Survey on Oracle Padding Attacks on Cryptographic Protocols

Anab Leila Abdi
CIISE, Concordia University
Montreal, CA

Tomas Navarro Munera
tomas.navarro@mail.concordia.ca
CIISE, Concordia University
Montreal, CA

Jainil Shah
jainil.shah@mail.concordia.ca
CIISE, Concordia University
Montreal, CA

Bergen Davis
CIISE, Concordia University
Montreal, CA

Diego Jaramillo
CIISE, Concordia University
Montreal, CA

Md Ashfaque Rahman
CIISE, Concordia University
Montreal, CA

Alcibiades Rogelio Pumajulca
alcibiades.pumajulcasalazar@mail.concordia.ca
CIISE, Concordia University
Montreal, CA

Shrutibahen Rana
shrutibahen.rana@mail.concordia.ca
CIISE, Concordia University
Montreal, CA

Juan Soto Chourio
CIISE, Concordia University
Montreal, CA

***Abstract*—Cryptographic protocols play an essential role in protecting information across diverse applications and systems. These systems range from finance, e-commerce, transportation and shipping. The integrity and reliability of these protocols are crucial for safeguarding the confidentiality of the data they facilitate. While modern-day protocols are generally effective in securing systems, there are instances where they can fall victim to malicious actors. These actors employ specially crafted techniques and tools to exploit various design flaws within the definition of the protocols libraries, its implementation etc, effectively bypassing the security mechanisms of cryptographic protocols.**

The Oracle Padding Attack, along with its variants, serves as an example of assaults on cryptographic protocols, padding oracle attacks enable the retrieval of either partial or full content of the original plain-text from encrypted messages. These attacks also impact the prevalent operational method of contemporary cryptographic protocols, particularly the cipher block chaining (CBC) mode. Consequently these attacks have the potential to compromise nearly all online communication channels that rely on such protocols for security In this survey report, we investigate and analyze the Oracle Padding Attack and its variants, along with several other attacks such as CRIME, BREACH, POODLE, DROWN, and BEAST. Our investigation delves into the background information of these attacks, their implementation details, and effective solutions against them

***Index Terms*—Cryptographic protocols, Oracle Padding Attack, systems, confidentiality**

I. Introduction

In today's modern society, a multitude of highly important operations are carried out over the internet or some electronic medium. This dependency on the digital era has correlated with an increase in cyber crimes across the technological industry. Therefore, when communicating, there is a demand for the confidentiality and integrity of components and properties of sent messages over these channels. In the case of the internet, the TLS/SSL protocol has been the conventional standard to secure communication. However, this protocol, among others, is not fully resistant to penetration. Actors can use specific implementations to penetrate and retrieve full or partial recovery of original plain-text messages. These attacks take advantage of unintended side channels revealed by cryptographic protocols. Consequently, they create an oracle that allows making inferences about the underlying plain-text by utilizing easily predictable padding bytes. This is why they are referred to as padding oracle attacks [1]. As a consequence, SSL/TLS protocols are not impervious to oracle attacks.

The SSL/TLS protocol has seen a significant change in its security and overall architecture over the last few years, culminating in its termination in 2015. The protocols were rolled out into the market in the early 1990s. However, as time progressed, the protocols were plagued by their insecure design and numerous vulnerabilities. Subsequently, the TLS standard was introduced as the successor to SSL, iterating over versions TLSv1.0 until today's current standard of TLS1.3, which has mitigated vulnerabilities that plagued its predecessors [2]

As the SSL/TLS version underwent enhancements for increased security, the Oracle padding attack progressed in sophistication and diversified into various forms. This survey delves into the Oracle padding attack and its multiple variants, presenting an examination of five attacks on the SSL/TLS protocol, encompassing BEAST, POODLE, DROWN, CRIME, BREACH, and LUCKY13.

II. Background

The oracle padding attack, introduced in the early 2000s, has proven to be an intricate attack that operates by utilizing the oracle padding algorithm to carefully examine the integrity of padding on plain-text when given any cipher-text. The padding oracle algorithm was designed to identify cases of valid padding. It operates on the condition that it returns a positive result if the padding is correct; however, it will return a negative result if incorrect. As is common with many standardized formats, this introduces a vulnerability for malicious actors to apply investigative techniques to determine the padding byte of a specific message, which directly results in the collapse of the system and the exposure of the plain-text message [3].

The oracle padding attack is a key attack within the realm of cryptographic attacks, showcasing its prominence by its ability to exploit a server's behavior during the decryption process of messages. Attackers are able to deduce sensitive information through side channel data such as prolonged decryption cycles or error code displays.

In the case of the oracle padding attack, we have designed a scenario where the attacker retrieves the cipher-text through his participation in a man-in-the-middle attack. The server functions as a padding oracle, validating the padding and issuing error messages when incorrect padding is detected. The attacker will perform a series of tests on each byte of the cipher-text, iterating through the set of 0 - 255 until the server validates the padding. Once successfully validated, the attacker will perform XOR computations to reveal the last byte of the plain-text. This process is repeated iteratively to reveal preceding bytes in each instance [3]. The vulnerability within the padding oracle attack underscores the need for well-designed protocols that are regularly revised to protect against security infractions.

A. *Padding Oracle History and Emergence of Variants*

The earliest version of the padding oracle attack can be traced back to 1998, demonstrated in Bleichenbacher's attack, an adaptive cipher-text attack that victimizes RSA with PKCS 1 v1.5 padding [4]. However, this attack was inefficient and ineffective. This improved in 2002 when Vaudenay made a significant breakthrough in symmetric cryptography, later known as the padding oracle attack [5]. Over the course of the next decade, many faults were discovered in the implementation of security protocols such as SSL and TLS. Regrettably, these weaknesses provided opportunities for cyber assaults to pilfer vital user credentials. Notably, widely-used encryption methods such as CBC are particularly susceptible to such breaches. Surprisingly, the robustness of the encryption doesn't consistently offer protection, as certain attacks can prevail even without the attacker having knowledge of the encryption key. Neglecting this led to the promotion of several variants of the oracle padding attack. These variants include BREACH [6], POODLE, ROBOT[7], LUCKY13, BEAST, and CRIME [8].

B. *Overview of Cryptography*

In this section, we provide an overview of fundamental cryptography concepts and terminology to assist readers in understanding attacks more comprehensively.

