Memory Deduplication: The Curse that Keeps on Giving

Erik Bosman, Ben Gras, Kaveh Razavi, Antonio Barresi

VU
VRJE
UNIVERSITEIT
AMSTERDAM

xorlab

33c3
EM ROE SKROW
GRUSBAH NCC
YANAREG
SI SHIT, OLLER
Who we are

Erik Bosman
erik@minemu.org
@brainsmoke
PhD candidate at VUSec
Research on building reliable and secure computing systems.
https://www.vusec.net/

Antonio Barresi
antonio.barresi@xorlab.com
@AntonioHBarresi
Co-founder of XORLAB
Interested in software and systems security topics.
https://www.xorlab.com
Acknowledgments

Cristiano Giuffrida
Herbert Bos
Mathias Payer
Bart Preneel
Mathias Payer
Thomas R. Gross
Our message today...
ONE DOES NOT SIMPLY ENABLE MEMORY DEDUPLICATION
Outline
Outline

> Memory deduplication
Outline

> Memory deduplication
> Side-channel
CAIN

Cross-VM leak, break ASLR
Memory deduplication
Intra-process read + write (Browser + JS)
Memory deduplication + Rowhammer
Cross-VM leak + write, system compromise
Memory deduplication + Rowhammer
Outline

> Memory deduplication
> Side-channel

> CAIN attack (2015)
> Dedup Est Machina (2016)
> Flip-Feng Shui (2016)
Outline

> Memory deduplication
> Side-channel

> CAIN attack (2015)
> Dedup Est Machina (2016)
> Flip-Feng Shui (2016)

> Conclusion
Memory deduplication
Memory deduplication

A method of reducing memory usage.
Memory deduplication

A method of reducing memory usage.

Used in virtualization environments,
Memory deduplication

A method of reducing memory usage.

Used in virtualization environments, (was) also enabled by default on Windows 8.1 and 10.
Memory deduplication

In virtualized environments it allows to reclaim memory and supports overcommitment of memory.
Memory deduplication

In virtualized environments it allows to reclaim memory and supports overcommitment of memory.

= run more VMs
Now we can sell even more VMs... $$$
Memory deduplication

physical memory

virtual machine A

virtual machine B
Memory deduplication

<table>
<thead>
<tr>
<th>Physical memory</th>
<th>Virtual machine A</th>
<th>Virtual machine B</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image.png" alt="Image" /></td>
<td><img src="image.png" alt="Image" /></td>
<td><img src="image.png" alt="Image" /></td>
</tr>
</tbody>
</table>
Memory deduplication

physical memory

virtual machine A

virtual machine B
Memory deduplication

physical memory

virtual machine A

virtual machine B
Memory deduplication

physical memory

virtual machine A

virtual machine B
Memory deduplication

physical memory

virtual machine A

virtual machine B
Kernel Same-page Merging (KSM)

> Enabled by default for KVM (Ubuntu Server)
> Out-of-band Content Based Page Sharing (CBPS)
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/sleep_millisecs * pages_to_scan = pages per second
  e.g., (1000/200ms) * 100 = 500 pages/sec
Memory deduplication: The Problem

Deduplicated memory does not need to have the same security domain.

(Unlike fork(), file-backed memory)

An attacker can use deduplication as a side-channel.
Deduplication side-channel attack

normal write
Deduplication side-channel attack

normal write

write
Deduplication side-channel attack

normal write

write

write

copy on write (due to deduplication)
Deduplication side-channel attack

normal write

write

write

copy on write (due to deduplication)

* trap
Deduplication side-channel attack

normal write

write

write

copy on write (due to deduplication)

trap

copy whole page
Deduplication side-channel attack

normal write

write

copy on write (due to deduplication)

* trap

copy whole page

update page tables
Deduplication side-channel attack

normal write

write

copy on write (due to deduplication)

trap

copy whole page

update page tables

resume process
Deduplication side-channel attack

normal write

write

copy on write (due to deduplication)

trap

copy whole page

update page tables

resume process

write
Deduplication side-channel attack

A 1-bit side channel which is able to leak data across security boundaries
Deduplication side-channel attack

