Silently Breaking ASLR In The Cloud

Antonio Barresi, Kaveh Razavi, Mathias Payer, Thomas R. Gross

black hat
EUROPE 2015

Amsterdam, November 12, 2015
Who we are

Antonio Barresi
antonio.barresi@xorlab.com
@AntonioHBarresi
Co-founder of
 xorlab
 Interested in software and systems security topics.

http://www.antoniobarresi.com
https://www.xorlab.com

Dr. Kaveh Razavi
kaveh@cs.vu.nl
Researcher at
 VU
 Research on building reliable and secure computing systems.

http://www.cs.vu.nl/~kaveh/
http://www.cs.vu.nl
Agenda

- Threat scenario
- Memory deduplication
- Side-channel
- CAIN attack (Cross-VM Address Space Layout INtrospection)
- Evaluation
- Post-CAIN exploitation
- Mitigations
Threat scenario

VM

VM

Virtual Machine Monitor

Hardware
Threat scenario

Attacker VM

Attacker has user privileges

Victim VM

Process

Victim OS

Virtual Machine Monitor

Hardware
Threat scenario

Attacker VM
- Attacker has user privileges
- Knows a memory-corruption vulnerability in Victim VM's process

Victim VM
- Process

Virtual Machine Monitor

Hardware
Threat scenario

Attacker VM

- Attacker has user privileges

Needs to know the exact location of code regions to construct a code-reuse attack

Victim VM

- Process
- Code

Victim OS

Address-Space Layout Randomization (ASLR)

Virtual Machine Monitor

Hardware
Threat scenario

Attacker VM

Attacker has user privileges

If the VMM uses page based same content memory deduplication

Victim VM

Process

Victim OS

Virtual Machine Monitor

Hardware
Threat scenario

Attacker VM
- Attacker has user privileges
- 0x7f9ffaa0000
- Infer randomized base address of libraries or executable

Victim VM
- Process
- Code

Victim OS

Virtual Machine Monitor

Hardware
Threat scenario

Attacker VM
- Attacker has user privileges
- Perform code-reuse attack with now known code locations
  0x7f9ffaa0000

Victim VM
- Process
- Code

Victim OS

Virtual Machine Monitor

Hardware
CAIN: Cross-VM ASL INtrospection

> New attack vector against memory deduplication
> CVE-2015-2877
> VU#935424 (https://www.kb.cert.org/vuls/id/935424)

> Leaks randomized base addresses (RBAs) of
  > libraries and
  > executables
  > in processes running on neighboring VMs
Memory deduplication

VM A

VM B

Physical memory

Virtual Machine Monitor
Memory deduplication

VM A

VM B

Physical memory

Virtual Machine Monitor

merge
Memory deduplication

VM A  VM B  Physical memory

Virtual Machine Monitor
Memory deduplication

VM A

VM B

Physical memory

Virtual Machine Monitor

Copy-on-Write

write
Memory deduplication

VM A

VM B

Virtual Machine Monitor

Physical memory
Memory deduplication
Memory deduplication side-channel

Attacker VM

Victim VM

Attacker has user privileges

Side-channel
Memory deduplication side-channel

Attacker VM

- Attacker has user privileges
- Memory page (guess)

Side-channel

Victim VM
Memory deduplication side-channel

Attacker VM

Attacker has user privileges

Memory page (guess)

wait(t)

Side-channel

Victim VM
Memory deduplication side-channel

Attacker VM

Attacker has user privileges

Memory page (guess)

write

Side-channel

Victim VM
Memory deduplication side-channel

Attacker VM

- Attacker has user privileges
- Memory page (guess)

Victim VM

- Side-channel
  - If write time > threshold
  - Page exists in another VM!
Memory deduplication side-channel

Attacker VM

Attacker has user privileges

Memory page (guess)

Write

Side-channel

If write time <= threshold

Page does not exist in another VM.

Victim VM
Memory deduplication side-channel

> Attacker can craft a page (guess)
Memory deduplication side-channel

> Attacker can craft a page (guess)

> Wait for a certain amount of time
Memory deduplication side-channel

> Attacker can craft a page (guess)

> Wait for a certain amount of time

> Write to page and measure time
Memory deduplication side-channel

> Attacker can craft a page (guess)
>
> Wait for a certain amount of time

> Write to page and measure time

> Write time will reveal if page was shared
CAIN: Cross-VM ASL INtrospection

Adaptive sleep time detection → Filtering → Verification
Suitable pages to break ASLR

> Mostly static
> Read-only in victim VM
> Known to exist
Suitable pages to break ASLR

