Software Exploitation: Hardware is the New Black

Cristiano Giuffrida
Vrije Universiteit Amsterdam

https://vusec.net

Bug-free Software: Expectations

Bug-free Software: Reality

Credits

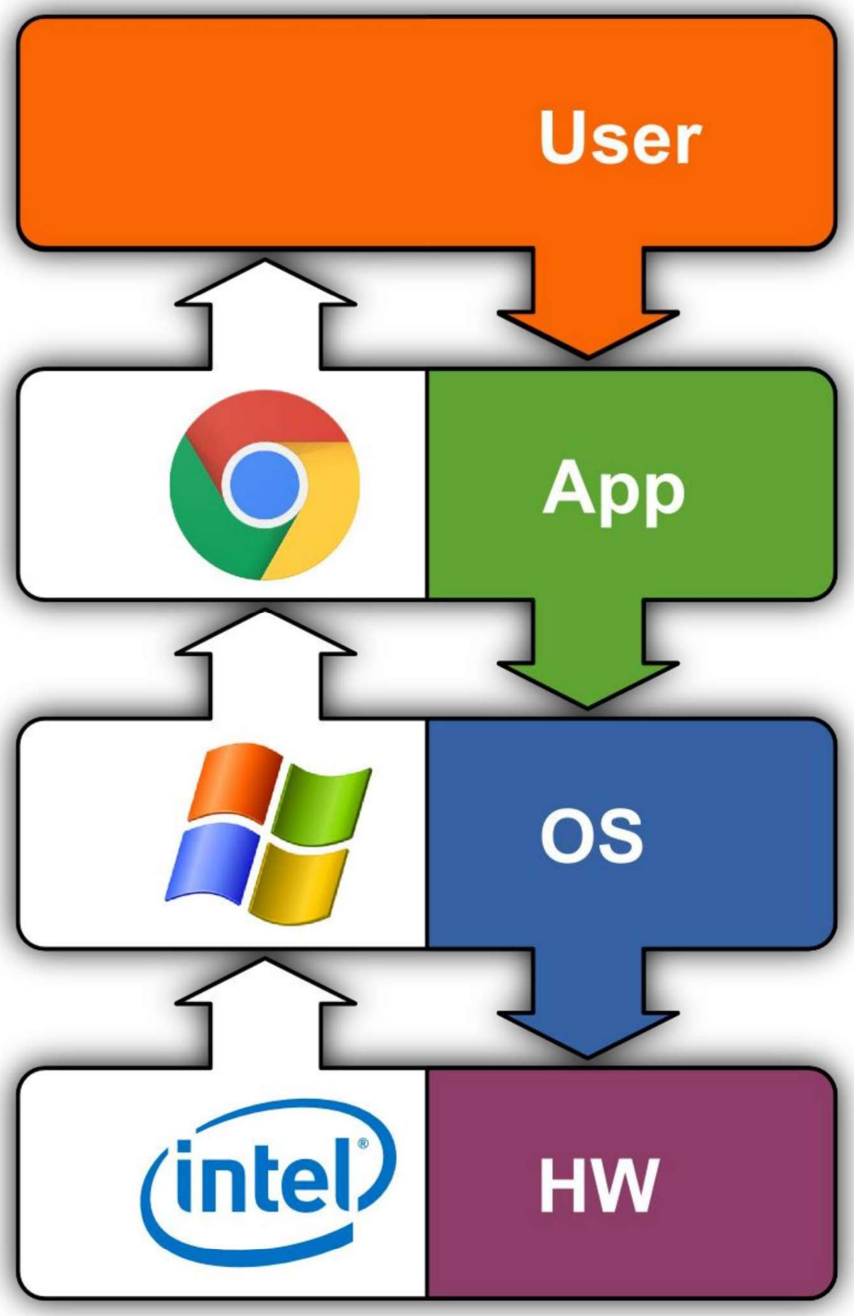
- Erik Bosman
- Ben Gras
- Kaveh Razavi
- Victor van der Veen
- Andrei Tatar
- Lucian Cojocar
- Herbert Bos