Encryption and Decryption: Encryption is the process of converting plain-text information into a secret code (cipher-text) to prevent unauthorized access. It is a crucial technique in cryptography for securing sensitive data during storage or transmission. Plain-text messages, denoted as 'm', can be sequences of bytes of any length (m E 0, 18n). The corresponding cipher-text, denoted as 'c', consists of byte sequences that are multiples of 'b', known as the block size. During encryption, a mathematical algorithm (encryption algorithm 'E') and a secret key 'k' are used to transform plain-text into cipher-text. An example of the encryption process is as follows:

E(m, k) = c.

Decryption involves the reverse process of converting encrypted cipher-text back into its original and readable form (plain-text). It is the inverse operation of encryption and requires the use of a key or algorithm to transform the cipher-text back into its original content. During decryption, a mathematical algorithm (decryption algorithm 'D') and the secret key 'k' are used to convert the cipher-text into plain-text. An example of the decryption process is provided below.

D(c,k) = M

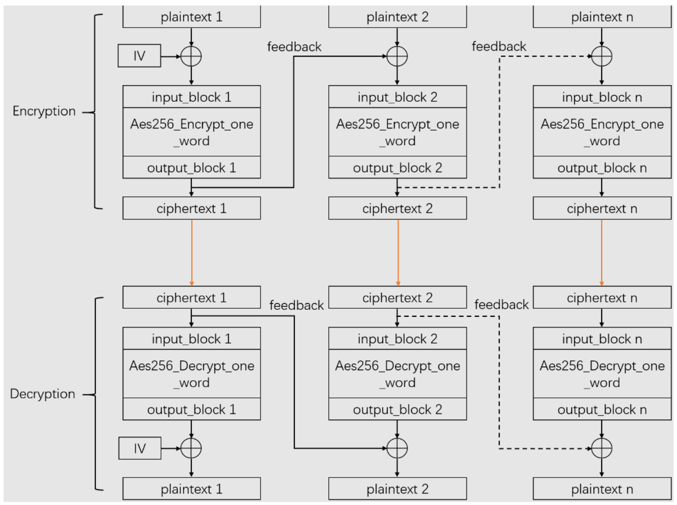

Fig. 1. Overview of CBC Encryption and Decryption [?]

C. *SSL/TLS*

The vulnerability leading to maximum padding oracle attacks stems from flaws in the implementation of SSL or TLS protocols. Let's delve briefly into these cryptography protocols:

SSL (Secure Sockets Layer) and its successor, TLS (Transport Layer Security), were devised to fortify internet communications by ensuring privacy, authentication, and data integrity.

Introduced by Netscape in 1995, SSL aimed to counter the risk of transmitting data in plain-text over the web, vulnerable to interception and unauthorized access.

SSL and TLS are pivotal in securing online transactions and communications. Websites utilizing SSL/TLS display URLs starting with ”HTTPS” instead of ”HTTP,” indicating a secure connection. These protocols encrypt transmitted data, rendering it unreadable to unauthorized entities. This encryption is especially crucial in safeguarding sensitive information like credit card details from interception and theft.

A significant feature of SSL/TLS is the authentication process initiated through a handshake between communicating devices. This handshake verifies the identity of both the client (user's device) and the server. SSL/TLS also employs digital signatures to ensure data integrity, affirming that transmitted information remains unaltered during transit.

SSL evolved into TLS over time, with the Internet Engineering Task Force (IETF) proposing updates in 1999. This transition in naming marked a change in ownership and development. Though SSL and TLS are often used interchangeably, TLS is acknowledged as the more contemporary and secure version. [30] TLS serves three primary functions: encryption, authentication, and integrity. Encryption shields data from unauthorized parties, ensuring confidentiality. Authentication validates the identities of involved parties, thwarting unauthorized access. Integrity ensures that transmitted data remains unchanged during communication. The TLS handshake, pivotal to the protocol, establishes a secure connection between the client and the server. It involves specifying the TLS version, selecting cipher suites for encryption, authenticating the server through its TLS certificate, and generating session keys for secure communication. [30]

Public key cryptography is instrumental in TLS, facilitating secure key exchange over an unencrypted channel. The server's public key, part of its TLS certificate, is utilized in authentication. Once data is encrypted and authenticated, it undergoes further protection via a message authentication code (MAC), ensuring data integrity akin to a tamper-proof seal on a product.

It is noted that, TLS-protected HTTPS has become standard practice for websites, with major browsers reinforcing security measures by highlighting non-HTTPS sites. The adoption of TLS encryption remains pivotal in shielding web applications from data breaches and cyber threats. [30]

III. The Padding Oracle Attack

The Padding Oracle attack exploits weaknesses in modern cryptography protocols and systems. The attack capitalizes on differences in error messages that occur during the decryption of cipher-texts. Through the investigation and analysis of these errors, malicious attackers can deduce critical data surrounding the validity of padding, allowing them to decrypt and compromise the security of the encryption process. The existence of this attack sheds light on the obstacles and challenges in the realm of securing encrypted communication systems and

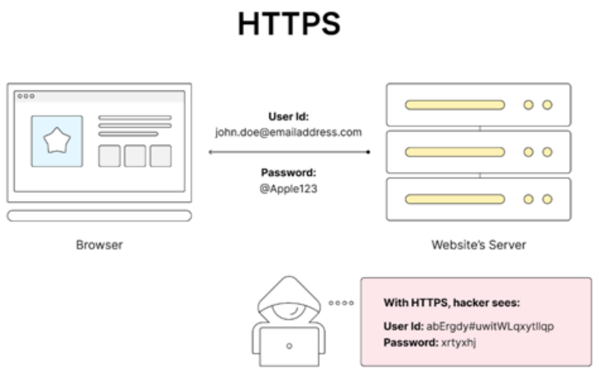

Fig. 2. Overview of HTTP/S [29]

showcases the importance of robust cryptography practices. [9]

A. Prerequisites for the attack

The success of the Padding Oracle Attack hinges on meeting specific conditions. Typically, the attacker needs access to encrypted communication and the Oracle. This access enables the attacker to discern the validity of the padding generated by the current plain-text. With this crucial information, the attacker can decrypt and reveal the plain-text, regardless of the encryption's strength or the selection of robust keys. [9] The prerequisites for executing the padding oracle attack encompass:

- Availability of an Oracle: Successful execution requires the attacker to have access to a padding oracle, a mechanism indicating whether the decrypted message's padding is correct. This could manifest as subtle error messages or other detectable behaviors.
- Cipher Block Chaining (CBC) Mode: This attack is particularly effective when encryption employs Cipher Block Chaining (CBC) mode, leveraging the interconnection of blocks to exploit padding.
- Familiarity with Padding Schemes: Understanding the padding scheme used in the cryptography protocol is essential for the attacker. Various schemes exhibit distinct error patterns that can be exploited. In our demonstration, we'll utilize the PKCS7 padding scheme.

