Cache Template Attacks:

Automating Attacks on Inclusive Last-Level Caches

Daniel Gruss, Raphael Spreitzer, and Stefan Mangard,
IAIK, Graz University of Technology

August 14, 2015

Cache Template Attacks

Cache Template Attacks

- State of the art: cache attacks are powerful
- Problem: manual identification of attack targets

Cache Template Attacks

- State of the art: cache attacks are powerful
- Problem: manual identification of attack targets
- Solution: Cache Template Attacks
- Automatically find any secret-dependent cache access
- Can be used for attacks and to improve software

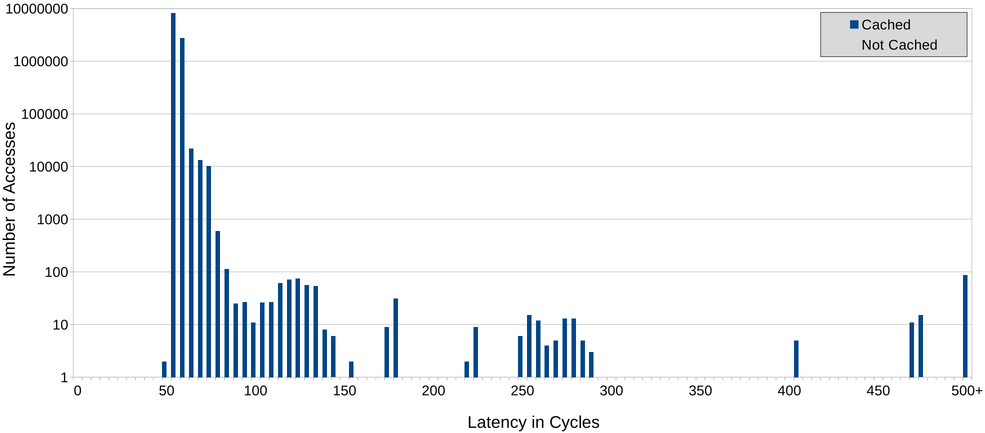
Cache Template Attacks

- State of the art: cache attacks are powerful
- Problem: manual identification of attack targets
- Solution: Cache Template Attacks
- Automatically find any secret-dependent cache access
- Can be used for attacks and to improve software
- Examples:
  - Cache-based keylogger
  - Automatic attacks on crypto algorithms

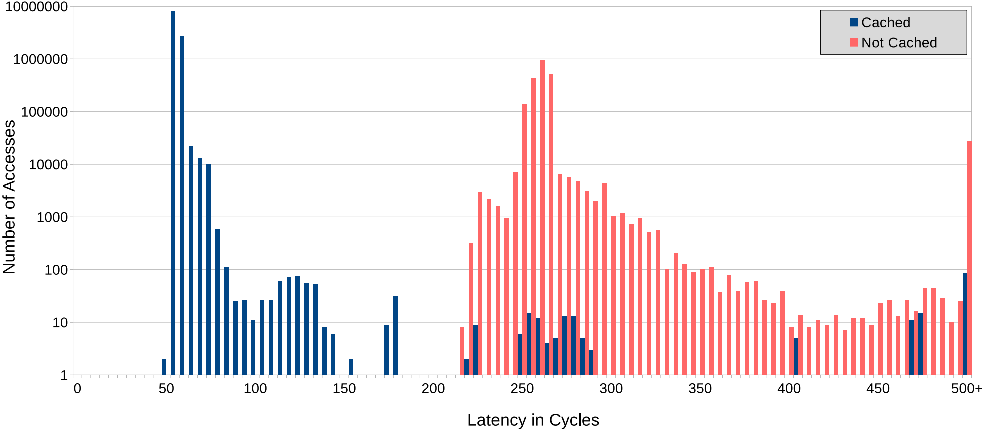
CPU Caches

- Reduce memory access latency

Memory Access Latency



Memory Access Latency



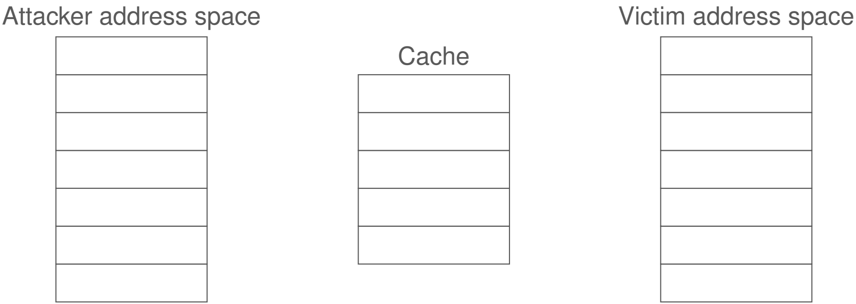
CPU Caches

- Reduce memory access latency
- Last-level cache on modern Intel CPUs:
  - Inclusive (to lower levels)
  - Not in Last-level cache $\Leftrightarrow$ not cached
  - Shared (between cores)

Flush+Reload

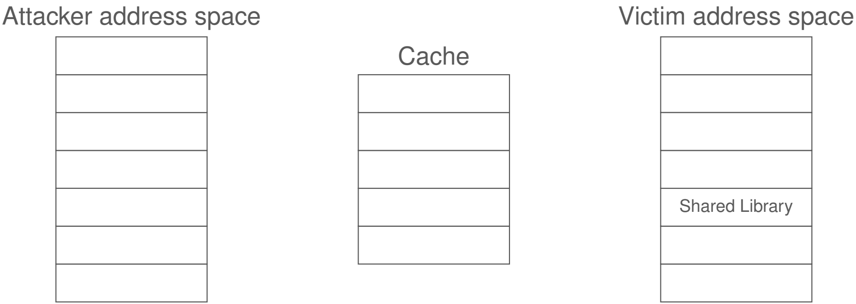
- Powerful cache attack
- Works on shared binaries/libraries
- Application on crypto algorithms

Flush+Reload



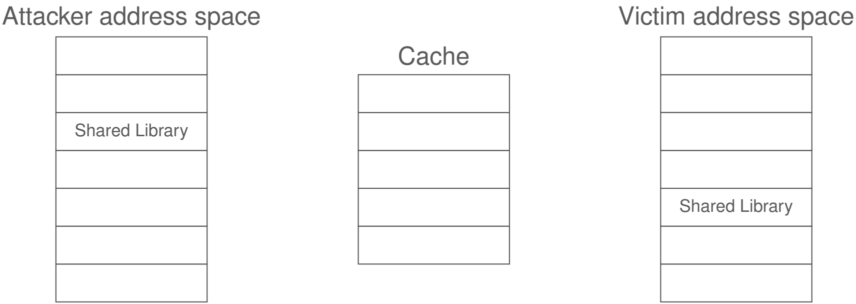
Cache is empty

Flush+Reload



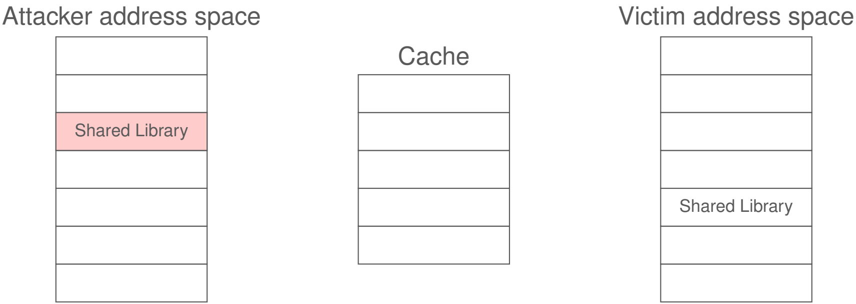
Victim maps shared library

Flush+Reload



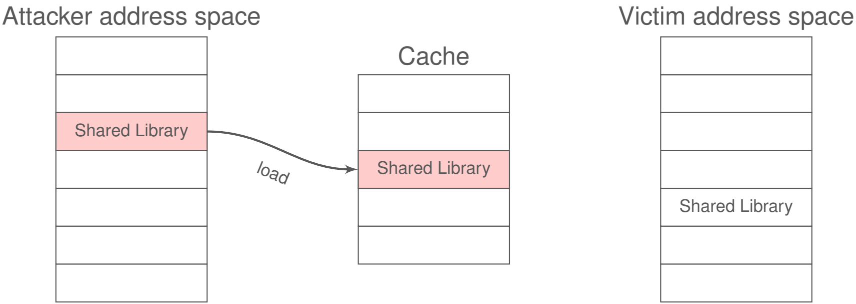
Attacker maps shared library

Flush+Reload



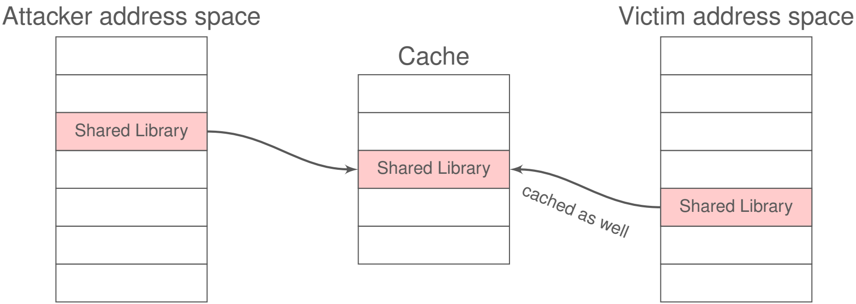
Attacker accesses shared library

Flush+Reload



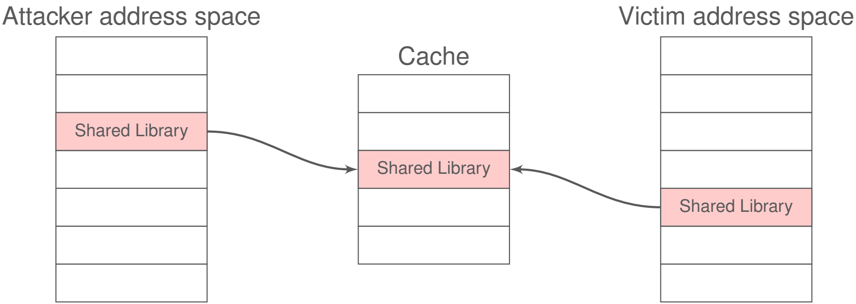
Loading into cache...

Flush+Reload



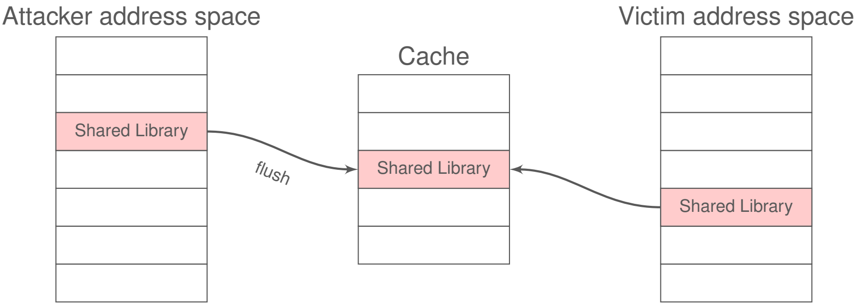
Loading into cache...

