KTH STH HANINGE

Side-Channel Attack

Login Timing Attack

Mathias Axtelius
Robin Duda

26/02-2016

Abstract

Timing attacks are one of several side-channel attack methods that can break and steal a cryptosystems keys. Through measuring the time for an encryption device to perform its operations, timing data is leaked. The attack depends on a jitter-free environment and high-resolution timers, the feasibility of such an attack is discussed in this paper.

Introduction

In cryptography side-channel attacks are based on information that can be collected by analysing patterns and statistics to predict the outcome of a current or future operation. Side-channel vulnerabilities include timing leakage (disk read, computations), electromagnetic attacks and power-monitoring that make use of the varying power consumption by the hardware during a computation. The side-channel attack implementations are a concern because they are very simple and can be implemented on cheap hardware that is available today. Another important factor is that they are becoming more feasible as the jitter present in networks continuously decrease.

Previous attacks were based on ciphertext knowledge that were used in ciphertext-only attacks, known-plaintext attacks chosen-plaintext attacks. These methods requires knowledge of the cipher in use and its inputs and outputs. In contrast a side-channel timing attack only requires a target and a stopwatch.

Methods

Timing attacks are based on measuring the time it takes for an encryption device to perform an operation. This method can be used to gather information about a secret key, that is used for encrypting a password or the password itself. By measuring the time that is required to perform a private key operation, an attacker could find the Diffie-Hellman exponents. Then factor the RSA (Rivest, Shamir, Adleman) keys and finally break the cryptosystem. As it is not uncommon for a system to reuse their private DH exponent, it is recommended that ephemeral (DHE) is configured.[1]

Optimizations present in the hardware of cryptosystems are the root cause of leaking timing data. Optimizations include caching operations in lower levels of the cache, storing data in RAM (Random Access Memory), conditional statements, branch prediction and a variety of other optimizations. Optimizations done in software have a greater impact on timing information as changes in high-level code translates to more CPU instructions. A critical example is the usual way of comparing strings where a function returns upon the first character that mismatches, in turn saving a load of CPU time (and leaks timing information).[2]

Computing the timing variances identification of the correct exponent bit is possible. The amount of samples required to gather enough information to solve the secret key is determined by the properties of the noise and signal. More samples is required if the amount of noise is high to solve the secret key (known as key recovery). Furthermore, there are error correction techniques that can reduce the number of samples required, these techniques require more memory and processing power.[3]

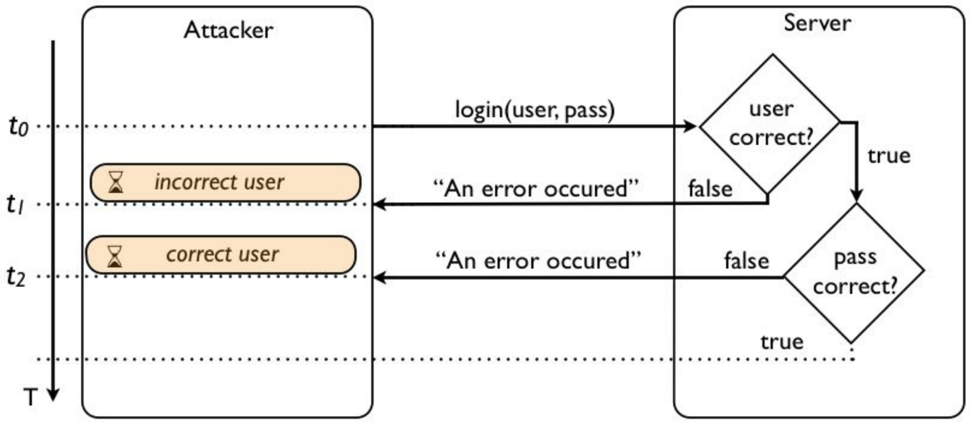

Figure 1. Login Timing Attack

To test the feasibility of extracting a plaintext password through timing a Cisco Router was virtualized within a GNS3 virtual machine. The version of the Cisco IOS used in the tests was "c3725-adventerprisek9-mz124-25". The router was configured to allow telnet connections and requiring only the password to gain access (reducing overhead). The originating system inside a VirtualBox virtual machine running Ubuntu Server 15.10 (1-40ns precision), provided better accuracy than the VM host system on Windows 10 which measured only a granularity of 300ns. A tool for measuring timing data with modular support for the Telnet protocol and local implementations of password comparison was developed.[4]

The test results was gathered by sending repeated login attempts starting with a known invalid password at the same length of the secret password. For every iteration another known letter was added to the query, using a known secret to verify the vulnerability is simpler than actually exploiting it. An iteration count was passed to the program that defines how many attempts should be made before adding the next known character. This improved the accuracy of the graphs while also removing the jitter introduced.