Contains base address of an executable image
Suitable pages to break ASLR

Contains values derived from the base address of an executable image
Suitable pages to break ASLR

Entropy = ASLR entropy
Suitable page under Windows

PE File Format on Disk

0x5a4d

DOS Header

COFF Header

ImageBase: 0x180000000

Optional Header

Section Table

[Code & Data]

PE File Format in Memory

0x5a4d

DOS Header

COFF Header

ImageBase: 0x7f9ffaa0000

Randomized DLL base address, 19 bits of entropy

4096 bytes 1st page of DLL in memory
Suitable page under Windows

We also found a page under Linux x64...
Guessing the right address

> Well, you still have to guess
Guessing the right address

> Well, you still have to guess
> $2^{19}$ base addresses for Windows x64
> 524'288 guesses
> One guess requires 1 page of memory
BRUTE FORCE

If it doesn't work, you're just not using enough.
Guessing the right address

> Well, you still have to guess
> $2^{19}$ base addresses for Windows x64
> 524’288 guesses
> One guess requires 1 page of memory

> Attacker VM has much more memory
> Fill up memory with all guesses
> $2^{19} \times 1$ page of 4 KB = 2 GB
Brute-force all addresses

0x7f9ffa70000
0x7f9ffa80000
0x7f9ffa90000
0x7f9ffaa0000
0x7f9ffab0000
0x7f9ffac0000
0x7f9ffad0000

...
Brute-force all addresses

```
detect_shared_pages()
```

```plaintext
0x7f9ffa0000
0x7f9ffa10000
0x7f9ffa20000
0x7f9ffa30000
0x7f9ffa40000
0x7f9ffa50000
```

Page contents?
Wait for how long?

> Depends on the memory deduplication implementation
Wait for how long?

> Depends on the memory deduplication implementation

> Varies depending on amount of memory used
Wait for how long?

- Depends on the memory deduplication implementation

- Varies depending on amount of memory used

- **Attacker trade-off**
  - Waiting too little obstructs the attack
  - Waiting too long increases attack time
Adaptive sleep-time detection

> Create random buffer
Adaptive sleep-time detection

> Create random buffer

> Copy every second page of 1st half to the 2nd half
Adaptive sleep-time detection

> Create random buffer

> Copy every second page of 1st half to the 2nd half

> Begin with $t_{\text{start}}$
  > Detect merged pages
  > Iterate and increase test time until detection rate is near 100%

$N = 10'000$, $t_{\text{start}} = 10 \text{ min}$
Adaptive sleep-time detection

> Create random buffer

> Copy every second page of 1st half to the 2nd half

> Begin with $t_{start}$
  > Detect merged pages
  > Iterate and increase test time until detection rate is near 100%

$t_{start} = t_1 = 10$ min, loop:

test_time($t_n$)

If detection rate > 95%
  return $t_n \times 1.2$

If detection rate < 50%
  $t_{n+1} = t_n \times 2$

Else
  $t_{n+1} = t_n \times (1/[\text{detection rate}])$
Detect merged pages

Non-shared

Merged

Non-shared
Detect merged pages

\[ t_1 \rightarrow \text{Non-shared} \rightarrow \text{Merged} \rightarrow \text{Non-shared} \]

Noise? Threshold?
Detect merged pages

\[ t_1 \rightarrow \text{Non-shared} \]
\[ t_2 \rightarrow \text{Merged} \]
\[ \text{Non-shared} \]

Noise? Threshold?

29

2667
Detect merged pages

Measure write time with rdtsc
(Read Time Stamp Counter)

t₁ → Non-shared 29

2667

Merger

t₂ → Merged

Non-shared

34
Detect merged pages

Measure write time with \texttt{rdtsc}
(Read Time Stamp Counter)

\[ t_2 > 2 \times \frac{(t_1 + t_3)}{2} \quad t_{1,3} < M = 1000 \quad t_1 < t_3, (t_3 - t_1) < \frac{t_3}{3} \]
Detect merged pages

These heuristics worked for different HW configurations

\[ t_2 > 2 \]

\[ 1000 \]

\[ t_1 < t_3, (t_3 - t_1) < t_3/3 \]
Handling noise

> Noise can affect write time
  > High write time for non-shared page
  > Missed shared page because of increased write time of adjacent pages

> Perform several rounds of detection
  > Noise will cancel out
  > Eliminate candidates
Handling noise

Round 1

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>G1</td>
<td>G2</td>
<td>G3</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gn</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Handling noise

Round 1

G1
G2
G3
...
Gn

detect()

G22
G37
G98

Noise? Threshold?
False positive
False positive
Handling noise

Round 1
- G1
- G2
- G3
- ...
- Gn

detect() ->

Round 2
- G22
- G37
- G98

Noise? Threshold?