B. Attack Analysis

The Padding Oracle Attack can target symmetric encryption using CBC mode, which employs the PKCS7 padding scheme. Initially, we'll delve into the mathematical principles underpinning the attack. Then, we'll explore the step-by-step decryption process for one byte, followed by subsequent bytes and blocks. Our approach assumes that we can submit any cipher-text to the server, which will indicate whether the decryption yields plain-text with valid padding.

a) Mathematical Foundation: In Cipher Block Chaining (CBC) encryption, each plain-text block undergoes XOR (exclusive OR) with the preceding cipher-text block before entering the cipher. Consequently, during CBC decryption, each cipher-text undergoes decryption by the cipher and subsequent XOR with the preceding cipher-text block to reveal the original plain-text. To decrypt a specific plain-text block Pn, we require both cipher-text blocks Cn and Cn -1 . For instance, to unveil P2, we need access to C2 and C1. In practice, C2 is initially transformed into an intermediate representation, I2, through the block cipher decryption function and the key.

The Padding Oracle attack operates by computing an temporary intermediate representation for each of the cipher-text. This representation holds significance as it facilitates decryption. Understanding I2 is crucial because once we have it, we can derive P2 using the following transformation:

I2 = C1 XOR P2

This equation implies:

P2 = I2 XOR C1 (1)

This equation serves as a compelling motivation to uncover the intermediate representations, which ultimately facilitate complete decryption. In our scenario, since C1 is already known as part of the cipher-text, determining I2 will enable us to readily deduce P2.

b) Computation of the Last Byte:

IV. Variants of the Attack

A. Padding Oracle Attack Against the SCP02 Protocol

The Secure Channel Protocols (SCP) specified by GlobalPlatform aim to secure communication between smart cards (or other secure elements) and external entities (off-card entity) like card readers and servers, some of these specifications are: SCP01, SCP02, SCP03, SCP04, SCP10, SCP11. In this section, the focus is on SCP02 which until today is used and improves upon SCP01 by introducing counter measures against replay attacks and providing more robust security features. SCP02 (based on 3DES) uses dynamic session keys derived from a base key and counters.

Section Outline: Part 1 describes the SCP02 protocol in detail. Part 2 describes the regular padding oracle attack. Part 3 details how to use the Padding Oracle Attack against SCP02. Part 4 specifies the settings and experimental results. Part 5 lists counter measures and conclusion of the section.

Notation: A byte value is written as 7E. The value 00^i corresponds to the byte string made of i bytes equal to 00. $b_i|b_{i+1}$ refers to the concatenation of the bytes b_i and b_{i+1}. C || B refers to the concatenation of the two DES blocks C and B. ENC indicates a symmetric encryption function, while ENC^{-1} indicates the corresponding decryption function.

1) GlobalPlatform SCP02 protocol:
   The protocol SCP02 is based on symmetric-key algorithms, and it aims at establishing a secure link between an "off-card entity" and a card [23]. The card and the party involved in the communication with the card (from now on "the server") share one or several sets of symmetric keys. Three keys make a set, from this set the session keys are computed every time a new channel is established. The card manages a sequence counter related to a given keyset with an initial value of 0, this value is incremented after each successful session. After the sequence number reaches its maximum value, the card should not start a session with that keyset. The commands sent by the server and the responses sent by the card are encrypted and protected with a MAC tag; although, this depends on the security level negotiated during the key exchange. The lowest security level is data integrity (regarding the commands) and only data encryption in not allowed. The plaintext is padded with a fixed string of bytes [25] where a byte equal to 80 is appended to the plaintext, then it's added as many null bytes as necessary, so the string length is multiple of a DES block. For data encryption is used 3DES in CBC mode with a null IV [24]. The MAC (8 byte) tag is computed on the command header, the plaintext, and a padding data. From figure 3 taken from [26]: The genuine header HDR can be retrieved from the header HDR' of the encrypted command. Besides, IV used for the next command is equal to the MAC tag computed on the previous command. The ciphertext and the MAC tag become then the data field of the server's command.

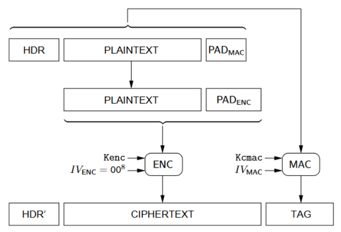

Fig. 3. Encryption and MAC computation of a command data with SCP02

2) Padding oracle attack:
   In Algorithm1 the attack is described, and it is assumed that the targeted block C includes at least one byte of padding, i.e., 80. When the plaintext (to retrieve) corresponds to more than two blocks $B_0, ..., B_{k-1}, k > 2$, and the encrypted blocks are $C_0 ... C_{k-1}$ after the attacker applies Algorithm1 to each pair of encrypted blocks $(C_{k-2}, C_{k-1}), (C_{k-3}, C_{k-2}) ... (C_0, C_1), (C_{-1}, C_0)$ where C_{-1} is the CBC IV (equal to 00^8 in SCP02). . .
3) Attack applied to SCP02:
4) Settings and Results
5) Countermeasures and Conclusion

B. Breach

C. POODLE Attack

The POODLE attack, short for Padding Oracle On Downgraded Legacy Encryption, targets a vulnerability within the SSL protocol, designated CVE-2014-3566 [12], with the aim of decrypting confidential data. Specifically, it seeks to compromise sensitive information like login details and encrypted exchanges between users and websites. [13] In the following section, we offer a comprehensive examination of this attack's mechanics and impact. Subsequently, we delve into the intricacies of the identified SSL cryptography weakness. Concluding our discussion, we outline our approach to implementation and suggest effective strategies for mitigating the risks posed by the POODLE attack.

