Time Trial
Racing Towards Practical Remote Timing Attacks

Daniel A. Mayer
@DanIAMayer
http://cysec.org

Joel Sandin
jsandin@matasano.com
Who we are...

Daniel A. Mayer
- Senior Appsec consultant with Matasano Security.
- Ph.D. in Computer Science (Security and Privacy).

Joel Sandin
- Appsec consultant with Matasano

Matasano Security
- Application Security Consultancy.
- Offices in New York, Chicago, Sunnyvale.
- Part of
Agenda

1. Timing Side-Channels
2. Remote Timing Attacks
3. Our Tool: Time Trial
4. Timing Attacks in Practice
5. Conclusion
Side-Channels
Side-Channel Attacks
Side-Channel Attacks
Examples of Side-Channels

- Power consumption
- RF emissions
- Sound
- Processing Time

- Really, anything that can be measured and is related to a secret.
“Regular Vulns” vs. Side-Channels

- Many vulnerabilities well understood
  - XSS, CSRF, SQL injection
  - Developers becoming more aware
  - Frameworks: Harder to introduce bugs

- Side-channels: Less so
  - Easy to introduce using “innocent” operators
  - Hard to observe and test for
  - Have to go out of one’s way to prevent them
Timing Side-Channels

- Response time differs depending on computation
- Attacker can learn information about system
  - sensitive credentials
  - internal system state
- Easy to introduce
- Exploitable remotely?
Timing Side-Channels

- Exploitable remotely?
post '/login' do
  if not valid_user?(params[:user])
    "Username or Password incorrect"
  else
    if verify_password(params[:user], params[:password])
      "Access granted"
    else
      "Username or Password incorrect"
    end
  end
end
Timing Attacks

- Reason about system based on response time

### Diagram

- **Attacker**
  - Start
  - POST /login with username, password
  - Invalid user: $t_0$
  - Valid user: $t_1$

- **Target**
  - Username or Password incorrect
  - valid_user?
    - Username or Password incorrect
  - verify_password
    - Access granted
Prior Work!

- Rich history of timing attacks in crypto, e.g.
  - **Kocher**, 1996
    *Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems*
  - **Brumley and Boneh**, 2005
    *Remote Timing Attacks are Practical*

- Excellent empirical studies, e.g.
  - **Crosby et al.**, 2009
    *Opportunities and Limits of Remote Timing Attacks*
  - **Lawson and Nelson**, 2010
    *Exploiting Timing Attacks In Widespread Systems*
Remote Timing Attacks
Local vs. Remote - Challenges

- **Local attacks**
  - Precise measurement of execution time
  - Can minimize external influences

- **Remote attacks**
  - Propagation time added to the measurement.
  - Network delays add jitter.

### Diagram

- **Signal**
- **Jitter**
- **Measured**

---

Daniel A. Mayer and Joel Sandin » Time Trial
Real Jitter

Additional Caveat:

- Distribution isn’t Gaussian, hard to model
- Skewed, multiple modes
Statistical Methods

- Measure a large number of response times
- **Measurement must be related to processing time!**
- Median and minimum not good indicators

---

**Graph Description:**
- **X-axis:** Server Processing Time
- **Y-axis:** Measured Response Time
- **Graph Title:** Based on Crosby et al.
- **Lines:**
  - Green line: 1 to ~20%
  - Red line: Minimum (0%)

---

*Daniel A. Mayer and Joel Sandin » Time Trial*
The Box Test

Compare intervals induced by percentiles

Percentiles to be determined empirically
Statistical Methods

- The Box Test
- Compare intervals induced by percentiles
- Percentiles to be determined empirically
Statistical Methods

- The Box Test
- Compare intervals induced by percentiles
- Percentiles to be determined empirically
Statistical Methods

- The Box Test
- Compare intervals induced by percentiles
- Percentiles to be determined empirically
New Tool: Time Trial
Why a tool for timing attacks?

