KeyDrown: Eliminating Software-Based Keystroke Timing Side-Channel Attacks

M. Schwarz, M. Lipp, D. Gruss, S. Weiser, C. Maurice, R. Spreitzer, S. Mangard

Feb 20, 2018—NDSS'18

Graz University of Technology

- Keystroke timing attacks infer typed words, passphrases or create user fingerprints
- 2 new attacks to recover keystroke timings
- Build an effective countermeasure

Background

Keystroke Timing Attacks

- Acquire accurate timestamps of keystrokes for input sequences
- Depend on bigrams, syllables, words, keyboard layout and typing experience
- Exploit timing characteristics to learn information about the user or the input
  - Infer typed sentences
  - Recover passphrases

Keystroke Timing Attacks

- Many ways to obtain keystroke timings have been presented:
  - SSH leaks inter-keystroke timings in interactive mode [Son+01]
  - Network latency with significant traffic [Hog+01]
  - Instruction and stack pointer, interrupt, network packet statistics [Zha+09]
  - CPU usage [Jan+12]
  - Wi-Fi Signals [Ali+15]
  - `/proc/interrupts` [Dia+16]
  - JavaScript Sensor API [Meh+16]
  - Cache attacks

Cache Attacks: Summary

- Cache side-channel attacks allow to monitor cache accesses by a victim process
- Measuring access time of an address, one can infer if the address is cached or not
  - Flush+Reload: Shared memory between victim process and attacker process
  - Prime+Probe: Applicable if no shared memory available, more noisy

- Processing a keystroke involves computations on all levels of the software stack
- Multiple possibilities to observe the input

Interrupt-timing attacks

- Idea: Continuously acquire a high-resolution timestamp and monitor differences between subsequent timestamps

- Idea: Continuously acquire a high-resolution timestamp and monitor differences between subsequent timestamps
- Requires unprivileged code execution and an accurate timing source (e.g., `rdtsc`)

```
1 int now = rdtsc();
2 while (true) {
3   int last = now;
4   now = rdtsc();
5   if ((now - last) > threshold) {
6     reportEvent(now, now - last);
7   }
8 }
```

```
1  int now = rdtsc();
2  while (true) {
3    int last = now;
4    now = rdtsc();
5    if ((now - last) > threshold) {
6      reportEvent(now, now - last);
7    }
8  }
```

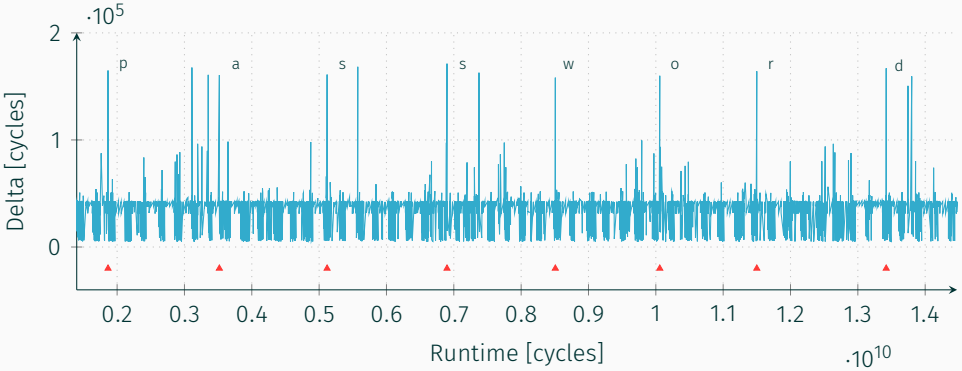
- Look at how much time has passed since the last measurement

```
1  int now = rdtsc();
2  while (true) {
3    int last = now;
4    now = rdtsc();
5    if ((now - last) > threshold) {
6      reportEvent(now, now - last);
7    }
8  }
```

- Look at how much time has passed since the last measurement
  - Significant differences occur when the process is interrupted

```
1  int now = rdtsc();
2  while (true) {
3    int last = now;
4    now = rdtsc();
5    if ((now - last) > threshold) {
6      reportEvent(now, now - last);
7    }
8  }
```

- Look at how much time has passed since the last measurement
  - Significant differences occur when the process is interrupted
  - More time the operating system consumes to handle the interrupt
    $\rightarrow$ higher timing difference

