

The TLS web server installed on the target system was the Apache 1.3.14 with mod_SSL 2.8.31-1.3.14. The SSL implementation on the attacker and the monitor machine in this experiments is the open source OpenSSL 1.0.0e [8]. The Apache SSL web server have been chosen due to its wide availability and use as shown by the survey at [54].

8.2 DoS tools and attack strategies

As already mentioned in this chapter, three different DoS tools are selected to measure the effectiveness of the DoS attack on the given Apache SSL server. These three different tools are based on three different strategies of the DoS attacks. This section explains the working of the these DoS tools along with the strategies they use.

8.2.1 thc-ssl-dos tool

The working of this tool is already explained in Chapter 4. Therefore, details of this tool are not given again in this chapter. This tool exploits the SSL renegotiation feature in the SSL protocols. This tool is considered in this experiment because no one has practically evaluated this tool. There are few articles providing the theoretical analysis of the impact of this tool. One of the theoretical analysis can be found at [42]. This experiment is an attempt to provide the practical impact of this tool and determine the effectiveness in comparison with the other attack strategies.

8.2.2 sslsqueeze tool

This tool is written by Michal Trojanara [18] and has been released under GNU general Public licence. It uses libevent2 library. This tool does not perform any cryptographic operations

1At least not till the time of performing this experiment
2The libevent is an even notification library which provides the mechanism to execute a callback function when a specific event occurs on a file descriptors. More information about the library can be obtained from
such as RSA encryption or decryption, calculating MACs etc. This tool makes use of libevent library for handling the events such as writing bytes to the server at right time and reading server’s response whenever necessary. This tool does not need to use threads to perform complex and lengthy SSL related operation.

In the SSL handshake protocol, RSA decryption is the most expensive operation to perform. It is performed by the server to obtain the pre-master secret sent by the client. The pre-master secret is encrypted by the client with the server’s public key. An attacker can send number of such decryption requests to the server through SSL handshake to computationally overload the server. But, to achieve this, the client needs to perform the RSA encryption which is not as expensive as RSA decryption but adds to the computational overhead on the client side. There are several strategies to avoid this overhead on the client side. One traditional strategy is to calculate the client key exchange message only once and replay it whenever necessary. The another strategy could be to send just the bogus data to the server instead of encrypted client key exchange message. The server is not going to detect such data as bogus before the decryption. This idea is also quoted at [42]. Once the server completes the decryption of the bogus data and understands that the data decrypted is not a valid key exchange message containing the pre-master secret, the harm (Computational overhead) is already done and hence the purpose of overlading the server with the RSA decryption (cryptographically expensive) is achieved. However, the bogus data should be constructed in such a way that it follows the standard RSA requirements such as in order to make the server decrypt the data, it(client key exchange message) cannot be greater than the length of the modulus of the server’s RSA key pair. OpenSSL checks this condition before decryption.

The sslsqueeze tool utilizes the second strategy from the above description. That is, it does not calculates the valid client key exchange message even once. However, it makes sure that the length of this message is according to what SSL server expects before staring the decryption. This is the additive value of the sslsqueeze tool.

In the sslsqueeze tool, the necessary (SSL) handshake messages are prepared initially before starting the attack. These messages are the Client Hello, the client key exchange, the change cipher spec message and the finished message. These message are formatted according to the SSL specification with the correct record header and record body format. The Client Hello message is filled with the Client Hello header and the message body. The client key exchange message is constructed according to the length of the RSA key size. It is required to know the RSA key length in advanced. The target server’s key length can be easily obtained using the OpenSSL s_client command. This is explained in the Appendix B in Section B.3.

The change cipher spec message is a message of length 1 byte having value 1. The finished message in this tool is a structure containing a random data. When server receives the client key exchange message and when it finds that client key exchange message is not correctly formatted, the server will disconnect the connection. However, for the sake of following the protocol, this tool sends the change cipher spec message and finished message in the correct order.

The tool starts with creating a new socket. This socket is used to write the first record structure containing the Client Hello message. The server response is read on this connection. Once the server hello message is confirmed to be received, it continues with sending the rest of the prepared message starting with client key exchange message followed by the change cipher spec message and finished message in a single record. The parsing of the data is necessary. Because sending the handshake messages randomly can result in connection termination from the server side. The server does not expect any data from the client until it sends its reply to the first message. In this case, the server does not expect any SSL handshake message after Client Hello is received and before it sends the server hello message in response to the Client
Hello message.

This SSL communication always results in the server issuing the fatal error since the decrypted data at the server side is not well formatted and is discovered by the server after each decryption process. But, this tool does not care about it since the only aim is to make server decrypt more and more data. In this way, the attack continues till the time either server blocks the attacker’s IP or the attacker stops the attack.

8.2.3 BFC (Brute Force with Crypto) tool

In response to the study done in this research and the attack strategies considered up till now, a new and simple DoS tool was constructed. This tool is based on the simple approach. This tool only creates a continuous and parallel SSL connections (with the target server). The concurrency of this tool can be adjusted programatically. The main objective behind constructing this tool was to compare and analyse the effect of parallel SSL connections with the target server when the standard SSL protocol is followed and all the cryptography needed to be used is used to connect to the server unlike the sslsqueeze tool described in the previous section. The structure of this tool and logic of processing is inspired from number of hacking tools available on the internet. The main characteristics of this tool are as follows:

- This tool uses cryptography and OpenSSL SSL library to implement SSL protocol.
- To connect to the server, this tool uses standard SSL_connect(SSL *s) function so as to initiate the SSL handshake protocol.
- The scalability of the tool is achieved using threads. The scalability refers to the capacity to create simultaneous and continuous SSL connections.
- This tool can produce number of parallel connections to the server from the single computer.

This tool is nothing but the program which continuously opens the SSL connection with the target server to be attacked. This tool starts with creating the number of connection ends (Connection ends refer to the number of virtual machines to create the SSL connections). This number can be easily adjusted by the user of the tool. This number signifies the number of SSL connections that will be possible to create simultaneously. Of course, this number cannot be set to infinite due to the limitation of the OS to create number of simultaneous sockets. Once the connection ends are created, the SSL handshake with the server are started. After completion of the SSL handshake, a proper shutdown is not done that is usually done by calling SSL_shutdown(SSL *ssl). Instead, the SSL object (SSL *ssl) along with the SSL context object (SSL_CTX *ctx) are removed from the cache memory (memory deallocation process). This ensures that the SSL session is not kept in the cache. Hence at the next iteration of SSL connection, the SSL connection resumption is avoided and full handshakes is performed. This is required since the expensive operations such as RSA decryption are not done during the SSL session resumption process.

Therefore, this tool uses cryptography throughout the attack and follows the SSL protocol by using the standard OpenSSL SSL library function calls. Comparing the results of impact of this tool with the impact of the other tools mentioned in this chapter will create a clear picture of the effectiveness of the given strategies to deploy the DoS attack and its effectiveness. It is also important to measure the resources required to perform these attacks on the attacker’s side. It is important that the attacker requires the lesser resources than the server for the attack to be efficient.

SSL connection resumption is a process of abbreviated SSL handshake and does not involve the RSA decryption process at the server side. This feature is already described in Chapter.
8.3 Experiments and Analysis

8.3.1 Performance benchmarking

In this experiment, the web server was tested with 500 requests for 50 times continuously. At the first time, 500 requests are made sequentially. That is, concurrency level of 1. After completing the 500 requests, this concurrency level is increased by 1. This is continued till the concurrency level of 50.

The choice of the 500 connections with the highest concurrency level of 50 connections is made after testing the apache server on the available system for the number of connections and concurrency level. The combination of 500 connections with concurrency level of 50 was found be the most effective in terms of time required and the capacity of the apache browser to handle the requests on the test virtual machine. This combination is expected to vary according to the system configuration. Therefore, the output of this experiment is subject to change from system to system. However, the goal of the experiment is not to measure the performance of the web server itself but measure the difference in the impact of the DoS tools. Therefore, when the test server used in this experiment is changed, the performance impact on different (changed) server will also change. However, the range in the difference in impact is likely to scale in proportion with the results obtained in this experiment.

The performance impact is also subject to the availability of the bandwidth and the hardware accelerators to compute heavy cryptographic operations.

Experiment 1: Throughput

Figure 8.1 shows the result of the server throughput measurement experiment. This figure gives the graph of the server throughput for 4 different conditions. The red line indicates the requests (per second) served by the server under normal condition. The green line represents the throughput of the server when server was under attack from the sslsqueeze DoS tool. Similarly, the blue and purple line represents the server performance when the server was under the attack from the BFC and the thc-ssl-dos tool.

The ab benchmarking tool was started to request the web server for the index page. The index page contains only single line of text. The attack tool is activated as soon as the 1st iteration of the 500 requests was over, the DoS tools were activated to attack the server. The tools were kept active until all the 50 iterations of the 500 requests were completed. Line chart shown in Figure 8.1 is the result of the average values obtained with the 3 iterations of the SSL server performance measurement as described in this section.

As seen in Figure 8.1, the thc-ssl-dos tool clearly outperforms the among all the rest of the tools. The BFC tool starts to take the early effect among other tools. With the number of greater requests, the requests served by the web server in case of the sslsqueeze tool are growing faster than the BFC tool and the thc-ssl-dos tool. In case of the thc-ssl-dos tool, the number of requests served by the server remains almost constant and much lesser than the original capacity of the server to serve the request.

The gradient observed in Figure 8.1 also indicates the effectiveness of these tools over the period of the time. As time increases, the impact of these tools were reduced with the exception of the thc-ssl-dos tool.

\footnote{The test server used in this experiment is not the best performing system since it is just the virtual machine.}
Analysis of Experiment 1

The web server under the test serves 25-30 requests per second in the normal conditions with the concurrency level of 1 to 50 connections.

When the system is under the attack from the thc-ssl-dos tool, the throughput of the server drops drastically. This tool has performed 80 (average) SSL handshakes per second. Originally, this tool creates the 400 peers on a single machine to create the DoS attack. However, 400 peers would have been a greater number to neatly measure the performance of the SSL web server that is installed on the given virtual machine. Therefore, this tool was modified to create only 100 peers. The other tools also create 100 peers to produce the DoS attack.

The number of connections required to perform more and more handshakes with the thc-ssl-dos tool is less. When this tool creates one connection to the server, the first handshake is performed. In order to perform the subsequent handshakes, this tool does not create a new connection but it just renegotiates. The renegotiation is started by sending a Client Hello message. Therefore, the 3 way TCP handshake to establish a new connection is not needed. This saves the time and latency involved in creation of a new connection. This results in a binding the server's resources continuously.