Results

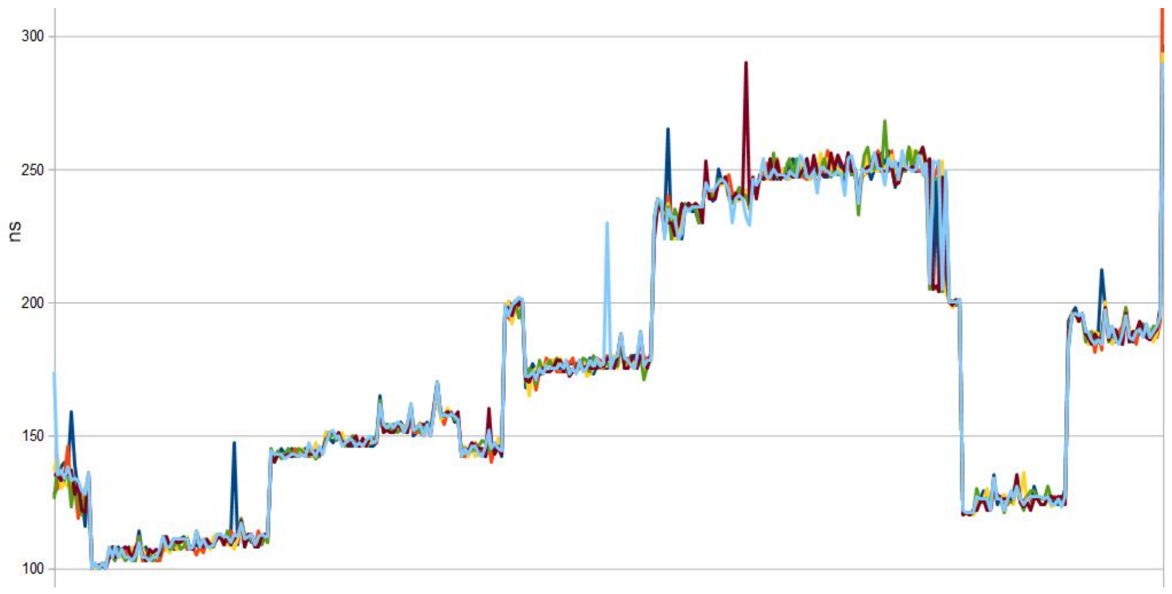

Figure 2. Local comparison - Leaking Data, Generated from *Appendix 1*

The test data from the local implementations proved that significant data is leaked that could be used to guess the correct password. Throughout the graph there is a constant increase in time taken to compare the tokens. The falls and rises of 50ns are clearly distinguishable as jitters that are not related to the comparing of the strings. Whether the information could be measured over networks is yet unknown although there is clear evidence that the problem exists.[Figure 2]

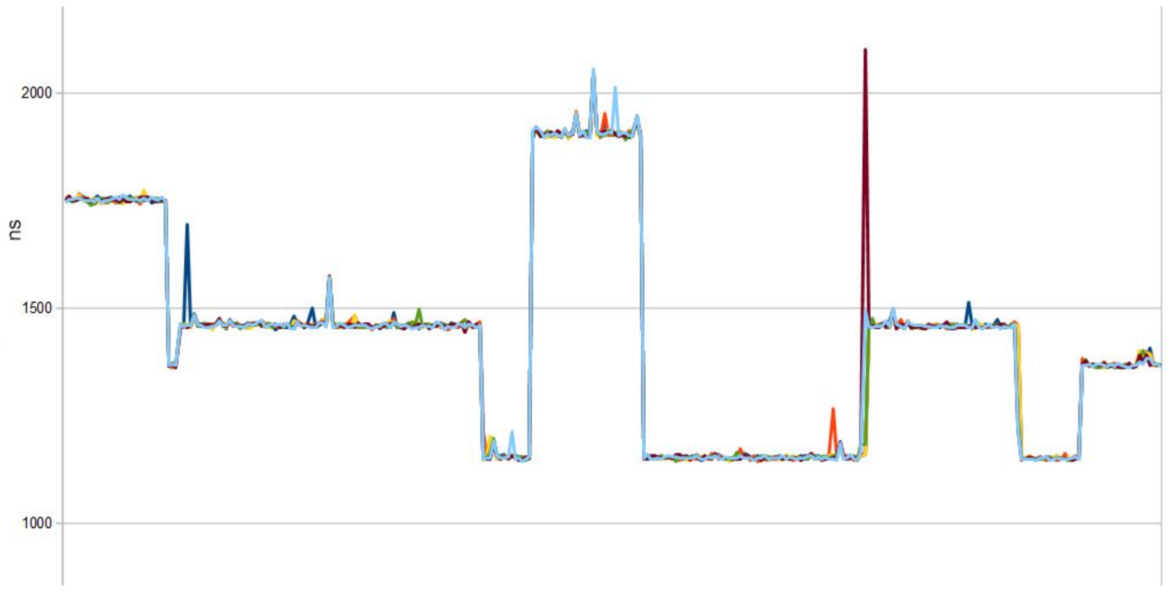

Figure 3. Local comparison - Time-constant Not Leaking, Generated from *Appendix 2*

