Timing attacks against OpenSSL

Juhani Toivonen

24.4.2013
OpenSSL

- An open source implementation of the SSL and TLS protocols and a collection of cryptographic ciphers and related tools.


- Based on SSLeay (discontinued in 1998).

- Usually used as a library by other programs to secure communication.
Popularity contest statistics for openssl, libgnutls26, libgnutls28, libnss3
Popularity contest statistics for openssl, libgnutls26, libgnutls28, libnss3

popcon graph

percent of submitters

date


libgnutls26 vote + libgnutls28 vote x libnss3 vote * openssl vote □
Side-Channel Attacks

- Attacks that utilize information that leaks from the real-life implementation.
Timing attacks

- Side-Channel Attacks based on execution time

- Big Question: What can we learn from how long it takes?
Timing attacks

Simple example

 Mell

Verifying password through simple string comparison.

```c
bool check_pass(char* real, char* guess) {
    int len = strlen(guess) + 1; /* +1 for terminator */
    for (int i = 0; i < len; i++) {
        if (real[i] != guess[i]) {
            return false;
        }
    }
    return true;
}
```
Timing attacks
Simple example

* For 8 character password, with alphabet [A-Z, a-z, 0-9], no rules

* Brute force: iterate through all the combinations

  * $62^8 = 218340105584896$ (worst case)

* Timing attack: Iterate through the alphabet to build combination one by one.

  * $62 \times 8 = 496$ (constant)
Timing attacks
Prerequisites in example

* Understanding of the underlying algorithm.

* Control over one of the inputs.

* A way of measuring time accurately.
The Brumley and Boneh attack

- The goal is to reveal the target’s private key.
- The attack is based on a timing difference caused by optimization features used in OpenSSL.
RSA
Initialization

* Start by choosing two distinct large prime numbers $p$ and $q$

* Compute modulus $n = pq$

* Choose encryption exponent $e$ such that $1 < e < \varphi(n)$ and $\gcd(e, \varphi(n)) = 1$

* Compute decryption exponent $d$ as $d^{-1} \equiv e \pmod{\varphi(n)}$

* Create a pair of keys, private key from $d$ and $n$, public from $e$ and $n$. 
RSA

Usage

• Encryption: \( c \equiv m^e \pmod{n} \)

• Decryption: \( m \equiv c^d \pmod{n} \)
RSA in OpenSSL

- OpenSSL supports the RSA encryption scheme and optimizes it with a couple of techniques.

- CRT, Chinese Remainder Theorem

- Sliding windows

- Montgomery multiplication

- Karatsuba’s algorithm
RSA in OpenSSL

Chinese remainder theorem

• Instead of performing the computations directly with $n$, do it in parts.

• $d_p = d \pmod{p-1}$, $d_q = d \pmod{q-1}$ and $q_{inv} = q^{-1} \pmod{p}$

• Decryption:

  • $m_1 = c^{dp} \pmod{p}$ and $m_2 = c^{dq} \pmod{q}$

  • $h = q_{inv} \ast (m_1 - m_2) \pmod{p}$

  • $m = m_2 + (hq)$. 
Splitting the operations allows operating on smaller factors, hence speeding up the process.

A timing attack could now gather information about the factors.
RSA in OpenSSL

Sliding windows

- The technique of decrypting the ciphertext a block at a time instead of one bit at a time is called sliding windows.
- Decrypting a chunk at a time results in less steps in multiplication, and is hence faster.
The Montgomery algorithm is used for performing multiplications on large integers.

The idea is to transform the integers into Montgomery form, do computations, then transform them back.

Montgomery form for $x$: $xR \mod q$
Where $R$ is some power of 2.

The Montgomery form is based on powers of 2, so computations are fast and can be implemented in hardware.
RSA in OpenSSL

Karatsuba’s algorithm

• Karatsuba’s algorithm is used for multiplying equal-length (word-wise) integers efficiently.
  