False positive

False positive
Handling noise

Round 1
- G1
- G2
- G3
- ...
- Gn

Round 2
- G22
- G37
- G98

detect()
Handling noise

Round 1
G1
G2
G3
...
Gn

Round 2
G22
G37
G98

detect()

Round 3
G22
G37

G22
G37
G98

False positive
False positive
False positive

Noise? Threshold?
Handling noise

Round 1

G1
G2
G3
...
Gn

detect()

G22
G37
G98

Round 2

G22
G37
G98

detect()

G22
G37

Round 3

G22
G37

detect()

G37

Noise?
Threshold?

False positive
False positive
False positive
Filtering

> As little memory per candidate as possible
> Once at the beginning to eliminate as many candidates as possible

Noise? Threshold?
Verification

> Verify remaining potential candidates

> Use more memory per candidate

> Eliminate remaining false positives
Does it work?
Does it work?

We implemented CAIN and evaluated the PoC
CAIN requirements

- Side-channel exists (memdedup is on)
- Suitable page exists (for breaking ASLR)
  - Other «secrets» might be interesting too
- Entropy is within brute-forceable range
  - For given amount of attacker memory
CAIN attack phases
CAIN attack phases

Adaptive sleep time detection

Goal:
detect a wait time \( t \) required to merge pages
CAIN attack phases

Adaptive sleep time detection

Goal: detect a wait time \( t \) required to merge pages

Filtering

Goal: reduce number of candidates with as little memory

1 round
CAIN attack phases

Adaptive sleep time detection

Goal: detect a wait time $t$ required to merge pages

Filtering

Goal: reduce number of candidates with as little memory

1 round

Verification

Goal: verify remaining candidates and remove false positives

N rounds
Evaluation setup
Evaluation setup

- Dual CPU Blade Server
  - 2 x AMD Opteron 6272 CPUs with 16 cores each
  - 32GB of RAM
  - VMM: KVM on Ubuntu Server 14.04.2 LTS x86_64
  - Linux Kernel 3.16.0

- 1 attacker VM with Ubuntu Linux 14.04
- 6 victim VMs with Windows Server 2012 (6.2.9200 Build 9200)
  - 4 vCPUs, 4 GB per VM
Kernel same-page Merging (KSM)

- Enabled by default for KVM (Ubuntu Server)
  - Out-of-band Content Based Page Sharing (CBPS)

/sys/kernel/mm/ksm/run  ‘1’ or ‘0’
/sys/kernel/mm/ksm/sleep_millisecs  e.g., 200 ms
/sys/kernel/mm/ksm/pages_to_scan  e.g., 100
Kernel same-page Merging (KSM)

> Enabled by default for KVM (Ubuntu Server)
> Out-of-band Content Based Page Sharing (CBPS)

/sys/kernel/mm/ksm/run  ‘1’ or ‘0’
/sys/kernel/mm/ksm/sleep_millisecs  e.g., 200 ms
/sys/kernel/mm/ksm/pages_to_scan  e.g., 100

\[
1000 / \text{sleep\_millisecs} \times \text{pages\_to\_scan} = \text{pages per second}
\]

\[
e.g., (1000/200\text{ms}) \times 100 = 500 \text{ pages/sec}
\]
ASLR in Windows x64

> High Entropy ASLR
> 33 bits for stacks
> 24 bits for heaps
> 17 bits for executables
> 19 bits for DLLS

System-wide at boot-time for certain images

http://www.nynaeve.net/?p=198
Attacking a single Windows VM

![Graph showing ASLR entropy over time](image)

**time t = 0 min**
Attacking a single Windows VM

19 bits of entropy
524,288 potential base addresses

time t = 0 min
Attacking a single Windows VM

19 bits of entropy
524,288 potential base addresses

Sleep-time detection = 96 min per round

time t = 0 min
Attacking a single Windows VM

After first round 6'550 candidates remain, entropy = 12.68

time t = 96 min
Attacking a single Windows VM

After the second round 5 candidates remain, entropy = 2.32

time t = 192 min
Attacking a single Windows VM

Down to 1 candidate, the actual base address

time $t = 288$ min
Attacking a single Windows VM

and various merge times
Attacking a Windows VM under load

Victim VM runs IIS webserver, load generated with a separate physical machine and AB (apache benchmark)

sleep_millisecs = 20
Attacking multiple Windows VMs

sleep_millisecs = 20
Demo
Post-CAIN exploitation

- De-randomized base address can now be used in code-reuse attack
- Against a single victim, use the inferred value
- Against multiple victims...
Post-CAIN exploitation

> What will the attacker really get?
Post-CAIN exploitation

> What will the attacker really get?

