Unsafe Time Handling In Smartphones

Abhilash Jindal
Y. Charlie Hu

Prahlad Joshi
Samuel Midkiff
Smartphones are Battery Constrained

- Increasing hardware/software functionality
- Limited form factor and weight
Paradigm Shift in Power Management: Aggressive Sleeping Policy

- **Desktop/Server:** CPU Default ON
  - CPU turned off when idle for *long* time

- **Smartphones:** CPU Default OFF
  - Smartphone OSes aggressively turn off Screen/CPU after *brief* user inactivity
  - Helps increasing standby time period
Time Induced Critical Sections

```c
public double do_memcpy ( memcpy_t fn, size_t len, .. ) {
    ...
    t1 = getTime ( );
    fn (dst, src, len);
    t2 = getTime ( );
    t_diff = (t2 – t1);
    return len / t_diff;
}
```

From tools/perf/bench/mem-memcpy.c
public double do_memcpy ( memcpy_t fn, size_t len, .. ) {
    ...
    t1 = getTime ( );
    fn (dst, src, len);
    t2 = getTime ( );
    t_diff = (t2 − t1);
    return len / t_diff;
}

From tools/perf/bench/mem-memcpy.c
Sleep Induced Time Bugs (2)

• SITB happens when the smartphone CPU/SOC is suspended in the middle of a time manipulation
  • Alters intended program behavior

• Hard to reproduce
  • Will only happen when CPU sleeps when the code execution is between time manipulation

“I think it will fix an odd issue I have seen in a log file (apparently was completely off track debugging it). As this very likely is a real world issue, I’d recommend applying the patch to the fixes branch”

– Android kernel developer
Power Control API-- Wakelocks

Foo(...){
    wake_lock ( ... )
    time manipulation
    wake_unlock ( ... )
}

- CPU is suspended only after last wakelock is released
Outline

• Sleep Induced Time Bugs

• Categorizing Time Usages and Vulnerabilities to SITB
  • Case 1 : Timed Callback
  • Case 2 : Time Setting
  • Case 3 : Time arithmetic
  • Case 4 : Logging

• Klock Design

• Evaluation
Time Usage In Android

• Collected list of time related APIs exposed at each software layer and grepped them

<table>
<thead>
<tr>
<th></th>
<th>Kernel</th>
<th>Android Framework</th>
<th>978 Apps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time usages</td>
<td>1072</td>
<td>1737</td>
<td>7798</td>
</tr>
</tbody>
</table>

• Usages belong to four categories
  • Timed Callback, Time setting, Time arithmetic, Logging
Case 1: Timed Callback

• Code wishes to perform a certain task at a future time
  • Register alarm with system specifying a callback function and a time interval

  drivers/serial/msm serial.c

```c
void msm_serial_clock_request_off(.., int timeout){
  clk_off_timer.function = msm_serial_clock_off;
  hrtimer_start(clk_off_timer, timeout );
}
```

• **Vulnerability:** CPU suspension before timer callback finishes alters intended semantics
Case 2: Time setting

- Code updates current system time

```java
void setTimeFromNITZString( .. ) {
    nitz = getTime ( );
    /* some processing */
    c = f ( nitz );
    setAndBroadcastNetworkSetTime ( c );
}
```

From com/android/internal/telephony/gsm/GsmServiceStateTracker.java
Case 2: Time setting vulnerability

• Code updates current system time

```java
void setTimeFromNITZString( .. ) {
    nitz = getTime ( );
    /* some processing */
    c = f ( nitz );
    setAndBroadcastNetworkSetTime ( c );
}
```

From com/android/internal/telephony/gsm/GsmServiceStateTracker.java

CPU sleeps before setTime would set stale time
Case 3: Time arithmetic

• Code collects two timestamps and performs arithmetic on them

```c
public double do_memcpy ( memcpy_t fn, size_t len, .. ) {
  ...
  t1 = getTime ( );
  fn (dst, src, len);
  t2 = getTime ( );
  t_diff = (t2 – t1);
  return len / t_diff;
}
```

```
6000/3 = 2000 MBps
```

From tools/perf/bench/mem-memcpy.c
Case 3: Time arithmetic vulnerability

CPU sleeps between obtaining two timestamps

```java
public double do_memcpy (memcpy_t fn, size_t len, .. ) {
    ...
    t1 = getTime ( );
    fn (dst, src, len);
    t2 = getTime ( );
    t_diff = (t2 – t1);
    return len / t_diff;
}
```

6000 MB

Time

0:00

5:00

300

6000/300 = 20 MBps

From tools/perf/bench/mem-memccpy.c
Case 3: Time arithmetic vulnerability (2)

Time is set between obtaining two timestamps

```java
public double do_memcpy (memcpy_t fn, size_t len, .. ) {
    ...
    t1 = getTime ( );
    fn (dst, src, len);
    t2 = getTime ( );
    t_diff = (t2 – t1);
    return len / t_diff;
}
```

Set time

6000 MB

6000 / -60 = -100 MBps

From tools/perf/bench/mem-memcpy.c
Case 4: Time logging

• Code obtains current time and logs it in conjunction with some event
  • Usually for postmortem debugging

• Vulnerability: CPU suspension in between event and its timestamping will result in an incorrect timestamp being logged for the event.
Overview