*1) **Overview:*** SSL, or Secure Sockets Layer, stands as a cryptography protocol with the primary purpose of ensuring secure communication across networks. Its role encompasses safeguarding the confidentiality and integrity of data exchanged between a client (such as a web browser) and a server. Although SSL 3.0 is an antiquated and insecure protocol, it has largely given way to more robust alternatives like the Transport Layer Security (TLS) protocol. Various iterations of TLS, including TLS 1.0, TLS 1.1, and TLS 1.2, retain compatibility with SSL 3.0 to facilitate seamless interaction with older systems and maintain user experience continuity. Consequently, many TLS clients adopt a mechanism known as a "protocol downgrade dance" to navigate around server-side interoperability issues when dealing with legacy systems. [14] During the initial handshake, the client presents the highest protocol version available. If this attempt fails, subsequent efforts, potentially repeated, involve trials with earlier protocol versions. This downgrading process differs from standard negotiation, where a server may respond to a client's proposal of a higher version with a lower one. However, in the downgrade dance scenario, downgrading can result not only from negotiation but also from network disruptions or deliberate interference by malicious actors. Consequently, attackers may confine communication to SSL 3.0. Once the protocol is successfully downgraded, the attacker gains the ability to decrypt the SSL session content, executing decryption on a byte-by-byte basis, often leading to the establishment of numerous connections between the client and server. [15]

The US-CERT has acknowledged a design flaw discovered in how SSL 3.0 manages block cipher mode padding. The POODLE attack serves as a demonstration of how attackers can exploit this vulnerability to decrypt and extract data from encrypted transactions. [20]

The vulnerability within SSL 3.0 arises from the encryption process of data blocks using a specific encryption algorithm within the SSL protocol. The POODLE attack capitalizes on the protocol version negotiation functionality embedded in SSL/TLS to compel the usage of SSL 3.0, subsequently exploiting this newfound vulnerability to decrypt specific content within the SSL session. This decryption occurs incrementally, byte by byte, leading to the establishment of numerous connections between the client and server. [20]

Despite SSL 3.0 being an outdated encryption standard and largely superseded by TLS, most SSL/TLS implementations retain backward compatibility with SSL 3.0 to facilitate interaction with legacy systems and uphold a seamless user experience. Even in cases where both client and server support a version of TLS, the SSL/TLS protocol suite allows for negotiation of protocol versions (commonly referred to as the "downgrade dance" in other discussions). The POODLE attack exploits the tendency of servers to revert to older protocols like SSL 3.0 when a secure connection attempt fails. By inducing a connection failure, an attacker can coerce the use of SSL 3.0 and proceed with the new attack. [20]

To effectively carry out the POODLE attack, two additional prerequisites must be fulfilled:

- Firstly, the attacker needs control over segments of the SSL connection's client side, enabling manipulation of input length.
- Secondly, the attacker must gain access to the resulting cipher-text. Typically, achieving these conditions involves adopting a Man-in-the-Middle (MITM) stance, which necessitates a distinct method of attack to acquire such access.

These requirements introduce complexity to the successful exploitation of the vulnerability. Environments predisposed to MITM attacks, like public WiFi networks, mitigate some of these obstacles.

*2) **Details of POODLE Attack (CVE-2014-3566):*** SSL 3.0 employs either the RC4 stream cipher or a block cipher in Cipher Block Chaining (CBC) mode for encryption purposes. However, it's essential to note that RC4 is known to exhibit biases. These biases imply that when the same secret, like a password or HTTP cookie, is transmitted across multiple connections and encrypted with different RC4 streams, increasing amounts of information will gradually become exposed. [15]

An issue of paramount concern with CBC encryption within SSL 3.0 arises from its block cipher padding, which lacks determinism and remains outside the purview of the Message Authentication Code (MAC). Consequently, during decryption, it becomes challenging to ascertain the integrity of the padding entirely. [15]

The exploit becomes most viable when encountering a padding block consisting of L-1 arbitrary bytes followed by a single byte with a value of L-1 (where L represents the block size in bytes). Upon processing an incoming cipher-text record $C_1, C_2, ..., C_n$ (with each C_i representing one block) and given an initialization vector C_0, the plain-text initially appears as $P_1, P_2, ..., P_n$ where $P_i = DK(C_i) \oplus C_{i-1}$ (with DK indicating block cipher decryption using the key K). Following this, the recipient conducts a validation and removal process for the padding, subsequently eliminating the MAC post-verification. Notably, the MAC size in SSL 3.0 CBC cipher suites typically amounts to 20 bytes. [15] As a result, beneath the CBC layer, an encrypted POST request assumes the following structure:

$$POST\ /path\ Cookie : name = value...body||20\text{byte MAC}||\text{padding}$$

For executing the POODLE attack, the perpetrator positions themselves as an intermediary between the victim and the server. By intercepting and altering the SSL records transmitted by the browser, the attacker strategically manipulates them to increase the likelihood of server acceptance without rejection. Once the modified record gains acceptance, the attacker gains the capability to decrypt a single byte of the message, such as cookies. With control over both the request path and body, the attacker orchestrates requests to fulfill two specific conditions [16]:

- The padding fills an entire block (encrypted into C_n)
- The cookies' first unknown byte appears as the final byte in an earlier block (encrypted into C_i).

Afterward, the intruder substitutes C_n with C_i and dispatches the altered SSL record to the server. Typically, the server will deny this record, prompting the attacker to retry with a fresh request. Occasionally, approximately once in every 256 requests, the server will approve the altered record. At this juncture, the attacker deduces that $DK(C_i)[15] \oplus Cn - 1[15] = 15$, signifying that $P_i[15] = 15 \oplus Cn - 1[15] \oplus Ci - 1[15]$. Consequently, the initial undisclosed byte of the cookie is uncovered, prompting the attacker to proceed to the subsequent byte by adjusting the dimensions of the request path and body concurrently, ensuring that the request size remains consistent while the header's position shifts. This iterative process persists until the attacker decrypts the desired portion of the cookies. [15]

*3) **Systems Affected**:* Every system and software employing Secure Socket Layer (SSL) 3.0 with cipher-block chaining (CBC) mode ciphers could potentially face vulnerabilities. Nonetheless, the POODLE (Padding Oracle On Downgraded Legacy Encryption) attack highlights this vulnerability predominantly within the domain of web browsers and servers, presenting a common avenue for exploitation.

