# Chapter 1

# Problem Statement and Introduction

Problem Statement – Giving proof of concept of major known vulnerabilities in SSL

Protocol and designing SSL Vulnerability Checker.

* Problem Definition

Proof of concept – “The proof of concept is a demonstration or prototype designed to check the feasibility of theoretical concept in real world given reasonable resources”.

In this project we are going to present proof of concept of major known vulnerabilities in SSL protocol (Protocol which is widely considered as part of application layer in TCP/IP model. One of such vulnerabilities is POODLE aka Padding Oracle on Downgraded Legacy Encryption.[1]

* Motivation

This vulnerability caused the SSL protocol to be widely abandoned. SSL protocol is one of the widely used protocol around the networks over the world. Most of the time this vulnerabilities are published in research papers which only demonstrate theoretical feasibility of concept and hence people usually are reluctant to upgrade their systems in order to mitigate the vulnerability but presenting proof of concept of any such vulnerability gives it enough ground to be considered as major vulnerability and software developers around the world then work in direction to cast aside the protocol to find the better alternative.

Also doing work on cryptographic protocols may bring to notice any hidden vulnerabilities and might give ideas to new cryptographic protocols.

* Objectives

1. To present prototypes to imitate major vulnerabilities in SSL protocol.
2. Prototypes must be computationally feasible.
3. Designing general purpose vulnerability detection software on secure server employing SSL/TLS protocols.

* Introduction

SSL protocol also known as Secure Socket Layer a protocol used for securing the traffic on the network. It is a layer protocol consisting of two layers –

1. SSL Record Layer
2. SSL Handshake Layer

This protocol is never published officially by Internet Engineering Task Force (IETF) an organization which produces official standards for internet protocols like SSL, HTTP, FTP, etc. But IETF produced historical document in 2011 [1] which was the draft of the standard which we are going to use throughout our project as standard for SSL.

The last version of SSL was 3.0 [2]. SSL protocol uses asymmetric cryptographic algorithms in handshake layer to establish the secure connection based on master secret shared by both client and the server while it uses symmetric cryptographic algorithms for application data encryption in record layer [1].

As we know symmetric cryptographic algorithms are of two types. One is stream cipher and other is block cipher. [3] SSL protocol supports RC4 stream cipher and block ciphers such as RC2, DES, 3DES, etc. But over the time many vulnerabilities have been discovered in SSL such as POODLE, CRIME, FREAK, etc. [4] this was primarily due to the information leak through various ways. Also this symmetric block ciphers are used in CBC aka cipher block chaining mode within SSL protocol which are vulnerable to padding oracle attacks. [5]

Again in case of attacks like CRIME aka Compression ratio info-leak made easy the information leak occurs through the data compression. Any protocol not only SSL but also other cryptographic protocols can be broken if they have some kind of information leak. [6]

Most of these vulnerabilities are presented in research papers authored by notable computer scientists, cryptologists and cryptanalyst. But there are none or very few practical proof of principle/concept given for these attacks. So the aim of our project is to give proof of concept for some of the major attacks like POODLE which can be theoretically implemented using feasible amount of resources and can be very effective attacks against the protocol if proved to be feasible in practice.

As auxiliary project to this we are creating a Vulnerability checker software which will scan the webserver for HTTPS port and then will perform tests on that port to check if that port is open and vulnerable for attacks then it will generate report of possible risks in server and the software will be designed in a way such that it will be modular and can be updated easily to add new vulnerabilities to be checked.

# Chapter 2

# Literature Review

* RFC 6101 – A standard draft of SSL v3.0.

URL - <https://tools.ietf.org/pdf/rfc6101.pdf>

Author - IETF

Summary – It basically explains all the micro details required for implementing SSL protocol but at the same time it is helpful to observe the weak points in protocol implementation

* POODLE BITES AGAIN -

URL - <https://www.openssl.org/~bodo/ssl-poodle.pdf>

Author – Bodo Moller, Thai Duong, Krzysztof Kotowicz

Summary – It explains beautifully what is POODLE attack, how to mitigate it, how to implement it. It is first paper related to POODLE. Even then theoretical explanation is quite simple to read and grasp

* Here comes the XOR Ninja –

URL - <http://netifera.com/research/beast/beast_DRAFT_0621.pdf>

Author - Thai Duong, Juliano Rizzo

Summary – This paper in details discusses methods to implement the BEAST attack and theoretical background behind it.

* Analysis of SSL 3.0 protocol

URL - <https://www.schneier.com/academic/paperfiles/paper-ssl.pdf>

Author - David Wagner, Bruce Schneier

Summary – Explores the replay attacks and version rollback attacks

* Basic Cryptography and Modular Arithmetic Courses

URL – <https://www.khanacademy.org/computing/computer-science/cryptography>

Author - Khan Academy Instructor

Summary – Explained basics of cryptography, modular arithmetic and Diffie

Hellman key exchange, RSA algorithm, One Time Pad in great detail

# Chapter 3

# Proposed Approach

* For proof of concept

1. The vulnerability which we found most feasible theoretically is POODLE. So in order to implement the POODLE vulnerability we need to be in the middle of connection between client and server. We will achieve this through either creating plain HTTP site which user will open while he will be at same time access some secure HTTPS site.[1]

Another way to do this will be using some libraries such as winpcap (On windows), libpcap (On Unix), etc.

