Practical Keystroke Timing Attacks in Sandboxed JavaScript

M. Lipp, D. Gruss, M. Schwarz, D. Bidner, C. Maurice, S. Mangard

Sep 11, 2017—ESORICS'17

Graz University of Technology

Motivation

- Keystroke timing attacks infer typed words, passphrases or create user fingerprints

- Keystroke timing attacks infer typed words, passphrases or create user fingerprints
  - Typically require native code execution

Motivation

- Keystroke timing attacks infer typed words, passphrases or create user fingerprints
  - Typically require native code execution
- First JavaScript-based keystroke timing attack

Motivation

- Keystroke timing attacks infer typed words, passphrases or create user fingerprints
  - Typically require native code execution
- First JavaScript-based keystroke timing attack
- Build classifiers to detect visited websites or to identify users and a covert channel

Motivation

- Keystroke timing attacks infer typed words, passphrases or create user fingerprints
  - Typically require native code execution
- First JavaScript-based keystroke timing attack
- Build classifiers to detect visited websites or to identify users and a covert channel
- Runs in the background and can monitor on other tabs and applications

Background

- Acquire accurate timestamps of keystrokes for input sequences

Keystroke Timing Attacks

- Acquire accurate timestamps of keystrokes for input sequences
- Depend on bigrams, syllables, words, keyboard layout and typing experience

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

- 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]

Interrupt-timing Attacks

- Idea: Continuously acquire a high-resolution timestamp and monitor differences between subsequent timestamps [Sch+17]

- Idea: Continuously acquire a high-resolution timestamp and monitor differences between subsequent timestamps [Sch+17]
- Requires unprivileged code execution and an accurate timing source (e.g., `rdtsc`)

Interrupt-timing Attacks

```
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  }
```