A 1-bit side channel which is able to leak data across security boundaries

> Cross-VM
Deduplication side-channel attack

A 1-bit side channel which is able to leak data across security boundaries

> Cross-VM

> Cross-process
Deduplication side-channel attack

A 1-bit side channel which is able to leak data across security boundaries

> Cross-VM

> Cross-process

> Intra-process, leak process data from JavaScript
Exploitation of the side-channel
Exploitation of the side-channel

attacker memory

victim memory
Exploitation of the side-channel

attacker memory

victim memory

secret page
Exploitation of the side-channel

guess page

secret page

attacker memory

victim memory
Exploitation of the side-channel

guess page

secret page

attacker memory

victim memory
Exploitation of the side-channel

wait(t) attacker memory

secret page victim memory
Exploitation of the side-channel

write

attacker memory

victim memory

secret page
Exploitation of the side-channel

write time $> \text{threshold}$
Exploitation of the side-channel

write time > threshold

attacker memory

victim memory

secret page
Exploitation of the side-channel

write time \leq \text{threshold}

attacker memory

victim memory

secret page
Exploitation of the side-channel

write time ≤ threshold

attacker memory

victim memory

secret page
Exploitation of the side-channel

\[\text{write time} \leq \text{threshold}\]

attacker memory

victim memory

secret page
CAIN:
Cross-VM Address Space Layout Introspection

Deduplication
(software side-channel)
CAIN:
Cross-VM Address Space Layout Introspection

Deduplication
(software side-channel)

Cross-VM leak / ASLR bypass

CVE-2015-2877 / VU#935424 (https://www.kb.cert.org/vuls/id/935424)
CAIN
> Page contents to leak ASLR? Secret page?
> Page contents to leak ASLR? Secret page?

> How long to wait?
CAIN

> Page contents to leak ASLR? Secret page?

> How long to wait?

> How to detect a merged page? Noise?
Suitable pages to break ASLR

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

- Page aligned
- 0x7f9ffaa0000: Known offset within the page
- Contains base address of an executable image
Suitable pages to break ASLR

Page aligned

Known offsets within the page

Contains values derived from the base address of an executable image
Suitable page under Windows

PE File Format on Disk

DOS Header

COFF Header

ImageBase: 0x180000000

Optional Header

Section Table

(Code & Data)

PE File Format in Memory

DOS Header

COFF Header

ImageBase: 0x7f9ffaa0000

Randomized DLL base address, 19 bits of entropy

4096 bytes
1st page of DLL in memory
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
Guessing the right address

> Well you still have to guess
> $2^{19}$ base addresses for Windows x64
> 524,288 guesses
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
Based on http://sourceforge.net/projects/mpimd5bruteforc/

BRUTE FORCE

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

> Attacker VM has much more memory
Guessing the right address

> Attacker VM has much more memory

> Fill up memory with all guesses
Guessing the right address

> Attacker VM has much more memory
  > Fill up memory with all guesses
  > $2^{19} \times 1 \text{ page of } 4 \text{ KB} = 2 \text{ GB}$
<table>
<thead>
<tr>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x7f9ffa70000</td>
</tr>
<tr>
<td>0x7f9ffa80000</td>
</tr>
<tr>
<td>0x7f9ffa90000</td>
</tr>
<tr>
<td>0x7f9ffaa0000</td>
</tr>
<tr>
<td>0x7f9ffab0000</td>
</tr>
<tr>
<td>0x7f9ffac0000</td>
</tr>
<tr>
<td>0x7f9ffad0000</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>
Brute-force all addresses

detect_shared_pages()

0x7f9ffaa0000
Wait for how long?
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

> Try to automatically detect sleep time
Adaptive sleep-time detection

> Try to automatically detect sleep time
Adaptive sleep-time detection

> Try to automatically detect sleep time

> After buffer creation, wait e.g. t = 10min
Adaptive sleep-time detection

> Try to automatically detect sleep time

> After buffer creation, wait e.g. \( t = 10\text{min} \)
  > Detect how many pages were merged
Adaptive sleep-time detection