  • $O(n^{\log_{2}3}) \approx O(n^{1.585})$

• If the integers are of different length, long multiplication is used instead.
  
  • $O(nm)$, for equal-length $n=m \rightarrow O(n^2)$

• Timing difference, reveals information about the factors!

Thursday, April 25, 13
The Brumley and Boneh attack

- Bombing the target with ciphertext-guesses $g$.

- Based on two observations:
  - Extra reduction step in Montgomery’s algorithm
  - Multiplication of large numbers
The Brumley and Boneh attack
Extra reduction step in Montgomery’s algorithm

• The algorithm makes an extra reduction step with probability
  \[ P = \frac{(g \mod q)}{2R} \]

• The closer \( g \) is to \( q \), the more extra reductions take place.

• Decrypting takes more time when \( g \) is just below \( q \) than when \( g \) is just over \( q \).
The Brumley and Boneh attack

Multiplication of large numbers

- When \( g \) is just below \( q \), they are likely to be of equal length word-wise. When \( g \) is just over \( q \), they are likely to be of different length.

- Equal-length integers are multiplied using Karatsuba’s algorithm, others are multiplied using long multiplication.

- Decrypting takes less time when \( g \) is just below \( q \), and more time when just above \( q \). The opposite effect of the Montgomery multiplication.
The Brumley and Boneh attack

The attack

- The attack is done by sending the target guesses for ciphertext \( g \) and measuring time it takes to decrypt them.

- Start by iterating through all possible combinations for the few most significant bits and measuring time it takes to decrypt them.

- Plotting the initial step produces a graph like this:
The Brumley and Boneh attack

The attack

- Pick a value that was close to, but below \( q \)
  - Toggle the most significant unknown bit. Compute \( u_g = gR^{-1} \) for both with the bit set to 0 and the bit set to 1.
  - Measure time it takes to decrypt them, call it \( t_0 \) and \( t_1 \) respectively.
  - If the difference in measured time is small, the corresponding bit in \( q \) is 1. If the difference is large, setting the bit to 1 made \( g \) larger than \( q \), meaning that the corresponding bit in \( q \) is 0.
The Brumley and Boneh attack

The attack

- If the difference was large and \( t_1 > t_2 \), the Montgomery reductions dominated the execution time. If \( t_1 < t_2 \), the multiplications dominated the execution time.

- The Sliding window optimization reduces the number of steps required for the multiplications, which makes detecting the difference somewhat more difficult.

- To overcome the sliding windows, the measurements should be made using not only \( g \), but it’s neighborhood: \( g, g+1, g+2 \ldots \) and summing their durations together.
The Brumley and Boneh attack

The attack

- The steps are repeated for each consecutive bit, until about half of the bits have been retrieved.

- When enough bits are known, the rest can be solved using the Coppersmith’s method. [http://en.wikipedia.org/wiki/Coppersmith_method](http://en.wikipedia.org/wiki/Coppersmith_method)
For timing attacks in general, the most used countermeasures are blinding and branch equalization.

- Blinding means adding a random delay to the computation; the attacker will get a mostly useless sum of “computation+random”

- Branch equalization means designing the algorithms in such a way, that computation takes equally long regardless of inputs.
Countermeasures
System Administrators

- Against the Brumley and Boneh attack, the simplest countermeasures are to keep your SSL library up to date.

- The OpenSSL development team has addressed the vulnerability by enabling blinding in OpenSSL version 0.9.7b.
Related work

- A more recently found vulnerability has been used in the Lucky Thirteen attack, which I will be discussing in the seminar paper.

- The Lucky Thirteen attack has also been addressed; it has been suggested to switch from a block cipher mode called CBC to a stream cipher called RC4.

- RC4 has since been found vulnerable as well.
Conclusions

✧ Side channel attacks are a serious threat.

✧ Keep your software up to date.

✧ Use libraries, they usually address many problems you have never even heard about.
Thank you for your interest!

• Questions?