Flush+Reload



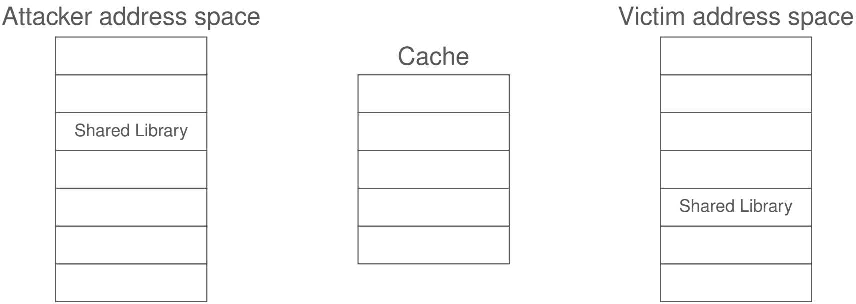
Attacker measures high latency

Flush+Reload



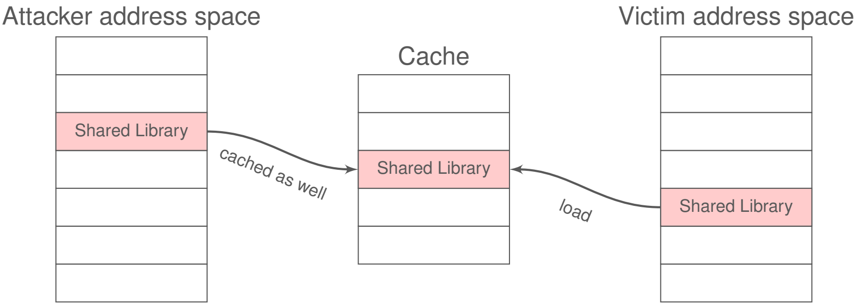
Attacker flushes shared library (“flush”)

Flush+Reload



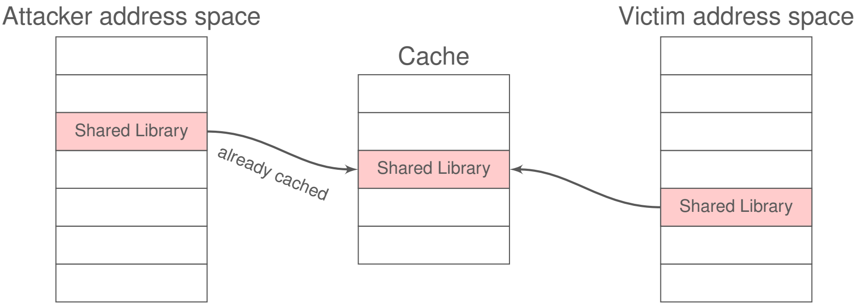
Cache is empty again

Flush+Reload



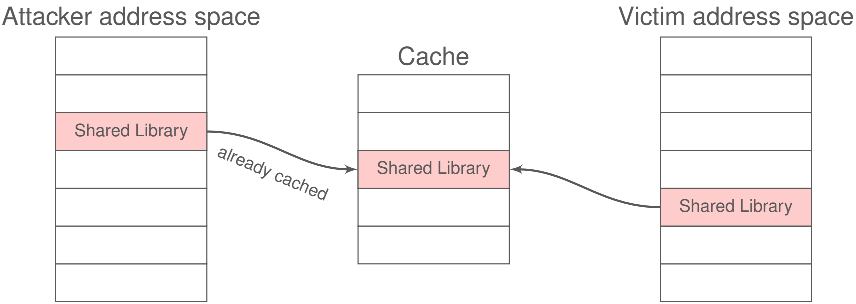
Victim accesses shared library

Flush+Reload



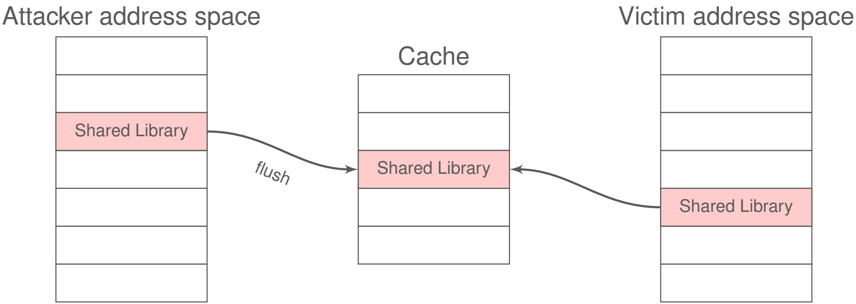
Attacker accesses shared library (“reload”)

Flush+Reload



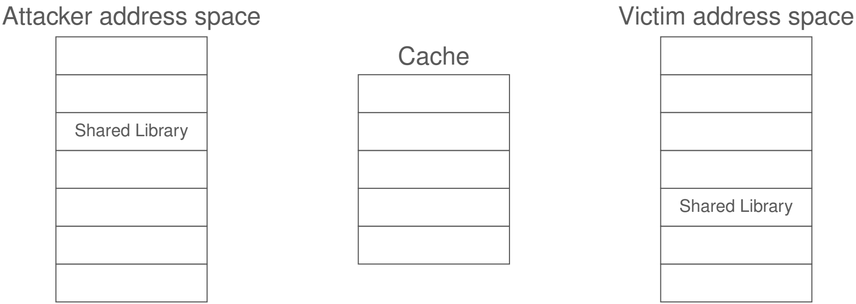
Attacker measures low latency

Flush+Reload



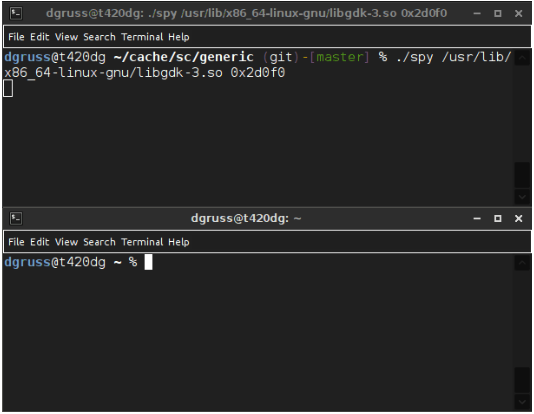
Attacker flushes shared library (“flush”)

Flush+Reload

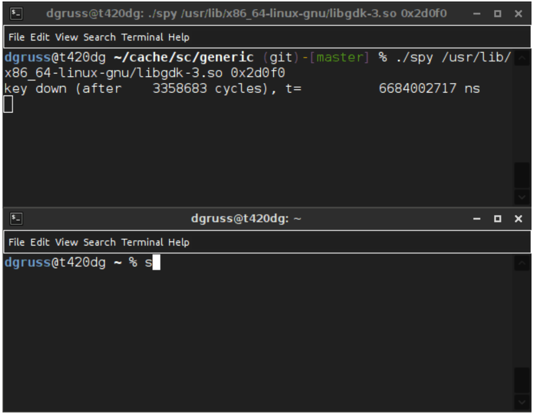


Cache is empty again

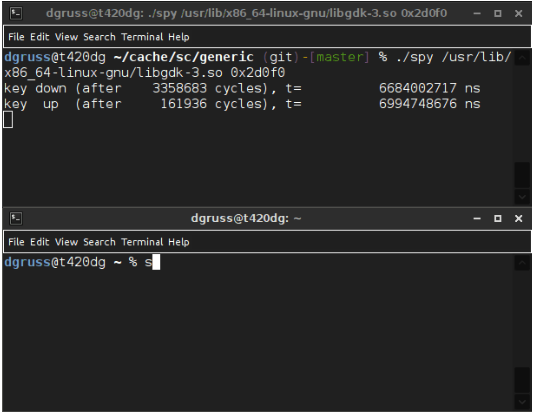
Can We Build a Cache-Based Keylogger?



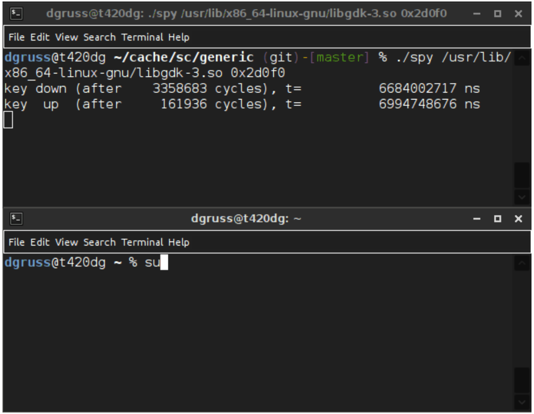
Can We Build a Cache-Based Keylogger?



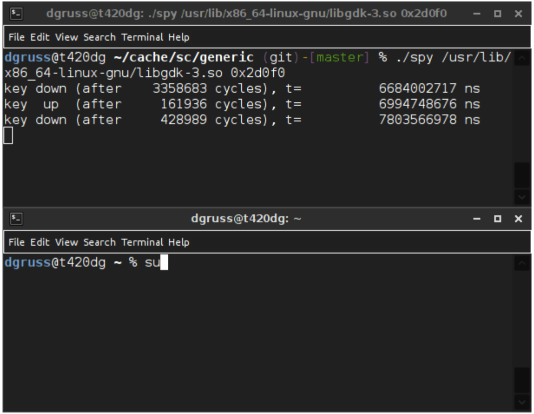
Can We Build a Cache-Based Keylogger?



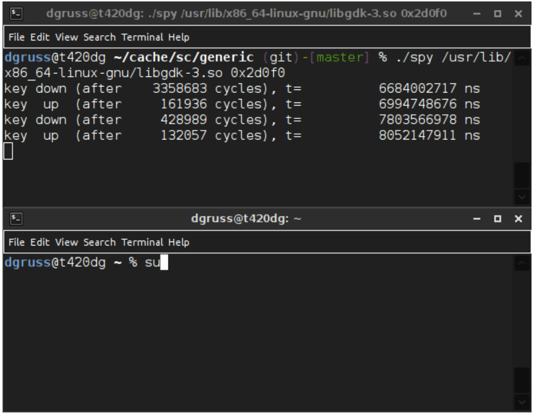
Can We Build a Cache-Based Keylogger?



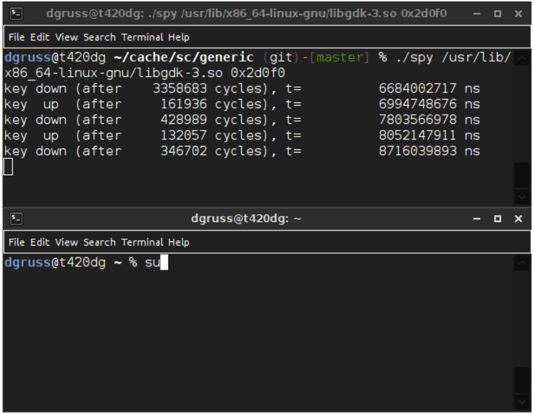
Can We Build a Cache-Based Keylogger?



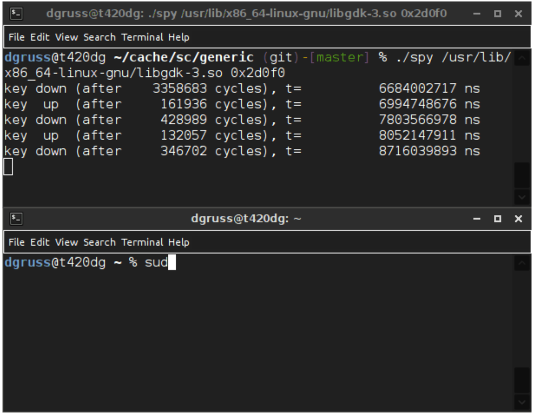
Can We Build a Cache-Based Keylogger?



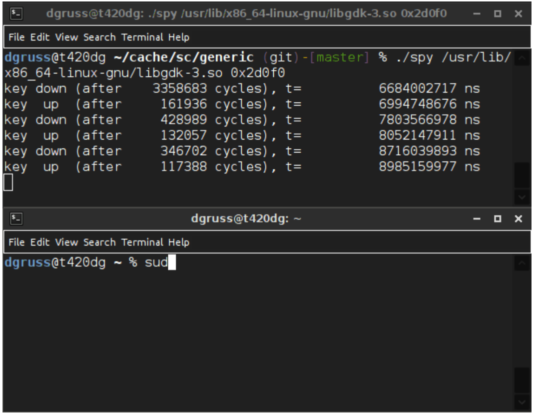
Can We Build a Cache-Based Keylogger?



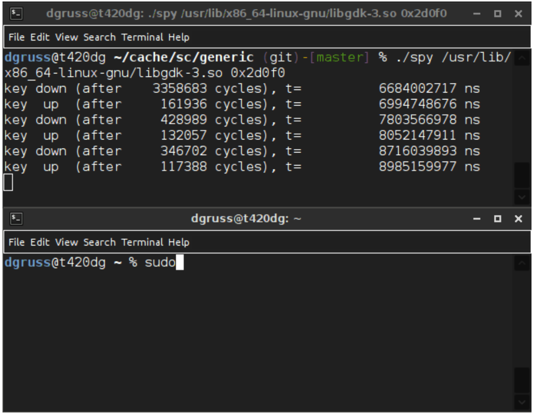
Can We Build a Cache-Based Keylogger?