> Try to automatically detect sleep time

> After buffer creation, wait e.g. t = 10min
  > Detect how many pages were merged
  > If detection rate > threshold (e.g. 90%)
Adaptive sleep-time detection

> Try to automatically detect sleep time

> After buffer creation, wait e.g. \( t = 10 \text{min} \)
  > Detect how many pages were merged
  > If detection rate > threshold (e.g. 90%)
    > Use \( t \)
Adaptive sleep-time detection

> Try to automatically detect sleep time

> After buffer creation, wait e.g. \( t = 10 \text{min} \)
  > Detect how many pages were merged
  > If detection rate > threshold (e.g. 90%)  
    > Use \( t \)
  > Else, increase \( t \) and try again
Detect merged pages

- Non-shared
- Merged
- Non-shared
Detect merged pages

$t_1$ → Non-shared
    → Merged
    → Non-shared

29
Detect merged pages

- $t_1 \rightarrow \text{Non-shared}$
- $t_2 \rightarrow \text{Merged}$
- $t_2 \rightarrow \text{Non-shared}$

Count: 29, 2667
Detect merged pages

Measure write time with rdtsc
(Read Time Stamp Counter)

- $t_1$ → Non-shared
- $t_2$ → Merged
- $t_3$ → Non-shared

Values:
- 29
- 2667
- 34
Detect merged pages

Measure write time with `rdtsc` (Read Time Stamp Counter)

- \( t_1 \rightarrow \text{Non-shared} \)
- \( t_2 \rightarrow \text{Merged} \)
- \( t_3 \rightarrow \text{Non-shared} \)

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

These heuristics worked for different HW configurations

\[ t_2 > 2 \]

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

> Be conservative and perform multiple rounds
Handling noise

> Be conservative and perform multiple rounds

> Probability that same guess is affected by noise in different rounds is low
Windows x64 ASLR

> 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 attack time (min)]
Attacking multiple Windows VM

sleep_millisecs = 20
[ATTACK - CREATE PAGES] napped 1st page to memory (0x7f791c979000)
[ATTACK - CREATE PAGES] napped page buffer (0x7f791b0c9000)
[ATTACK - RUN - FILTERING] filtering rounds are completed, remaining can
[ATTACK - RUN - VERIFICATION] total attack time so far 720 s / 12 min
[ATTACK - RUN - VERIFICATION] reccuring 3527 attack pages

[ATTACK - RUN - VERIFICATION] win64_server_2012.create_attack_pages()
[ATTACK - RUN - VERIFICATION] unmap previous buffer
[ATTACK - RUN - VERIFICATION] 1st page file dump opened (bin\win2012\win2012_n
[ATTACK - RUN - VERIFICATION] napped 1st page to memory (0x7f791c979000)
[ATTACK - RUN - VERIFICATION] wait for pages to be merged (approx. 12 m
[ATTACK - RUN - VERIFICATION] verification round 1 done

[ATTACK - RUN - VERIFICATION] recceating 38 attack pages
[ATTACK - RUN - VERIFICATION] win64_server_2012.create_attack_pages()
[ATTACK - RUN - VERIFICATION] napped page buffer (0x7f791b0c9000)
[ATTACK - RUN - VERIFICATION] verification rounds are completed

[ATTACK - RUN - RESULTS] *** HIT: 00007ffe59f0000, rating: 2/2 (address

[ATTACK SUMMARY]
> ATTACK TIME 1440 s / 24 min
> HTS 1
> FILTERING ROUNDS 1
> VERIFICATION ROUNDS 1
> TOTAL ROUNDS 2

[done]
Speed improvements

> Many ways to increase speed of attack
Speed improvements

> Many ways to increase speed of attack

> Allocate more random pages in-between
Speed improvements

> Many ways to increase speed of attack

> Allocate more random pages in-between

> Use more than one guess page (redundancy)
Speed improvements

> Many ways to increase speed of attack

> Allocate more random pages in-between

> Use more than one guess page (redundancy)
  > Different guess pages for same secret
    e.g. relocated code pages 😊
Big limitation

