Ciupitu Andrei-Valentin 341C4
Sterian Cosmin-Cristian 341C4

DROWN: Breaking TLS using SSLv2

Introduction

The DROWN (Decrypting RSA with Obsolete and Weakened eNcryption) attack is a cross-protocol security bug that attacks servers supporting modern TLS protocol suites by using their support for the obsolete, insecure, SSLv2 protocol to leverage an attack on connections using up-to-date protocols that would otherwise be secure. DROWN can affect all types of servers that offer services encrypted with TLS yet still support SSLv2, provided they share the same public key credentials between the two protocols. Additionally, if the same public key certificate is used on a different server that supports SSLv2, the TLS server is also vulnerable due to the SSLv2 server leaking key information that can be used against the TLS server.

TLS is one of the main protocols responsible for transport security on the modern Internet. TLS and its precursor SSLv3 have been the target of a large number of cryptographic attacks in the research community, both on popular implementations and the protocol itself. Prominent recent examples include attacks on outdated or deliberately weakened encryption in RC4, RSA, and Diffie-Hellman, different side channels including Lucky13, BEAST, and POODLE, and several attacks on invalid TLS protocol flows.

Comparatively little attention has been paid to the SSLv2 protocol, likely because the known attacks are so devastating and the protocol has long been considered obsolete. Most modern TLS clients do not support SSLv2 at all. Yet in 2016, our Internet-wide scans find that out of 36 million HTTPS servers, 6 million (17%) support SSLv2.

Bleichenbacher's attack

Bleichenbacher's attack is a padding oracle attack — it exploits the fact that RSA ciphertexts should decrypt to PKCS#1 v1.5-compliant plaintexts. If an implementation receives an RSA ciphertext that decrypts to an invalid PKCS#1 v1.5 plaintext, it might naturally leak this information via an error message, by closing the connection, or by taking longer to process the error condition. This behavior can leak information about the plaintext that can be modeled as a cryptographic oracle for the decryption process. Bleichenbacher demonstrated how such an oracle could be exploited to decrypt RSA ciphertexts.

Breaking TLS with SSLv2

In this section, we describe our cross-protocol DROWN attack that uses an SSLv2 server as an oracle to efficiently decrypt TLS connections. The attacker learns the session key for targeted TLS connections but does not learn the server's private RSA key. We consider a

server accepting TLS connections from clients. The connections are established using a secure, state-of-the-art TLS version (1.0–1.2) and a TLS_RSA cipher suite with a private key unknown to the attacker.

The same RSA public key as the TLS connections is also used for SSLv2. For simplicity, our presentation will refer to the servers accepting TLS and SSLv2 connections as the same entity.

Our attacker is able to passively eavesdrop on traffic between the client and server and record RSA-based TLS traffic. The attacker may or may not be also required to perform active man-in-the-middle interference, as explained below.

The attacker can expect to decrypt one out of 1,000 intercepted TLS connections in our attack for typical parameters. This is a devastating threat in many scenarios. For example, a decrypted TLS connection might reveal a client's HTTP cookie or plaintext password, and an attacker would only need to successfully decrypt a single ciphertext to compromise the client's account. In order to collect 1,000 TLS connections, the attacker might simply wait patiently until sufficiently many connections are recorded. A less patient attacker might use man-in-themiddle interference, as in the BEAST attack.

Special DROWN, The OpenSSL "extra clear" oracle

We discovered multiple vulnerabilities in recent (but not current) versions of the OpenSSL SSLv2 handshake code that create even more powerful Bleichenbacher oracles, and drastically reduce the amount of computation required to implement our attacks. The vulnerabilities, designated CVE-2016-0703 and CVE-2016-0704, were present in the OpenSSL codebase from at least the start of the repository, in 1998, until they were unknowingly fixed on March 8 4, 2015 by a patch designed to correct an unrelated problem. By adapting DROWN to exploit this special case, we can significantly cut both the number of connections and the computational work required.

Prior to the fix, OpenSSL servers improperly allowed the ClientMasterKey message to contain clear_key_data bytes for non-export ciphers. When such bytes are present, the server substitutes them for bytes from the encrypted key. For example, consider the case that the client chooses a 128-bit cipher and sends a 16-byte encrypted key k[1], k[2],..., k[16] but, contrary to the protocol specification, includes 4 null bytes of clear_key_data. Vulnerable OpenSSL versions will construct the following master_key:

[00 00 00 00 k[1] k[2] k[3] k[4] ... k[9] k[10] k[11] k[12]]

This enables a straightforward key recovery attack against such versions. An attacker that has intercepted an SSLv2 connection takes the RSA ciphertext of the encrypted key and replays it in non-export handshakes to the server with varying lengths of clear_key_data. For a 16-byte encrypted key, the attacker starts with 15 bytes of clear key, causing the server to use the master_key:

[00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 k[1]]

The attacker can brute force the first byte of the encrypted key by finding the matching ServerVerify message among 256 possibilities. Knowing k[1], the attacker makes another connection with the same RSA ciphertext but 14 bytes of clear key, resulting in the master_key:

[00 00 00 00 00 00 00 00 00 00 00 00 00 00 k[1] k[2]]

The attacker can now easily brute force k[2]. With only 15 probe connections and an expected $15 \cdot 128 = 1{,}920$ trial encryptions, the attacker learns the entire master_key for the recorded session. As this oracle is obtained by improperly sending unexpected clear-key bytes, we call it the Extra Clear oracle. This session key-recovery attack can be directly converted to a Bleichenbacher oracle. Given a candidate ciphertext and symmetric key length ℓk, the attacker sends the ciphertext with ℓk known bytes of clear_key_data.

The oracle decision is simple:

• If the ciphertext is valid, the ServerVerify message will reflect a master_key consisting of those ℓk known bytes.

• If the ciphertext is invalid, the master_key will be replaced with ℓk random bytes (by following the countermeasure against the Bleichenbacher attack), resulting in a different ServerVerify message.

This oracle decision requires one connection to the server and one ServerVerify computation. After the attacker has found a valid ciphertext corresponding to a ℓk-byte encrypted key, they recover the ℓk plaintext bytes by repeating the key recovery attack from above. Thus our oracle OSSLv2-extra-clear(c) requires one connection to determine whether c is valid. After ℓk connections, the attacker additionally learns the ℓk least significant bytes of m. We model this as a single oracle call, but the number of server connections will vary depending on the response.

Results

We implemented the OpenSSL “extra clear” oracle as explained in the DROWN attack paper [0] and some helper functions for OpenSSL communication.

At first we tried to capture TLS handshakes using Wireshark, but we encountered difficulties when we tried to convert the original ClientKeyExchange message(the PreMaster secret) from TLS conformant ciphertext into an SSLv2 conformant ciphertext, so we decided to simulate SSLv2 connections only. Also, we used SSL_DES_192_EDE3_CBC_WITH_MD5 as the cipher suite, allowing the attacker to recover 24 bytes of key at a time.

In order to check the guess key, the oracle has to establish a new connection each time and send a ClientHello handshake to the server before sending the actual guess key. As mentioned before, the current byte is correctly guessed if the server replies with a ServerVerify message containing the same key as the one sent by the oracle.

[0] - DROWN attack paper