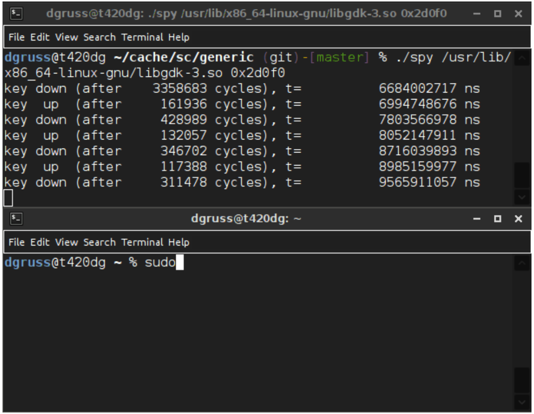
Can We Build a Cache-Based Keylogger?



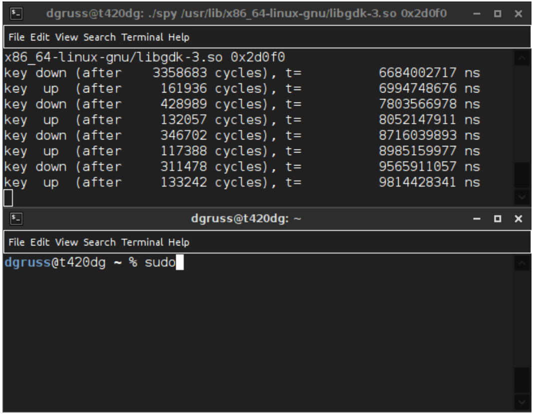
Can We Build a Cache-Based Keylogger?



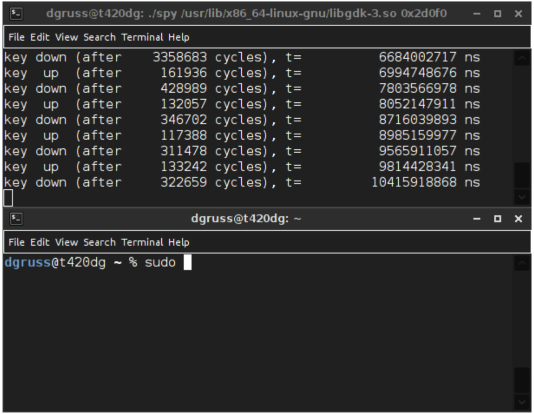
Can We Build a Cache-Based Keylogger?



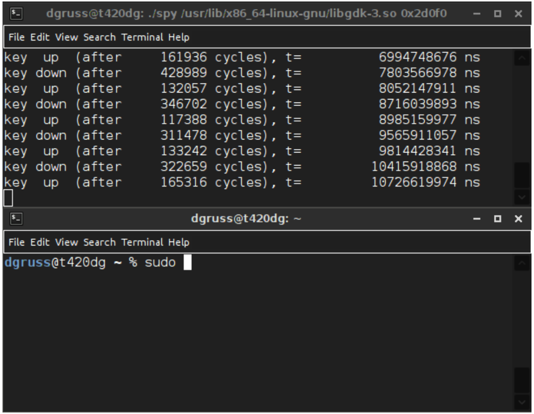
Can We Build a Cache-Based Keylogger?



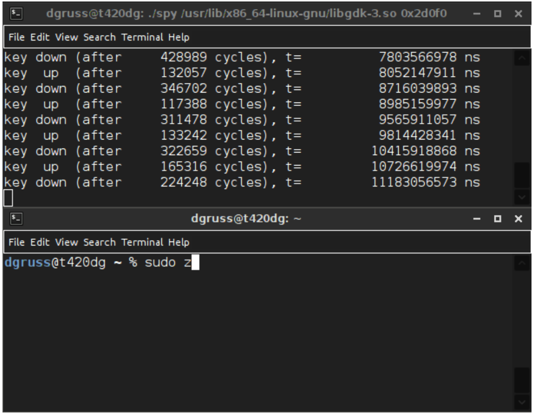
Can We Build a Cache-Based Keylogger?



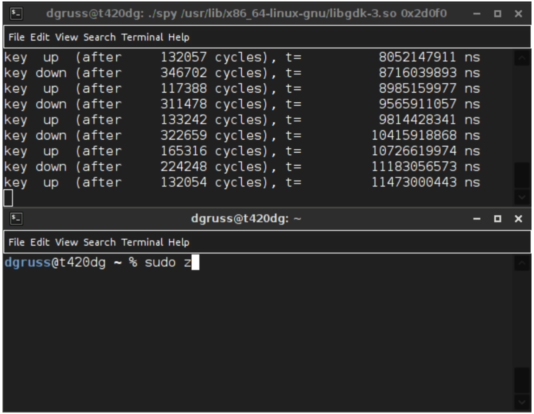
Can We Build a Cache-Based Keylogger?



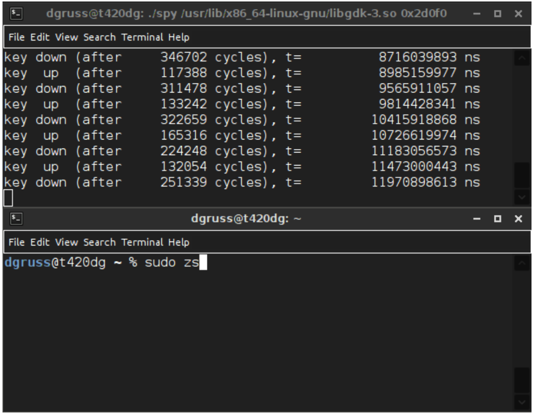
Can We Build a Cache-Based Keylogger?



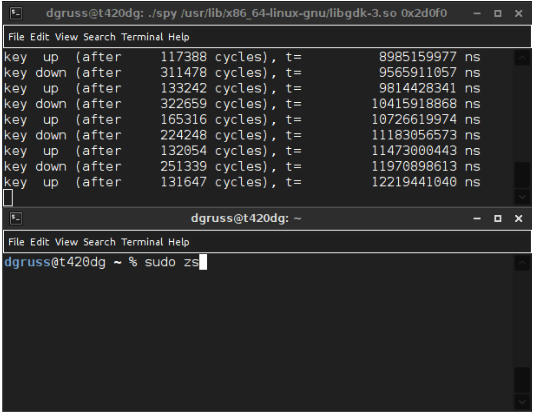
Can We Build a Cache-Based Keylogger?



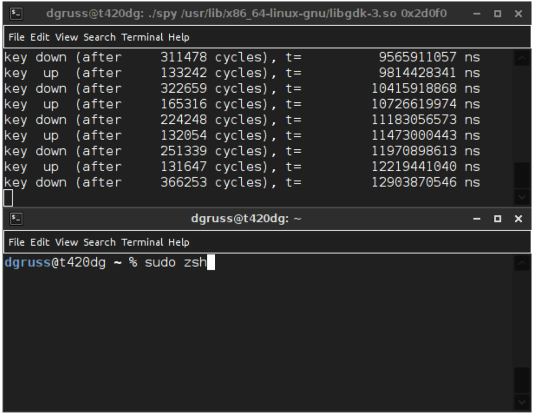
Can We Build a Cache-Based Keylogger?



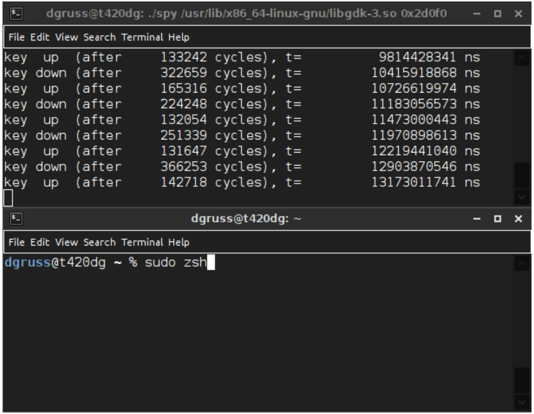
Can We Build a Cache-Based Keylogger?



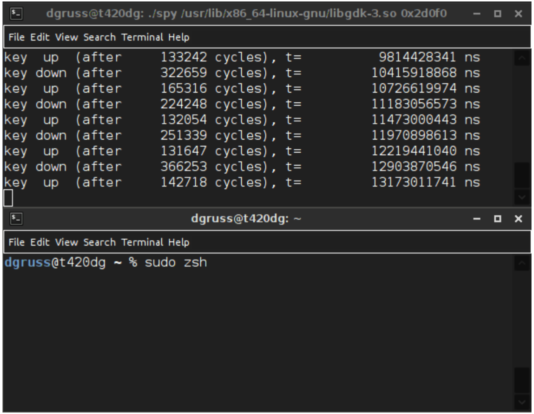
Can We Build a Cache-Based Keylogger?



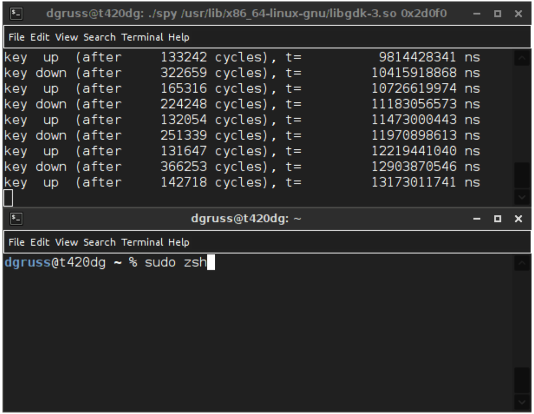
Can We Build a Cache-Based Keylogger?



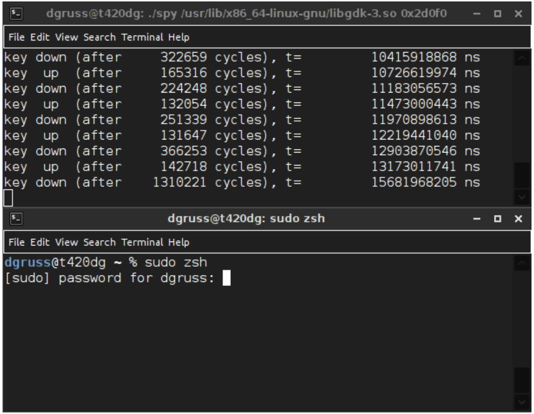
Can We Build a Cache-Based Keylogger?



Can We Build a Cache-Based Keylogger?



Can We Build a Cache-Based Keylogger?



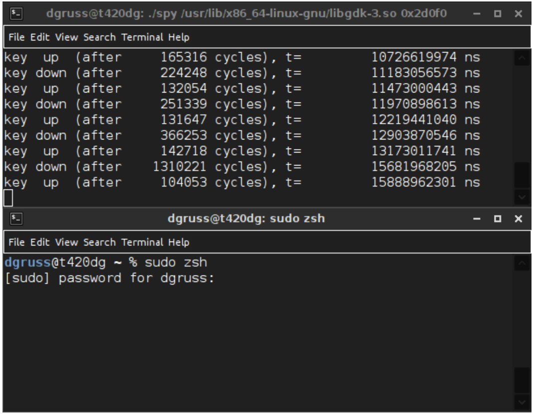
Can We Build a Cache-Based Keylogger?

```
dgruss@t420dg: ./spy /usr/lib/x86_64-linux-gnu/libgdk-3.so 0x2d0f0
File Edit View Search Terminal Help
key  up  (after     165316 cycles), t=          10726619974 ns
key down (after     224248 cycles), t=          11183056573 ns
key  up  (after     132054 cycles), t=          11473000443 ns
key down (after     251339 cycles), t=          11970898613 ns
key  up  (after     131647 cycles), t=          12219441040 ns
key down (after     366253 cycles), t=          12903870546 ns
key  up  (after     142718 cycles), t=          13173011741 ns
key down (after    1310221 cycles), t=          15681968205 ns
key  up  (after     104053 cycles), t=          15888962301 ns
```

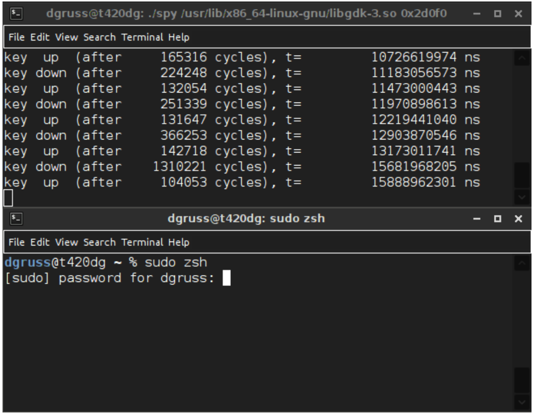
```
dgruss@t420dg: sudo zsh
File Edit View Search Terminal Help
dgruss@t420dg ~ % sudo zsh
[sudo] password for dgruss:
```


Daniel Gruss, IAIK, Graz University of Technology

Can We Build a Cache-Based Keylogger?