The other two tools, has almost same impact on the SSL server. However, the impact of the BFC tool remains greater than the impact of the sslqueeze tool. The impact of the BFC tool is lesser than the thc-ssl-dos tool because, the BFC creates the more number of connections to the server when compared with the-ssl-dos tool. The thc-ssl-dos tool creates one connection and is preserved for a longer time. Unlike in the BFC tool, the connection is created and immediately terminated. This creates time gap for the normal traffic to request to the SSL server and server can serve the normal traffic slightly efficiently when compared to the-ssl-dos tool. When the traffic on the SSL server is increased (larger number of concurrent normal traffic), the impact shown by the BFC tool is gradually decreasing. But the SSL server under the attack by the BFC tool cannot serve the normal traffic with the same throughput as it served while not under the attack. This is quite obvious.
The SSL server under the attack of sslsqueeze tool shows a better performance as compared with other two tools. This is something that was not expected. Since this tool does not make use of any cryptography and attempts to request the server continually. However, there is one fact that needs to be considered. When sslsqueeze tool is attacking the server, the SSL connection is never completed. Whenever a server finishes the RSA decryption (decryption of the bogus data sent by the sslsqueeze), the connection terminates. Therefore, this avoids the transmission of the change cipher spec message and finished message from the server. In addition, the server does not wait for the application data messages from the attacker since the connection already has terminated right after the RSA decryption. The RSA decryption process does impact the server performance and hence the throughput of the web server is reduced under the attack of sslsqueeze tool. However, the impact is not as large as the impact shown by the thc-ssl-dos and BFC tool.

**Experiment 2: Performance**

In this experiment, the web server is requested for 1000 connections with the concurrency level of 50. In this experiment, the response time is recorded for each connection. Figure 8.2 displays the graph of the (total time of the request) total response time on different conditions. The conditions remains the same as described and explained in the earlier experiment in this section. The total response time depicted on Y-axis is the total time required to complete a given request. This time also includes the time required to open the respective TCP connection.

![Figure 8.2: Total request time under different categories of attacks](image)

Analysis of Experiment 2: Total Response Time

As expected, the response time at the normal connections is the lowest. The thc-ssl-dos tool produces the greatest impact. A step-like pattern can be observed from the graph in 8.2 for the thc-ssl-dos impact line. When the tool is started, it continuously attempts to connect to the server. In other words, the starting phase is the time when this tool creates
the number of SSL connections with the target SSL server. Then comes the phase, where number of SSL renegotiation is performed. It should be noted that the SSL renegotiation does not require to open a new TCP connection. The renegotiation is initiated just by sending one encrypted handshake message and hence the process of SSL renegotiation is faster than creating a new TCP connection. This process continues further. Whenever the tool creates a TCP connection, the impact of the tool remains almost unchanged. Under the renegotiation phase, the response time increases drastically.

The fact that BFC tool performs better than sslsqueeze tool is a bit unexpected. Since, BFC uses cryptography and sslsqueeze does performs any cryptographic operations. There are few things which needs to be explained here. Figure 8.2 gives the impact of the tool on the server and not on the attacker machine. Therefore, when sslsqueeze does not perform the cryptographic operations and BFC does perform the cryptographic operations, the impact should be seen on the attacker's machine and not the target machine. As a matter of fact, the response time of the server is greater for the BFC because the full SSL handshake is performed at the server side. Where as, when the sslsqueeze tool is used, the server attempts to perform the RSA decryption to obtain the pre-master secret from the client key exchange message, the process results in the failure and hence the SSL connection is aborted and no further SSL handshake protocol messages are exchanged. In case of the BFC tool attack, the server does the extra operations sending the change cipher spec message and encrypted server finished message. The server also receives and processes the change cipher spec and the client finished message. After completing the handshake, the SSL connection is terminated from the attacker’s side. These extra operations takes a while and hence it results in increased response time compared to the response time when sslsqueeze tool is used. Therefore, the waiting time for the BFC tool is greater than the waiting time for the sslsqueeze tool. This different is clearly identified in Figure 8.4.

Figure 8.3: The Connection Time: brute force with crypto (BFC) Vs brute force without crypto (sslsqueeze) with respect to normal connections

To obtain the clear picture of the change in response time as seen in Figure 8.2, Figure

\[\text{Note that these messages are also sent in the case of sslsqueeze tool but are not processed by the server}\]
8.3 Experiments and Analysis

8.3 displays the comparison of the total time required to serve the request under the normal condition (normal traffic as described initially), under the attack by the BFC tool and under the attack by the sslsqueeze tool respectively with red, blue and green lines.

Analysis of Experiment 2: Total Wait Time

Figure 8.4 gives the line graph of the total wait time under different conditions. These conditions are normal traffic condition and the conditions when the server was under attack by the BFC tool and sslsqueeze tool. The wait time during a period of 1000 requests with 50 concurrent requests is indicated for normal traffic, under attack by BFC tool and under attack by sslsqueeze tool is indicated with red, green and blue line respectively. The measurement for the-ssl-dos tool is intentionally skipped in this figure since it has very much higher wait time that can also be easily understood from Figure 8.2 and including it in the graph makes the distinction between the wait times of other three conditions unclear.

![Figure 8.4: Total Wait Time: brute force with crypto (BFC) Vs brute force without crypto (sslsqueeze) with respect to normal connections](image)

The wait time indicates the time between writing the last byte of the request and the receiving the first byte of the response. The normal traffic used in this experiment only requests for the index page containing one text line. Intuitively, the wait time for the requests when there is no attack on the system is less. When the system is under attack from the sslsqueeze tool, the wait time is seen to be greater than the normal traffic. This is because when the system is under attack, it has to serve few more SSL handshake request. However these requests are never completed unlike in BFC tool attack where the wait time is even greater than the wait time during the sslsqueeze attack.

The wait time for all three conditions(normal traffic, BFC attack, sslsqueeze attack), has a gradual increasing behavior and at the end the wait time drastically drop just before completing the 1000 request. This is because how Apache benchmarking tool `ab` works. This experiment is conducted for 1000 requests with 50 concurrent connections. Therefore, the `ab` starts with the 50 concurrent connections. When one or more of the 50 concurrent connection...
is finished before rest of the 50 connections, these completed connections are replaced by the next few connections to meet the requirement of the experiment to be the 50 concurrent connections. Finally, at the end a situation arrives where there are no more connections that needs to be requested to the server. For instance, when 970 connections are completed, there will be only 30 concurrent connections. Hence, there is a drastic drop seen at the end of the graph.

**Analysis of Experiment 2: Requests per second**

Table 8.2 gives another statistics obtained in this experiment. The first column gives the name of the condition. The second column in this table gives the time required to perform 1000 requests. This time signifies the performance of the web server. For instance, the time given in the second column is time required for the web server to serve 1000 requests with concurrency level of 50. The third column gives the requests per second served by the web server in this experiment. The last two columns are most important columns in this table. It can be observed from that the requests served per second is reduced when the server is under the attack. The impact of the thc-ssl-dos tool is the most. That is, when thc-ssl-dos tool is activated, the server serves the least number of requests. The last column gives the average response time per request. These values signifies the response time that legitimate customers are going to see or experience when they visit the web server. Clearly, the response time for the web server when attacked with thc-ssl-dos tool is the highest.

<table>
<thead>
<tr>
<th>Condition</th>
<th>Total time required for 1000 requests (sec)</th>
<th>Requests per second</th>
<th>Time per request (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal Connections</td>
<td>32.778</td>
<td>30.51</td>
<td>1638.882</td>
</tr>
<tr>
<td>Under sslsqueeze attack</td>
<td>63.071</td>
<td>15.86</td>
<td>3153.544</td>
</tr>
<tr>
<td>Under BFC tool attack</td>
<td>130.095</td>
<td>7.69</td>
<td>6504.742</td>
</tr>
<tr>
<td>Under thc-ssl-dos</td>
<td>327.742</td>
<td>3.05</td>
<td>16387.102</td>
</tr>
</tbody>
</table>

Table 8.2: Comparison of web server performance under different attack scenarios

**8.3.2 Micro-benchmarking**

The micro-benchmarking is done for each condition of the attack scenarios mentioned in this chapter, each for the attacker’s machine and the client machine. It is very important to measure the impact of using the tool on the attacker’s machine along with the target machine. An ideal DoS tool should take as less resources as possible on the attacker’s machine to achieve the greater magnitude of degrading impact.

**Measuring techniques and environment**

In order to understand the measurement of the micro-benchmarking, it is important to understand the environment of the experiment. The systems (attacker and target machines) used were the same as described earlier in this chapter. In order to perform the micro-benchmarking on the victim’s machine, the utility called Monit was installed. It was configured to monitor the system, record the result and send it to my personal email address for later analysis.

---

This can happen, even if the connections are concurrent the connection completion time depends upon how server can performs and can serve these 50 connections.
The peak memory usage is calculated by constantly monitoring the output of `ps` utility in the Linux machine. Since, observing the output of the `ps` utility manually to record the peak memory usage can be error prone, it is done programmatically. The memory usage is monitored and recorded for each 0.1 second to achieve accurate results. The memory usage signifies the requirement of memory (computer RAM) while conducting the attack to achieve the result described in Section 8.3.1.

The CPU usages were obtained by using two mechanisms. The simplest method was to monitor the output of the `top` utility on the Linux machine. The CPU usage fluctuates time to time. Therefore, whenever necessary the range of minimum to maximum CPU usage observed were recorded. These results were confirmed using the other utility called `Monit`.

The load averages are measured using the same utility called `Monit`. The load averages are the three numbers representing the load on the system over progressively longer periods of the time. The higher numbers represent a problem or an overloaded machine. The load average should be as minimum as possible. If it is just less than or equal to 1 then it represents the overloaded system.

### Impact on attacker’s machines

Table 8.3 summarizes the micro-benchmarking done for the attacker’s machine. The first column in this table gives the name of the DoS tool that was running on the attacker machine while recording the measurements mentioned in next three columns.

The second column gives the memory usage by the DoS tool. Since, the sslsqueeze tool does not make use of any cryptography, it requires the least memory to store the data structure etc. The BFC tool uses the standard OpenSSL libraries and threading mechanism to attack the target machine and performance all the cryptography required to connect to the target server. The peak memory usage is greater for BFC tool compared with that of sslsqueeze tool. The thc-ssl-dos tool uses the maximum (peak) memory of all three tool. This is the result of the greater number of data structures maintained by the thc-ssl-dos tool. This tool writes data to the SSL target and reads the data sent by the server occasionally. This increases the memory requirement of this tool.

<table>
<thead>
<tr>
<th>Tool</th>
<th>Peak memory usage (kB)</th>
<th>CPU (min-max) (%)</th>
<th>Load on the system [1 min][5 min][15 min]</th>
</tr>
</thead>
<tbody>
<tr>
<td>sslsqueeze</td>
<td>1344</td>
<td>6-8</td>
<td>[0.30][0.35][0.32]</td>
</tr>
<tr>
<td>BFC</td>
<td>57400</td>
<td>12-14</td>
<td>[0.98][0.62][0.59]</td>
</tr>
<tr>
<td>thc-ssl-dos</td>
<td>107872</td>
<td>14-17</td>
<td>[1.59][0.95][0.64]</td>
</tr>
</tbody>
</table>

Table 8.3: Micro-benchmarking for Attacker’s machine

The third column in Table 8.3 gives the CPU usage by the DoS tools. The thc-ssl-dos tool uses the 14%-17% CPU. This is maximum amount amongst all three tools. The sslsqueeze tool requires the much lesser CPU and uses the CPU in the range of only 6%-8%. Thats is, the CPU requirement for the sslsqueeze is almost half with compared with the BFC tool and one third when compared with the thc-ssl-dos tool.

The fourth column in Table 8.3 gives the load (averages) on the attacker’s system. When the system used the sslsqueeze tool, the load on the system was normal, i.e. the load does not change from normal condition. When the system uses BFC tool, the 1 minute average load is almost 1. The tool thc-ssl-dos makes system overloaded with the 1 minute average load.