Furthermore, certain implementations of Transport Layer Security (TLS) are also susceptible to the POODLE attack.

*4) **Implementation**:* The architecture of our implementation is depicted in Fig. 4. In this setup, the attacker assumes the role of a middleman situated between the victim's browser and the HTTPS server. Within the victim's browser, the request generator operates, implemented as a static JavaScript file named *(POODLEClient.js)*. The HTTP server hosts the static request generator code and responds to requests from the generator concerning the parameters of the generated HTTPS requests directed to the target server under attack. For this purpose, the Python module *http.server* was utilized. The TLS Proxy intercepts and modifies the TLS packets generated by the HTTPS requests and monitors the responses from the server. Implementation-wise, the module *socketserver* was employed for the TLS proxy functionality. Given that the TLS proxy and the HTTP server operate on different threads, Python classes such as Manager and BaseManager from *multiprocessing.managers* were leveraged to instantiate objects accessible from various threads and even remotely. [17]

To execute the POODLE attack, we proceeded with the subsequent steps. Initially, we downgraded the TLS protocol to SSL 3.0 by interfering with the TLS handshake process. Subsequently, we augmented the size of the POST body by one byte until the ciphertext expanded by a block size. This adjustment was made to align the URL and POST length in a manner that the final block of the ciphertext constituted padding. Ultimately, we conducted a copy operation on each generated TLS packet and determined the leaked byte upon the server's acceptance of the modified packet. [17]

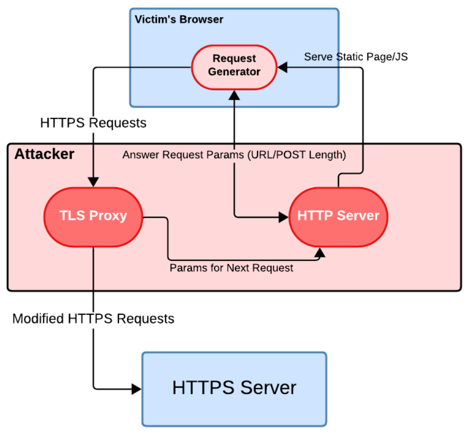

Fig. 4. POODLE Attack Implementation Architecture

As the latest iterations of Mozilla Firefox and Google Chrome remain secure, we opted to set up Firefox v37.0.0 in our testing environment, which was Kali Linux. Additionally, to render the browser susceptible to the POODLE attack, configuration adjustments were necessary, specifically involving enabling SSL downgrade, achieved in the following manner:

$$security.tls.version.min = 0$$

$$security.tls.version.max = 0$$

$$security.tls.version.fallback - limit = 0$$

$$security.ssl3.*\_rc4\_* = false$$

A custom script named `TestHTTPServer.py` was developed to serve as the target server for the attack. This script forwards incoming connections to the HTTP server. Additionally, a script titled *POODLE-dev.sh* was created to initiate the *httpserver*, *sslserver*, and *attacker-nodebug* components. The attack is initiated by requesting the URL `http://localhost:8000` from a browser vulnerable to exploitation. Consequently, it takes approximately 240 seconds on average to leak 8 bytes of data [17].

*5) **Mitigation Techniques:**:* Within TLS servers, the utilization of the $TLS_FALLBACK_SCSV$ parameter guarantees that SSL 3.0 is exclusively employed when interfacing with legacy systems, thereby thwarting protocol downgrades and mitigating the risk of POODLE attacks. Additionally, to fortify against the POODLE attack, it is imperative to deactivate SSL 3.0 support on both servers and browsers.

D. DROWN Attack

The DROWN (Decrypting RSA with Obsolete and Weakened eNcryption) security breach leverages weaknesses within SSLv2 to decipher RSA encrypted communications. It specifically focuses on servers retaining support for the outdated and vulnerable SSLv2 protocol, despite their predominant usage of more recent TLS versions. This susceptibility often arises from servers maintaining SSLv2 compatibility for legacy purposes.

The attack methodology is depicted as follows: the attacker monitors numerous connections between the targeted client and web server, which involves capturing traffic over an extended period or enticing the user to access a website that rapidly establishes multiple connections to another site in the background. These connections can utilize any SSL/TLS protocol version, including TLS 1.2 with the RSA key exchange method [20], [21], [27].

Subsequently, the attacker initiates multiple connections to the SSL 2.0 server and transmits modified handshake messages containing alterations to the RSA ciphertext extracted from the victim's connections. The server's response to each of these probes hinges on whether the modified ciphertext decrypts into a plaintext message with the correct structure. This response pattern aids the attacker in uncovering secret keys, which are then exploited for the victim's TLS connections [20], [21], [27].

Thus, for contemporary servers and users supporting the SSL 2.0 protocol, DROWN poses a significant threat. Servers are susceptible to DROWN attacks for the following reasons [20], [21]:

- Utilizing SSL 2.0 connections.
- Employing its private key on any other server permitting SSL 2.0 connections, even across different protocols.

*1) **Overview:*** In contemporary times, the majority of up-to-date servers eschew SSLv2 in favor of alternative protocols such as TLS. However, a study conducted in 2016 revealed that among 36 million HTTPS servers examined, a notable 6 million still maintained support for SSLv2 [20]. This deprecated protocol becomes a vulnerability exploited by attackers who capitalize on the SSLv2 handshake process to decipher encrypted key ciphertext.

*2) **SSLv2 handshake:*** During the SSLv2 handshake process, the client initiates communication by transmitting the clientHello message and awaits the ServerHello message from the server.