> Assume a vulnerability where:
  > Attacker can reliably hijack EIP/RIP
  > Attacker has some control over data
  > No infoleak for ASLR (code) bypass
Post-CAIN exploitation

Phrack #68 from 2012-04-14

Exploiting MS11-004

Microsoft IIS 7.5 remote heap buffer overflow

by redpantz

http://phrack.org/issues/68/12.html#article
Post-CAIN exploitation

--- [ 8 - Conclusion

Although Microsoft and many others claimed that this vulnerability would be impossible to exploit for code execution, this paper shows that with the correct knowledge and enough determination, **impossible turns to difficult.**
Post-CAIN exploitation

--- [ 8 - Conclusion

Although Microsoft and many others claimed that this vulnerability would be impossible to exploit for code execution, this paper shows that with the correct knowledge and enough determination, **impossible turns to difficult.**

.....

10) **This will obtain EIP with multiple registers pointing to user-controlled data.** From there ASLR and DEP will need to be subverted to gain code execution. Take a look at DATA_STREAM_BUFFER.Size, which will determine how many bytes are sent back to a user in a response.

Although full arbitrary code execution wasn't achieved in the exploit, it still proves that a remote attacker can potentially gain control over EIP via a remote unauthenticated FTP connection that can be used to subvert the security posture of the entire system, instead of limiting the scope to a denial of service.
Real-world environments

- Memory deduplication is used in commercial public cloud environments
  - some providers disable it
  - some VMMs do not support it at all
  - usually not apparent to customers if used or not

- If KVM is used, it is likely enabled

- Most providers offer 4 GB and more
  - 10 – 30 VMs per host is usual
# Affected VMMs

<table>
<thead>
<tr>
<th>Vendor</th>
<th>Memdedup</th>
<th>cert.org*</th>
<th>We think</th>
<th>Enabled by default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux KVM</td>
<td>KSM</td>
<td>Affected</td>
<td>Yes</td>
<td>n.a.</td>
</tr>
<tr>
<td>Ubuntu (KVM)</td>
<td>KSM</td>
<td>-</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Red Hat (KVM)</td>
<td>KSM</td>
<td>Affected</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Parallels</td>
<td>-</td>
<td>Affected</td>
<td>Not sure</td>
<td>-</td>
</tr>
<tr>
<td>Microsoft Hyper-V</td>
<td>None</td>
<td>Not Affected</td>
<td>Not Affected</td>
<td>n.a.</td>
</tr>
<tr>
<td>Xen</td>
<td>None (yet?)</td>
<td>Not Affected</td>
<td>Not Affected</td>
<td>n.a.</td>
</tr>
<tr>
<td>Oracle VirtualBox</td>
<td>PageFusion</td>
<td>Unknown</td>
<td>Probably</td>
<td>No</td>
</tr>
<tr>
<td>VMware</td>
<td>TPS</td>
<td>Unknown</td>
<td>Probably</td>
<td>No</td>
</tr>
</tbody>
</table>

* [https://www.kb.cert.org/vuls/id/935424](https://www.kb.cert.org/vuls/id/935424)
Disclosure & vendor responses
Disclosure & vendor responses

> Well, who should fix it?
Disclosure & vendor responses

> Well, who should fix it?

> We informed several VMM vendors that we thought might be affected
> Advisory sent on June 4, 2015, all replied
> Overall perception: low severity issue
Disclosure & vendor responses

> Well, who should fix it?

> We informed several VMM vendors that we thought might be affected
  > Advisory sent on June 4, 2015, all replied
  > Overall perception: low severity issue

= won’t fix
Mitigations

> VMM layer: Deactivation of memory deduplication

> System layer: Attack detection

> ASLR layer: Increase ASLR entropy

> Process layer: More entropy in sensitive memory
Mitigations

> VMM layer: Deactivation of memory deduplication

> System layer: Attack detection

> ASLR layer: Increase ASLR entropy

> Process layer: More entropy in sensitive memory
Black Hat Sound Bytes

- Memory deduplication considered harmful
- CAIN breaks ASLR of co-located VMs
- Don’t use memory dedup in public clouds
Memory deduplication considered harmful

CAIN breaks ASLR of co-located VMs

Don’t use memory dedup in public clouds

Antonio Barresi
antonio.barresi@xorlab.com
@AntonioHBarresi

Kaveh Razavi
kaveh@cs.vu.nl