\footnote{Normal stands for less than 1. Lesser the load, better is the system status}
of 1.59 and 5 minute average load of almost 1. Therefore, the sslsqueeze tool did not increase
the load on the system significantly. The BFC tool moderately overloaded the system. The
thc-ssl-dos tool overloaded the system the most.

**Impact on target machine**

Table 8.4 summarizes the micro-benchmarking done for the victim’s system when it was
under attack. It was not practically possible to observe the system (using `ps` or `top`) when it
was under the attack because the machine was almost non-responsive to the user interaction
while the attack was active.

The first column gives the name of the attack. The rest of the columns gives the respective
measurements for these attacks.

The second column gives the percentage CPU (user) time required by the system when it
was under attack. The values are the average of the maximum values noted for 5 iterations
of the 10 minutes tests performed on the system. Similarly, the third column specifies the
percentage CPU (system) time required by the system. The fourth column gives the load
averages on the system when it was under the attack. For this measurement, the load average
for 15 minutes is less important than the 1 minute and the 5 minute load averages measured
because the attack was in progress for only 10 minutes while recording the results. From
the experiment results described earlier in this chapter, it is understood that all the attacks
starts overloading the server within a short time (less than a minute) and impact remains
almost constant throughout the experiment under the given environment. In addition, the
result described in table 8.4 gives the averages of the peak values obtained for each category
for 5 sets of iterations.

<table>
<thead>
<tr>
<th>Tool</th>
<th>CPU (user) (%)</th>
<th>CPU (system) %</th>
<th>Load on the system [1 min][5 min][15 min]</th>
</tr>
</thead>
<tbody>
<tr>
<td>sslsqueeze</td>
<td>81.5</td>
<td>17.3</td>
<td>[21.33][12.75][5.50]</td>
</tr>
<tr>
<td>BFC</td>
<td>77</td>
<td>17.7</td>
<td>[82.74][63.18][34.41]</td>
</tr>
<tr>
<td>thc-ssl-dos</td>
<td>78.3</td>
<td>6.9</td>
<td>[91.76][45.77][18.12]</td>
</tr>
</tbody>
</table>

Table 8.4: Micro-benchmarking for Victim’s machine

From Table 8.4, it can be observed that the percentage user CPU used by the system under
all the attacks is almost the same. Still, the system appears to require highest percentage of
user CPU when sslsqueeze tool is active. The system CPU used by the target system when
the thc-ssl-dos attack is active is the least. The other two tools has the almost same effect on
the system CPU of the target system. Since, system requires greater user time than system
time, these tools are processor intensive.

The thc-ssl-dos tool produces the highest load on the system. This is followed by the BFC
tool and the least load produced was due to sslsqueeze tool attack. The next two fields in
the 4th column are the values that indicate the average load on the system for last 5 minutes
and 15 minutes respectively. These values are nothing but the average of the 1 minute load
on the system for last 5 minutes and 15 minutes respectively. These values should also be
as much less as possible. Lesser values indicate the ideal condition. The significance of 5
minute and 15 minute average is to measure the continuous load on the system. In the set of
experiments performed and described in this chapter, it has been observed that load on the
(attacker) system remains almost the constant throughout the attack period.

---

*The user CPU time indicates the time spent on the processor (user space) and the system CPU time is
the time spent on the Operating System kernel*
8.4 Summary

This Chapter has provided a comparative analysis of the impact of the three DoS attack strategies on the SSL web server. These three strategies were implemented using three different tools to test their performance. These three strategies are listed below.

- Brute force using cryptography (BFC).
- Brute force without using cryptography (sslsqueeze).
- Brute force based on the SSL renegotiation (thc-ssl-dos).

The purpose of this experiment was to assess the DoS risk factor involved in using the number of DoS attacking strategies. The results showed that the impact created by the thc-ssl-dos tool was the highest among other two attacking strategies. The thc-ssl-dos tool exploits the SSL renegotiation based computational vulnerability in the SSL protocols. The BFC tool that uses the cryptography and focuses on executing the number of concurrent SSL connections with the SSL server gave next better performance. The most lightweight tool (sslsqueeze) gave the minimal performance.

Even if the thc-ssl-dos tool was proved to be the most effective tool, it is very easy to mitigate the SSL renegotiation based computational DoS attack. If the server does not support the SSL renegotiation (to be more specific, the client initiated SSL renegotiation), thc-ssl-dos tool is ineffective on such servers. Whereas, the other two tools can execute the successful DoS attack on the SSL servers provided enough resources (individual machines with sufficient processing power and internet connection) are available to them.
This chapter is divided into two sections. Section 9.1 discusses the literatures that are related to the SSL protocols and focuses on improving the SSL server performance. The findings and conclusion in these literatures are analyzed on the basis of findings obtained in this thesis work.

Section 9.2 talks about two random topics studied in this thesis. Section 9.2.1 talks about the concept of session tickets and its effectiveness in reducing the load on the SSL servers. Further, it discusses the remote possibility of using the session tickets as the cause of the DoS attacks. Section 9.2.2 provides the argument regarding the difference in DoS attack possibilities among the different versions of the SSL/TLS protocols. Finally, Section 9.3 summarizes this chapter.

9.1 Related Work

9.1.1 Reverse SSL

The reverse SSL [6] aims at relieving server from the heavy public key decryption. There are two categories of the reverse SSL protocol depending upon whether the client authentication is performed or not.

Reverse SSL: With client authentication

Figure 9.1(a) shows the reverse SSL handshake when the client authentication is performed. The authentication role of the client and the server is reversed in reverse SSL. The server produces the pre-master secret, encrypts it using the client’s public key obtained from the client certificate to generate the server key exchange message and sends it to the client. The client obtains the pre-master secret by decrypting the server key exchange message. Hence if the client can decrypt and obtain pre-master secret, it proves that the client does know the private key corresponding to the public key included in the client’s certificate. Hence the client is authenticated by public key encryption. The server is authenticated to the client since the server sends the certificate verify message to the client. The certificate verify message is the signature of the hash value of all prior handshake protocol messages.

The server has to perform the signature generation operation and it is private key operation. However, the server makes use of online/offline signature method [58] to perform most
of the expensive operation offline, that is before the message to be signed is available. This saves the computational power and time at the server side. In [6], it is assumed that the server has varying load with respect to the time. The under-utilized time of the server can be used to perform the necessary pre-computation.

Figure 9.1: Reverse SSL Handshake: (a) with client authentication and (b) without client authentication (source: [6]).
**ReverseSSL: Without client authentication**

In most of the scenarios, the client authentication is not very common. Therefore, the reverse SSL protocol explained earlier cannot be widely used. Therefore, a mechanism is needed where the server needs a key that can be used to encrypt the *pre-master secret* and only the client can decrypt it. In addition, the protocol should not invite any security flaws in the design.

Figure 9.1(b) shows the operation where **client puzzles** are used to solve the issue of the safe key exchange and to safeguard the server from DoS attacks since the client will have to solve the computationally expensive client puzzle before the server can perform any computationally intensive operation. The describes the two puzzles for the application in the reverse SSL. The central idea of these puzzles is to request the client to produce the RSA key pair on the fly with modulus in the range specified by the server. The puzzle answer includes the public key generated, the modulus and the signature generated on the public key. This enables the server to verify the signature. This puzzle has one issue where the attacker can send any key pair prepared in advanced and satisfying the range specified by the client. This issue is addressed using the pre-image resistant hash technique. In this case, the server sends some part of the data that should be included in the modulus in the hashed format. The client first has to find original data, that matches the hash data sent by the server. With this arrangement, before verifying the signature, the server can verify if the public modulus is what expected and then proceed with the signature verification.

The server has to perform following operations while using client puzzles in ReverseSSL:

- Prepare the puzzle request (one hash computation, m-bit random number generation where m is the length of the big random number that the server sends to the client to specify the range in which the client should generate the modulus of the RSA key pair.)
- Verify the puzzle ((RSA) signature verification).

The client has to perform following operations while using client puzzles in ReverseSSL:

- $2^{p-1}$ hash computation where p is the length of the message that was hashed initially by the server and the client has to find this number by brute force technique.
- RSA key generation.
- RSA signature generation.

**Analysis of reverse SSL**

- If either of the communicating party does not support reverse SSL, one of the earlier version of the SSL protocol can be used to preserve the backward compatibility. An attacker can always choose not to support reverse SSL and conduct the DoS attack in the traditional way. Therefore, simply implementing reverse SSL does not guarantee the DoS resistance.

- Reverse SSL mechanism utilizes the concept of online/offline signature schemes to allow the server to perform most of the public key computation offline before the client requests for the SSL connection. Assumes that the server load is varying with respect to the time and under utilized time of the server can be used to perform necessary pre-computation. However, the attacker can attack on the server when it is under utilized and affect the pre-computation process. As a result, when the legitimate clients requests for the SSL connection, the performance of the server can be significantly
degraded while serving the clients since the server is not likely to be prepared with the pre-computations to be used in the online phase of the protocol. Finally, this could be the additional cause for the DoS attack.

- The attacker can always choose to select the maximum public key exponent while generating the RSA public key. With this arrangement, the client will require very less time to produce the RSA signature and the server will have to invest greater time verifying it. This fact is already confirmed and proved in Chapter 7.

- In the reverse SSL approach, the server gets authenticated at the end. Therefore, the attacker can create the malicious web sites and the divert the clients to such malicious web site. Whenever the client visits such web site, the malicious server sends much computationally expensive puzzle request to the legitimate client. This may overload the client to produce the client side DoS attack. This can affect the reputation of the company that owns the original server and can result in losing legitimate customers. This kind of attack is the client side DoS attack and hence it is not investigated further in this research. However, online service is always about the availability of the service and client satisfaction and hence such attacks should also be mitigated.

9.1.2 Improving secure server performance by re-balancing SSL/TLS handshakes

[60] focused on the adjusting the computational imbalance in the SSL handshake protocol. It introduced the idea of the Client Aided RSA (CA-Aided). The motivation of this idea was taken from the concept of Server Aided RSA [61] that aims at offloading the expensive RSA computation from weak devices (such as smart cards) to powerful but un-trusted server [60].

In the technique of re-balanceing the SSL handshake, the server sends the vector of elements and requests the client to perform some operation. The result of this operation is used by the server to decrypt the pre-master secret that is sent by the client. A care has been taken so that an attacker cannot derive the private key of the server from the vector that is sent by the server. An overview of the operations performed to re-balance the SSL handshake are described below.

- The client hello and the server hello are the same as in the standard SSL handshake.

- The server’s certificate message includes the vector \( B = (b_1, b_2, ..., b_k) \). The server’s private exponent \( d \) can be expressed as follows:

\[
d = (a_1 * b_1 + a_2 * b_2 + ... + a_k b_k) mod \Phi(n)
\]  

(9.1)

Where,

\( a_i \) and \( b_i \) are random vector elements of size \( c \) bits and \( |n| \) bits respectively.

- The client encrypts the pre-master secret (a 48 bit number) with the server’s public key that is obtained from the server’s certificate to obtain the value \( y \). This process is same as the standard SSL handshake. Here, \( y \) is the encrypted pre-master secret.