Can We Build a Cache-Based Keylogger?

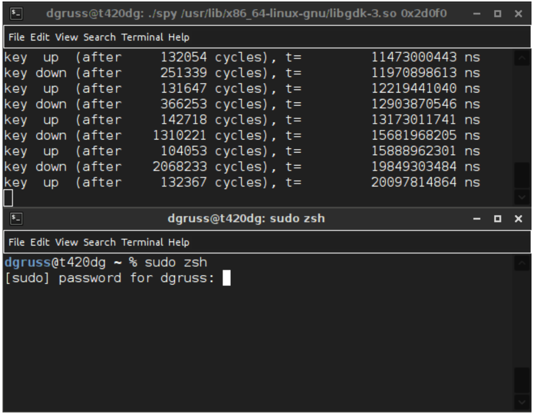


Can We Build a Cache-Based Keylogger?

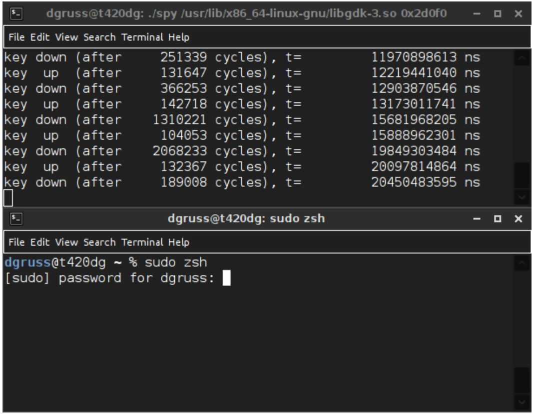
```
dgruss@t420dg: ./spy /usr/lib/x86_64-linux-gnu/libgdk-3.so 0x2d0f0
File Edit View Search Terminal Help
key down (after    224248 cycles), t=        11183056573 ns
key  up  (after    132054 cycles), t=        11473000443 ns
key down (after    251339 cycles), t=        11970898613 ns
key  up  (after    131647 cycles), t=        12219441040 ns
key down (after    366253 cycles), t=        12903870546 ns
key  up  (after    142718 cycles), t=        13173011741 ns
key down (after   1310221 cycles), t=        15681968205 ns
key  up  (after    104053 cycles), t=        15888962301 ns
key down (after   2068233 cycles), t=        19849303484 ns
```

```
dgruss@t420dg: sudo zsh
File Edit View Search Terminal Help
dgruss@t420dg ~ % sudo zsh
[sudo] password for dgruss:
```

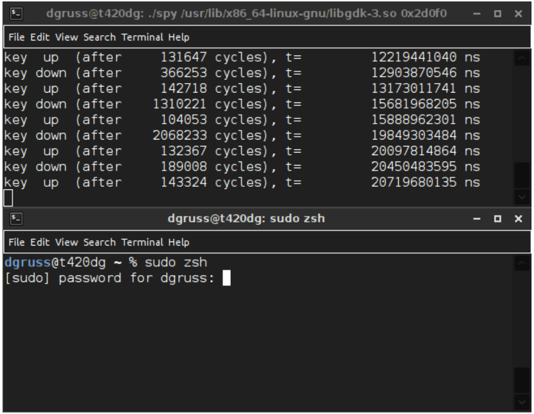
Can We Build a Cache-Based Keylogger?



Can We Build a Cache-Based Keylogger?



Can We Build a Cache-Based Keylogger?



Can We Build a Cache-Based Keylogger?

dgruss@t420dg: ./spy /usr/lib/x86_64-linux-gnu/libgdk-3.so 0x2d0f0

File Edit View Search Terminal Help

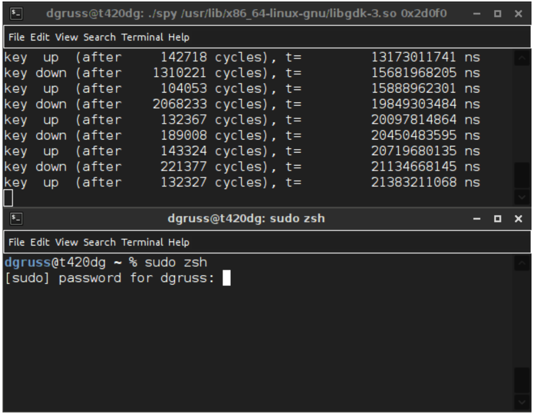
```
key down (after    366253 cycles), t=        12903870546 ns
key  up  (after    142718 cycles), t=        13173011741 ns
key down (after   1310221 cycles), t=        15681968205 ns
key  up  (after    104053 cycles), t=        15888962301 ns
key down (after   2068233 cycles), t=        19849303484 ns
key  up  (after    132367 cycles), t=        20097814864 ns
key down (after    189008 cycles), t=        20450483595 ns
key  up  (after    143324 cycles), t=        20719680135 ns
key down (after    221377 cycles), t=        21134668145 ns
```

dgruss@t420dg: sudo zsh

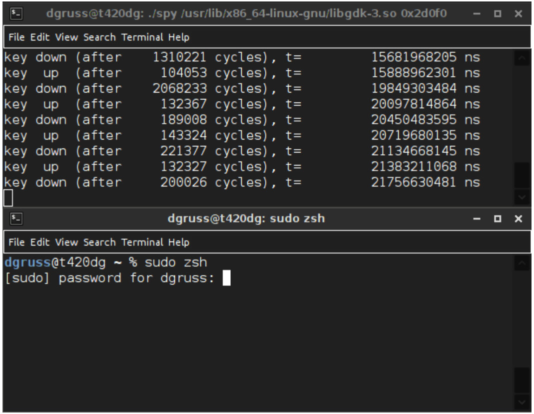
File Edit View Search Terminal Help

```
dgruss@t420dg ~ % sudo zsh
[sudo] password for dgruss:
```

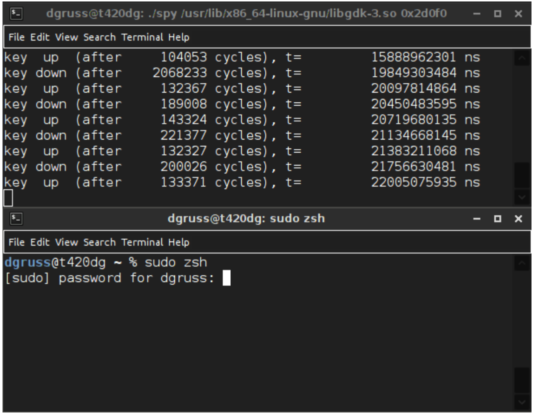
Can We Build a Cache-Based Keylogger?



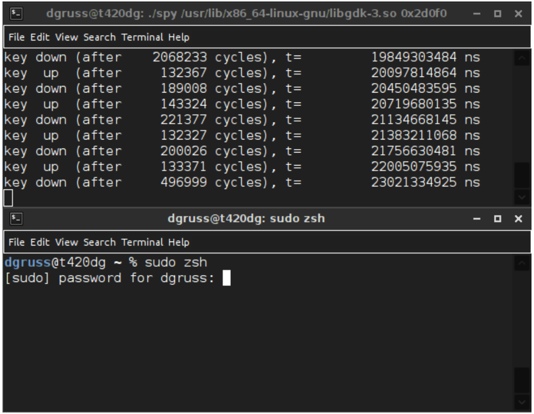
Can We Build a Cache-Based Keylogger?



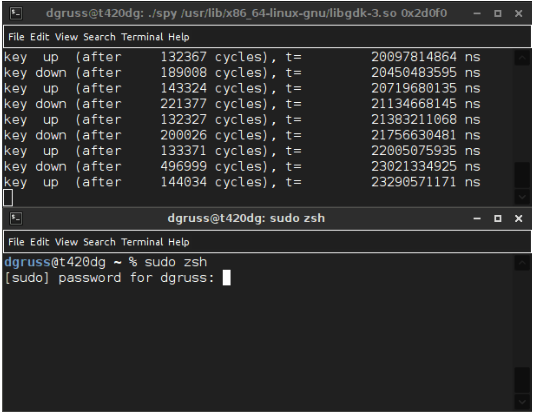
Can We Build a Cache-Based Keylogger?



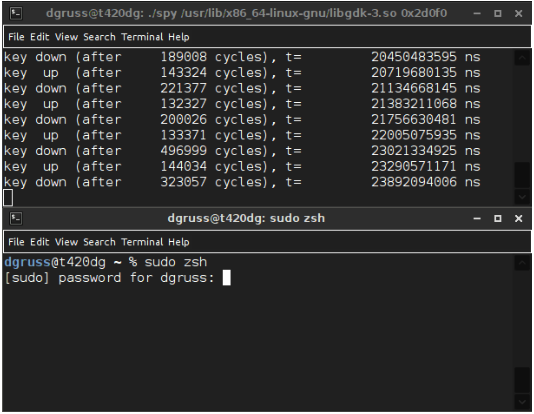
Can We Build a Cache-Based Keylogger?



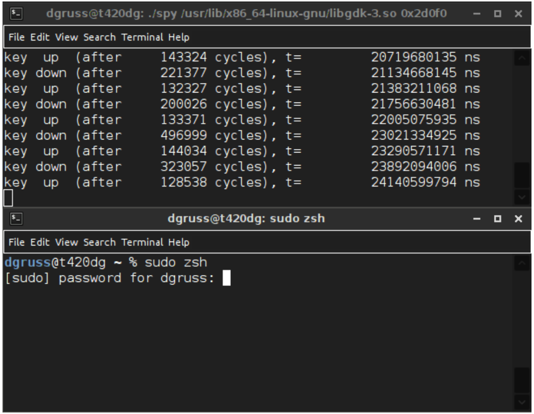
Can We Build a Cache-Based Keylogger?



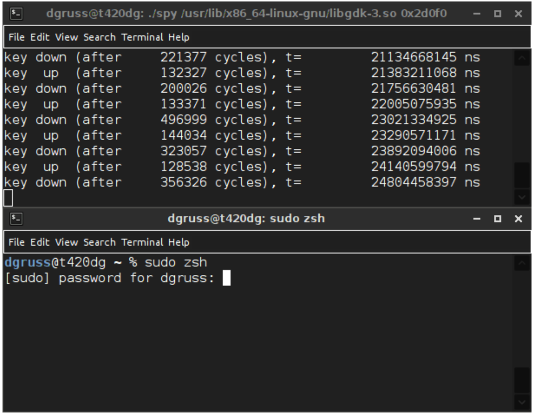
Can We Build a Cache-Based Keylogger?



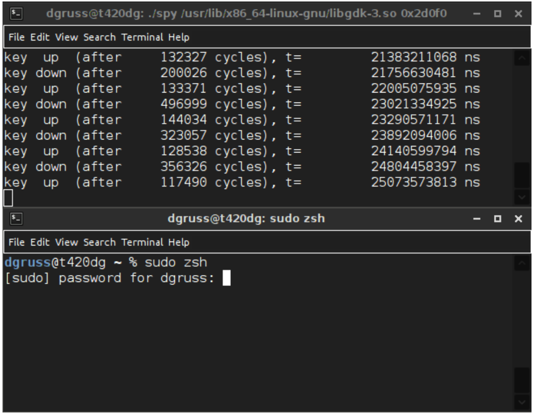
Can We Build a Cache-Based Keylogger?