1. Once we are in the man in middle position we can easily send the server requests which will be locally send from user’s browser using JavaScript. In this way secure site won’t recognize requests and we can easily infiltrate into the secure session of client.
2. Now we can start the padding oracle attack assuming downgrade dance has been successful (We are working on getting this done. As most browsers now try to mitigate the downgrade dance it has become tougher. But anyways this is not important task in proving proof of concept of POODLE.)
3. Now in padding oracle attack we will change the bytes of ciphertext blocks and guess the new byte and try sending it. Of all 255 out of 256 time the padding will be invalid and connection will be closed but at one instance connection will be established and we have successfully guessed one correct byte out of plaintext.
4. In this manner we can get all the bytes of that block and we can successfully get that block through javascript again. Even though one byte is small. The session token are often only few bytes long and then this become computationally very cheap to get highly sensitive data.
5. Furthermore one after 255 attempts is scenario of worst case in reality it will be much shorter because our guess will be random and due to probability being equally likely for all 256 characters we are bound to get better results.

* For Vulnerability Checker

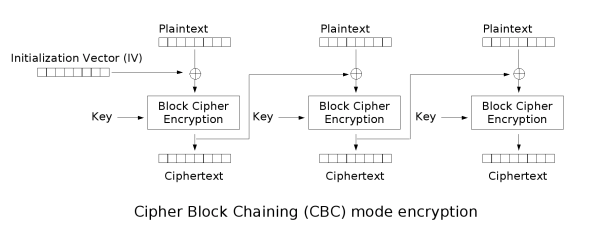
1. We will scan for the HTTPS service on the URL given to program. If it is open then we will go to next step
2. We will try to initiate connection and obtain initial header information about SSL version running on that site. Depending on which protocol version it is using it will be easy to check for selected vulnerability.
3. This program can also be designed in modular way as we are just checking header we can use database to store corresponding vulnerabilities to corresponding protocol implementations.

Theory behind the Padding Oracle Attack -

Padding oracle means a theoretical machine which can tell us for sure whether a given ciphertext is properly padded or not. In our case when connection will drop that means our theoretical deterministic oracle gave answer “NO” and when connection is established that means our theoretical deterministic oracle gave answer “YES”.

More particularly CBC mode of block ciphers are vulnerable to this kind of padding oracle attack. CBC (Cipher Block Chaining) mode works in the following way.

* Encryption –

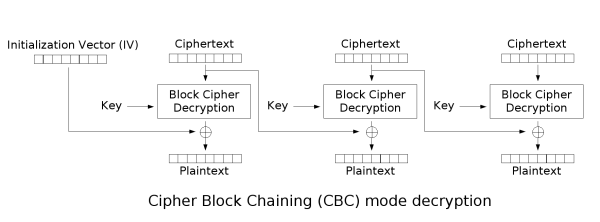


Here IV is considered C0.

Equation used for encryption are -



* Decryption -



Here IV is considered C0

Equations for decryption are –



Method to follow to launch attack –

1. We change the last byte of the ciphertext block former to our block of interest.
2. This causes paintext of former block to change completely due to avalanche effect. But only last byte of plaintext block corresponding to our ciphertext block will change.
3. This can be then checked for padding and byte can be guessed in worse condition after 255 tries (but this is still less )
4. This procedure can be followed to get subsequent bytes.

# Chapter 4

# Workflow and Schedule of Work

Work Done till Date -

1. Devised method to launch POODLE attack.
2. Designed some basic primitives to test in other vulnerabilities. The primitives include integer factorization of integers of size of 10^18.
3. All implementation of SSL is well studied.
4. Written Website to be used for POODLE attack.

Work Schedule –

1. Within next three weeks – Implement the POODLE primitives and setup server for attack. Solve the problem of downgrade dance.
2. After next three weeks – Implement the vulnerability checker software.
3. If time remains explore some other vulnerabilities to make them feasible.

# Chapter 5

# References

[1] <https://www.openssl.org/~bodo/ssl-poodle.pdf>

[2] <https://tools.ietf.org/pdf/rfc6101.pdf>

[3] <https://en.wikipedia.org/wiki/Symmetric-key_algorithm>

[4] <https://wolfssl.com/wolfSSL/security/vulnerabilities.php>

[5] <https://en.wikipedia.org/wiki/Padding_oracle_attack>

[6] <https://en.wikipedia.org/wiki/Information_leakage>

[7] <http://netifera.com/research/beast/beast_DRAFT_0621.pdf>

[8] <http://cve.mitre.org/cgi-bin/cvename.cgi?name=cve-2012-4929>

Other Reading References –

 <https://en.wikipedia.org/wiki/Transport_Layer_Security>

 <https://en.wikipedia.org/wiki/Symmetric_cryptography>

 <https://en.wikipedia.org/wiki/Ciphertext>

 <https://en.wikipedia.org/wiki/Non-repudiation>

 <https://en.wikipedia.org/wiki/Key-agreement_protocol>

 <https://en.wikipedia.org/wiki/Pre-shared_key>

 <https://en.wikipedia.org/wiki/Semantic_security>

 [https://en.wikipedia.org/wiki/Key\_(cryptography)](https://en.wikipedia.org/wiki/Key_%28cryptography%29)

 <https://en.wikipedia.org/wiki/Digital_signature>

 <https://en.wikipedia.org/wiki/Stream_cipher>

 <https://en.wikipedia.org/wiki/Block_cipher>

 <https://en.wikipedia.org/wiki/Public_key_infrastructure>

 <https://en.wikipedia.org/wiki/Web_of_trust>

 <https://en.wikipedia.org/wiki/Data_Encryption_Standard> (DES)

 [https://en.wikipedia.org/wiki/RSA\_(cryptosystem)](https://en.wikipedia.org/wiki/RSA_%28cryptosystem%29) (RSA)

 <https://en.wikipedia.org/wiki/Digital_Signature_Algorithm> (DSS)

Suggestions By board Members