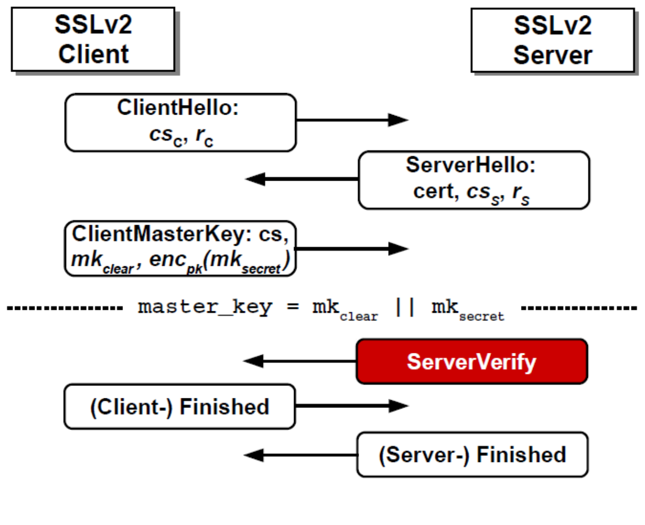
SSLv2 Client
SSLv2 Server
ClientHello: cs_c, r_c
ServerHello: cert, cs_s, r_s
ClientMasterKey: cs, mk_{clear}, $enc_{pk}(mk_{secret})$
master_key = mk_{clear} || mk_{secret}
ServerVerify
(Client-) Finished
(Server-) Finished

Fig. 5. SSLv2 connection handshake [20]

Following this exchange, the client proceeds to encrypt the master key using the server's key before sending it back. Upon receipt, the server encrypts the cipher-text and, should the message format comply with protocol standards, encrypts a new message using the symmetric key to transmit to the client. However, if the format proves to be invalid, the server resorts to generating a random key for encrypting the message instead. This particular aspect of the protocol presents an opportunity for attackers to assess the validation of the message format by sending a message twice. A discrepancy in the keys for the responses signifies that the server has generated two distinct random keys, indicating an invalid format. [22]

In the context of the DROWN attack, assailants exploit the SSLv2 protocol to disrupt TLS connections. Illustrated in Figure 6, when a client lacking SSLv2 support sends a request to a server using the TLS protocol, there remains a possibility that the server supports SSLv2 or that another server supporting SSLv2 shares the same key with the former server. In such scenarios, the attacker intercepts and captures the traffic between the client and the server. Subsequently, the attacker sends the captured traffic to the SSLv2 server multiple times, each time with slight modifications aimed at gathering information. Leveraging this acquired information, the attacker can decrypt the RSA cipher-text, employing the Bleichenbacher attack for decryption.

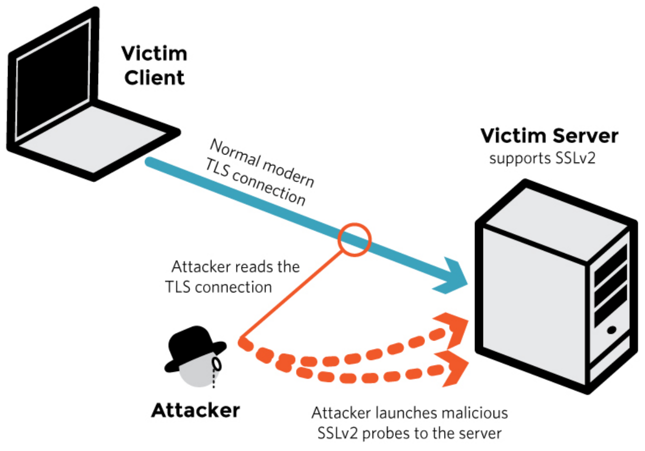
Victim Client
Normal modern TLS connection
Victim Server
supports SSLv2
Attacker reads the TLS connection
Attacker
Attacker launches malicious SSLv2 probes to the server

Fig. 6. Attacker uses SSLv2 to break TLS [20]

*3) **Details of decryption using Bleichenbacher:*** Bleichenbacher's padding oracle attack constitutes an adaptive chosen ciphertext exploit targeting PKCS#1 v1.5, the RSA padding standard utilized in SSL and TLS protocols. This attack facilitates the decryption of RSA ciphertexts by exploiting a server's ability to differentiate between correctly and incorrectly padded RSA plaintexts, earning it the moniker "million-message attack."

In the encryption padding process of PKCS#1 v1.5, depicted in Figure 7, data undergoes encryption with a predefined format. Upon receiving encrypted data, the server decrypts it and assesses the format's validity. If any issues arise with the

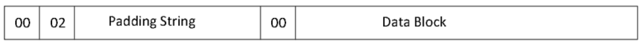

Fig. 7. PKCS#1 v1.5 block format for encryption [20]

format, the server returns a response of 1; otherwise, it returns 0 for a properly formatted input.

In other words, the valid format would be: $2B \leq m \leq 3B - 1$ where $B = 2^{8*(L(m)-2)}$.

Commencing with the cipher-text c0, the attacker transmits it to the oracle server and observes the result. Subsequently, the attacker iteratively adjusts the cipher-text, narrowing down the potential solution set. The modified cipher-text then undergoes further evaluation.

$$c = (c_0 \cdot s^e) \mod N = (m_0 \cdot s)^e \mod N$$

Upon receiving a response of 1, the attacker proceeds to adjust the cipher-text accordingly. Otherwise, he can deduce for some value r, $2B \leq m_0 s^{-r} \mod N < 3B$. And can figure out the possible range form as below:

$$(2B + rN)/s \leq m_0 < (3B + rN)/s$$

In contrast to Bleichenbacher's attack, which targets TLS using a 384-bit key length, DROWN focuses on vulnerabilities associated with short secret keys typically found in export-grade cryptography, where the key length is a mere 40 bits. Furthermore, in TLS, the server autonomously selects the cipher suite type, leaving us with limited insight into the precise length of the key.

*4) **Implementation**:*

a) Intercepting and recording: Capture approximately 1000 TLS connections by intercepting network traffic.

b) Morph TLS connection: As the captured connection lacks compatibility with the SSLv2 oracle for decryption, efforts are made to identify the SSLv2 format for the cipher-text.

c) decrypyt the key using bleichenbacher method: Transmit modified cipher-text multiple times to progressively narrow down potential solutions until only one remains viable.

*5) **Mitigation**:* Various measures can be taken to reduce the susceptibility to this attack:

a) Complete Disabling of SSLv2: Eliminating SSLv2 entirely helps thwart potential exploits associated with its vulnerabilities. It's advisable to deactivate SSLv2 on both client and server ends whenever feasible.

b) Verification of Public Key Usage: Employ distinct public/private key pairs for SSLv2 and TLS implementations if both are necessary. This precaution is particularly crucial as the DROWN attack relies on key reuse.

c) Updating OpenSSL: Vulnerabilities are present in OpenSSL versions before 1.0.2f and 1.0.1r. Ensure systems are upgraded to newer OpenSSL versions with appropriate patches to mitigate risks.