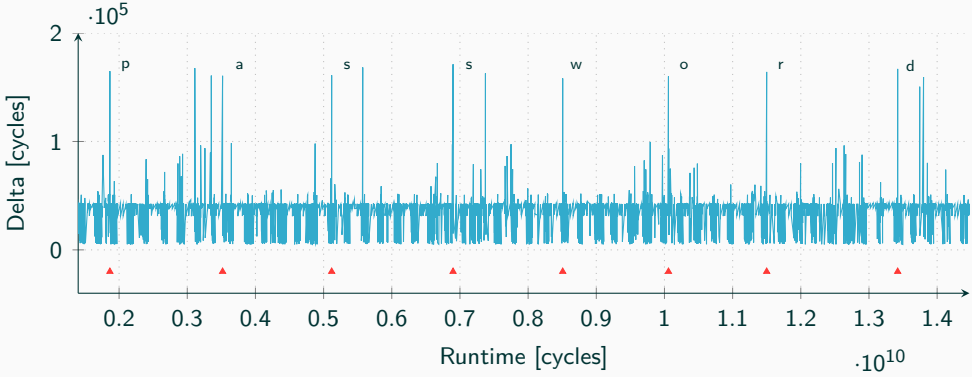
- Look at how much time has passed since the last measurement

```
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
    → 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

- High Resolution Time API (`performance.now`)

- High Resolution Time API (`performance.now`)
- Utilized to mount various attacks:
  - Page deduplication [Gru+15]
  - Cache attacks [Ore+15]

- High Resolution Time API (`performance.now`)
- Utilized to mount various attacks:
  - Page deduplication [Gru+15]
  - Cache attacks [Ore+15]
- W3C standard now recommends a resolution of 5 µs

Sandboxed Keystroke Timing Attacks without High-Resolution Timers

- Two phases:

- Two phases:
  - *Online phase*: Acquire timing traces

- Two phases:
  - *Online phase*: Acquire timing traces
  - *Offline phase*: Post-processing and evaluation

Online phase

- How can we mount the attack in JavaScript?

Online phase

- How can we mount the attack in JavaScript?
  - Native instruction (`rdtsc`) not available

Online phase

- How can we mount the attack in JavaScript?
  - Native instruction (`rdtsc`) not available
  - `performance.now` limited resolution

Online phase

- How can we mount the attack in JavaScript?
  - Native instruction (`rdtsc`) not available
  - `performance.now` limited resolution
- Implement a monotonic clock

Online phase

- How can we mount the attack in JavaScript?
  - Native instruction (`rdtsc`) not available
  - `performance.now` limited resolution
- Implement a monotonic clock
  - Constantly increment a value

Online phase

- How can we mount the attack in JavaScript?
  - Native instruction (`rdtsc`) not available
  - `performance.now` limited resolution
- Implement a monotonic clock
  - Constantly increment a value
  - Number of increments is proportional to the time the function is scheduled
    - Interrupt → lower increments

Online phase

- Single-threaded event loop

Online phase

- Single-threaded event loop
  - Browsers do not allow endless loops and warn the user

Online phase

- Single-threaded event loop
  - Browsers do not allow endless loops and warn the user
  - `setTimeout`/`setInterval` enforce a minimum pause of 4 ms

Cooperative endless-loop slicing

- Slice endless loop into smaller finite loops

Cooperative endless-loop slicing

- Slice endless loop into smaller finite loops
  - Every loop as an execution time of $\sim$4 ms

Cooperative endless-loop slicing

- Slice endless loop into smaller finite loops
  - Every loop as an execution time of $\sim$4 ms
- Before running the loop, we schedule the next loop with a timeout of 4 ms

Cooperative endless-loop slicing

- Slice endless loop into smaller finite loops
  - Every loop as an execution time of $\sim$4 ms
- Before running the loop, we schedule the next loop with a timeout of 4 ms
- The next slice of the loop is executed immediately after the current slice

Cooperative endless-loop slicing

- Slice endless loop into smaller finite loops
  - Every loop as an execution time of $\sim$4 ms
- Before running the loop, we schedule the next loop with a timeout of 4 ms
- The next slice of the loop is executed immediately after the current slice
- Higher priority events (user inputs) can still be processed $\rightarrow$ browser remains responsive

Cooperative endless-loop slicing

- Slice endless loop into smaller finite loops
  - Every loop as an execution time of $\sim$4 ms
- Before running the loop, we schedule the next loop with a timeout of 4 ms
- The next slice of the loop is executed immediately after the current slice
- Higher priority events (user inputs) can still be processed $\rightarrow$ browser remains responsive
- Minimum timeout is reduced to 1000 ms if the user switches the tab

Cooperative endless-loop slicing

- Slice endless loop into smaller finite loops
  - Every loop as an execution time of $\sim$4 ms
- Before running the loop, we schedule the next loop with a timeout of 4 ms
- The next slice of the loop is executed immediately after the current slice
- Higher priority events (user inputs) can still be processed $\rightarrow$ browser remains responsive
- Minimum timeout is reduced to 1000 ms if the user switches the tab
  - Utilize Web Worker API to execute code in background

Cooperative endless-loop slicing


```
1  function measure_time(id) {
2    setTimeout(measure_time, 0, id + 1);
3    counter = 0;
4    begin = window.performance.now();
5    while ((window.performance.now() - begin) < 5) {
6      counter = counter + 1;
7    }
8    publish(id, counter);
9  }
```

Cooperative endless-loop slicing

```
1  function measure_time(id) {
2    setTimeout(measure_time, 0, id + 1);
3    counter = 0;
4    begin = window.performance.now();
5    while ((window.performance.now() - begin) < 5) {
6      counter = counter + 1;
7    }
8    publish(id, counter);
9  }
```

- Low impact on the system and browser performance

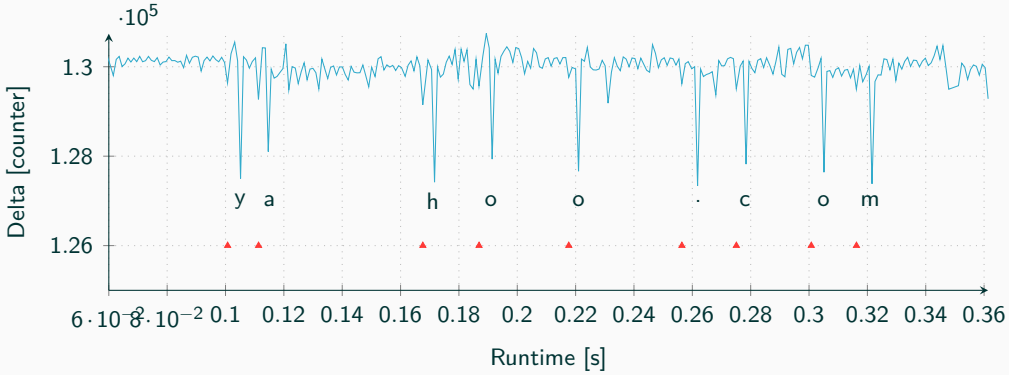
Cooperative endless-loop slicing