> No control over victim memory layout
Big limitation

> No control over victim memory layout

> Some control would help a lot 😊
Big limitation

- No control over victim memory layout
  - Some control would help a lot 😊

- No write primitive
Big limitation

> No control over victim memory layout
  > Some control would help a lot 😊

> No write primitive
  > Rowhammer 😊
memdedup for Windows

> MS enabled memory deduplication for Windows 8.1 + 10
memdedup for Windows

> MS enabled memory deduplication for Windows 8.1 + 10

THERE ARE NO BAD IDEAS

ONLY GREAT IDEAS THAT GO HORRIBLY WRONG
CAIN
Est
Dedup
Flip-
Machina
Feng
Shui
Dedup est Machina

Deduplication
(software side-channel)
Dedup est Machina

Deduplication
(software side-channel)
+
Rowhammer
(hardware bug)
Dedup est Machina

Deduplication
(software side-channel)
+
Rowhammer
(hardware bug)

Exploit MS Edge without software bugs
(from JavaScript)
Outline:

Deduplication
- leak heap & code addresses

JavaScript Array

+0.0
+3.141592
42.
1
NaN
Outline:

**Deduplication**
- leak heap & code addresses

JavaScript Array:

- +0.0
- +3.141592
- 42.
- 1
- NaN

chakra.dll
Outline:

Deduplication

- leak heap & code addresses
- create a fake object
Deduplication
- leak heap & code addresses
- create a fake object

Rowhammer
- create reference to our fake object
Outline:

Deduplication
- leak heap & code addresses
- create a fake object

Rowhammer
- create reference to our fake object
Leaking existing pages is slow and the gained information is limited.

What if we can manipulate the contents of the victim's memory to leak secrets hand-picked by the attacker.
Challenge 1:

The secret we want to leak does not span an entire page.
Turning a secret into a page
Turning a secret into a page

secret  \[\rightarrow\]  known data

secret page
Challenge 2:

The secret we want to leak has too much entropy to leak all at once.
Primitive #1: alignment probing

- secret

- known data

- secret page
Primitive #1: alignment probing

secret

known data

secret page
Primitive #2: partial reuse

secret → known data

secret page
Primitive #2: partial reuse

known data

secret

secret page
Outline:

Deduplication
- leak heap & code addresses
JIT function epilogue (MS Edge)

mov RCX,0x1c20  mov RAX, [code address]  jmp RAX  trap  trap  trap  trap  trap  trap  trap  trap  trap  trap  trap ...

known data
JIT function epilogue (MS Edge)

mov RCX, 0x1c20  mov RAX, [code address]

jmp RAX  trap  trap  trap  trap
       trap  trap  trap  trap  trap
       trap  trap  trap  trap  trap
       trap  trap  trap  trap  trap
       trap  trap  trap  trap  trap
JIT function epilogue (MS Edge)

mov RCX, 0x1c20
mov RAX, [code address]
jmp RAX
trap
Outline:

Deduplication

- leak heap & code addresses
Outline:

Deduplication
- leak heap & code addresses
What if leaking a heap pointer in stages is not possible...

We need to guess a page containing the complete pointer.
Heap pointer entropy in Edge

0x5F48143540
Heap pointer entropy in Edge

advertised ASLR (24 bit)

\[ 0x5F48143540 \]
Heap pointer entropy in Edge

advertised ASLR (24 bit)

non-deterministic bits (+/- 36 bit)

0x5F48143540

64G  * redundancy

256T  * redundancy
Slab allocator for JavaScript objects

array object

array data
Slab allocator for JavaScript objects

1M VirtualAlloc()
Slab allocator for JavaScript objects

1st after VirtualAlloc() call

1M VirtualAlloc()
Slab allocator for JavaScript objects

1st after VirtualAlloc() call

Timing side-channel :-D

1M VirtualAlloc()
Heap pointer entropy in Edge

advertised ASLR (24 bit)

non-deterministic bits (+/- 36 bit)