- The client uses the vector \( B \) from the server’s certificate message and does the following operation to obtain \( z \). This is an extra operation that the client needs to perform extra when compared to the standard SSL handshake.

\[
z_i = y^{b_i} mod \Phi(n)
\]  

(9.2)
for $1 \leq i \leq k$

The vector $Z$ such that, $Z = (z_1, z_2, ..., z_k)$ is sent to the server in the client key exchange message.

- The server is supposed recover the pre-master secret from the encrypted client key exchange message. Therefore, originally, the server needs to compute the $x$ as follows: $x$ is nothing but the pre-master secret.

$$x = y^d \mod(n)$$ (9.3)

Where,

$$d = (a_1 * b_1 + a_2 * b_2 + ... + a_k b_k)$$ (9.4)

Therefore,

$$x = \prod_{i=1}^{k} y^{a_i b_i} \mod(n)$$ (9.5)

Finally,

$$x = \prod_{i=1}^{k} z_i^{a_i} \mod(n)$$ (9.6)

- As shown in Equation 9.6, in the re-balanced SSL handshake, the server works less than that in the standard SSL handshake. The rest of the SSL handshake remains unchanged.

In this way, the server shifts a part of the heavy private key decryption operation at the client side.

Analysis

The server makes use of the CRT (Chinese remainder Theorem) to perform the exponentiation operation to decrypt pre-master secret message in the SSL handshake as described above. The details of the CRT are not described here for brevity. The confidentiality is appropriately maintained through the search space of $2^{ck}$. In this technique, the values of $c$ and $k$ are selected in such a way that the difficulty of exhausting the search space is equivalent to underlying RSA cryptosystem. The exhaustive search of $2^{ck}$ values is equivalent to searching for all possible keys in a symmetric-key cryptosystem (E.g. DES, AES etc.) [60]. The well known study done by Verheul and Lenstra [62] gives the formulae to determine the equivalent key sizes that gives the same security as the keys sizes in asymmetric cryptosystem. [60] considers study [62] while deciding the $2^{ck}$ search space\(^\dagger\).

The CA-RSA achieves the speedup in the range of 11 for 1024 bit key and 19 for 2048 bit key at the server side processing. In addition, the client faces an additional computational overhead varying according to the RSA key sizes. The bandwidth requirement is increased since the client has to send the an extra bytes in client key exchange message. However, since SSL is purely CPU bounded [37], adding the small data in the handshake protocol messages does not affect the bandwidth requirement and hence the performance of SSL.

The CA-RSA enables the server to perform better by offloading some of the RSA decryption work to the client-side. This proposal makes the DoS attack expensive to produce because of two reason. The first one is due to the fact that the server can handle more SSL handshake simultaneously and the client or the attacker will have to work more. However, the attacker with adequate resource and computational power can still execute the DoS attack on the SSL server.

\(^\dagger\)More details about the scheme can be referred from [60], it is not explained here since it is out of the scope of this paper
An attacker can start initiating the SSL handshake with the server that uses CA-RSA, when the server requests for an extra computation (of calculating the vector $Z$ as described in this section), the attacker can still send the bogus data. With this strategy the server will have to work considerably more than the client and hence DoS attack is still possible. Of course, the impact of DoS will be lesser than the DoS attack on the server that uses the traditional SSL handshake protocol. The difference in the impact can be anticipated by investing more resources (eg. bots) in the attack.

9.1.3 Performance analysis of SSL web servers

This section lists the important observations made in [37]. Researchers have studied the SSL web server performance under different conditions to determine the vectors affecting it. The results in [37] shows that the RSA operations costs significant overhead (13% to 58%) to the SSL web servers. However, it is not the only factor that accounts for overloaded SSL web servers [37].

- The optimizations aimed at reducing the volume of network traffic have little effect on the SSL server throughput.

- The cost of operations such as message fragmentation, assembling records (copy message fragments into SSL records), calculating sequence numbers and computing master key block from the pre-master secret is between 6% to 15% of the total performance cost.

- The usage of RSA accelerators gives the varying results. The RSA accelerators are most effective when the session reuse is lower and the data to be transferred is less. The RSA accelerators in such cases have shown to result in 46-111% performance improvements. However, the web servers where the session re-use rate is lower and larger web data is to be handled, the RSA accelerators results in 11-24% performance improvement. These results are regardless of the how fast the RSA accelerators are [37].

- The CPU cost associated with SSL connection setup have a greater impact on the SSL throughput than the CPU costs associated with the SSL data exchange [37]. As CPU becomes faster, the relative cost of cryptographic operations in the SSL such as RSA, MD5, RC4 decreases. This results in shifting the load to the non-SSL components such as the web server and the operating system, kernel [37]. Therefore, with the faster CPU, the difference between the performance of the insecure (non-SSL) web server and the SSL web server will be reduced [37].

- According to the results obtained in the experiment at [37], the performance given by the dual CPU server is greater or at the worst equal to the performance given by the single CPU with RSA accelerators.

Some of these results are also confirmed during the experiments performed in this research.

9.1.4 Using client puzzles to protect SSL

In [63], the researchers have included the new record type (along with standard record types such as the handshake, the change cipher spec, the alert, and the application data) for the puzzle message. They have modified the OpenSSL library to implement the prototype of the SSL that uses the client puzzles to protect the SSL web servers. The idea of client puzzle is originally proposed at [59].

The idea behind this prototype is to send the puzzle request to the client only when the load on the server is beyond certain predefined threshold. The threshold is measured by
employing few extra callback functions that allows the OpenSSL application to understand if the client puzzle request needs to be sent. The client puzzle request is sent just after the (server) certificate message and before the server hello done message. The server hello done message is sent only after receiving the reply to the client puzzle message. The difficulty level of the puzzle structure proposed in [63] can vary. The variation can be decided by the server according to the load on the server.

Analysis

Solving the puzzle should be finished in a reasonable amount of time and should not take unnecessary computational resources at the client side. This requirement is considered in order not to harm the legitimate clients with the slow machine should be satisfied.

The other implications of the usage of the client puzzles is already discussed in Section 9.1.1. [63] suggests the important functionality that enables the usage of the client puzzles at run-time.

9.2 Random Discussions: DoS attacks on SSL

9.2.1 Session Tickets

Two mechanisms can be used to accomplish the abbreviated handshake. One mechanism is already explained in this thesis. It is called session resumption. In session resumption, the client includes the session identifier from the last session in the client hello. If the server can find and agrees to resume the session that is identified by the session ID, an abbreviated negotiation takes place, that is expensive operations such as private key decryption are not performed. The session resumption, however, adds the cost of remembering the whole session to the server.

In the new approach, on each successful SSL negotiation, the server sends a ticket to the client. This ticket is called session ticket. This ticket consist of all the information needed by the server to perform the abbreviated handshake. Whenever the client re-visits the server, the client can present the ticket to the server and request for the abbreviated SSL handshake.

The session ticket is opaque to the client [64]. The session ticket consist of following items.

key_name This field helps the server to identify the set of keys used to protect the ticket. These keys include one 128-bit AES (Advances Encryption Standard) in CBC (Cipher Block Chaining) mode key and other 256-bit key for HMAC-SHA-256.

IV The initialization Vector that is used in the CBC mode encryption.

Encrypted State An encrypted session state information. The session state consist of the information negotiated during the last session including the time stamp and the authentication method used. The time stamp helps the server to decide whether to validate or invalidate the session ticket presented by the client.

MAC This field contains the MAC data. This data is calculated using the key_name, IV, length of encrypted state field and its contents.

The session tickets are most useful in the following scenarios as listed in RFC 5077 [64].

- The server that handles large number of transactions from the different users.

---

2 The details of the puzzle is not provided here, interested readers can refer [63] for more information
3 HMAC: Hash based Message Authentication Code
The server that desires to cache sessions for long time.

The servers can share the cache. This will enable to load the session across the servers of the single service provider.

The server that has little memory such as embedded servers.

Whenever the session ticket is presented to the server, server searches the matching key_name in its session cache. It helps the server to decide whether the ticket is valid and issued by the server or not. The successful look up for the key_name is followed by the decryption of the session ticket to obtain the session information. The session resumption using session tickets reduces the need of the amount of the session cache with the cost of few cryptographic operations.

Usage of session tickets as a DoS attack vector

An attacker cannot create the forged session tickets due to the fact that the key_name field is generated randomly at the server side. In addition, the ticket is protected using the strong encryption and HMAC algorithms.

However, the attacker can collect the valid session tickets from the server over a period of time and use them to request the server to verify them [64]. The verification process consist of 256-bit HMAC calculation followed by the symmetric key decryption. These processes are comparatively cheaper than the private key decryption. However, there is a possibility to overload the server with such requests. The impact of such attacks cannot be higher than the traditional DoS attack discussed through out in this thesis.

9.2.2 DoS attacks with respect to different SSL/TLS protocol versions

Table 9.1 gives the SSL protocol version history. As observed from this table, TLSv1.2 is the latest version available and it is published quite a while ago. However, most of the browsers do not support the TLSv1.2. The browsers like Mozilla firefox, support the TLS1.0 as its latest version.

<table>
<thead>
<tr>
<th>Protocol Versions</th>
<th>SSLv1</th>
<th>SSLv2</th>
<th>SSLv3</th>
<th>TLSv1.0</th>
<th>TLSv1.1</th>
<th>TLSv1.2</th>
</tr>
</thead>
</table>

Throughout this thesis, we referred these protocols as one irrespective of their version. A brief study is performed to investigate if the different versions of the SSL/TLS protocols affect the possibility of implementing the DoS attack on the secured servers. That is, if any particular protocol creates any additional opportunity to execute DoS attack to create greater impact of the server side.

This study was required to create a guideline for those who are interested in upgrading the older versions of the SSL protocols employed on their servers. The newer version of the SSL protocols differ from their accentors, mainly in terms of the support for newer cipher suites and improved checking of the protocol messages and added mitigation techniques for attacks, for instance, TLSv1.1 [32] and TLSv1.2 [49] protects against CBC attacks (E.g. Beast attack [65]).

We did not find any differences among SSL protocol versions (SSLv3.0, TLSv1.0, TLSv1.1, TLSv1.2) in terms of increased or decreased possibilities of DoS attacks. This is because, the
basic functionality, the default protocol messages and the expensive nature of any cryptographic operation performed in the SSL protocols has remained the same among the different versions of the SSL protocols.

9.3 Summary

This chapter described four different literatures those are studied during the course of this thesis. In addition, this chapter presented a short description about the session tickets and the impact of the DoS attacks with respect to different versions of the SSL protocols.

The reverse SSL aims at reversing the roles of client and the server in the SSL handshake when client authentication is used and employing the client puzzle techniques when the client authentication is not used. This proposal of change in the protocol focuses on decreasing the computational requirements of the server during the process of SSL handshake by using the technique called online/offline signature scheme. We argued that this proposal does not mitigate the risk of DoS attack on the servers those use reverse SSL.

In Section 9.1.2 we studied the literature that focused on adjusting the computational imbalance between the server and the client by shifting some of the heavy operations from server-side to the client-side. However, this technique does not completely mitigate the risk of DoS attack. In Section 9.1.3 we listed findings from literature [37]. These results confirm the fact that the SSL protocol is processor intensive. Adding more processors can increase the performance given by the SSL server. Section 9.1.4 describes the idea of application of the "client puzzle" concept in the SSL protocol to request the client to solve the computationally intensive puzzle only when the SSL server is overloaded. The original work can be found at [63].