- No way to demonstrate impact
- Separate theoretical issues from exploitable vulnerabilities
- Reframes the debate about practicality of these attacks
Time Trial

What Time Trial is:
- A framework for capturing precise timing
- A tool for feasibility analysis
- A generator of visual proof-of-concepts

What Time Trial is NOT (yet):
- A read-to-use exploit framework
- An automated attack tool
Goals and Design

- Separate “racer” sensor from analytic front end.
  - Front end: Python + Qt
  - Racer: C++
- Schedule trials and analyze results

1. Define Trials in the GUI
2. Redis-Backed RQ
3. Racer Executes
4. Results in Queue
5. View/Analyze Results
How to do precise time measurements?
How to do precise time measurements?
How to do precise time measurements?
Optimizations

- Use `clock_gettime` for nanosecond timer
  - Using MONOTONIC clock

- Used fixed, reserved CPU core
  - `GRUB_CMDLINE_LINUX_DEFAULT="maxcpus=2 isolcpus=1"`
  - CPU affinity

- Run with real-time priority

- Disable frequency scaling
DEMO: Time Trial
Lets get some data!
Data across different networks

- Analyzed response time distributions for different networks:
  - LAN
  - Internet at large
  - Cloud environments

- In order to exploit: distinguish response times.
  - Was the response $t_0$ or $t_1$ for given input?
Feasibility Based on Echo Trials

- What timing differences can be distinguished in practice?
  - Similar to the approach by Crosby et al.

![Diagram]

- Attacker
- Target
- Start
- "1,000"
- measured time
- "1,000"
- sleep for 1,000 ns
Timing Resolution: LAN

1,000 Repetitions
Timing Resolution: LAN

1,000 Repetitions
Timing Resolution: LAN

1,000 Repetitions

100 μs
110 μs
Timing Resolution: LAN

1,000 Repetitions
Timing Resolution: LAN

1,000 Repetitions

10,000 Repetitions

100,000 Repetitions
Timing Resolution: LAN Limit

- 100 ns difference clear
- < 100 ns inconsistent

![Graph showing frequency distribution of time measurements with 100 ns and 200 ns resolution. The graph has a y-axis labeled "Frequency" and an x-axis labeled "Time / [us]". The graph shows two peaks, one for 100 ns and one for 200 ns, with the 100 ns peak being much narrower. The number 1,000,000 is displayed at the bottom of the slide.]

Daniel A. Mayer and Joel Sandin » Time Trial
Better than 30 ns
Timing Resolution: WAN Limit

100,000
Timing Resolution: EC2 Limit

100,000
Timing Resolution: EC2 Limit

![Graph showing the comparison between 100 ns and 200 ns timing resolutions. The x-axis represents Time in microseconds (μs), and the y-axis represents Frequency. Two peaks are visible: one at approximately 360 μs for 100 ns and another broader peak for 200 ns.]

100,000
## Overview of Results

<table>
<thead>
<tr>
<th></th>
<th>1 ms</th>
<th>1 µs</th>
<th>100 ns</th>
<th>&lt; 100 ns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loopback</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>LAN</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>❌</td>
</tr>
<tr>
<td>EC2</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>❌</td>
</tr>
<tr>
<td>WAN</td>
<td>✔</td>
<td>✔</td>
<td>❔</td>
<td>❌</td>
</tr>
</tbody>
</table>
Impact on Real-world Applications

WHAT DOES THAT

EVEN MEAN?
Timing Attacks in Practice
String comparison

- Most string comparison return early
  - Leaks timing information about which byte differed

![String Comparison Diagram]

Valid Credential: e993

<table>
<thead>
<tr>
<th>Attacker</th>
<th>Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>e9aa</td>
</tr>
<tr>
<td>t_0</td>
<td>Credential invalid</td>
</tr>
<tr>
<td>t_1</td>
<td>Credential invalid</td>
</tr>
</tbody>
</table>

```
# Valid Credential: e993

$cred = \text{'e993'}$
```

```
if $cred[i] != input[i]$
    $i++$
```

```
# Start: t_0
# e9aa == e993
```

```
# t_1
# Credential invalid
```

```
e == e
9 == 9
a == 3
```