E. CRIME Attack

CRIME (Compression Ratio Info-leak Made Easy) is a security vulnerability exploitation technique against the HTTPS and SPDY [32] protocols when compression is used in HTTP requests to transport data [31]. When this attack is executed and successful, the HTTP requests header can be retrieved and recovered completely, then later, that information can be used by an attacker to perform other types of attacks. CRIME was exposed for the first time in 2012 by security researchers Juliano Rizzo and Thai Duong at Ekoparty security conference [9]. It showcased, for the first time, a real-life example on how information leakage occurs through data compression first reported in 2002 by John Kelsey [33]. The vulnerability that made the attack possible is CVE-2012-4929 [34].

*1) **Overview**:* HTTP requests could contain significant authentication related information such as cookies. When those cookies are leaked, an attacker is able to use them to hijack the session. The main objective of the CRIME attack is to reveal the content from HTTP requests. It exploits the compression techniques of TLS and SPDY [**?**] protocols when they use DEFLATE [**?**] and gzip data compression schemes [**?**].

Deflate is a lossless data compression file format that uses a combination of LZ77 [**?**] and Huffman coding. Deflate Compression LZ77 is best explained using an example. So let's consider the following compressed string:

Security is $s(-14, 7)$

LZ77 decompression works as follows: From the position in the string where the number -14 is inserted as a byte value, the pointer is moved 14 positions to the left – it then points to the letter e of "security". From there, seven characters are copied, and this string – the string "ecurity" – is inserted into the string instead of the pair of numbers (-14,7). The decompressed string is:

Security is security!

During LZ77 compression, the second occurrence of a string is replaced by a pointer to the beginning of the first occurrence of this string and the length of the string. Pointers can also point to other pointers, as the following Figure 8 shows.

CRIME relies on a combination of chosen plaintext attacks and inadvertent information leakage through compression. Attacker observes the change in size of the compressed request payload, which contains both the secret cookie that is sent by the browser to the target site, and variable content created by the attacker. When the size of the content is reduced, it can be said that it is probable that some part of the injected content matches some part of the source, giving the attacker a hint to guess the secret cookie correctly.

*2) **Prerequisite of CRIME attack**:* Executing CRIME attack requires special environments and requirements. To execute this attack, an attacker should have and ensure the following abilities [**?**] [**?**]:

- Compression techniques, such as DEFLATE, gzip, etc. are used in HTTP requests.

Figures/deflate.png

Fig. 8. Complex example of LZ77 compression.

Figures/scrime1.jpg

Fig. 9. An example of HTTP request

- He should be able to measure the size of encrypted traffic.
- Must be able to inject partially chosen plaintext into a victim's HTTP requests.

By this, an attacker can leverage the information leaked by compression to recover the targeted parts of the plaintext.

3) ***Implementation:*** CRIME capitalizes on the data compression feature present in SSL and TLS protocols. Compression occurs at the SSL/TLS level, encompassing both the header and body. The compression algorithm employed by SSL/TLS and SPDY [**?**] is DEFLATE, which eliminates duplicate strings, thereby compressing HTTP requests. CRIME exploits the manner in which duplicate strings are eliminated, employing a systematic brute force approach to guess session tokens. Each occurrence of a duplicate string is substituted with a pointer to the initial instance of the string. Consequently, the level of redundancy in the data directly influences the compression amount. Greater data redundancy results in more compression, leading to a reduced length of the HTTP request. The attacker leverages this dynamic. The CRIME attack process is depicted below.

In Figure 9, we observe an illustration of an HTTP request from a website. The compression techniques of gzip and deflate are employed for compressing or encoding the HTTP request. The header of the request contains a cookie that holds a session secret. The CRIME attack capitalizes on the use of compression techniques for HTTP requests, fulfilling the initial condition for its execution.

In Figure 10, the attacker injects guesses into the HTTP request, satisfying the second condition for the CRIME attack. Utilizing DEFLATE compression, the system recognizes multiple occurrences of the "Cookie: secret=" part. The second

Figures/scrime2.jpg

Fig. 10. Attacker injects some text (e.g. by javascript)

instance is replaced with a small token pointing to the location of the first occurrence, reducing the request length by 15 (the length of the string "Cookie: secret="). Although the attacker cannot directly observe the data, they monitor the length changes in the request. The attacker employs a brute-force approach, systematically trying different values for the secret until the correct one is compressed. For instance, the attacker repeats the process with secret = 2, secret = 3, and so forth until the correct value is identified.

Figures/scrime3.jpg

Fig. 11. Attacker injects some correct text (e.g. by javascript)

In Figure 11, when secret=7 the request's length decreases by 16. Thus the attacker realizes that he has retrieved the first character of the cookie. The same process is repeated until the entire cookie value is retrieved.

*4) **Practicality**:* Executing the CRIME attack successfully requires a remarkably small number of requests, merely six per cookie byte [**?**]. Despite its minimal demand in terms of requests, the ramifications of this attack are substantial, especially when considering the widespread use of TLS across various online platforms and services. The attack's efficiency in obtaining sensitive information with such limited interaction underscores the urgency for robust countermeasures to safeguard against such threats in the realm of secure communication protocols.

*5) **Mitigation Techniques**:* The mitigation techniques related to CRIME attacks are described below:

- **Disable compression:** CRIME attack was executable in TLS version 1.0 [**?**]. Because of that, TLS 1.1 and TLS 1.2 versions were introduced, where compression mechanisms were disabled at TLS levels and thus mitigated this problem [**?**].
- **Compression algorithm:** In TLS protocol version 1.2, client sends a compression algorithm in its ClientHello message, and the server picks one of them and returns it back in its ServerHello message. So, if the client offers or the server chooses **"none"** as a compression algorithm, no compression will take place [**?**] and thus prevent this problem.
- **Keep Browser Updated:** As newer attacks are introduced everyday, it is necessary to keep the browsers updated all time to prevent the attacks.

V. Conclusion

References

[1] R. Fedler,"Padding Oracle Attacks,"Supervisor: B. Hof, M.Sc.,Seminar Innovative Internet Technologies and Mobile Communications, SS 2013,Chair for Network Architectures and Services,Department of Computer Science, Technische Universität München,2013.