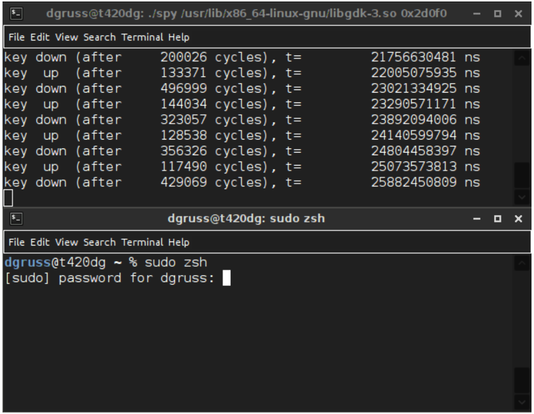
Can We Build a Cache-Based Keylogger?



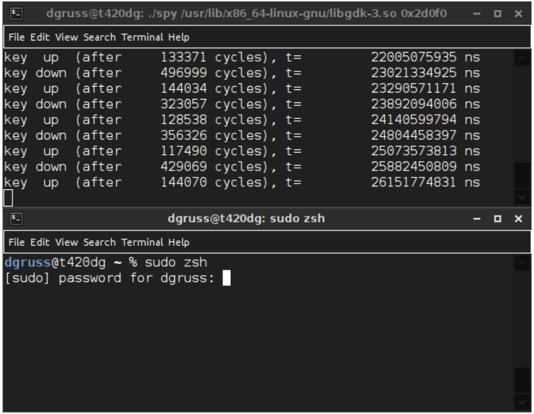
Can We Build a Cache-Based Keylogger?



Can We Build a Cache-Based Keylogger?



Can We Build a Cache-Based Keylogger?

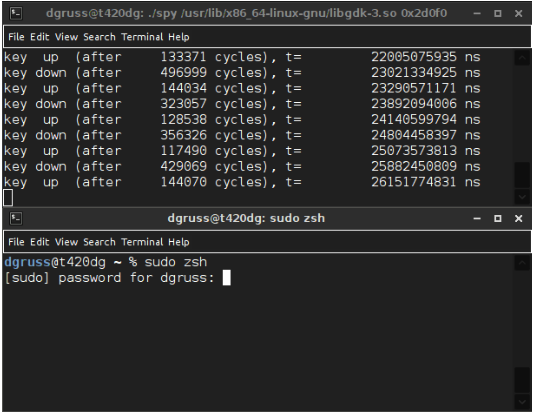


Can We Build a Cache-Based Keylogger?

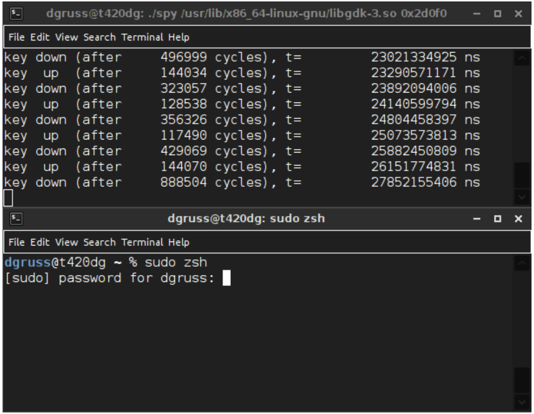
```
dgruss@t420dg: ./spy /usr/lib/x86_64-linux-gnu/libgdk-3.so 0x2d0f0
File Edit View Search Terminal Help
key  up  (after     133371 cycles), t=          22005075935 ns
key down (after     496999 cycles), t=          23021334925 ns
key  up  (after     144034 cycles), t=          23290571171 ns
key down (after     323057 cycles), t=          23892094006 ns
key  up  (after     128538 cycles), t=          24140599794 ns
key down (after     356326 cycles), t=          24804458397 ns
key  up  (after     117490 cycles), t=          25073573813 ns
key down (after     429069 cycles), t=          25882450809 ns
key  up  (after     144070 cycles), t=          26151774831 ns
```

```
dgruss@t420dg: sudo zsh
File Edit View Search Terminal Help
dgruss@t420dg ~ % sudo zsh
[sudo] password for dgruss:
```

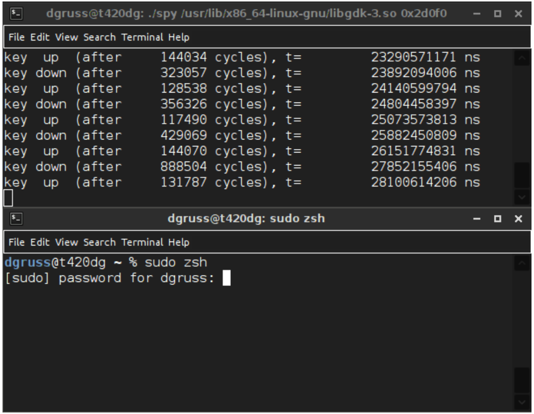
Can We Build a Cache-Based Keylogger?



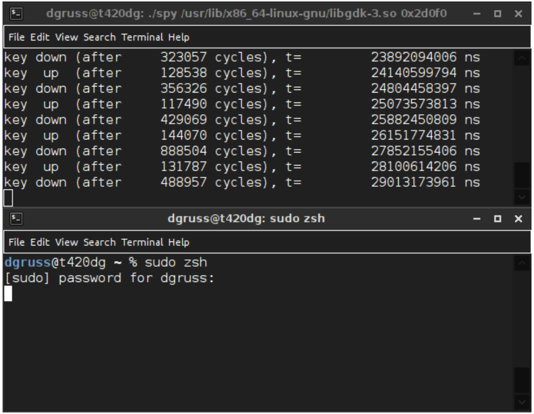
Can We Build a Cache-Based Keylogger?



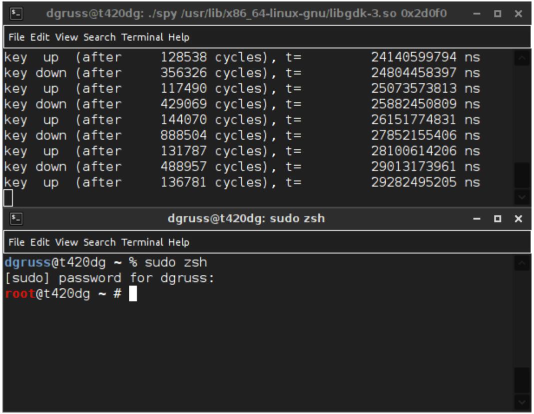
Can We Build a Cache-Based Keylogger?



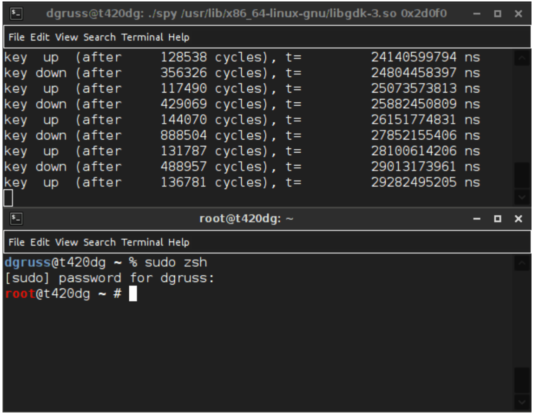
Can We Build a Cache-Based Keylogger?



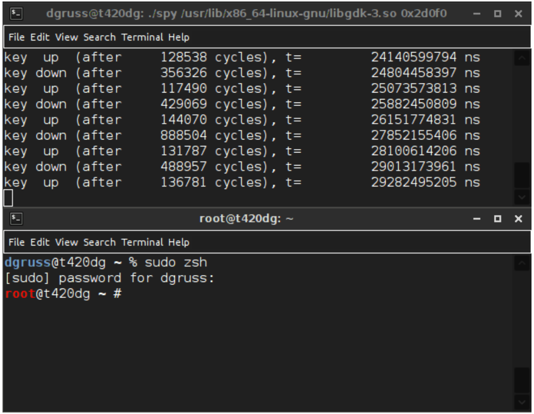
Can We Build a Cache-Based Keylogger?



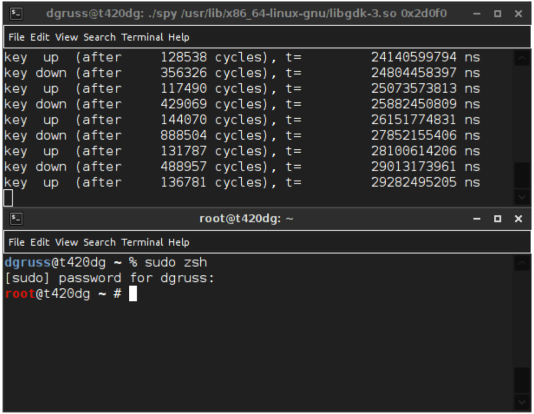
Can We Build a Cache-Based Keylogger?



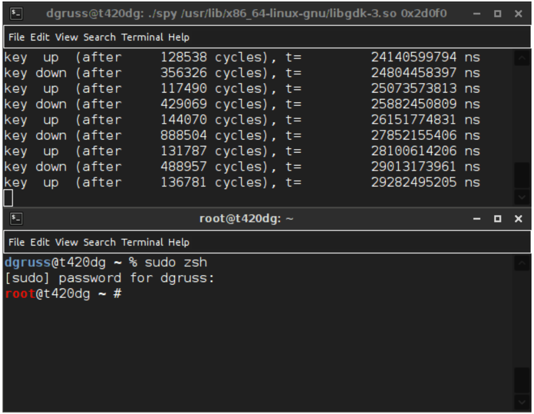
Can We Build a Cache-Based Keylogger?



Can We Build a Cache-Based Keylogger?



Can We Build a Cache-Based Keylogger?



Challenges

- How to locate key-dependent memory accesses?

Challenges

- How to locate key-dependent memory accesses?
- It's complicated:
  - Large binaries and libraries (third-party code)
  - Many libraries (gedit: 60MB)
  - Closed-source / unknown binaries
  - Self-compiled binaries

Challenges

- How to locate key-dependent memory accesses?
- It's complicated:
  - Large binaries and libraries (third-party code)
  - Many libraries (gedit: 60MB)
  - Closed-source / unknown binaries
  - Self-compiled binaries
- Difficult to find all exploitable addresses

Cache Template Attacks

Cache Template Attacks

Profiling Phase

- Preprocessing step to find exploitable addresses automatically
    - w.r.t. “events” (keystrokes, encryptions, ...)
    - called “Cache Template”

Cache Template Attacks

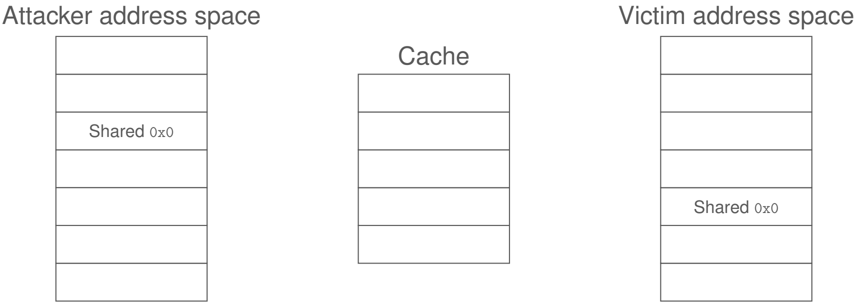
Profiling Phase

- Preprocessing step to find exploitable addresses automatically
  - w.r.t. “events” (keystrokes, encryptions, ...)
  - called “Cache Template”

