Timing attacks against OpenSSL

Juhani Toivonen
Department of Computer Science
P.O. Box 68 (Gustaf Hällströmin katu 2b)
FI-00014 University of Helsinki, Finland
juhani.toivonen@cs.helsinki.fi

ABSTRACT

OpenSSL is the most widely used open source SSL/TLS implementation on the internet and an immense amount of sensitive communication is trusted to be secured by it. The related cryptographic algorithms themselves are indeed very secure. However implementing the models in hardware or software introduces new kinds of channels that are not present in the mathematical model, but which can nonetheless be abused. Using information retrieved using these channels can reveal the secrets about the communication to the attacker. In this paper we discuss two known attacks against OpenSSL and their implications.

Categories and Subject Descriptors
E.3. [Data Encryption]: Public key cryptosystems, Code Breaking, Standards (RSA)

General Terms
DESIGN, SECURITY

Keywords
RSA, CBC, OpenSSL, Side-channel attack

1. INTRODUCTION

Side channel attacks are historically somewhat neglected field of security. Secure systems are usually designed so that they follow a provably secure mathematical model, but it is often forgotten that when placed in a context, new channels appear. SSL and TLS protocols are highly important, they are used throughout the internet to secure communications between clients and servers. Every time you see the "https://" and the lock image in your browser, your communication is going through one of them.

For this paper we chose one well explored side channel and its implications for the OpenSSL protocol and cipher suite. We explore two timing attacks and demonstrated against OpenSSL, The Brumley and Boneh attack, which attempts to reveal the long term secret, the private key of the remote server, and the very recent Lucky Thirteen attack, which can be used to decrypt captured ciphertext. We also discuss the current state of OpenSSL in regard to these attacks.

In section two, we discuss the nature of side channel attacks, and the important details regarding implementation of things in OpenSSL. In section three we introduce the attacks, and finally in section four, some proposed countermeasures. We’ll also discuss related work in section 5.

2. BACKGROUND

2.1 Side channel attacks

Side channel attacks are attacks against secured systems that utilise information unintentionally leaked by the systems. Essentially, they are attacks against the crypto systems’ implementations rather than against their mathematical models. Side channel attacks is a broad subject and ranges from discreetly and undetectably observing the devices’ operation to drilling holes and inserting probing needles into their components to see what happens inside with a certain input. Zhou et al. provided background and explored a number of known side channel attacks in [8]. Figure 1 shows the different channels giving output from a cryptographic system, usually not present in their mathematical models.

Figure 1: Secure message M from Alice to Bob, including channels present in implementation [8].

Timing attacks, the main focus of this paper, attempt to use the execution time as a side channel to gain information about the secrets. They can be classified as active, non-intrusive attacks. The attacks are non-intrusive because no tampering with the hardware is necessary, and active because interacting with the system using its usual, "legal" interfaces usually is. The attacks involve inserting inputs into the system and measuring the time it takes for it to carry
out its normal operation. A somewhat more passive attack using a carefully crafted spy process has been demonstrated by Aciqmez et al. [1]. The differences in the measured times can leak information about events such as an encountered mismatch during string comparison. Such information can then be used to make a better guess about the expected input, e.g. about a secret key.

The root cause of vulnerability to timing attacks is the conditionally branched nature of many operations. If a condition is matched, one branch is taken, if it’s not, another one is taken. The usual ways of programming along with various ways of optimising, result in the different branches requiring different amounts of computation and hence different amount of time. By trying out different inputs, Branch equalisation and artificial addition of noise have been proposed and used as counter measures against timing attacks.

2.2 OpenSSL

OpenSSL is an open source implementation of the SSL and TLS protocols. It provides a suite of encryption ciphers, that can be used in a number of operational modes. It is the most widely used SSL/TLS implementation used on web servers throughout the internet. For performance reasons, OpenSSL’s RSA implementation contains certain optimisation features. Namely it makes use of the Chinese Remainder theorem, Sliding windows, the Montgomery multiplication algorithm and Karatsuba’s algorithm.

SSL/TLS implementations are usually used as a library. Hence their features, including vulnerabilities, affect all software that uses the specific implementation. Fortunately this also means that fixes for the vulnerabilities affect all such software as well.

Besides OpenSSL, other open source implementations of the SSL and TLS protocols include GnuTLS, CyassL, PolarSSL and NSS. Proprietary implementations such as Microsoft’s Secure channel exist as well.

2.3 RSA

RSA is an algorithm commonly used in public-key cryptography. It is often used as a key exchange protocol while initiating a secure communication channel, for securely agreeing on a key to use in a (computationally cheaper) symmetric encryption scheme, but can be used to encrypt any kind of message.