[2] IETF. The transport layer security (tls) protocol version 1.3, 2018. RFC 8446, Section 1.1.

[3] Rafael Fedler. Padding oracle attacks. Network, 83, 2013.

[4] Daniel Bleichenbacher. Chosen ciphertext attacks against protocols based on the rsa encryption standard pkcs 1. In Advances in Cryptology— CRYPTO'98: 18th Annual International Cryptology Conference Santa Barbara, California, USA August 23–27, 1998 Proceedings 18, pages 1–12. Springer, 1998

[5] Serge Vaudenay. Security flaws induced by cbc padding—applications to ssl, ipsec, wtls... In International Conference on the Theory and Applications of Cryptographic Techniques, pages 534–545. Springer, 2002.

[6] Breach attack website. https://www.breachattack.com/.

[7] Serge Vaudenay. Security flaws induced by cbc padding—applications to ssl, ipsec, wtls... In International Conference on the Theory and Applications of Cryptographic Techniques, pages 534–545. Springer,2002.

[8] Hanno B¨ock, Juraj Somorovsky, and Craig Young. Return of Bleichenbacher's oracle threat. In 27th USENIX Security Symposium (USENIX Security 18), pages 817–849,2018.

[9] Juliano Rizzo and Thai Duong. The crime attack. Retrieved September 21, 2012, via Google Docs: https://docs.google.com/document/d/1l31tFHbUxZcblU1DWXq3nD4BG8bMr9pKu6JH

[10] T. Duong and J. Rizzo. Padding oracles everywhere (presentation slides). In Ekoparty 2010, 2010.http://netifera.com/research

[11] Klima, V., Rosa, T.: Side Channel Attacks on CBC Encrypted Messages in the PKCS 7 Format. Cryptology ePrint Archive, Report 2003/098 (2003)

[12] Cve-2014-3566. NVD. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-3566

[13] Poodle. Wikipedia. https://en.wikipedia.org/wiki/POODLE.

[14] Downgrade attack. Wikipedia. https://en.wikipedia.org/wiki/Downgrade attack.

[15] Poodle. Google Blog. https://www.openssl.org/bodo/ssl-poodle.pdf.

[16] What is the poodle attack? Acunetix. https://www.acunetix.com/blog/web-security-zone/what-is-poodle-attack/.

[17] Poodle implementation. Thomas Patzkes Personal Website. https://patzke.org/implementing-the-poodle-attack.html.

[18] Bodo Möller, Thai Duong and Krzysztof Kotowicz. This POODLE Bites: Exploiting The SSL 3.0 Fallback, 2014.

[19] Abeer E. W. Eldewahi, Tasneem M. H. Sharfi, Abdelhamid A. Mansor, Nashwa A. F. Mohamed, Samah M. H. Alwahbani. SSL/TLS attacks: Analysis and evaluation. 2015 International Conference on Computing, Control, Networking, Electronics and Embedded Systems Engineering (ICCNEEE), 2015. 10.1109/ICCNEEE.2015.7381362.

[20] https://www.cisa.gov/news-events/alerts/2014/10/17/ssl-30-protocol-vulnerability-and-poodle-attack

[21] Nimrod Aviram, Sebastian Schinzel, Juraj Somorovsky, Nadia Heninger, Maik Dankel, Jens Steube, Luke Valenta, David Adrian, J Alex Halderman, Viktor Dukhovni, et al. DROWN: Breaking TLS using SSLv2. In 25th USENIX Security Symposium (USENIX Security 16), 2016.

[22] O. Ivanov, V. Ruzhentsev and R. Oliynykov, "Comparison of Modern Network Attacks on TLS Protocol," 2018 International Scientific Practical Conference Problems of Infocommunications. Science and Technology, Kharkiv, Ukraine, 2018, pp. 565-570, doi: 10.1109/INFOCOMMST.2018.8632026.

[23] Drees JP, Gupta P, Hüllermeier E, Jager T, Konze A, Priesterjahn C, Ramaswamy A, Somorovsky J. Automated detection of side channels in cryptographic protocols: DROWN the ROBOTs!. InProceedings of the 14th ACM Workshop on Artificial Intelligence and Security 2021 Nov 15 (pp. 169-180).

[24] GlobalPlatform. GlobalPlatform Card Specification Version 2.3.1, March 2018. Reference GPC-SPE-034. Available online: https://globalplatform.org/wp-content/uploads/2018/05/GPC-CardSpecification-v2.3.1-PublicRelease-CC.pdf

[25] ISO/IEC 10116:2017 – Information technology – Security techniques – Modes of operation for an n-bit block cipher, available online : https://www.iso.org/obp/ui/iso:std:iso-iec:10116:ed-4:v1:en

[26] ISO/IEC JTC 1/SC 27. ISO/IEC 9797-1:2011 – Information technology – Security techniques – Message Authentication Codes (MACs) – Part 1: Mechanisms using a block cipher, available online: https://www.iso.org/standard/50375.html

[27] Avoine, G., Ferreira, L. (2018). Attacking GlobalPlatform SCP02-compliant Smart Cards Using a Padding Oracle Attack. IACR Transactions on Cryptographic Hardware and Embedded Systems, 2018(2), 149–170. https://doi.org/10.13154/tches.v2018.i2.149-170

[28] “The DROWN Attack” [Online]. Available: https://drownattack.com

[29] What Is HTTPS and How Does It Work? [Explained],” Semrush Blog. Accessed: Mar. 17, 2024. [Online]. Available: https://www.semrush.com/blog/what-is-https/

[30] Oppliger, Rolf. SSL and TLS: Theory and Practice. Artech House, 2023.

[31] CRIME. Wikipedia. https://en.wikipedia.org/wiki/CRIME.

[32] Spdy: An experimental protocol for a faster web. Chromium Developer Documentation. Retrieved 2009-11-1.

[33] John Kelsey. Compression and information leakage of plaintext. In International Workshop on Fast Software Encryption, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[34] Cve-2012-4929. NVD. https://nvd.nist.gov/vuln/detail/CVE-2012-4929

[35] P. Deutsch. Deflate compressed data format specification. Technical report, RFC 1951, RFC Editor, 1996

[36] Accessed: Mar. 17, 2024. [Online]. Available: https://xilinx.github.io/Vitis$_Libraries/security/2019.2/guideL1/internals/cbc.html$