```
1  function measure_time(id) {
2    setTimeout(measure_time, 0, id + 1);
3    counter = 0;
4    begin = window.performance.now();
5    while ((window.performance.now() - begin) < 5) {
6      counter = counter + 1;
7    }
8    publish(id, counter);
9  }
```

- Low impact on the system and browser performance
- Less than 256 bytes of code

```
1  function measure_time(id) {
2    setTimeout(measure_time, 0, id + 1);
3    counter = 0;
4    begin = window.performance.now();
5    while ((window.performance.now() - begin) < 5) {
6      counter = counter + 1;
7    }
8    publish(id, counter);
9  }
```

- Low impact on the system and browser performance
- Less than 256 bytes of code
- Can be hidden in modern JavaScript frameworks or online advertisements


$\cdot 10^5$
1.3
1.28
1.26
Delta [counter]
y a h o o · c o m
$6 \cdot 10^{-2}$
$8 \cdot 10^{-2}$
0.1
0.12
0.14
0.16
0.18
0.2
0.22
0.24
0.26
0.28
0.3
0.32
0.34
0.36
Runtime [s]

Offline phase

- Process and analyze traces of the *online phase*

Offline phase

- Process and analyze traces of the *online phase*
  - Filter the measured trace to reduce noise

Offline phase

- Process and analyze traces of the *online phase*
  - Filter the measured trace to reduce noise
  - Detect threshold for keystroke events

- Process and analyze traces of the *online phase*
  - Filter the measured trace to reduce noise
  - Detect threshold for keystroke events
- Features of recorded measurements are strong enough that simple techniques (k-nearest neighbours (KNN)) allow to build an efficient and accurate classifier

Practical Attacks and Evaluation

URL Classification

- Infer URLs a user enters into the browsers address bar
  - Intel i7-6700K and Firefox 52.0

URL Classification

- Infer URLs a user enters into the browsers address bar
  - Intel i7-6700K and Firefox 52.0
- Train a classifier with the input sequences of the top 10 most visited websites

- Small timing variations when the user starts typing and whenever the user presses a key

- Small timing variations when the user starts typing and whenever the user presses a key