Section 9.2.1 described the concept of the session ticket. The session tickets can be used for initiating the abbreviated handshake with the server. This method is preferable over the traditional way of session resumption (using session ID). Finally, in Section 9.2.2 we state that the risk of the DoS attack is same irrespective of the different versions of the SSL protocols (SSLv3.0, TLSv1.0, TLSv1.1, TLSv1.2).
In this chapter, we summarize the observations made during the course of this thesis work and certain key areas that should be explored further as future work. We first revisit the problem of DoS attacks and then consider each attack vector experimented in this thesis. Next, this thesis is concluded on the basis of the analysis of the findings of the test cases and the results obtained in the comparative experiments in this thesis work.

The SSL protocol consists of the number of sub-protocols. The SSL protocol is complex in structure. There are several reasons that make the SSL protocol a potential target to deploy DoS attack. As the DoS attacks on the TCP may also be applicable on the SSL protocols, care needs to be taken while implementing the SSL protocol under any high level application layer protocol (E.g. HTTP) that originally runs on the transport level protocol such as TCP.

10.1 Attack Vectors

In this research, the SSL is studied in detail, and some of the attack vectors are identified that can be exploited to build DoS attack on the SSL server.

This section considers all the attack vectors identified in this thesis. The attack vectors are referred to the processes that are conducted in the SSL protocol. For example, this thesis identified few of the processes such as client authentication, compression, cryptographic operations and SSL renegotiation. This section discusses the likelihood of exploiting these processes to produce DoS condition.

More specifically, this section summarizes the work done in this thesis on these attack vectors and discusses the extent to which DoS attacks can be successfully executed on the SSL protocol exploiting these attack vectors.

10.1.1 Client authentication using RSA certificates

The SSL protocol provides the functionality of authenticating the client to the server. In some applications where the client authentication is required, the server requests the client to send its certificate. If the client is configured with the certificates, it sends its certificate or chain of certificates that provides the trust about the client’s identity to the server. However, the verification of the client’s certificate introduces some extra work for the server.

Therefore, the impact of the ’client authentication’ is investigated.
Certificate chain construction

**Experiment.** *Send the list of certificates in such a way that it creates a loop at the server side while constructing the certificate chain.*

The idea behind this experiment was to construct the certificate chain in such a way that the server will not be able to construct the valid certificate chain due to the formation of the loop during construction of the certificate trust path. It is observed that when the depth of the parsed certificate becomes greater than the maximum allowed depth at the server side, the server simply breaks out of the method responsible for the certificate chain building. The SSL connection is then immediately terminated by issuing the handshake failure alert. Therefore, The SSL protocol performance is not affected by malicious construction of the certificates that forms a loop while constructing the valid certificate chains. There is no DoS attack possible using this technique.

**Length of certificates in a chain & size of public exponent**

**Experiment.** *Send the chain of the client certificates with maximum possible depth. Measure the performance of the server for:*

1. The different public exponent values.
2. The different depth of the certificate chain.

**Impact of size of public key exponent** The certificate verification is the process of signature verification. Whenever the RSA certificates are used, the server performs the RSA signature verification operation. According to the results obtained in Chapter under section 7.3.3, the length of the public exponent does impact the server performance. The server requires more time verifying a certificate with an increase in the length of the public exponent of the RSA key that is used to sign the certificate.

The maximum possible size of the public exponent allowed in the OpenSSL is 3071 bits. Therefore, an experiment is performed to verify the impact of the size of the public exponent where the certificate is signed with the RSA key with public exponent of size 3071 bits. The server completed the 1000 handshakes in 8.96 seconds when the client sent the RSA certificate that had the RSA signature by the RSA key with public exponent of 65537. The same server required 80.249 seconds to complete 1000 handshakes when the client sent the RSA certificate with 3071 bits public exponent RSA signature. Hence, the performance given by the server reduced by the 482.61%.

Therefore, The server’s performance is drastically reduced when the client sent a certificate with largest possible public exponent. Therefore, the certificates with larger public exponent is a high risk DoS attack vector.

**Impact of size of certificate chain** Whenever the server requests the client to provide its certificate for the process of client authentication, the client can send the chain of certificates that inherently create the trust through series of certificates required by the server. Contrary to the only single certificate, whenever a group of certificate is sent, the server has to perform the number of steps to verify the identity of the client. At first, the server constructs the certificate chain. Once the certificate chain is built, the server verifies the number of constraint provided in the certificate. One of the important constraints is known as 'basic constraint'.

---

1 Almost all the applications use the RSA certificates.
2 Largest possible public exponent in the Openssl
This is one of the extensions included in the X509 v3.0 certificate that provides some basic information about the certificate. There are two attributes in the "basic constraint". The CA attribute suggests whether the certificate is an end-entity certificate or the CA certificate. If CA = TRUE then it is a CA certificate else (CA = FALSE) it is an end-entity certificate. The end-entity certificate cannot be used to sign any other certificate. The second attribute is a 'path length' attribute. This attribute gives the number of (intermediate CA) certificates that can exist between the current certificate and the end entity certificate.

The OpenSSL (SSL) server implementation correctly verifies if these attributes are correctly pursed and then only proceeds with the signature verification. An experiment is performed where the impact of the length of the certificate on the SSL server is tested. It has been confirmed that the server does require more time when the chain of 10 certificates is sent as opposed to the single certificate to the server for the client authentication. In this experiment, the maximum certificate chain of depth 10 is chosen due to the fact that the Apache web server allows the chain of 10 certificates to be constructed at the server side when the client sends the set of certificates. This is the default value in the SSL configuration file of the Apache web server. However, the scenario where the chain of 10 certificates is allowed is not common in the real world. However, verifying the impact of 10 certificate verification process is worth investigating because of such default configuration files. There is always a possibility in the real world where the certificate chain of depth 10 is allowed.

It has been found that the length of the certificate chain degrades the server performance. Whenever the greater public exponent (a bit less than the modulus) is used to produce the signature inside the certificate, the impact on the server’s performance is even larger. Whenever a standard public exponent value (65537) is used, the server’s performance is reduced by 9.17 % for three certificates in the chain and by 39.14 % for 10 certificates in the chain.

When the public exponent of value 3071 bits is used, the server performance is decreased by 255.50% for three certificates in the chain and by 766.31 % for 10 certificates in the chain.

**Final Remarks**

<table>
<thead>
<tr>
<th>Size of Certificate Chain</th>
<th>Size of public exponent</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>&lt; 32-bits</td>
</tr>
<tr>
<td>Single Certificate</td>
<td>No problem</td>
</tr>
<tr>
<td>3 Certificates</td>
<td>No problem</td>
</tr>
<tr>
<td>10 certificates</td>
<td>Needs attention</td>
</tr>
</tbody>
</table>

Table 10.1: Certificate verification experiment: Summary of risk analysis of the client authentication based DoS attacks

Finally, the malicious certificates which create a loop while constructing the certificate chain cannot impact the server’s performance and hence does not count for any type of DoS attack.

The server’s performance is significantly degraded when RSA keys with larger public exponents are used to sign the certificate. The impact of certificate verification is even greater when the length of the certificates to be verified at the server-side increases in a number. Hence, such process of certificate verification can be counted as one of the causes for the computational DoS attack on the SSL servers. Table 10.1 summarizes the result of investigation of the certificate verification experiment.
10.1.2 Record processing

Several experiments were performed to investigate how the SSL records are processed in the SSL protocol. It has been found that the SSL record layer addresses fairly standard problems of DoS attacks that are possible on transport layer protocols such as TCP. However, some of the DoS attacks such as TCP SYN flood, are still possible on the web-servers that uses SSL protocols. This DoS attack is not on the SSL protocols but on the TCP. These attacks will always be possible since SSL runs on the top of TCP. It uses TCP for reliable delivery of the SSL-related data.

10.1.3 Compression in SSL

The SSL enables the use of the compression. The compression method is negotiated along with the cipher suite renegotiation. The client and the server are mandatory to support the compression method. This method indicates that no compression will be used in the SSL communication. In reality, only Google chrome browser supports the compression. Few standard web sites such as www.google.com, www.amazon.com, www.twitter.com does opt for the compression method. Hence it can be concluded that these web sites do not support SSL compression. Compression is very much useful when there is a larger amount of content (specially text files) that needs to be transmitted. Most of the e-commerce web sites and internet banking applications involve a very little information (credit card number, passwords, personal information) that needs be compressed. This could also be one of the reasons for the web sites not enabling the SSL compression.

Compression burst handling

**Experiment.** Send a maliciously crafted SSL record such that when it is in compressed format, it follows the boundary limit imposed by the SSL specification, but when decompressed, it expands into a large amount of data that is not allowed by the SSL specification and/or the SSL implementation cannot handle it.

A theoretical experiment is done to verify whether the server handles the compressed data safely without introducing any vulnerability to produce any memory-related attacks such as buffer overflow. The OpenSSL systematically checks the message boundary before starting decompression to confirm that it follows the limits on the compressed data boundary required by the standard. The OpenSSL also makes sure that the compressed data does not expand into larger size so as to occupy unnecessary memory available to the server. If the compressed data crosses the limit specified by the SSL standard, the process is immediately stopped and it results in SSL connection termination.

Due to the fact that, the compression requires to maintain the history at the both sides of communication. Therefore, in order to produce a compression overhead at the server side, the client will also have to invest significant amount of resources. But, this cost of compression on the client side can be easily alleviated by compressing the data in advanced using the compression history from earlier sessions.

It would be quite interesting to measure the difference (if any) in the overhead at the client side and the server side when compression is used at SSL level. Due to lack of time, the practical results could not be generated to provide the evidence of this issue and measure the impact of SSL compression on the server side.

Finally, compression burst handling is correctly handled in the SSL protocols.
10.1 Attack Vectors

10.1.4 Cryptographic operations

The cryptographic operations are one of the important attack vectors when DoS attacks on the SSL protocols are considered. As discussed in this thesis, the SSL protocols involve the number of expensive cryptographic operations. Especially, the server has to perform the RSA private key operations which are expensive due to several factors discussed throughout in this thesis.

This section summarizes the problems that may lead to DoS attacks with substantial performance impact.

Role of key lengths

It has been found that the usage of long private keys degrades the server’s performance significantly. The experimental results obtained in Chapter 6 are summarized here.

<table>
<thead>
<tr>
<th>SSL connection side</th>
<th>Time required for key lengths (bits)</th>
<th>Percentage change for time required between</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1024</td>
<td>2048</td>
</tr>
<tr>
<td>Server</td>
<td>4.88</td>
<td>12.21</td>
</tr>
<tr>
<td>Client</td>
<td>2.12</td>
<td>2.33</td>
</tr>
</tbody>
</table>

Table 10.2: Comparison of time required by the server & the client to perform 1000 handshakes for different RSA private key sizes

It is observed that the CPU time requirement of the server increases exponentially with an increase in private key size. Therefore, the selection of the private key sizes should be done carefully. It should be noted that the statistics provided in Table 10.2 is done for RSA key exchange method. One of the important observations from this table is even if the computational overhead on the server is increasing exponentially on the server side, the increase in the overhead on the client side remains significantly lower. However, the server is likely to be equipped with greater processing power compared to the client. But, the server will also have to serve a multiple number of clients.