The key generation phase of the algorithm starts by choosing two distinct large prime numbers $p$ and $q$. The modulus $n$ for the encryption and decryption keys is computed $n = pq$. Then the encryption exponent $e$ is chosen such that it satisfies $1 < e < \Phi(n)$ and $gcd(e, \Phi(n)) = 1$, where $\Phi$ is Euler’s totient function. Finally the decryption exponent $d$ is determined as the multiplicative inverse of $e$ (mod $n$).

The key generation phase produces a pair of keys, called a public key and a private key. The public key can be used for encrypting messages, and the private key can be used for decrypting them. The keys consist of their respective exponent $e$ or $d$, and the modulus $n$, that are needed for the operations. Encryption and decryption are then simple to express. $m$ being the plaintext message and $c$ being the encrypted ciphertext, encryption can be expressed $c \equiv m^e \pmod{n}$ and decryption $m \equiv c^d \pmod{n}$.

2.3.1 RSA in OpenSSL

OpenSSL and many others rely on the Chinese remainder algorithm to speed up decryption. Rather than computing $m$ directly, the computation is split in two parts. From $p$ and $q$, are computed $d_p = d \pmod{p - 1}$, $d_q = d \pmod{q - 1}$ and $q_{cds} = q^{-1} \pmod{p}$. The decryption is performed by computing $m_1 = c^{d_p} \pmod{p}$ and $m_2 = c^{d_q} \pmod{q}$, then $h = q_{cds} \ast (m_1 - m_2) \pmod{p}$ and finally $m = m_2 + (hq)$. What seems like more work here is faster, because the numbers to be operated on are smaller. As the CRT uses $q$ and $p$, the factors of $n$, a timing attack that reveals information about how long it takes to compute the decryption can be used to find out the factors, and eventually derive the decryption key.

To further optimise, OpenSSL uses techniques called sliding windows and Montgomery reduction. Sliding windows are an optimisation for computing the $g^e \pmod{q}$ a block of bits at a time, instead of a bit at a time with the ordinary method. Montgomery reduction, or Montgomery multiplication is an algorithm for performing modulo arithmetic on large numbers quickly. The numbers to be multiplied are transformed to a Montgomery form, where the integers reduced to modulo some power of 2 denoted by $R$. The Montgomery form for a number $x$ is $xR \pmod{q}$, and multiplication is done by first computing $aR + bR = cR^2$, and reducing it $cR^2 \ast R^{-1} = cR \pmod{q}$. Computations in this form are fast and can be implemented in hardware. As the results are also in Montgomery form, multiplication operations can be chained without further transformations in between. The basic idea is to transform the numbers to a form where computation is fast, do the computations, and transform them back to the original form.

Dealing with large numbers usually requires the use of a library for multi-precision integers. OpenSSL uses two algorithms for multiplication of large numbers. When the integers to be multiplied are of an equal length, the OpenSSL optimises the computation by using the Karatsuba multiplication algorithm. Otherwise the traditional long multiplication algorithm is used.

2.4 Cipher-Block Chaining

Cipher-Block Chaining is a mode of operation for a cryptographic block cipher. It is used to increase confidentiality of the encrypted message.

In CBC, the ciphertext of each block is used, along with the key, as part of the input while encrypting the next block. This way the ciphertext for a block is influenced by the encryption of all the previous blocks, instead of just the key. For encrypting the first block, there is no prior ciphertext, an initialisation vector is used instead. Figure 2 represents encryption using CBC.

For proper decryption, the key and the used initialisation vector must be known. Most of the plaintext can be decrypted using only the key and adjacent blocks of ciphertext, but the initialisation vector is needed for the first block. Latter blocks can be decrypted even if the plaintext of the first block could not be recovered because only the ciphertext is used.

In CBC, all the blocks are of an equal size, called the block size. When encrypting a message whose length is not a multiple of the block size, padding must be added to the last block. The form of the padding is not meaningless, it can be used to determine whether there is something wrong with the message. Non-constant time operations in checking the padding also make grounds for timing attacks.
Figure 2: Encryption in Cipher Block Chaining mode [7].

3. ATTACKS

3.1 The Brumley and Boneh attack

In 2005, Brumley and Boneh presented a timing attack against OpenSSL, that aims to find out the complete RSA decryption key (private key) of a server [4]. The attack is based on two observations about how properties of the input data correlate with the execution time.