0x5F48143540

64G * redundancy

256T * redundancy
Heap pointer entropy in Edge

advertised ASLR (24 bit)

0x5F48100000

entropy after 1MB alignment (20 bit)

64G * redundancy

4G * redundancy
Birthday problem
Birthday problem
Birthday problem
Birthday problem
Birthday problem
Birthday problem
Birthday problem
Birthday problem
Birthday problem
Birthday problem
Birthday problem
Birthday problem
Primitive #3: birthday heapspray

physical memory

attacker memory

victim memory
Primitive #3: birthday heapspray

physical memory

attacker memory

victim memory
Primitive #3: birthday heapspray

Physical memory

Attacker memory

Victim memory
Primitive #3: birthday heapspray

physical memory

attacker memory

victim memory
Primitive #3: birthday heapspray

physical memory

attacker memory

victim memory
Primitive #3: birthday heapspray

physical memory

attacker memory

victim memory
Primitive #3: birthday heapspray

physical memory

attacker memory

victim memory
Primitive #3: birthday heapspray

<table>
<thead>
<tr>
<th>Physical Memory</th>
<th>Attacker Memory</th>
<th>Victim Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Physical Memory Grid" /></td>
<td><img src="image2.png" alt="Attacker Memory Grid" /></td>
<td><img src="image3.png" alt="Victim Memory Grid" /></td>
</tr>
</tbody>
</table>
Primitive #3: birthday heapspray

physical memory

attacker memory

victim memory
Primitive #3: birthday heapspray

physical memory

attacker memory

victim memory
Primitive #3: birthday heapspray

physical memory

attacker memory

victim memory
Creating Secret Pages

1M Aligned objects
Creating Secret Pages

Array data

1M Aligned objects
Creating Secret Pages

1M Aligned objects
Creating Secret Pages

page
page
page
page

1M Aligned objects
Creating Guess Pages

typed array data
Creating Guess Pages

guessed aligned addresses, 128M apart
Creating Guess Pages

guess X
guess Y
guess Z
guess Q

guessed aligned addresses, 128M apart
Birthday heap spray

+1M, +1M, +1M, ...

\[ \text{Diagram of a large heap with injections at 1M and 128M.} \]
Birthday heap spray

+1M, +1M, +1M, ...

secret pages (allocated addresses)

+128M,
+128M,
+128M,
...

?? ?? ?? ?? ?? ?? ?? ?? ?? ??
Birthday heap spray

+1M, +1M, +1M, ...

secret pages (allocated addresses)

guess pages (containing guessed addresses)
Birthday heap spray

+1M, +1M, +1M, ...

secret pages (allocated addresses)

guess pages (containing guessed addresses)
Outline:

Deduplication
- leak heap & code addresses
- create a fake object

Rowhammer
- create reference to our fake object
Fake Uint8Array object
Pointer pivotting
Pointer pivotting

array
header

JavaScript Array
Pointer pivotting

array header

array data

JavaScript Array

JavaScript Array
Pointer pivotting

array header

array header

array data

JavaScript Array

JavaScript Array
Pointer pivotting

JavaScript Array

array
header

array
header

array
data

JavaScript Array
Rowhammer attack
Rowhammer attack

rows

row buffer cache
Rowhammer attack

text:

- Rowhammer attack
- rows
- row buffer cache
Rowhammer attack

rows

row buffer cache
Rowhammer attack

row buffer cache

rows
Rowhammer attack

rows

row buffer cache
Rowhammer attack

Rows

Row buffer cache
Rowhammer attack way make the system unstable and crash.
Pointer pivotting

```
array header
```

```
array header
```

```
array data
```

```
JavaScript Array
```

```
JavaScript Array
```
Pointer pivoting

Array header

Array header

Array data

JavaScript Array

JavaScript Array
Flip Feng Shui

Rowhammer
(hardware bug)
Flip Feng Shui

Rowhammer (hardware bug) + Deduplication (more than a software side-channel)
Flip Feng Shui

Rowhammer (hardware bug) + Deduplication (more than a software side-channel)