Therefore, the increased key lengths can impact the server’s performance drastically. This vector can play an important role in executing the DoS attacks on the SSL server.

Role of authentication method and key exchange method

**Key exchange method** The Diffie-Hellman key exchange method requires both the server and client to perform the exponentiation operation. However, the server can perform its exponentiation operation prior to starting the handshake and hence it requires less CPU time than the client as already shown in Chapter 6.

The RSA key exchange method involves the cheaper public key encryption operation at the client side and the expensive private key decryption operation at the server side increasing the work overhead on the server side. The difference in overhead is already described in this thesis.

**Authentication method** According to one of the empirical studies in [47], a majority of the web sites uses the RSA certificates for authentication. Alternatively, the DSA certificates may also be used. The DSA signature scheme is balanced unlike the RSA signature scheme. In addition, DSA signature verification requires more time than the DSA signature generation. Therefore, the client is likely to require greater time than the server, as the client verifies the server’s DSA certificate (containing DSA signature).
However, the usage of the DSA certificates are rare. This could be because of the flexibility and ease provided by the RSA certificates. The RSA certificates can play the dual role as the authentication method as well as the key exchange method. Where as in the case DSA method, it can be used only for the authentication and not for the key exchange. For key exchange, a separate method such as DH needs to be used. This will result in an increased complexity.

**Distribution of work overhead**

One of the important reasons for the SSL to be the target of the DoS attacks is the unbalanced work overhead. The server is the one who has to perform the RSA decryption operation as opposed to the client who performs the comparatively cheaper operation such as RSA encryption.

The other operations involved in the SSL protocol are same on the both connection ends and therefore, do not contribute to the uneven distribution of the work overhead. These operations include the MAC calculation, master secret generation, etc. Once, the SSL handshake is completed, the symmetric key operations are very much cheaper than the public-key operations performed during handshake and these operations need to be performed on the both sides.

**10.1.5 SSL renegotiation**

The SSL renegotiation feature is present in the SSL to enable the renegotiation of the key materials and/or re-authentication of the end-entity. The applications of the renegotiation are described in Chapter 4. On one hand it enables the flexibility in renegotiating a new session, and on the other hand, it introduces the complexity in the already existing SSL protocol.

The SSL renegotiation is one of the important attack vectors that can be exploited to build a hazardous DoS attack. The renegotiation feature can be exploited to initiate the number of SSL handshake protocol. The handshake protocol is itself an expensive part of the SSL protocol. In renegotiation, this fact can be completely exploited to deplete server’s resources such as CPU power. The results in Chapter 8 confirm the harmful impact of renegotiation based DoS attack over the other traditional flood attacks.

**10.2 Attack Strategies**

This section summarizes the impact of the three DoS tools considered and compared in this thesis. These three DoS tools work on three different attack strategies. These three strategies focus on requesting the server to perform as many handshakes as possible. The difference is the way they request the server to perform the SSL handshake.

**10.2.1 sslsqueeze tool**

The sslsqueeze tool [18] does not use any cryptography to produce the number of SSL handshakes with the server. All the standard SSL Handshake protocol messages and Change Cipher Spec messages are prepared in advanced (in raw hex format). This tool requires target IP address, port number (on which the SSL server is running on the target server) and the key size of the target server. The key size is required in order to construct the client key exchange message of appropriate size according to the target’s key size. This is required since, an implementation such as Openssl [8] verifies the size of the client key exchange message before starting the decryption process.
Once this tool is started, it opens a new socket and starts the SSL Handshake protocol by sending client hello message. When some message is available to read, it is read from the socket to understand the server’s response. It is necessary to understand the server’s response. Once server replies with the server hello, this tool sends the rest of the messages in the SSL handshake protocol.

**Problem with sslsqueeze tool**

It is observed that the sslsqueeze tool does not perform better than the other two tools discussed in Chapter 8. The main reason behind this anomaly is that sslsqueeze tool does not complete the total SSL handshake with the server. This is because, this tool does not compute the valid key exchange message and it is caught by the server. Therefore, the server terminates the SSL connection immediately. With this scenario, the time required to prepare and transmit the change cipher spec message and the server finished message is absent in transaction spawned by the sslsqueeze tool. In addition, this also saves the server’s effort for one signature and symmetric encryption required to prepare the server finished messages.

These factors impact the performance given by the sslsqueeze tool when compared with other tools. The BFC and thc-ssl-dos tools create the SSL connections with the target server such that these connections are successfully completed before these are terminated either by the server or the control written in the tool itself.

### 10.2.2 Brute Force with Cryptography

This tool makes use of standard OpenSSL library [8] to initiate the SSL handshake with the target server. To start the multiple SSL connection simultaneously, it uses threading mechanism.

### 10.2.3 thc-ssl-dos

This tool uses the standard cryptographic libraries provided by the Openssl [8] to create multiple SSL connections with the server and to computationally overload the server. This tool is originally obtained from [1]. This tool uses the SSL renegotiation feature. Whenever the SSL renegotiation is not supported at the server side, this tool cannot be used to attack the SSL server. This tool creates a single SSL connection with the target server and performs a number of SSL handshakes within a single connection.

### 10.2.4 Final remarks

This section gives some final remarks about the three experimental DoS tools considered in this thesis work.

Table 10.3 gives an overview of the applicability of DoS attack strategies on different versions of the SSL protocols. The thc-ssl-dos tool cannot be used on the servers equipped with SSLv2.0 [33] protocol. This is because, SSLv2.0 does not support SSL renegotiation.

Table 10.4 gives an overview of DoS strategies’ comparison experiment. The first half of the table gives the impact statistics of the target system which was attacked, and the other half gives the impact on the machine which was attacking the target machine.

The impact on the target server’s machine is discussed below.

**load** The load on the target server created by the thc-ssl-dos tool is the highest. This signifies that, when thc-ssl-dos tool was attacking the server, there were a large number of processes (instructions) those were waiting to be completed.

---

3BFC tool closes the connection once full SSL handshake is completed.
Table 10.3: Applicability of DoS attack strategies on different versions of the SSL protocols

<table>
<thead>
<tr>
<th>DoS Tool</th>
<th>SSLv2.0</th>
<th>SSLv3.0</th>
<th>TLS v1.0-v1.2</th>
</tr>
</thead>
<tbody>
<tr>
<td>thc-ssl-dos</td>
<td>N/A</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>BFC</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>sslsqueeze</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

**CPU (user)** The CPU (user) time signifies the time spent on the processor. Again, all the DoS tools create a situation, where most of the user CPU is busy. This indicates that, these tools are computationally processor intensive. Therefore, the impact of such tools depends upon the number of processors available on the target server.

**CPU (system)** The CPU (system) time signifies the time spent on the OS kernel. This factor is differently affected by different tools unlike CPU (user) time. The thc-ssl-dos requires least time and other two tools require nearly same time. This is obvious because, the thc-ssl-dos tool makes lesser network calls than other two tools. In other words, The thc-ssl-dos tool connects once and creates a number of handshakes in a single connection, whereas other two tools focus on creating as many connections as possible to create a single handshake at a time. Hence, the BFC and the sslsqueeze tools end up creating multiple network calls.

Table 10.4: Overview of DoS strategies comparison Experiment: Impact on the target machine and client machine

<table>
<thead>
<tr>
<th>DoS Tool</th>
<th>Impact on target server</th>
<th>Impact on attacker’s machine</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>load (%)</td>
<td>CPU(user) (%)</td>
</tr>
<tr>
<td>thc-ssl-dos</td>
<td>91.76</td>
<td>78.3</td>
</tr>
<tr>
<td>BFC</td>
<td>82.74</td>
<td>77</td>
</tr>
<tr>
<td>sslsqueeze</td>
<td>21.33</td>
<td>81.5</td>
</tr>
</tbody>
</table>

The impact on the attacker’s machine is greatest for the thc-ssl-dos tool when compared with other two tools. The memory requirement for the sslsqueeze tool is the least since it does not make use of any cryptography and hence it has least investments in data structures and OpenSSL library. The CPU usage is also less for the sslsqueeze tool. In addition, the sslsqueeze tool does not create a significant load on the attacker’s system.

Finally, load on the attacker’s system remains significantly lower than the load on the target server’s system for each attack strategy. All of these tools are processor intensive. The impact of the thc-ssl-dos tool is the highest. However, The main draw-back behind the thc-ssl-dos tool is that it only works if SSL renegotiation is enabled on the server side. With this respect, the tools like BFC are most relevant as this tool will work for any SSL servers. These final remarks are summarized in Table 10.5. The last column in this table specifies that the mitigation technique for the thc-dos-ssl tool is "easy" and for other tools, it is specified as 'hard'. This is due to the fact that the SSL renegotiation based attack can be mitigated just by disabling the support for the SSL renegotiation. The other two attacks cannot be easily disabled since they do not depend upon any protocol specific extra feature. These tools follow the SSL protocol in a systematic way. One of the mitigation techniques for these tools would be to monitor and limit the SSL handshake request from a source. A broad overview...
of the overall mitigation techniques based on the observations made during this research is
given in next section.

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Attack Strategies</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SSL renegotiation based [thc-ssl-dos]</td>
</tr>
<tr>
<td>Load on server</td>
<td>High</td>
</tr>
<tr>
<td>SSL renegotiation support</td>
<td>Required</td>
</tr>
<tr>
<td>Cryptography support</td>
<td>Required</td>
</tr>
<tr>
<td>Computational requirement at server side (Impact on the processor)</td>
<td>High</td>
</tr>
<tr>
<td>Mitigation</td>
<td>Easy</td>
</tr>
</tbody>
</table>

Table 10.5: Applicability of DoS attack strategies on different versions of the SSL protocols

10.3 DoS Attack Mitigation Techniques

This section is dedicated to provide some mitigation techniques for DoS attacks on the SSL
protocol.

Since the significant overhead on the SSL server is due to the expensive cryptographic
operations performed during the SSL handshake protocol, the rate of SSL handshake should
be reduced at the first place. This can be done by using the session cache technique. Using
the abbreviated handshakes (SSL session resumption), expensive cryptographic operations
are not performed.

10.3.1 Mitigation by monitoring

- Detect the sources that send an excessive number of requests for a secured object.
Automated attack tools almost always request web pages (or any other resources) more
rapidly than standard users. The SSL connections that are terminated continuously just
after RSA decryption are most probably DoS attack traffic. Therefore, monitoring and
blocking such connection requests can mitigate the SSL DoS attack those are based on
gratuitous requests to create the computational burden on the server. However, there
is no explicit mechanism (at least in OpenSSL) to detect whether RSA decryption
is successful. Still monitoring the SSL error messages can be useful to employ such
mitigation technique.

- The sources that requests for some standard and small pages such as index pages and
issues the SSL renegotiation request in an excessive number is likely to be the sign of
the DoS attack. This would be the SSL renegotiation based resource exhaustion attack.
Monitoring and blocking such traffic can be useful in mitigating such DoS attacks. one
of such mitigation techniques is already described in Chapter 4.