Exploitation Phase

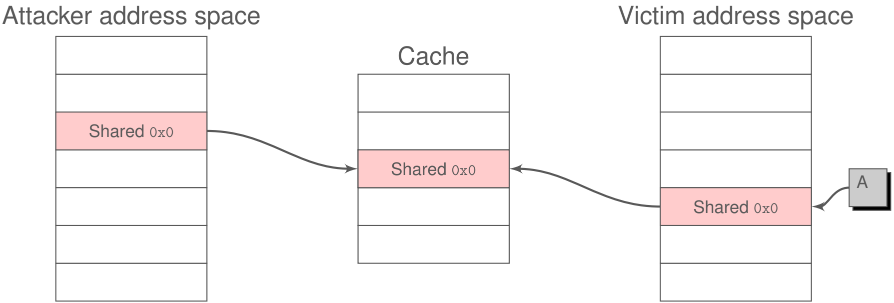
- Monitor exploitable addresses

Profiling Phase



Cache is empty

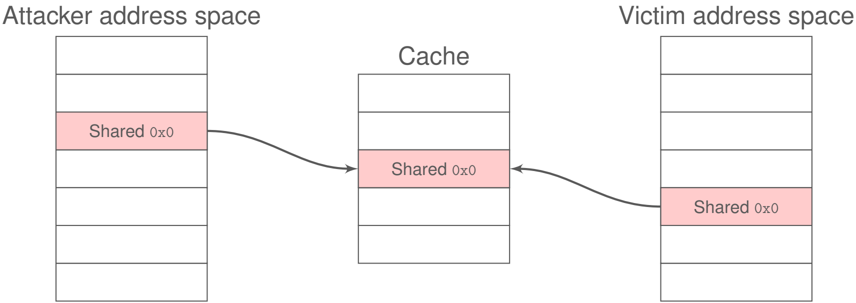
Profiling Phase



Attacker triggers an event

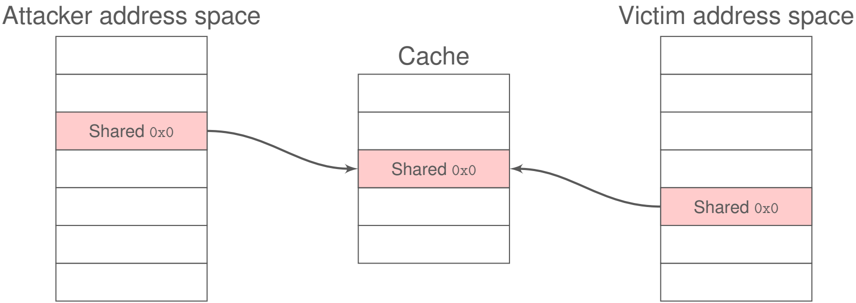
Daniel Gruss, IAIK, Graz University of Technology

Profiling Phase



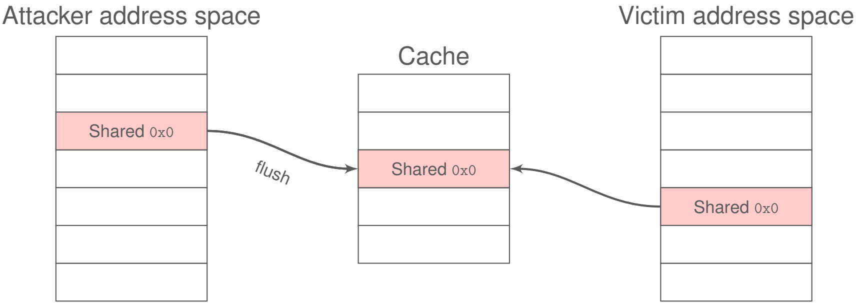
Attacker checks one address for cache hits (“Reload”)

Profiling Phase



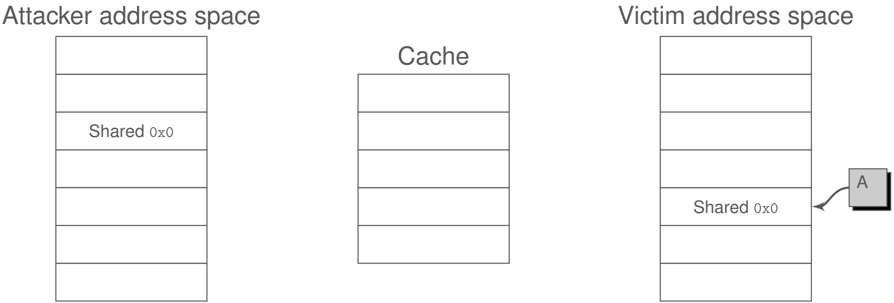
Update cache hit ratio (per event and address)