String comparison

- Introduced when attacker-controlled data is compared to a secret

- Commonly prone to timing attacks:
  - HMACs (e.g., session state)
  - Web API keys
  - OAuth token checks
  - Middleware authentication

- Exploitable remotely?
String Comparison: Conclusions

- Most 64-bit OSes compare 8 bytes at a time!
Internet of Things

- BeagleBone Black: 1 GHz ARM Cortex-A8
  - Java benchmarks put it within reach, exit on first byte:
## Microbenchmarks (in nanoseconds)

<table>
<thead>
<tr>
<th>Language</th>
<th>Function</th>
<th>Lawson 2010*</th>
<th>i5-3210M 2.50GHz</th>
<th>Cortex-A8 1GHz</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>per byte</td>
<td>per word</td>
<td>per byte</td>
</tr>
<tr>
<td>C</td>
<td><code>memcmp</code></td>
<td>0.719</td>
<td>0.243</td>
<td>1.37</td>
</tr>
<tr>
<td>C</td>
<td><code>strcmp</code></td>
<td>-</td>
<td>0.41</td>
<td>4.04</td>
</tr>
<tr>
<td>Ruby</td>
<td><code>str ==</code></td>
<td>0.840</td>
<td>0.36</td>
<td>1.75</td>
</tr>
<tr>
<td>Python</td>
<td><code>str ==</code></td>
<td>1.400</td>
<td>0.224</td>
<td>1.48</td>
</tr>
<tr>
<td>Java</td>
<td><code>String.equals</code></td>
<td>40.594</td>
<td>7.65</td>
<td>18.91</td>
</tr>
</tbody>
</table>

- Resolution < differences of multiple bytes
- **Remote exploitation highly unlikely in practice!**

* Hardware: AMD Athlon X2 2.7 GHz
Branching

- Different code path based on secret state
- Timing difference depends on application
- Which operation performed in each code path?

Diagram:
- valid_user?
- verify_password

Flowchart:
- Start
- $t_0$
- $t_1$
- POST /login with username, password
- Username or Password incorrect
- Access granted
Branching

- User enumeration (SHA-256)
  - (Not a SHA-256 attack!)

LAN

WAN

Daniel A. Mayer and Joel Sandin » Time Trial
**Time-Based Padding Oracle**

- AES CBC Padding Oracle
- Distinguish
  - Wrong Padding
  - Other Processing Error

![Diagram of Time-Based Padding Oracle](image)

- **GET /decrypt?data=[encrypted data]**
- **t_0**: An error occurred.
- **t_1**: Data processed successfully.
- **decryption_successful?**
- **process_data**
Time-Based Padding Oracle

- Perform SQLite query when decrypt successful
  - Actual difference depends on application!
DEMO: Time-Based CBC Padding Oracle

Hacker shields on full power

I'll use some magic
Take Away: Microbenchmarks

- Computing performance continues to improve
  - Comparison-based vulnerabilities difficult to exploit.

- Branching-based often feasible

- Embedded systems at greater risk
  - Java on ARM a feasible target
  - Attacking string-comparison on Arduino realistic.
Preventing timing attacks

- Ensure sensitive operations take constant time
  - Analyze for branching side-channels
  - This is hard!

- Use constant time comparison functions
  - See our white paper

- Best practices
  - Throttle or lock out misbehaving clients
  - Monitor for failed requests
Future Plans

- More empirical studies
- Implement (feasible!) attacks
- Jitter changes over time
  - Alternate long and short measurements

Send bug reports, feature / pull requests!
Thanks!

Questions?

https://github.com/dmayer/time_trial

http://matasano.com/research/

Daniel A. Mayer
mayer@cysec.org
@DanlAMayer

Joel Sandin
jsandin@matasano.com