- Compute the correlation for different alignments

- Small timing variations when the user starts typing and whenever the user presses a key
- Compute the correlation for different alignments
- Evaluate classifier using k-fold cross-validation

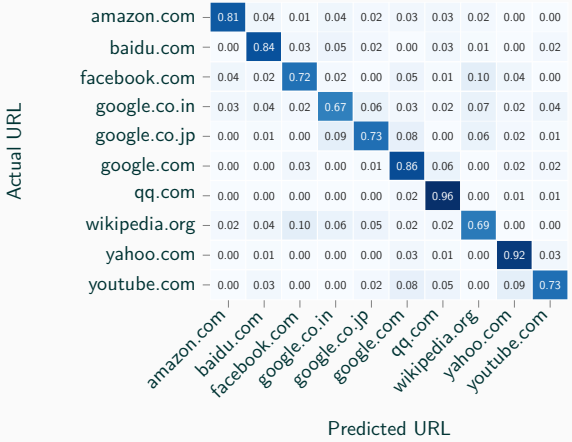


Figure 1: Confusion matrix for URL input.

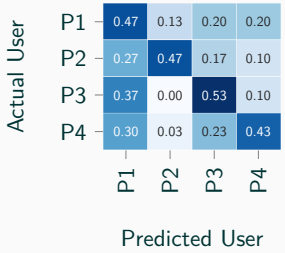


Figure 2: Confusion matrix for input by different users.

- Evaluate attack on mobile devices

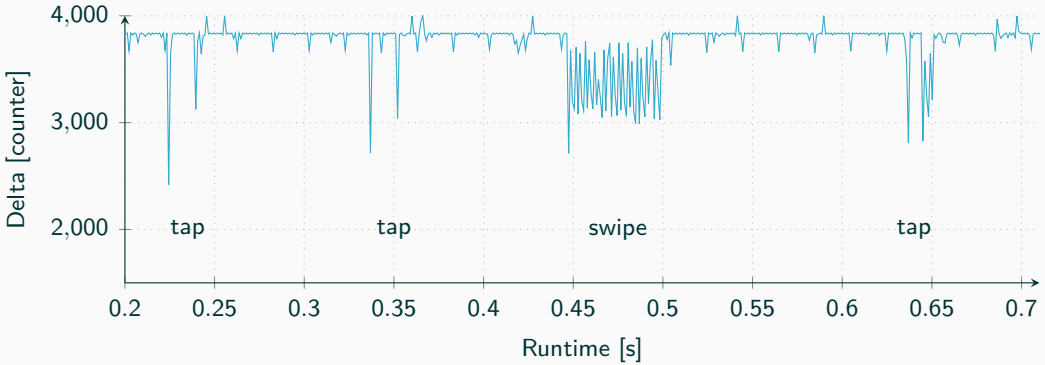


Figure 3: Keystroke timing attack running in a native app on the Google Nexus 5.

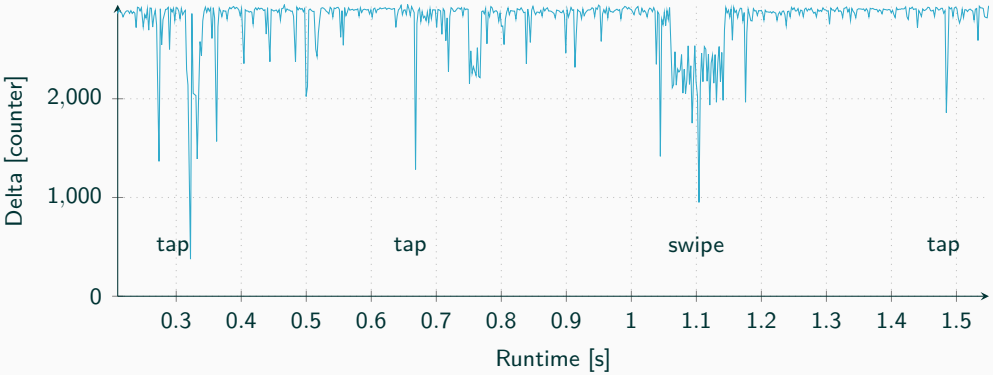


Figure 4: Keystroke timing attack running in JavaScript on the Google Nexus 5.

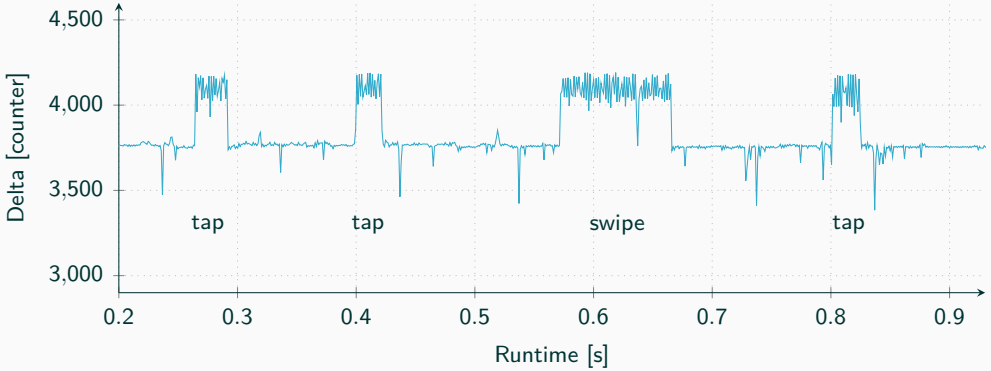


Figure 5: Keystroke timing attack running in JavaScript on the Xiaomi Redmi Note 3.

Spying on other applications and PIN unlock

- Attack allows monitoring of every other event triggering interrupts

Spying on other applications and PIN unlock

- Attack allows monitoring of every other event triggering interrupts
- Allows to monitor keystrokes in different tabs and other applications

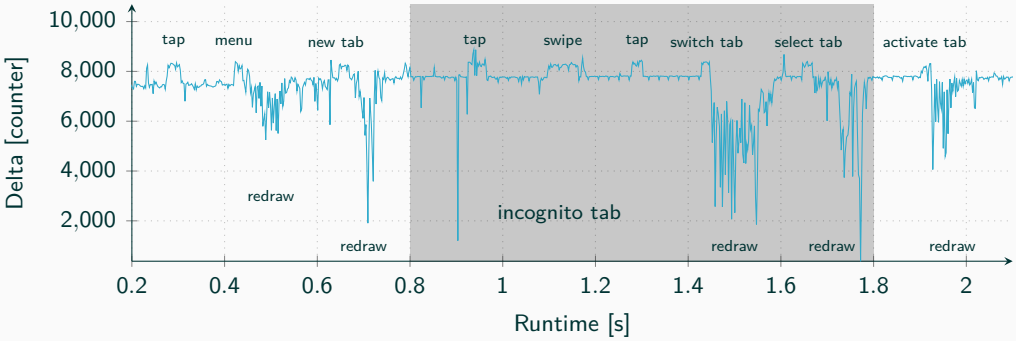


Figure 6: Keystroke timing attack running while switching to a different tab in the Chrome browser on the Xiaomi Redmi Note 3.

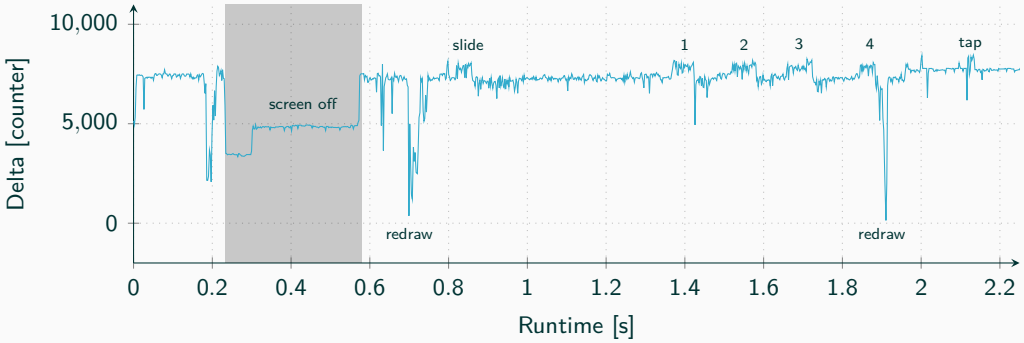


Figure 7: Keystroke timing attack running in the Firefox browser on the Xiaomi Redmi Note 3. While the user locked the screen, the application still detects keystrokes as long as it is executed on the last used tab.

Device	SoC	Keystrokes	Screen lock
Google Nexus 5	Qualcomm MSM8974 Snapdragon 800	✓	-
Xiaomi Redmi Note 3	Mediatek MT6795 Helio X10	✓	✓
Homtom HT3	MediaTek MTK6580	✓	✓
Samsung Galaxy S6	Samsung Exynos 7420	-	✓
OnePlus One	Qualcomm MSM8974AC Snapdragon 801	✓	✓
OnePlus 3T	Qualcomm MSM8996 Snapdragon 821	-	-

Table 1: Mobile test devices.

- Establish a unidirectional covert channel

- Establish a unidirectional covert channel
  - Sending a 1: Issue interrupt

Covert channel

- Establish a unidirectional covert channel