- The SSL traffic that connects to the server and requests smaller resources at some
interval of time repeatedly to keep the connection SSL connection alive can be a part of
the SSL DoS attack where an attacker tries to engage the server in an idle connection
and slowly depleting the server’s capacity to serve new SSL connections. Such requests
should be monitored and phased out.
10.3.2 Mitigation by load balancing and abbreviated handshakes

- Load balancers (especially layer 7 load balancers) should be configured in such a way that all the requests from the same source IP addresses are directed to the same servers. This will minimize the need of performing full length handshakes each time and hence will save the net CPU time of the servers belonging to an organization provided session cache are maintained at the server. In addition, the server throughput will increase due to fewer handshakes, enhancing the user experience in terms of short response time.

- Efficient use of session cache will enable the frequent use of abbreviated handshakes. If the user re-visits the web site within a short duration, the session ID included in its client hello message can be used to resume the session remembered by the server in its session cache. It is assumed that the user (browser used by the user) properly includes the session ID in the client hello message.

10.3.3 Mitigation by strategic changes

This section describes some mitigation techniques that involves either the change in protocol or the change in the way SSL protocol can be implemented.

- The client certificates can be used to execute the DoS attack as discussed in Chapter 7. Whenever the client certificates are used, the client has to send the client certificate message, and the certificate verify message. The certificate verify message is the signed handshake messages exchanged so far in the current SSL communication. This signature operation is not cheap. Therefore, the client can send a fake certificate verify message to the server in order to save some time and computational power. The OpenSSL server implementation first verifies the certificate chain and then verifies the certificate verify message. Whenever, a single client certificate is used, there is no problem since, the server will have to perform at least one signature verification operation. But, whenever multiple client certificates are used (certificate chain), the OpenSSL can opt to verify the certificate verify message first. If the verification is successful, then it can continue with the process of constructing and verifying the client certificate chain. One important thing is that the server performs the RSA private key decryption (decryption of the client key exchange message) before verifying the certificate verify message. The operation of decryption is expensive and should not be performed if certificate verify message can not be successfully verified. This is also because, the certificate message can be stolen easily by sniffing the network traffic and hence can be mis-used by the attacker. But, the certificate verify message cannot be faked since to prepare the certificate verify message, private key of the certificate is required. This mitigation technique is also listed by Vincent Bernat [7].

- Number of literatures, including [63], provides the number of SSL protocol amendments. These literatures are already discussed in this thesis. Such changes in the SSL protocol can be activated when the server is experiencing the load. For instance, if the load on the server is increased beyond a threshold, the server can compulsorily request the clients to solve the computationally expensive problems in order to maintain the server’s availability during the high load conditions. This can reduce the server’s throughput for the legitimate users. However, reduced throughput is better than the total outage. As soon as the load on the server is reduced below the threshold level the SSL server

---

4 The certificate message is sent in the plain-text format.
5 Co-incidentally, this idea of mitigation technique is already discussed by Vincent Bernat [7].
can fall back to follow standard SSL protocol. One of such experiment is performed by

10.3.4 Mitigation by configurational changes

- As discussed and observed throughout this thesis, the basic cause for the DoS attack is due to the computational requirement of the SSL protocol. In addition, the computational requirement is unbalanced. The server needs to perform more expensive operations than the client. Last, but not the least, the client can spawn a larger number of SSL handshake requests to the server without much investment in the resources. Therefore, one full-proof mitigation to such computational DoS attack would be increasing the server’s processing powers. This can include, more processors, hardware accelerators, greater memory, etc.

- The SSL servers can be the victim of the DoS attacks that focus on slowing down the connections. This is because, such attacks exploit the vulnerabilities in the base protocols such as TCP, and the SSL protocol runs on the top of it. Therefore, the SSL server should be equipped with the sufficient connection pool to accommodate a larger number of concurrent connections.

- Configure the server with DSA certificates for authentication purpose.

- Make sure that, the server does not accept the client certificates with either longer trust chain or with longer public key exponent.

10.4 SSL Server Performance Improvement Techniques

This section provides important SSL server performance improvement techniques and their effectiveness.

10.4.1 Hardware accelerators and session cache

SSL hardware accelerators are used to offload the processor-intensive modular arithmetic operations such as RSA private key decryption to speed up the SSL handshake operations. Literature [37] analyses the SSL server performance for different server configurations. It has been shown that the SSL hardware does improve the server performance, but they do not give the optimum results as one expects. In their experiments, the performance given by the dual CPU server gives the better performance than the single CPU server with the RSA accelerator. It is suggested that instead of investing in a relatively expensive hardware accelerator, a better choice would be to purchase a faster CPU.

The usage of RSA accelerators gives the varying results for the SSL web servers. The RSA accelerators are most effective when there is a low session reuse and data to be transferred is less. The RSA accelerators in such cases have shown to result in 46-111% performance improvements. However, the web servers where the session re-use rate is lower and larger web data is to be handled, the RSA accelerators results in 11-24% performance improvement. These results are regardless of the how fast the RSA accelerators are [37].

10.4.2 Elliptic curve cryptography

Elliptic Curve Cryptography (ECC) is emerging as an attractive alternative to widely used RSA cryptosystem. Certain advantages of using ECC cryptography over RSA cryptogra-
phy are ECC can offer equivalent security with smaller key sizes, which results in faster computation, lower power consumption, as well as bandwidth savings [66].

In [66], the SSL server performance is analyzed with the use of ECC cryptography over the RSA cryptography. It has been shown that the ECC can perform twice as fast as RSA when RSA 1024 bits key and ECC 163 bits keys were used. The speed-ups while using ECC and RSA with greater key size increases.

10.5 Future Scope

This thesis describes the details of the implementation of the compression in the SSL protocol. The results of the code review done on the OpenSSL [8] source code for the compression utility looks promising and does not appear to create any anomalies that can cause the DoS attack on the SSL protocol functioning. However, it would be interesting to test the computational impact of the compression on the SSL protocol. Due to lack of time, practical performance testing experiments for the compression are not performed, and left for a future work.

The usage of session tickets enables the session resumption without the server-side state unlike session resumption using the session identifiers. However, the usage of the session tickets introduces a small overhead on the SSL servers. This overhead is contributed by one symmetric key operation and one MAC verification. Although, these operations are not as expensive as other cryptographic operations in the SSL protocol, it would be interesting to examine the impact of the session resumption using session tickets. This experiment is left for a future work.

There are several DoS mitigation techniques mentioned throughout this thesis. Some of them are based on heuristic approach, for instance, the ip table entry to filter the probable SSL renegotiation traffic to combat the malicious SSL renegotiation requests (see Chapter 4). The efficiency of these techniques to catch the attack traffic and mitigate the DoS attacks needs to be measured. This will enable the evaluation of such techniques. This evaluation is left for a future work.

Finally, the effectiveness of the different DoS attack strategies is tested on the Apache web server. As already mentioned, the apache web server was selected for the experiments due to its wide usage. However, a comparative study where these attack strategies are tested on the different web server such as IIS (Internet Information Services) to measure their comparative impact is the next step. It should be noted that, the SSL renegotiation based DoS attack is impossible on IIS web server version 6 and later, IIS does not support the SSL renegotiation.

10.6 Conclusion

An increasing number of websites, including internet banking, e-commerce, social networking are using the SSL protocols to secure their communication over the internet. The SSL protocols enable the authentication along with the protection of the application level data. The SSL protocol starts with its sub protocol called the handshake protocol. The handshake protocol is responsible for negotiating the cipher suites and compression method, the authentication and the key exchange. These processes involve cryptographic operations and are usually expensive in terms of CPU usage and memory requirements. This thesis analyzed the various attack vectors those can be exploited to degrade the server’s performance and ultimately produce the Denial of Service attack.

This thesis investigated the server based Denial of Service attacks on the SSL protocols. The existing SSL protocols [31, 30, 32, 49] are unbalanced in terms of the computational requirements between the server and the client. There are no existing explicit literatures that
cover all the aspects of the SSL protocols those models the DoS threat for the SSL protocols. The existing literature [6] [60] [63] [61] focuses on the reducing the load on the SSL server either by shifting some of the cryptographic computation to the client-side or requesting the client to work sufficiently more before the server performs any expensive operations. The literature [37] suggests that the extra load on the web servers due to the SSL-related operations can be reduced significantly by employing the faster and/or more CPUs.

This thesis focused on following aspects:

- Study the DoS attack strategies that are possible to successfully create DoS attacks on the SSL web servers.
- Study the various aspects of the SSL protocol and investigate the possibilities to exploit any of these to execute the DoS attack. Measure the impact of the identified possibilities.
- Provide the effectiveness of some of the existing DoS strategies on the SSL protocol.

The DoS attacks on the SSL protocol cannot be completely avoided. The SSL protocol is an application layer protocol and runs transparently above the transport layer protocols such as TCP. The packet based mitigation techniques cannot sufficiently mitigate the DoS attacks which exploit the problems in the SSL protocol since such mitigation techniques act according to the signatures (definitions) of the attack those are provided to them. Generally, the attack traffic targeted at the SSL protocol appear to be the normal traffic at the transport level.

The main problem in the SSL protocols is the heavy cryptographic operations. These operations need to be performed at the starting of the SSL protocol in the phase called SSL handshake protocol. To make the situation worse, to perform the same SSL handshake, the client requires much lesser time and computational resources than the server. In addition, the client can request the server to perform many handshake requests. Therefore, this imbalance in the workload is the main cause of the DoS attacks. This kind of DoS attack depletes the server's computational resources ultimately leading to the DoS condition.

The costs of operations other than the public key cryptography are relatively balanced in the SSL protocol. That is, operations such as symmetric key operations (symmetric key encryption-decryption), MAC calculations, random number generation and data handling (fragmentation, data copy) costs the same at the client as well as at the server side. This has been practically proven at [37].

The SSL protocol allows either communicating party to authenticate each other. In general, the server is the one who authenticates itself to the client, mostly by providing the certificates. The client can also authenticate itself to the server. The client can only authenticate itself to the server if the server requests the client to do so. The process of client authentication can add an extra work load on the server, degrading the server's performance. The certificate verification consists of the process of certificate signature verification. The RSA signature verification need not be an inexpensive operation. The computational requirements for the RSA certificate verification depend upon the size of the public exponent of the RSA key that is used for signing the certificate under consideration. Therefore, care should be taken that the certificates are signed with the RSA public key that does not have a higher public exponent. The extra work load on the server due to client certificate verification can lead to DoS attack. However, the likelihood of such attack is rare. Using certificates to attack the server cannot be safe for the attacker. Since, the certificate contains the identity of the user. However, maliciously created or the stolen certificate can allow the attacker to use the client authentication as the DoS attack vector. In addition, the legitimate users can degrade the server's performance if the client’s certificates require higher time for verification due to
reasons explained so far in the thesis. Therefore, attention should be paid while accepting
the certificate authorities (the public exponent of the key belonging to CAs should not be
higher). In addition, the allowed certificate chain should be kept as short as possible.

The compression boundaries are systematically checked and followed in the OpenSSL
implementation of the SSL protocol. Based on this implementation, there are no signs of a
possibility of creation any buffer overflow attacks on the SSL server side.

The SSL renegotiation based DoS attack outperforms among the other two variants of
the computationally intensive brute force DoS attacks. The other two variants follow the
basic idea of simply opening the SSL connection with the SSL server. One DoS tool does
not use the cryptography and does not send the valid client key exchange message to the
server. Therefore, the server terminates the SSL connection immediately once decryption
of the client key exchange message fails. Another DoS tool makes use of cryptography
and is implemented using the standard OpenSSL library. With respect to the impact on
the victim’s machine, this tool performs better than the tool that does not use cryptography.
This is because, when the SSL handshake terminates just after decrypting the key exchange
message, the server does not have to wait for the cipher spec protocol message and the
finished message from the client. In addition, it does not have to send these two messages
from its side to the client. This saves the (waiting) time for the server and most importantly,
the server saves few cryptographic operations. With these savings, server can serve more
legitimate users.