• Sleep Induced Time Bugs
• Categorizing Time Usages and Vulnerabilities to SITB
• Klock Design
  • Primer on Reaching definition, UD and DU chains
  • Identifying Protected Statements
  • Identifying Time Critical Sections
  • Implementation

• Evaluation
• Conclusion
Reaching Definition DataFlow problem

\[
y = 3; \\
x = 10; \\
y = 11; \\
\text{if}(y > 5) \{ \\
    x = 1; \\
    y = 2; \\
}\text{else} \{ \\
    z = x; \\
    x = 4; \\
\}\]
Use-Def (UD) Chains

Links each use of variable x to DEF which reach that use

Closure: Recursively following UD chains show all DEFs that impact 1 variable use
Def-Use (DU) Chains

Links each definition of variable x to those USE which that definition can reach.

Closure: Recursively following DU chains show all USEs impacted by 1 definition
Klock Overview

Program source code → Standard Compiler Transformations
- Alias analysis
- CFG construction

Identifying Protected Statements
- Reaching Definition Analysis
- Protection mechanisms

Identifying Time Critical Sections (TICS)
- Case 1: Timer registration APIs
- Case 2: UD chains
- Case 3: DU chains

Compare and report
BUGS = TICS - SAFE
RDA to Identify Protected Statements

```
// Don't let CPU sleep

if (a condition)
    wakelock.acquire();

wakelock.acquire();

wakelock.release();

// CPU is now free to sleep
```
Klock Overview

Program source code

Standard Compiler Transformations
- Alias analysis
- CFG construction

Identifying Protected Statements

Reaching Definition Analysis
Protection mechanisms

Identifying Time Critical Sections (TICS)

Case 1
Timer registration APIs

Case 2: UD chains
setTime APIs

Case 3: DU chains
getTime APIs

Compare and report
BUGS = TICS - SAFE
Identifying Time Critical Section

Case 1: Timer Callback

For every timer registration site
• Find callback function target
• Conservatively add callback function to TICS

```c
void msm_serial_clock_request_off(.., int timeout){
    clk_off_timer.function = msm_serial_clock_off;
    hrtimer_start(clk_off_timer, timeout);
}
```

```c
msm_serial_clock_off (struct hrtimer timer){
    clk_disable(msm_port->clk);
}
```
Identifying Time Critical Section
Case 2: Time Setting

void setTimeFromNITZString(..) {
    nitz = getTime();
    /* some processing */
    c = f(nitz);
    setAndBroadcastNetworkSetTime(c);
}

From com/android/internal/telephony/gsm/GsmServiceStateTracker.java

For all statements, where time is set
• Recursively find DEFS using UD chains
• Add all paths from DEFS to set time into TICS
Identifying Time Critical Section
Case 3: Time Arithmetic

public double do_memcpy ( memcpy_t fn, size_t len, .. ) {
    ...
    t1 = getTime ( );
    fn (dst, src, len);
    t2 = getTime ( );
    t_diff = (t2 - t1);
    return len / t_diff;
}

For all definitions that get time
• Find closure of USES using DU chains

If a statement has variables from two different closures (t1, t2)
• Must be arithmetic between t1, t2
• Add all statements between getting t1 and getting t2 to TICS
Implementation

• Built on LLVM compiler infrastructure
  • 1 custom pass to build call graph
  • 4 custom passes for identifying protected statements and identifying TICS (case 1, 2, 3)

• ~5 KLOC

• Available at http://github.com/klock-android
Overview

• Sleep Induced Time Bugs
• Categorizing Time Usages and Vulnerabilities to SITB
• Klock Design
• Evaluation
Evaluation

• Ran Klock on 5 different kernel versions
  • Nexus 1, Nexus 7, Nexus 10, Nexus S and x86 (with wakelocks enabled)

• Found 63 bugs
  • 4 timed callback bugs, 0 time setting bugs, 59 time arithmetic bugs
  • 14 have been fixed, 7 files have been removed in newer kernel versions
  • Out of 42 remaining, 7 developers replied so far confirming the bugs
Bugs (63) breakdown

• Correctness related bugs (18)
  • 6 drivers incorrectly measure pulse width hence reading wrong received data
  • 5 radio drivers incorrectly measure clock rate necessary to decode the incoming data
  • 7 other miscellaneous bugs

• Performance related bugs (15)
  • 8 drivers spin for an extended period of time leaving device unusable
  • 4 code locations call sleep for a long time
  • 3 drivers keep their devices on longer than necessary wasting energy

• Benchmark bugs (30)
False Positives

• 106 False positives

• Suspension does not affect program semantics
  • driver generates a random number using timer arithmetic

• System calls
  • System calls (eg sys_settime) are just wrappers of actual time setting APIs and do not have suspension prevention mechanism
Conclusion

• Sleep Induced Time Bugs
  • Time manipulation form Time Critical Sections
  • CPU suspension during Time Critical Sections lead to the bugs

• Time is widely manipulated in Android Ecosystem
  • Timed callback, Time setting, Time arithmetic and Logging

• Klock
  • Static checker built using reaching definition analysis, UD/DU chains
  • Found 63 bugs in the kernel
  • http://github.com/klock-android