The first observation is a finding by Schindler [5], who reported that the Montgomery reduction algorithm makes an extra reduction step with a probability following equation 1. Whether the extra reduction step takes place or not causes a timing difference in the execution time of the algorithm, revealing information about the key. Figure 3 shows the number of extra Montgomery reduction steps involved in the decryption process as a function of $g$, where $g$ is the ciphertext.

$$P(\text{extra reduction}) = \frac{g \mod q}{2R}$$ (1)

The second observation relates to how OpenSSL handles multiplications of large numbers. Libraries for handling large numbers usually internally present them as a sequence of words. When multiplying numbers consisting of an equal amount of words, OpenSSL optimises the computation by using the Karatsuba multiplication algorithm. The Karatsuba multiplication algorithm works for numbers of equal length and runs in $O(n^{\log_23}) \approx O(n^{1.585})$. Numbers consisting of an unequal number of words are computed using normal long multiplication, which runs in $O(nm)$, where $n$ and $m$ are the lengths of the input numbers. Multiplying unequal length numbers should therefore take longer than equal length numbers. This is again useful for a timing attack.

The attack demonstrated by Brumley and Boneh works by making approximations of $q$, the modulus, and iterating revisions of it until half of the most significant bits are known. Then the complete factorisation is derived using the Coppersmith’s algorithm. By sending guesses $g$ and measuring time it takes to decrypt them, peak points as presented by figure 3 at the locations of $q$ and $p$ can be seen. By toggling the most significant unknown bit and measuring time it takes for decryption, we can determine whether turning the bit to one takes us over the modulus $q$. If the difference in decryption time whether the bit is one or zero is small, we are still under the modulus, and the number with the bit set to one is closer to the modulus, hence we choose one. If the difference is large, setting the bit to one made our guess larger than the modulus and the time is dominated either by the Montgomery reductions or the multi-precision multiplications.

The sliding window reduces the amount of steps for multiplications, which makes it harder to get trustworthy single measurements. To counter the effect of the sliding window, the raw measurements with $g$ are not used alone. Instead neighbourhood of $g$ is queried and the average is used.

3.2 The Lucky Thirteen attack

In 2013, AlFardan et al. demonstrated another timing attack, effective against many TLS implementations including OpenSSL [2]. It aims to reveal the contents of an encrypted message. The attack is a modification of an earlier attack described by Vaudenay et al. in [6]. It takes advantage of a vulnerability in the CBC padding scheme. Figure 4 shows the structure of the padding.

The time it takes to check a padding depends on whether it’s good or not. When the checking is done, a server replies with an error message if the padding or MAC (Message Authentication Code) was bad, and terminates the session. The time it takes from sending the request until the time an error message is received is measured. By combining a target ciphertext $C^*$ (a captured encrypted message) and a chosen ciphertext $\Delta$ with a valid message, the attacker can cause its plaintext to be misinterpreted as padding. The decryption of the message can then result in three likely outcomes, one of which (case 2 in original paper) can be distinguished through the difference in execution time. The attacker can find the $\Delta$ that produces the case 2 outcome by trying out
all the different values from a limited set. Finding the case 2 at this phase reveals the last 2 bytes of the target plaintext $P^*$. This can be repeated to work out the bytes from left to right, and eventually the complete plaintext. More details and further considerations can be found in [6].

4. COUNTERMEASURES

The attack of Brumley and Boneh can be countered by using RSA blinding. RSA blinding is a technique where before decryption, $x = r^e \cdot c \mod n$, where $r$ is a newly generated random number, is computed. Decrypting $x$ and dividing the message by $r$ reveals the original plaintext. This introduces a random factor to how long the computation takes, rendering the timing measurements in the attack unusable.

For the The Lucky Thirteen attack, AlFardan et al. suggest switching from CBC-mode block encryption to using RC4 stream encryption. Since a stream cipher does not use padding, the weak spot for the attack is not present. As a drawback, a block cipher with padding hides the exact length of the message, while a stream cipher does not. Other countermeasures they suggest are adding random delays and switching to an Authenticated Encryption algorithm. However they admit that the random delay method is really not as secure as it would intuitively appear. Also the support for Authenticated Encryption algorithms is not very good yet.

Countermeasures for the known timing attacks have been implemented and are enabled by default in the newest versions of OpenSSL. The blinding against the Brumley and Boneh attack is enabled by default since 0.9.7.b [4]. Countermeasures for the Lucky Thirteen attack have been addressed in OpenSSL versions 1.0.1d, 1.0.0k and 0.9.8y [2].

From the user’s point of view the simplest countermeasure is to keep your software up to date. From the system administrator’s point of view, in addition to the previous, configure their servers to reject use of unsafe ciphers and use an intrusion detection/prevention framework.

5. RELATED WORK

In 2011 Brumley et al. discussed another timing attack vulnerability, this time involving ECDSA elliptic curve cryptography [3].

6. REFERENCES