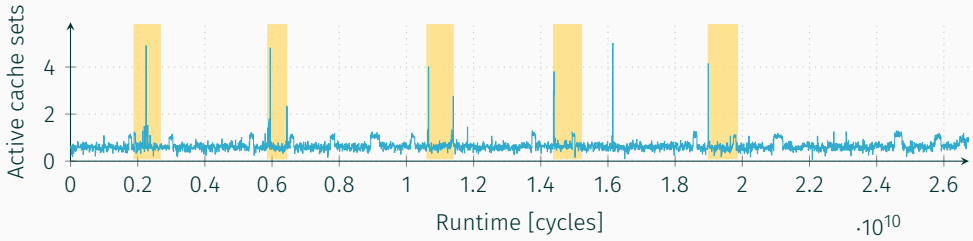

·10^5
2
1
0
Delta [cycles]
p
a
s
s
w
o
r
d
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
1.1
1.2
1.3
1.4
Runtime [cycles]
·10^10

Multi-Prime+Probe Attack on the Kernel

Multi-Prime+Probe Attack on the Kernel

- Attack keyboard interrupt handler within the kernel
- Observing cache activity in the cache sets used by the interrupt handler
- Reduce influence of system noise by combining simultaneous Prime+Probe attacks on different addresses
- First highly accurate keystroke timing based on Prime+Probe on the last-level cache

Multi-Prime+Probe Attack on the Kernel



Building a counter measure

Attack Model

- Unprivileged code execution on a recently updated system
- Continuously monitor a side-channel to obtain traces for all user input
  - Generally only a single trace for a user input sequence
  - Multiple traces for password input

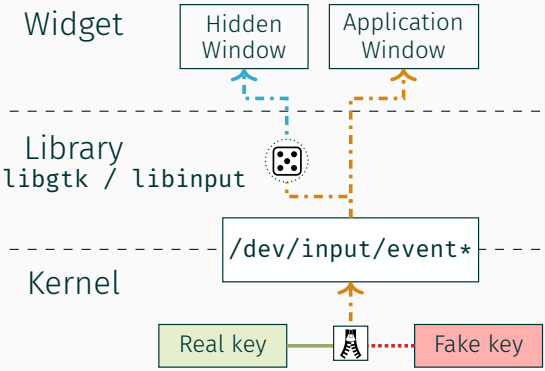
Requirements

- R1: Minimize Side Channel Accuracy
  - Keystroke timing attacks require a high precision and a high recall
- R2: Reduction of Statistical Characteristics in Password Input
  - An attacker can combine information from multiple traces (password input) and exploit statistical characteristics
  - Number of required traces must be impractically high
  - Studies estimate that users have 1-5 different passwords, enter 5 passwords per day on average and 56% change passwords in 6 months: 913 traces
- R3: Implementation Security
  - Implementation of the countermeasure itself could leak side-channel information
- If all requirements are met, a side-channel attack in the presence of the counter measure is not practical anymore

Simplistic approach

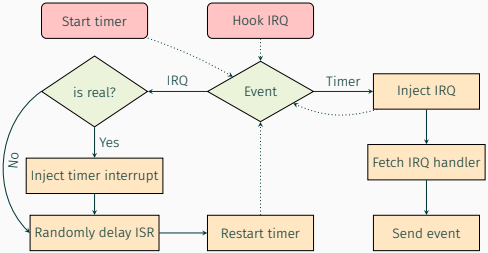
- Disallow unprivileged access to statistics/APIs?
  - ⇒ Different side-channels exist
- Restrict high-resolution timers (*rdtsc*/*perf*)?
  - ⇒ One can build his own timers
- Instead: Adding noise by injecting fake key strokes

KeyDrown



- Multi-layered countermeasure
- Injecting fake key strokes at the root traveling through the entire software stack

- Protection against interrupt-based attacks and timing-based attacks by artificially injecting interrupts
  - Real interrupt replaces one fake interrupt within a multiple of fake interrupts
  - Implementation ensures that interrupt density uniform over time and, thus, independent of real interrupts



- Implemented as a kernel module that handles hardware interrupts from the input device and timer interrupts:
  - Timer interrupts, it injects a keyboard interrupt
  - Keyboard interrupt, it injects a non-periodic one-shot timer with a random delay
- For real and fake keystrokes, both interrupts occur

Second Layer: Protecting libraries

- Protects library handling the user input against Flush+Reload and Prime+Probe attacks
  - Real and injected keystrokes should have the same code paths and data accesses to the library
  - Injected keys are valid, but are typically unused keys
    - Might not have the same code path within the library
  - Duplicate key, randomize its value and send it to a hidden window

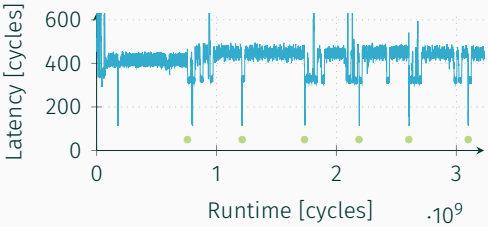
Third Layer: Protecting password entry

- Protect the actual password entry
- Generating cache activity on the cache lines that are used by the password's buffer
- Access the buffer whenever a real or fake keystroke is received
- Mitigates Prime+Probe attacks on the buffer