  - Sending a 1: Issue interrupt
  - Sending a 0: Idle

Covert channel

- Establish a unidirectional covert channel
  - Sending a 1: Issue interrupt
  - Sending a 0: Idle
- Utilize `XMLHttpRequest` to fetch a network resource from an invalid URL to implicitly issue an interrupt

- Cross-tab

Covert channel

- Cross-tab
  - Breaks Same-Origin policy (SOP)

Covert channel

- Cross-tab
  - Breaks Same-Origin policy (SOP)
  - Breaks HTTP Strict Transport Security (HSTS) policy

Covert channel

- Cross-tab
  - Breaks Same-Origin policy (SOP)
  - Breaks HTTP Strict Transport Security (HSTS) policy
- Cross-browser

Covert channel

- Cross-tab
  - Breaks Same-Origin policy (SOP)
  - Breaks HTTP Strict Transport Security (HSTS) policy
- Cross-browser
  - Circumvents process-per-site or process-per-tab policy

Covert channel

- Cross-tab
  - Breaks Same-Origin policy (SOP)
  - Breaks HTTP Strict Transport Security (HSTS) policy
- Cross-browser
  - Circumvents process-per-site or process-per-tab policy
  - Transmission from Firefox to Chrome

Covert channel

- Cross-tab
  - Breaks Same-Origin policy (SOP)
  - Breaks HTTP Strict Transport Security (HSTS) policy
- Cross-browser
  - Circumvents process-per-site or process-per-tab policy
  - Transmission from Firefox to Chrome
  - Established with browsers running in incognito mode

Covert channel

- Cross-tab
  - Breaks Same-Origin policy (SOP)
  - Breaks HTTP Strict Transport Security (HSTS) policy
- Cross-browser
  - Circumvents process-per-site or process-per-tab policy
  - Transmission from Firefox to Chrome
  - Established with browsers running in incognito mode
- Transmission rate
  - Raw transmission rate of 25 bps using a sample interval of 40 ms

Countermeasures

Countermeasures

- Generic Countermeasures

- Generic Countermeasures
  - Inject phantom keystrokes that will be intercepted by malware [Mye17]

- Generic Countermeasures
  - Inject phantom keystrokes that will be intercepted by malware [Mye17]
  - Analyze the statistical properties of noise necessary to impede real keystroke detection [Ort12]