Profiling Phase



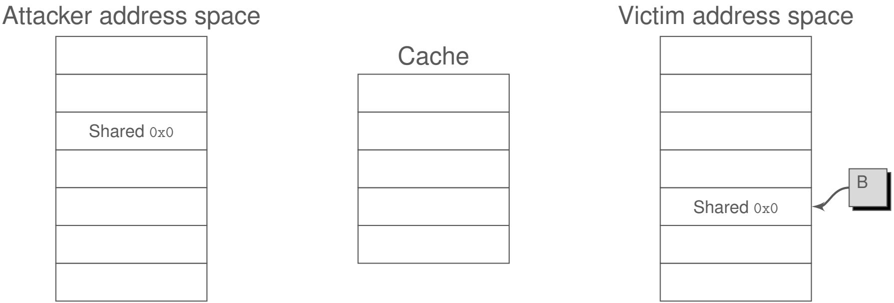
Attacker flushes shared memory

Profiling Phase



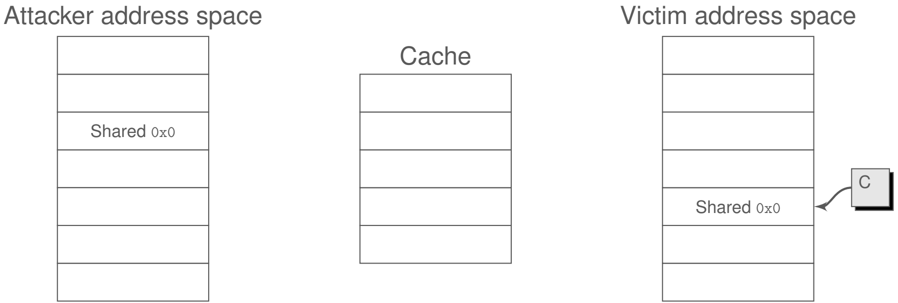
Repeat for higher accuracy

Profiling Phase



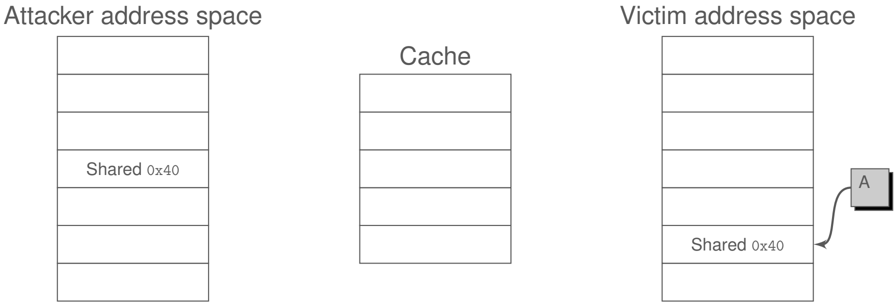
Repeat for all events

Profiling Phase



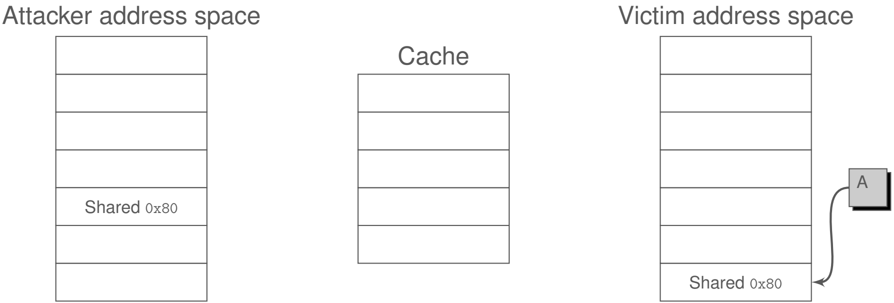
Repeat for all events

Profiling Phase



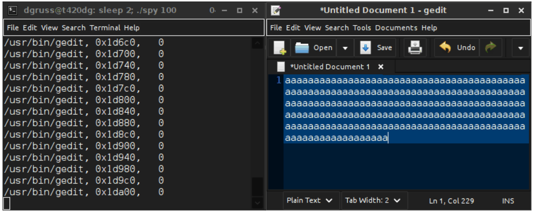
Continue with next address

Profiling Phase

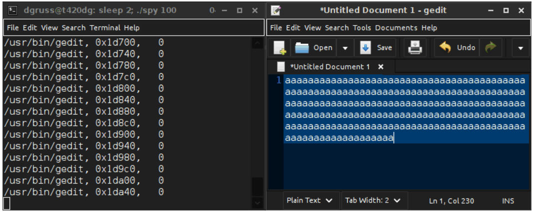


Continue with next address

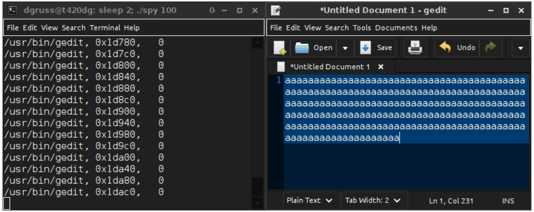
Profiling a Single Event



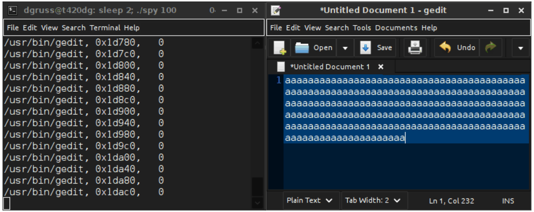
Profiling a Single Event



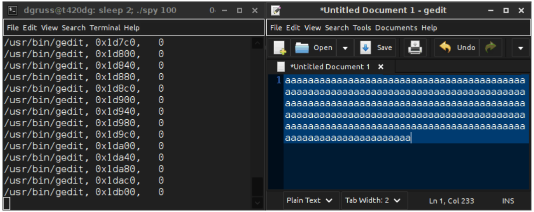
Profiling a Single Event



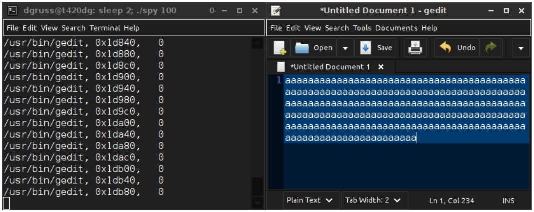
Profiling a Single Event



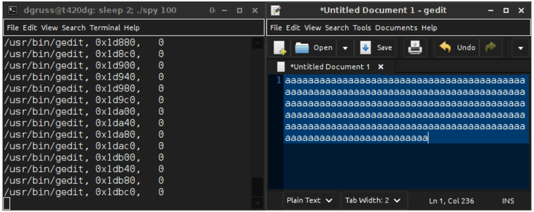
Profiling a Single Event



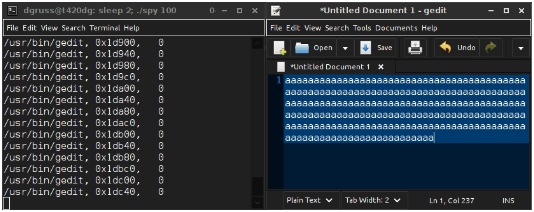
Profiling a Single Event



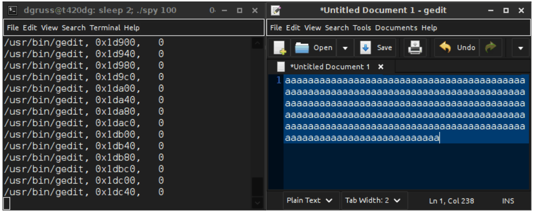
Profiling a Single Event



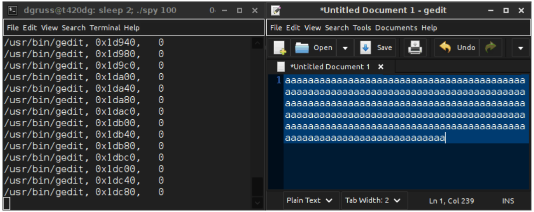
Profiling a Single Event



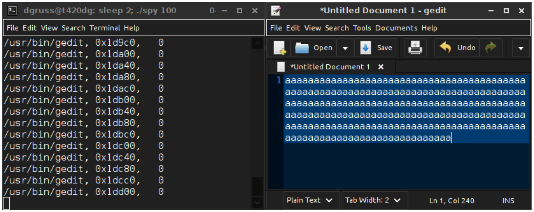
Profiling a Single Event



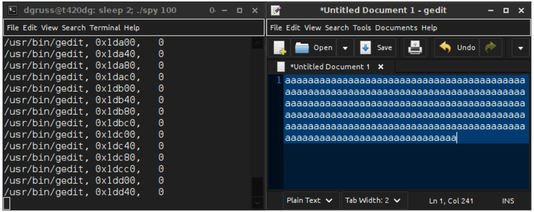
Profiling a Single Event



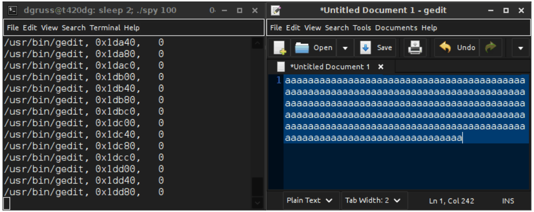
Profiling a Single Event



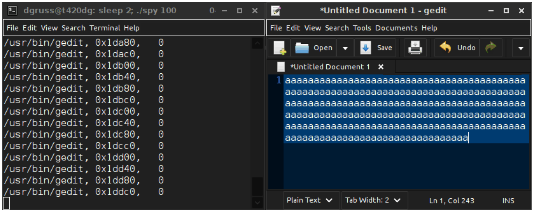
Profiling a Single Event



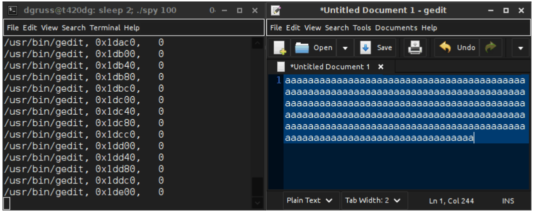
Profiling a Single Event



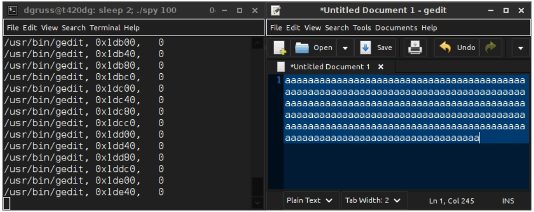
Profiling a Single Event



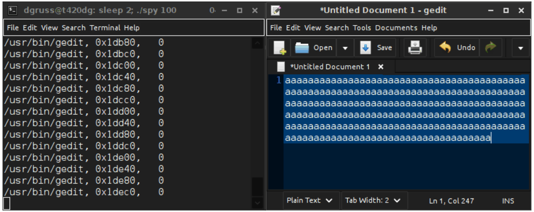
Profiling a Single Event



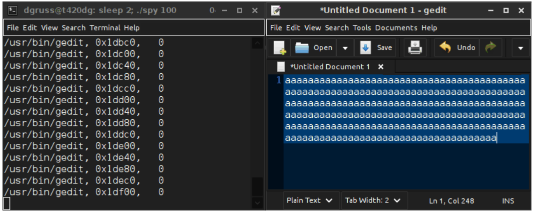
Profiling a Single Event



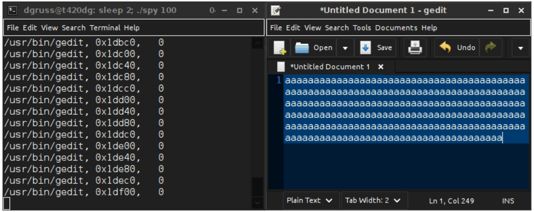
Profiling a Single Event



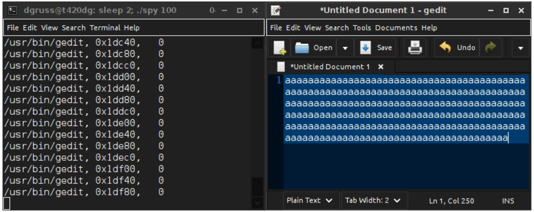
Profiling a Single Event



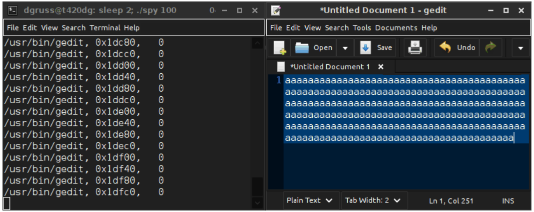
Profiling a Single Event



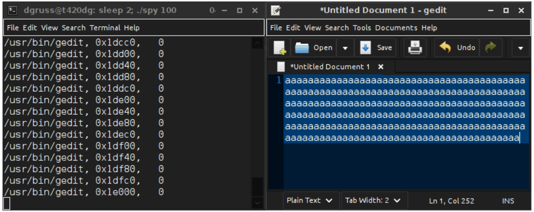
Profiling a Single Event



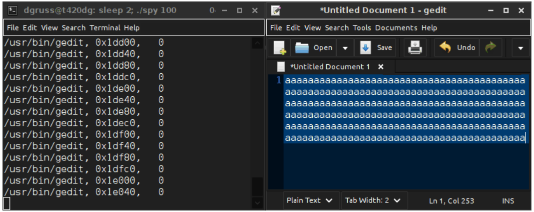
Profiling a Single Event



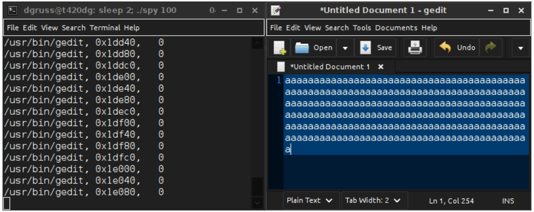
Profiling a Single Event



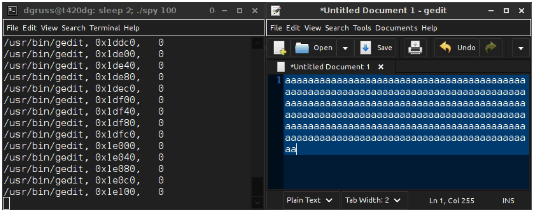
Profiling a Single Event



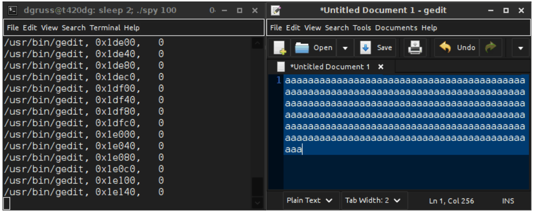
Profiling a Single Event



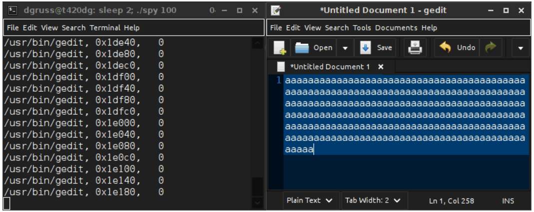
Profiling a Single Event



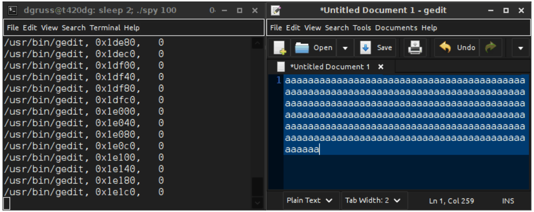
Profiling a Single Event



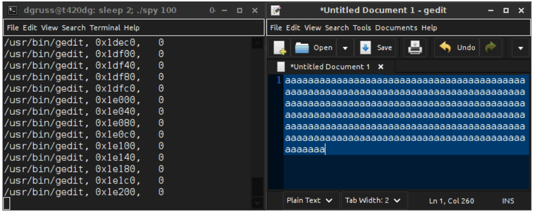
Profiling a Single Event



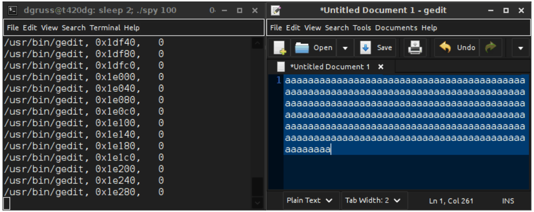
Profiling a Single Event



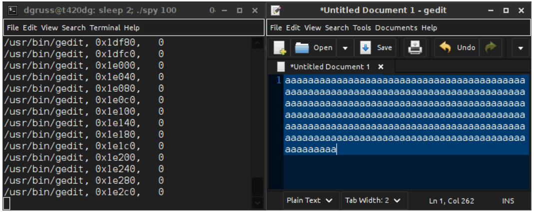
Profiling a Single Event



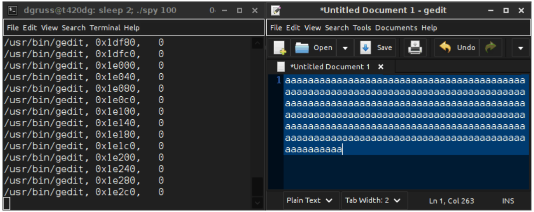
Profiling a Single Event



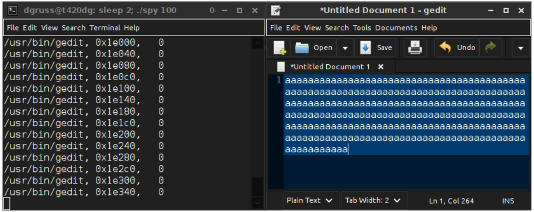
Profiling a Single Event



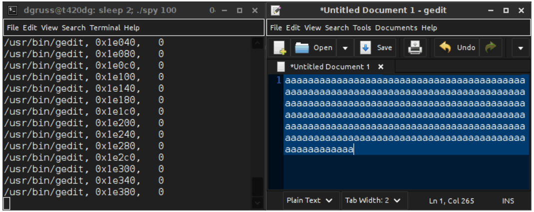
Profiling a Single Event



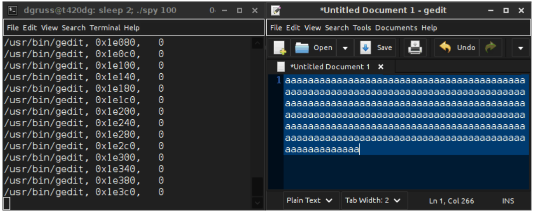
Profiling a Single Event



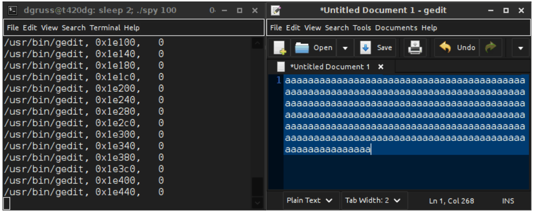
Profiling a Single Event



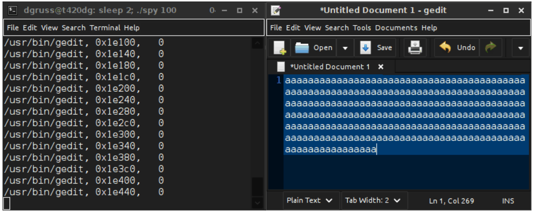
Profiling a Single Event



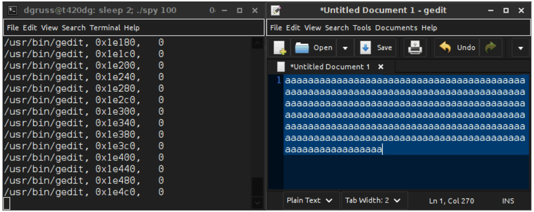
Profiling a Single Event



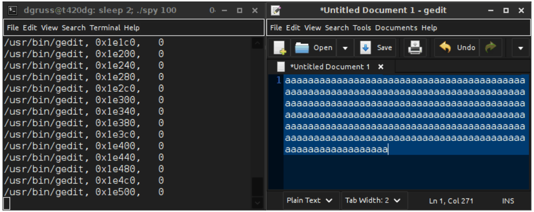
Profiling a Single Event



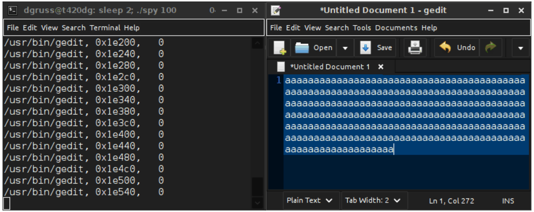
Profiling a Single Event



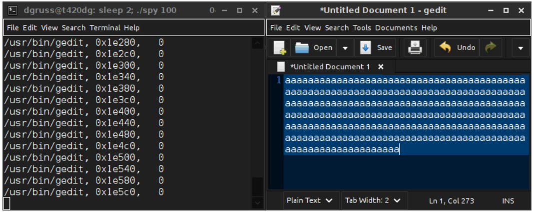
Profiling a Single Event



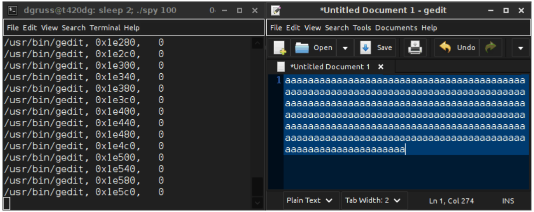
Profiling a Single Event



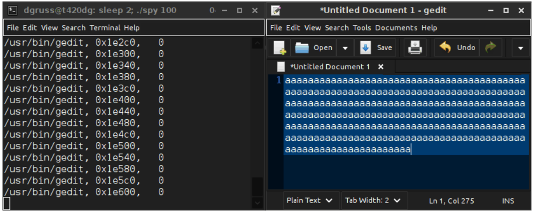
Profiling a Single Event



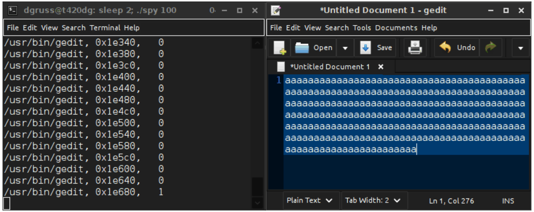
Profiling a Single Event



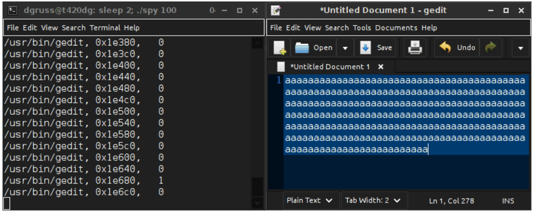
Profiling a Single Event



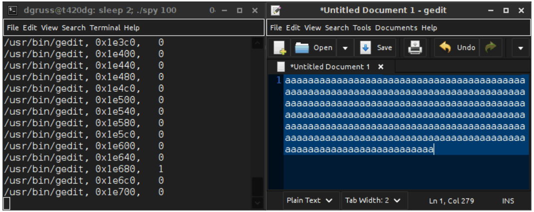
Profiling a Single Event



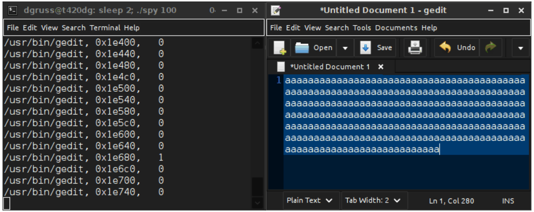
Profiling a Single Event



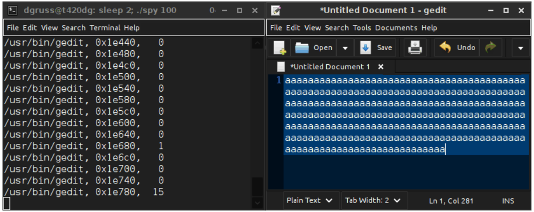
Profiling a Single Event



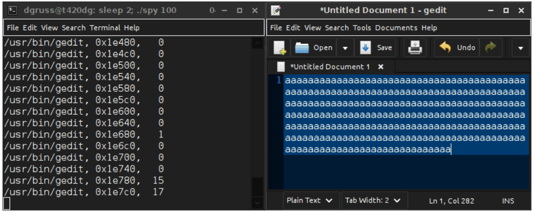
Profiling a Single Event



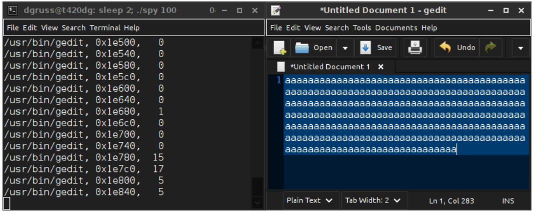
Profiling a Single Event



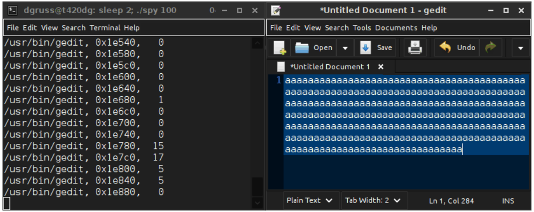
Profiling a Single Event



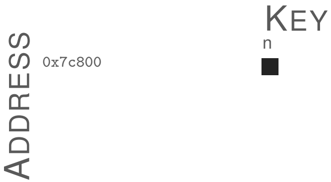
Profiling a Single Event



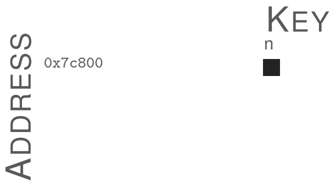