↓

Cross-VM compromise
Rowhammer bit flips:

1) Unpredictable on which (virtual) page
2) Unpredictable where in the page
3) Repeatable once you've found a flip
Flip Feng Shui goal:

> Find victim pages with known content which allow for exploitation when certain bits are flipped

> Land this victim page in a physical memory location where this bit is flippable
Deduplication implementation: Windows 10

physical memory

attacker memory

victim memory
Deduplication implementation:
Windows 10

physical memory

attacker memory

victim memory
Deduplication implementation: 
KVM on Linux (KSM)

physical memory

attacker memory

victim memory
Deduplication implementation: KVM on Linux (KSM)

physical memory

attacker memory

victim memory
Deduplication implementation: KVM on Linux (KSM)

physical memory

attacker memory

victim memory
Deduplication implementation:
KVM on Linux (KSM)

physical memory

attacker memory

victim memory
Deduplication implementation:
KVM on Linux (KSM)

physical memory

attacker memory

victim memory
Deduplication implementation: KVM on Linux (KSM)

physical memory

attacker memory

victim memory
Deduplication implementation: KVM on Linux (KSM)
Deduplication implementation: KVM on Linux (KSM)
Deduplication implementation: KVM on Linux (KSM)

physical memory

attacker memory

victim memory
Example 1: OpenSSH

Target: ~/.ssh/authorized_keys
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQC52/Uk84iUmmicel7ESr+/D/PWZ6Ljkhlu8yv35bEEoTwXm9eGxJyzV+1s68tRyzpD3VQvwSHiKqDnCg+0taAo0KvCqZcoBQFB9XawIfJI5dSeGtcUBuokUv+TlmAZ+D9MNNAxjuSBBH0ShbaiH65imlauISfR3VZWFE7uy6sB26j52LhWG5BRwSkMnMRN2E2fqHaP96J9R0FlHuykw8jwUXJw4kJ8vRo1uhX0SVu8Z9wGrKR5b+GQWJ3Ph7vjoMVU/KoAbWnNnYKR8ITBnkPD0LrEyAKRygEfi7gwcix0vQR79by8lL6ypJ4kM5eyobSBsNCjmghxQj8RRzGUtd1 victim@laptop

Exponent

Modulus (p * q)
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQC52/Uk84iUmmicel7ESr+/D/PWZ6Ljkhlu8yv35bEEoTwXm9eGxJyzV+1s68tRyzpD3VQvwSHiKqDnCg+0taAoOkvCqZcoBQFB9XawIfJI5dSeGtcUBuokUv+TlmAZ+D9MNAXjuxSBBH0ShbaiH65imlauISfR3VZWFED7uy6sB26j52LhWG5BRwSkMnMRN2E2fqHaP96J9R0FlHuykw8jwUXJw14kJ8vRo1uhX0SVu8Z9wGrKR5b+GQWJ3Ph7vjoMVU/KoAbWnNnYKR8ITBnkPDOLrEyAKRygEfi7gcix0vQR79by8ll6ypJ4kM5eyobSBsNCjmghxQj8RRzGUTdl victim@laptop

Exponent

Modulus (p' * q' * r' ...)
Example 1: OpenSSH

Target: ~/.ssh/authorized_keys

> Flip a bit in the RSA modulus
> Factorize it
> Reconstruct the new private key
Example 2: GPG & apt-get

Targets: sources.list

flip package repository domain name
eg. ubuntu.com -> ubunvu.com
Example 2: GPG & apt-get

Targets: sources.list
      + GPG keyring

corrupt signing key
Conclusion
Conclusion

> Memory deduplication is dangerous
Conclusion

> Memory deduplication is dangerous

> Be aware of the security implications
Conclusion

> Memory deduplication is dangerous

> Be aware of the security implications

> Well, or just disable it
Erik Bosman
erik@minemu.org
@brainsmoke

Antonio Barresi
antonio.barresi@xorlab.com
@AntonioHBarresi
<table>
<thead>
<tr>
<th>Physical memory</th>
<th>Sprayed page tables</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>