- Generic Countermeasures
  - Inject phantom keystrokes that will be intercepted by malware [Mye17]
  - Analyze the statistical properties of noise necessary to impede real keystroke detection [Ort12]
  - Do not prevent interrupt-timing attacks

Countermeasures

- Generic Countermeasures
  - Inject phantom keystrokes that will be intercepted by malware [Mye17]
  - Analyze the statistical properties of noise necessary to impede real keystroke detection [Ort12]
  - Do not prevent interrupt-timing attacks
- Fine-grained Permission Model for JavaScript

Countermeasures

- Generic Countermeasures
  - Inject phantom keystrokes that will be intercepted by malware [Mye17]
  - Analyze the statistical properties of noise necessary to impede real keystroke detection [Ort12]
  - Do not prevent interrupt-timing attacks
- Fine-grained Permission Model for JavaScript
  - Per-page level access control to APIs, e.g., web workers

Conclusion

Conclusion

Conclusion

- First JavaScript-based keystroke timing attack

Conclusion

- First JavaScript-based keystroke timing attack
  - independent of browser and operating system

Conclusion

- First JavaScript-based keystroke timing attack
  - independent of browser and operating system
- Infer accurate timestamps of keystrokes as well as taps and swipes

Conclusion

- First JavaScript-based keystroke timing attack
  - independent of browser and operating system
- Infer accurate timestamps of keystrokes as well as taps and swipes
- Built classifiers to detect visited websites and identify users and a covert channel

Conclusion

- First JavaScript-based keystroke timing attack
  - independent of browser and operating system
- Infer accurate timestamps of keystrokes as well as taps and swipes
- Built classifiers to detect visited websites and identify users and a covert channel
- Highly practical, as it runs in background to spy on other tabs and applications

Practical Keystroke Timing Attacks in Sandboxed JavaScript

M. Lipp, D. Gruss, M. Schwarz, D. Bidner, C. Maurice, S. Mangard

Sep 11, 2017—ESORICS'17

Graz University of Technology