Profiling a Single Event



Profiling Phase: 1 Event, 1 Address



Profiling Phase: 1 Event, 1 Address



Example: Cache Hit Ratio for (0x7c800, n): 200 / 200

Profiling Phase: All Events, 1 Address

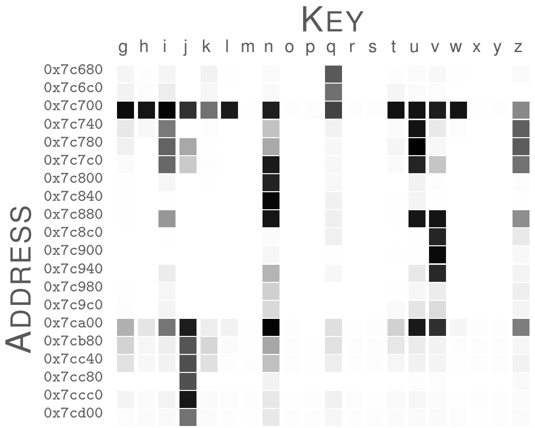
Profiling Phase: All Events, 1 Address

Example: Cache Hit Ratio for $(0x7c800, u)$: 13 / 200

Profiling Phase: All Events, 1 Address

Distinguish `n` from other keys by monitoring `0x7c800`

Profiling Phase: All Events, All Addresses



Exploitation Phase

- Monitor addresses from Cache Template

Exploitation Phase

- Monitor addresses from Cache Template
- Report to log file / attacker

Exploitation Phase

- Monitor addresses from Cache Template
- Report to log file / attacker
- Manual analysis of log file
  - Find password in keypress log, etc.

Example Attacks

Attack 1: Keystroke Timings

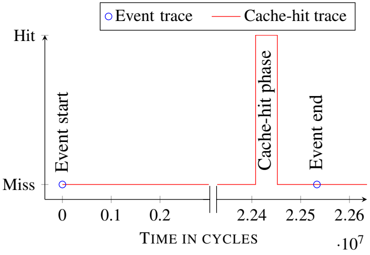
- Spy on keystroke timings on Linux, Windows and OS X

Attack 1: Keystroke Timings

- Spy on keystroke timings on Linux, Windows and OS X
- Sub-microsecond accuracy

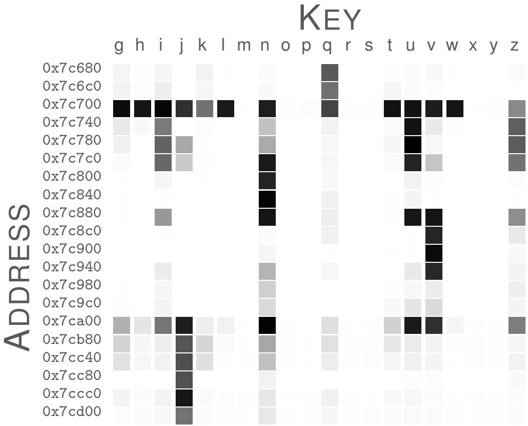
Attack 1: Keystroke Timings

- Spy on keystroke timings on Linux, Windows and OS X
- Sub-microsecond accuracy
- Derive text input from timings



Attack 2: Keylogging

- Linux with GTK: monitor keystrokes of specific keys
- Detect groups of keys
- Some keys distinct



Attack 3: Locate AES T-Tables

AES T-Table implementation from OpenSSL 1.0.2

Attack 3: Locate AES T-Tables

AES T-Table implementation from OpenSSL 1.0.2

- Most addresses in two groups:
    - Cache hit ratio 100% (always cache hits)
    - Cache hit ratio 0% (no cache hits)

Attack 3: Locate AES T-Tables

AES T-Table implementation from OpenSSL 1.0.2

- Most addresses in two groups:
  - Cache hit ratio 100% (always cache hits)
  - Cache hit ratio 0% (no cache hits)
- One 4096 byte memory block:
  - Cache hit ratio of 92%
  - Cache hits depend on key value and plaintext value
  - The T-Tables

Attack 4: AES T-Table Template Attack

AES T-Table implementation from OpenSSL 1.0.2

- Known-plaintext attack
- Events: encryption with only one fixed key byte

Attack 4: AES T-Table Template Attack

AES T-Table implementation from OpenSSL 1.0.2

- Known-plaintext attack
- Events: encryption with only one fixed key byte
- Profile each event

Attack 4: AES T-Table Template Attack

AES T-Table implementation from OpenSSL 1.0.2

- Known-plaintext attack
- Events: encryption with only one fixed key byte
- Profile each event
- Exploitation phase:
  - Eliminate key candidates

Attack 4: AES T-Table Template Attack

AES T-Table implementation from OpenSSL 1.0.2

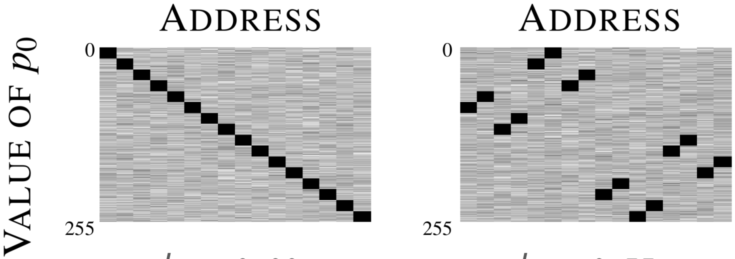
- Known-plaintext attack
- Events: encryption with only one fixed key byte
- Profile each event
- Exploitation phase:
  - Eliminate key candidates
  - Reduction of key space in first-round attack:
    - 64 bits after 16–160 encryptions

Attack 4: AES T-Table Template Attack

AES T-Table implementation from OpenSSL 1.0.2

- Known-plaintext attack
- Events: encryption with only one fixed key byte
- Profile each event
- Exploitation phase:
  - Eliminate key candidates
  - Reduction of key space in first-round attack:
    - 64 bits after 16–160 encryptions
  - State of the art: full key recovery after 30000 encryptions

Attack 4: AES T-Table Template



$k_0 = 0x00$ $k_0 = 0x55$

(transposed)

Conclusion

- Novel technique to find any cache side-channel leakage
  - Attacks
  - Detect vulnerabilities

Conclusion

- Novel technique to find any cache side-channel leakage
  - Attacks
  - Detect vulnerabilities
- Marks a change of perspective:

Conclusion

- Novel technique to find any cache side-channel leakage
  - Attacks
  - Detect vulnerabilities
- Marks a change of perspective:
  - Large scale analysis of (unknown) binaries
  - Large scale automated attacks

Cache Template Attacks:

Automating Attacks on Inclusive Last-Level Caches

Daniel Gruss, Raphael Spreitzer, and Stefan Mangard,
IAIK, Graz University of Technology

August 14, 2015