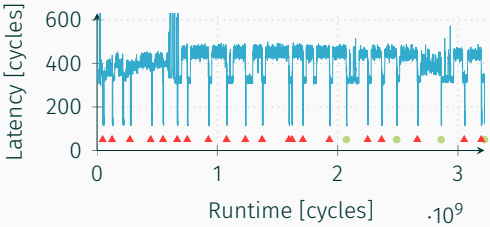
Evaluation

- Uniform key-injection interval [0,20ms]
- Any real key replaces the currently scheduled key injection
- Distribution of real keystrokes in these 20ms are uniform
- Uniform interrupt density function with 100 events per second

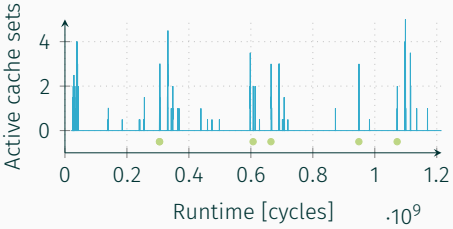
- Compare results to a always-one oracle and a random-guessing oracle
  - Random guessing oracle: F-Score 0.14
  - Always-one oracle: F-Score 0.15
- If side-channel yields an F-Score of this value or below, provides no useful information



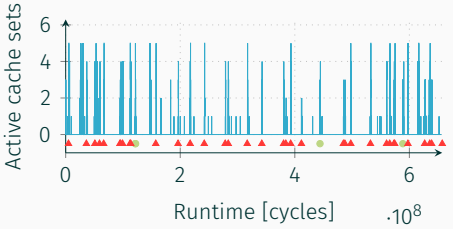
Without Keydrown, F-Score 0.99



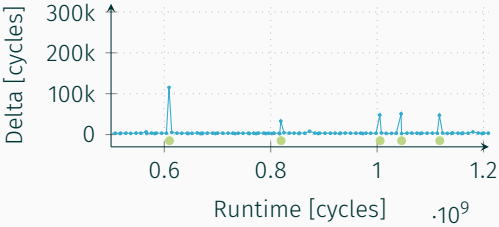
With Keydrown, F-Score 0.09



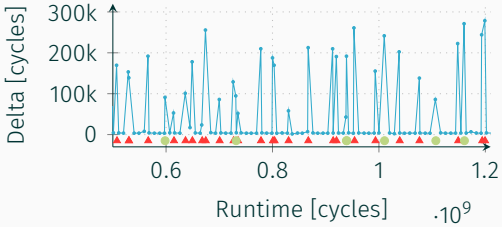
Without Keydrown, F-Score 0.81



With Keydrown, F-Score 0.11



Without Keydrown, F-Score 0.94



With Keydrown, F-Score 0.14

R2: Combining multiple traces

- Model powerful attacker:
  - Noise-free side-channel
  - Perfect (re-)alignment
  - Known length of the password
- Far stronger than a practical attacker

R2: Combining multiple traces

- Simulated typing variance $\pm$40ms (bit less than reported By Lee et al) for trained text sequences
- Generated 300 000 traces
  - Containing 8 keystrokes within 2 seconds
- How many randomly chosen traces have to be combined to extract the correct positions of keystrokes
  - 2458 traces, more than deemed to be secure in R2

- First layer
  - Runs in the kernel and can only be attacked by Prime+Probe
  - In general, execution flow and data accesses should be the same
  - For few derivations, we perform same memory accesses for non-executed paths
- Second layer
  - User space binary and could theoretically be attacked by Flush+Reload
  - Second layer does not know whether an event is generated from a real or injected keystroke $\Rightarrow$ Attacker cannot learn additional information
- Third layer
  - Third layer relies on the same source as second layer
  - Cache activity stays the same

Conclusion

Conclusion

- New interrupt-based attack and the first Multi-Prime+Probe attack on kernel interrupt handlers
- KeyDrown, a new defense mechanism injecting random keystrokes
  - Performance overhead of 2.5% (PARSEC 3.0) and 6.9% (lmbench)
  - Battery consumption increase by 4.6%
- Attackers cannot distinguish fake from real key strokes on commodity notebooks and smartphones
- KeyDrown eliminates any advantage an attacker can gain using software-based side-channel attacks

KeyDrown: Eliminating Software-Based Keystroke Timing Side-Channel Attacks

M. Schwarz, M. Lipp, D. Gruss, S. Weiser, C. Maurice, R. Spreitzer, S. Mangard

Feb 20, 2018—NDSS'18

Graz University of Technology