Therefore, the brute force attack, where the attacker successfully completes the SSL
handshake with the server is more effective than the brute force attack that uses canned
messages and/or does not perform valid cryptographic operations. The most effective
DoS attack on the SSL server remains to be the one that exploits the renegotiation feature
in the SSL protocols. Therefore, the SSL renegotiation should be carefully configured on the
server or should be disabled if not required by the business model.

Finally, this thesis reviewed some of the proposals those aim at improving the SSL server
performance either by shifting the load to the client-side and reducing the load on the server
side or requesting the client to work more before server starts performing expensive opera-
tions. These optimizations have number of limitations and possibility of the DoS attack is
not completely removed. In addition, some of these techniques introduce a new possibilities
of DoS attacks. These possibilities are discussed in this thesis in respective sections.

Therefore, the SSL server should be carefully configured. The impact of the DoS attacks
can be alleviated by employing sufficient bandwidth, processing power and correct usage of
the session cache. When under attack, the network traffic should be actively monitored and
attack sources should be identified according to the some of the techniques discussed in this
thesis.

As a final remark of this research and experiments, there is a need of new protocol or
change in the protocol where the server can change its behavior at run-time; for instance to
request the clients to work more whenever needed.
CHAPTER 11

Acknowledgments

This thesis is written as part of the master project that I have performed at the Department of Mathematics and Computer Science at Eindhoven University of Technology, the Netherlands, in cooperation with Rabobank, Utrecht, the Netherlands. Upon successfully finishing the Master project comes the conclusion of my study master of Science, specialized in the track Information Security Technology at Eindhoven University of Technology.

I have enjoyed working on this project, mainly because of the freedom I was given to explore into the topic of DoS attacks and SSL protocols. I learnt a lot of things not only about the core topic of this thesis but also about a number of applications and technologies associated with it. I express my gratitude to my supervisors. I would like to thank Eric Verheul for initiating this project and giving numerous amounts of new insights and direction along the route. I would like to thank Berry Schoenmakers for supervising my work from Eindhoven University of Technology, for providing great advice and giving magnificent support. I especially want to thank both Eric and Berry for teaching me how to perform a research and various aspects of writing a technical paper.

I would like to thank Henny van de Pavert and Remco Ruiter for providing me all the necessary information, platform and an opportunity to discuss the ideas and giving me all the freedom to perform this research. I would also like to thank Benne de Weger from TU/e and Yorick Koster, David Vaartjes and Pieter Anbeek from Rabobank for their helpful discussions and suggestions. Furthermore, I thank all the folks at Rabobank for the pleasant working and social environment. I would like to specially thank Hans Bielok from Rabobank for motivating me throughout my research.

Finally, a big thanks goes to my friends and fellow colleagues Roberto Lie and Alok Lele for advising me on various aspects of my studies.

Last, but not the least, I would like to thank my parents for their great support and encouragement throughout my study years.
References


REFERENCES


[33] K. E. Hickman, “SSL0.2 PROTOCOL SPECIFICATION.”

[34] “Openssl, cryptography and SSL/TLS toolkit.”


[40] “Server-Gated Cryptography.”


[45] “Option codes.”

 http://zlib.net/ May 2012.


[48] V. Bernat, “SSL-DoS.”


A.1 RSA

This section explains the difference in overhead i.e. why the server needs more computational power in the case of RSA.

During handshake, client encrypts pre-master secret using RSA public key obtained from server’s certificate message. The server decrypts this encrypted message using corresponding private key. These are most expensive operation in handshake when a cipher suite involves RSA as a key exchange algorithm. For instance, TLS_WITH_RSA_AES256_SHA.

From experimental results in this section, it is clear that RSA decryption is more expensive than encryption. Working of RSA is given to understand the difference in RSA encryption and decryption.

Following steps are followed to generate the RSA key pair (public and private keys) in the OpenSSL.

• Select \( e \). Where \( e \) is a public exponent. Default value for \( e \) in OpenSSL is 65535. This can be also changed to 3.

• Choose two distinct prime numbers such that they follow following properties.

\[
1 < e < \Phi(n) \quad (A.1)
\]

And

\[
GCD(\Phi(n), e) = 1 \quad (A.2)
\]

Where,

\[
\Phi(n) = (p - 1) \times (q - 1) \quad (A.3)
\]

• This means, a public exponent \( e \) is a value such that it is co-prime with \( \Phi(n) \).

• A private exponent \( d \) is calculated as follows:

\[
d = e^{-1} \mod \Phi(n). \quad (A.4)
\]

• The public key is represented as \((n, e)\) and private key as \((n, d)\).
• A message \( m \) is encrypted to obtain cipher text \( c \) as follows:

\[
c = m^e \mod n
\] (A.5)

• This cipher text \( c \) is decrypted to obtain \( m \) as follows:

\[
m = c^d \mod n
\] (A.6)

\( d \) is inverse of \( e \) and is normally a big number with no special property and therefore, RSA decryption process is more costly and slow.

A.2 Diffie-Hellman

This section explains the working of DH algorithm and explains the difference in the work overhead between client and server in the process of agreeing on the shared secret. General explanation of DH algorithm can be given as follows. The two communicating parties are treated as the server and the client.

• The server has two public values \((p,g)\). Where, \( p \) is prime modulus and \( g \) is generator. The server also has one secret value \((s)\). The public value of the server \((Y_s)\) is generated as shown in Equation A.7.

\[
Y_s = g^s \mod p
\] (A.7)

The triplet \((p,g,Y_s)\) is sent to the client.

• The client generates a random number \((c)\) and it is kept secret. The client also generates its public value \((Y_c)\) as shown in Equation A.8.

\[
Y_c = g^c \mod p
\] (A.8)

The client sends \(Y_c\) to the server.

• The server and the client calculates the common secret \( S \) as shown in Equation A.9 and A.10 respectively.

\[
S = Y_c^s \mod p
\] (A.9)

\[
S = Y_s^c \mod p
\] (A.10)

• In this way, the server and client obtains the shared secret \( S \) which is known to only the server and the client.

Whenever DH is used as a method of key exchange, the DH parameters are specified by the server. These parameters are included in the server certificate otherwise a separate server key exchange message is sent containing the DH public values.

When the client authentication is requested, the client can either include the DH parameters in the client certificate (if the client can generate the certificate that has suitable client’s DH public value\(^1\)) or send the client key exchange message that consist of the suitable client’s DH public value.

The server and the client generates the pre-master secret as already explained in Equations A.9 and A.10.

In the DH key exchange method, the server need not have to generate new DH parameters. But, the client has to calculate the suitable DH public value as shown in Equation A.8. Therefore, the client performs one extra exponentiation operation while using DH key exchange method when compared with the server.

\(^1\) A suitable DH public value is the one whose parameters (group and generator) match the parameters sent by the server.
B.1 Rate Limiting SSL Handshakes

This section provides the technical details of the ip-table entry B.1. It is originally considered in Section 4.3.

Listing B.1: Heuristic based Rate Limiting SSL handshakes iptable Entry (source: [7]).

A line by line details are as follows:

Line 1 The length of the IP header is given by a 4 bit value. The IP header length field in the IP packet is positioned at 5th bit from the starting of the packet. In this line $ 0 \gg 22 \& 0x3C @ 12 \gg 26 \& 0x3C @$ computes the number of bytes in the IP header and @ makes this the new offset into the (IP) packet. This is the starting of the TCP packet. The length of the TCP header is stored at the left half of the byte 12 of the TCP header. The $ 12 \gg 26 \& 0x3C $ computes the length in bytes. At the end, @ traverse the IP header of that length and makes the end of the byte as a new offset. Finally, payload field on Line [1] gives the offset of the TCP payload that is offset of the SSL data inside the TCP packet.

Line 2 This line is used to set-up the rule given by the described specification.

Line 3 The packet of protocol TCP should be checked.

Line 4 This extension is used to match the TCP packet where only PSH (PUSH) flag is set.

Line 5 This is an extended matching module. The name of the module is given as u32.
Line 6 The module u32 is defined on this line. The payload is already defined on the first line. The $0 >> 8$ selects the first 4 bytes of the TCP payload that is first 4 bytes of the record header and only first three bytes are considered. These three bytes are later checked whether they match $0x160300 : 0x160303$. This pattern stands for if the first three bytes are $0x160300$ (the SSL version 3.0 handshake) or $0x160301$ (the TLS version 1.0 handshake) or $0x160303$ (the TLS version 1.2 handshake). That is, if the first byte is equal to 16 (the SSL handshake message) and if the next two bytes indicates the valid SSL protocol version is checked.

Line 7 A new packet matching module called hashlimit is used.

Line 8 and 9 The hashlimit options are defined. If the SSL renegotiation (as defined in this ip table entry) from the source is more than 5 times per minute (after finding 3 renegotiation packets) then it take action.

Line 10 If the condition defined by the rule matches then DROP the packet.

This is the simple IP table entry as a work-a-round for limiting SSL renegotiation and hence avoid the DoS due to computational exhaustion caused due to repeated SSL renegotiation requests. This IP table entry is taken from [7].

B.2 Wireshark Screen Shots

B.2.1 Compression Method Support in Browsers

Figure B.1 shows Client Hello message as produced by the chrome browser. It can be observed that the Client Hello message of the Chrome browser includes two compression methods. One method is the standard and mandatory compression method `compression.null` as specified by the SSL specifications [31, 30, 32, 49]. The other compression method is `DEFLATE` compression method.

![Wireshark Screen-shot: Compression Support in Chrome Browser](image)

Similar tests were performed with other standard browsers such as Mozilla Firefox, Windows Internet Explorer and Safari. it has been found that these browsers only supports the `compressionMethod.null` method and no other compression method.
B.3 OpenSSL command line options

B.3.1 Obtaining RSA key length of the Target

The OpenSSL [8] has the application called s_client to emulate the SSL client. This client can be used to connect to and communicate with the SSL server. This program has several options. One of them is `connect`. One of the example usage is given in Listing B.2

```
OpenSSL s\_client connect www.example.com:443
```

Listing B.2: Openssl s_client usage (source: [8]).

This command gives the output with connection details and bytes exchanged during the SSL handshake. This output also mentions the size of the RSA public key of the server. For instance, the example given in Listing B.2 will give the public key of the server residing at the address `www.example.com`.

B.4 CA path-length constraint

Figure B.2 highlights that the "path length" constraint under the "basic constraint" extension of the certificate is set to `NONE`. This indicates that virtually unlimited number of certificates can exists between this certificate and the end entity certificate. Please note than this is a CA certificate. This is just an example of the certificate that indicates that the `"path length"` constraint can be left blank that could later cause the DoS attack mentioned.

![Certificate Path Length Constraint](image)

Figure B.2: The basic constraint "path length constraint" is set to none.
in this thesis. However, if the certificate issued by this certificate has definite value for "path length" field, then the certificate path length starting with issued certificate is automatically restricted to the value of the path length.