In [Figure 3] a safe implementation was tested and proved to not reveal any timing information. The lines are flat and any rise and fall is from jitter alone. This implementation is not vulnerable to remote timing attacks as it is not vulnerable to local attacks.

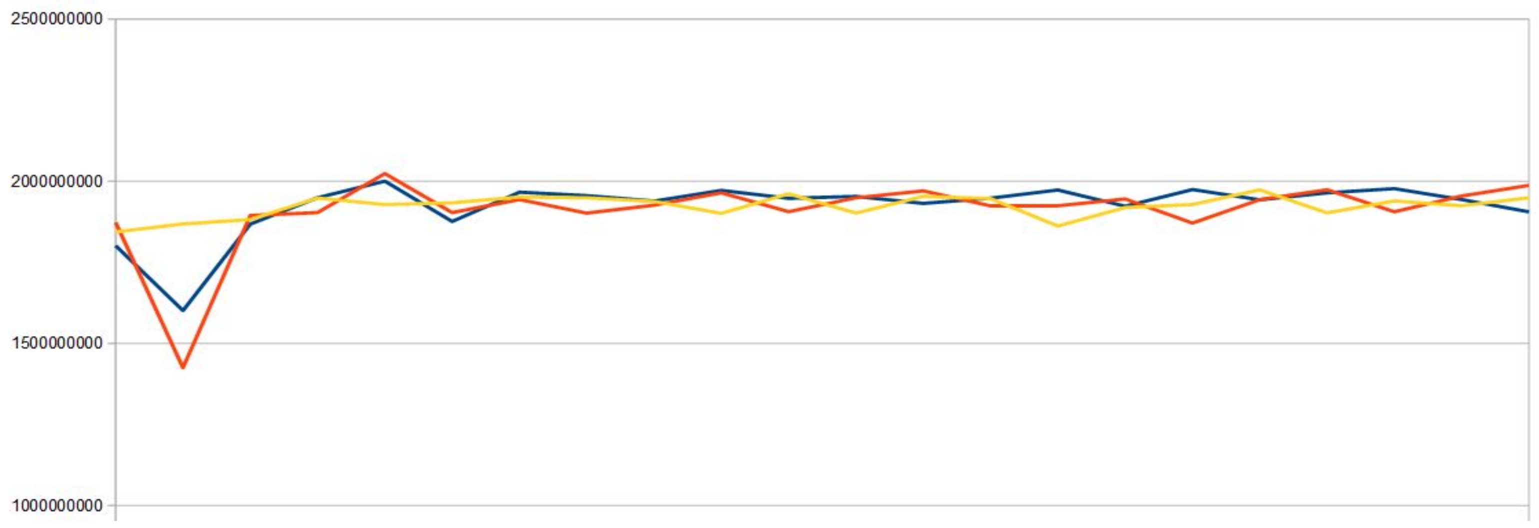

Figure 4. Telnet from Ubuntu Server 15.10 VM to GNS3 Cisco IOS

The measurements to the Cisco router produced data that is too inaccurate to be analyzed. The ping from the attack origin to the Cisco router varied from 1-10ms where measurements of 1ns is required. Whether the Cisco router is vulnerable or not is yet undecided, further tests are required.

Analysis & Discussions

Threats

Cryptography is one of the primary building tool for achieving security. Important information rely on cryptography for securing the information, for example by protecting applications, communications and securing payments. But if an attacker gain access to the secret keys, the encryption could be vulnerable by an attacker. The secret keys can later be used to steal private information from an active communication, ongoing payment or from an active application.

There are several reasons why an attacker would like to gain access to private information, or even gain access to do eavesdropping on a secure protocol. These attacks cause a serious threat to the encryption devices that the companies use.[1] Other key targets for login attempts through timing attacks are CMS (Content-Management System) platforms, such as Joomla, Drupal and WordPress. A successful attack to gain unauthorized access would allow the attackers to upload malicious backdoors based on PHP (Hypertext Preprocessor). In some cases, this compromise could also benefit an attacker to find the full path to a hosting server.[5]

Effectiveness

Theoretical

The effectiveness of the attack has a huge theoretical advantage compared to brute-forcing.

Brute force = keyspace ^ length
*Timing attack = tokens * length (* accuracy)*

An example with a keyspace of 25 characters and a length of 16 requires only 400 attempts when assuming perfect accuracy. Compared to brute forcing which requires 23 283 064 365 386 962 890 625 attempts. The number of attempts required in timing attacks may also be reduced by using chosen hash-prefixes or determining the hash from its prefix in a table.

Practical

One of the major drawbacks with GNS3 and virtualization is that all virtualization techniques is adding a lot of overhead. This overhead takes CPU processing time that a program/application instead could use. Most of the CPU virtualization overhead translates into a reduction of the overall performance of a server.[6] If Qemu based appliances are being used, such as GNS3, nested virtualization technique is preferred for better performance and throughput because the use of leverage KVM (Kernel-based Virtual Machine).[7]

A better prepared attacker would utilize special hardware for measuring time, high precision real-time clocks and a real time operating system.

Prevention

Defending against side-channel attacks, partially timing attacks should be part of any comprehensive effort to improve and deal with the known security flaws. By looking at the prevention techniques of what makes the attack possible and correct them is the primary solution to those attacks. Built-in methods to check the validation between user's input and the stored password value should not be used. This include Java's equals method or a simple, ==, operator which does an early exit on the first mismatching character.[8]

The difference between a secure comparison and an insecure is demonstrated in the Appendix 1 and 2. Another way to perform a secure comparison is to use key blinding, its purpose is to mask the actual values being compared while still producing timing information. It is preferable to prevent timing information leakage rather than masking.

An inefficient method that should not be employed in order to prevent a timing attack is the generation of 'random' noise. This is because cryptographically secure and true random number generators are more costly and error prone to implement. Another issue is that a defendant may add jitter that is constant or too far from the timing data. For example if a random noise of 200-1000 ns was added to each request the plateaus would have been bigger. The more granular measurements of 1-25 ns would be unaffected and no security was achieved that day.

Other techniques can be used such as denying root access through SSH from the outside, untrusted network. Creating public key logins for SSH would also benefit in securing the devices, as a public/private key pair is then used for authentication. For Cisco IOS, a login block-for statement can be added to deny frequent login failures from outside and/or internal hosts. Using access lists and firewall rules can block of untrusted hosts that could cause malicious activity on the network.

Another technique are password policy, that includes best practices for user accounts and their passwords. Password policy should be used for properly securing a system account, for example accounts with usernames, mysql, oracle, root, and admin. It should also specify acceptable reuse of old passwords and a minimum complexity and length of those passwords. The passwords must be strong enough to withstand brute-force attempts and other known attack methods.[9]

References

[1] Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems
cryptography.com/timingattack

[2] Performance Optimization and Tuning Techniques for IBM Power Systems Processors Including IBM Power8
redbooks.ibm.com/redbooks/pdfs/sg248171.pdf

[3] Introduction to Side Channel Attacks
gauss.ececs.uc.edu/Courses/c653/lectures/SideC/intro.pdf

[4] Side-channel timing tool
github.com/chilimannen/sidechannel-timing-tool

[5] Cisco 2014 - Annual Security Report
cisco.com/web/offer/gist_ty2_asset/Cisco_2014_ASR.pdf

[6] Performance Implications of CPU Virtualization
pubs.vmware.com/vsphere-55/index.jsp?topic=%2Fcom.vmware.vsphere.resmgmt.doc%2FGUID-9986F393-4D36-48AB-B839-8FF25ECEEF94.html

[7] Using the GNS3 VM
gns3.com/support/docs/-what-is-the-gns3-vm

[8] Login Timing Attacks For Mischief and Mayhem
security-assessment.com/files/documents/presentations/TimingAttackPresentation2012.pdf

[9] Observations of Login Activity in an SSH Honeypot
http://www.cisco.com/c/en/us/about/security-center/ssh-honeypot.html

[Figure 1] https://www1.cs.fau.de/filepool/sketch-timing-attack.png
[Figure 2] Local comparison - Leaking Data
[Figure 3] Local comparison - Time-constant Not Leaking
[Figure 4] Telnet from Ubuntu Server 15.10 VM to GNS3 Cisco IOS

Fetched 26/02-2016

Appendix

Appendix 1 - Unsafe implementation of password/token comparison.

```
public void authenticate(String match) {
     boolean equals = password.equals(match);

     if (equals)
       // do something to prevent the method from being removed.
  }
```

Appendix 2 - Safe and OpenSSL approved comparison.

```
  public void authenticate(String match) {
    int equals = 0;

    if (match.length() != password.length())
       return;

     for (int i = 0; i < match.length(); i++) {
        equals |= match.charAt(i) ^ password.charAt(i);
     }

     if (equals == 0)
        // do something to prevent the method from being removed.
  }
```