VUSec

https://vusec.net

Software Exploitation:

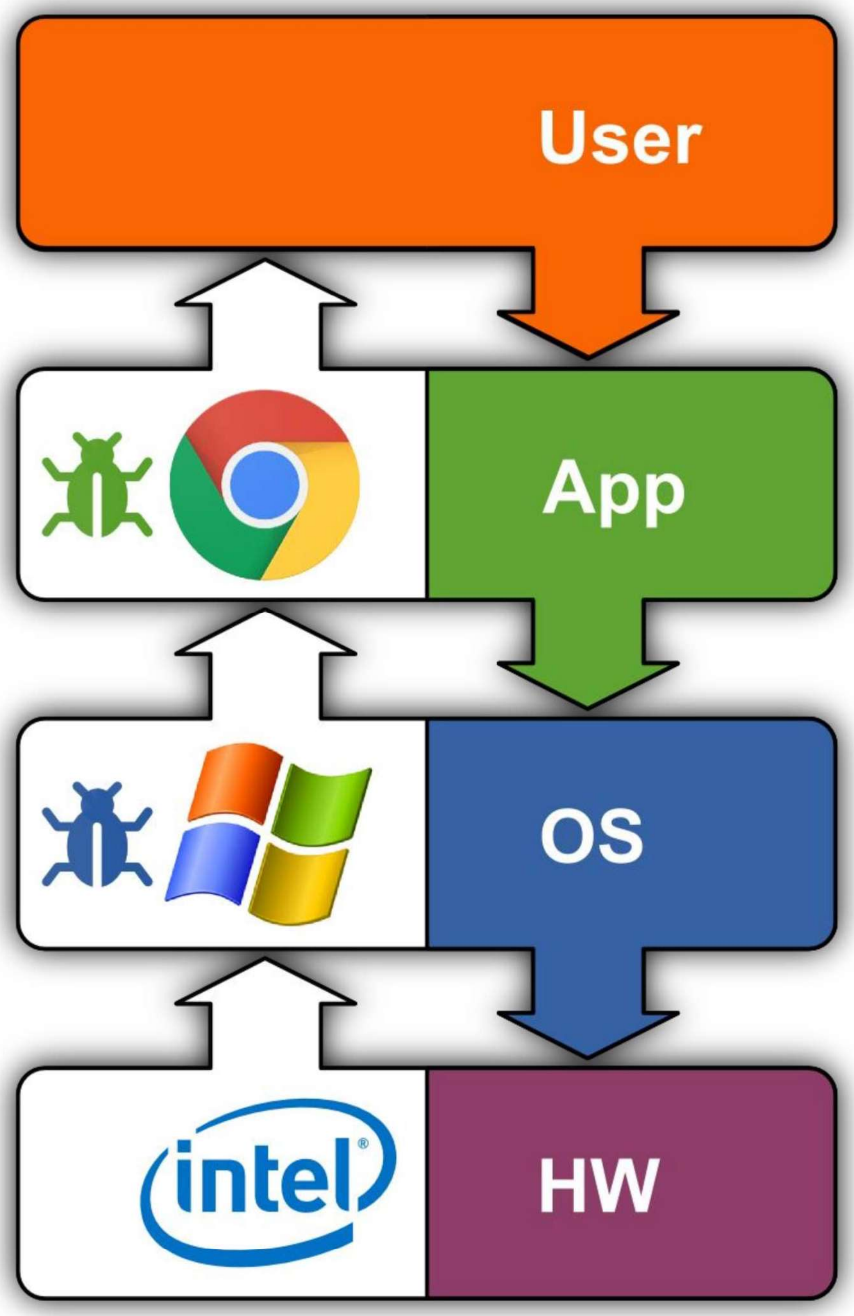
2014



Software Exploitation:

2014

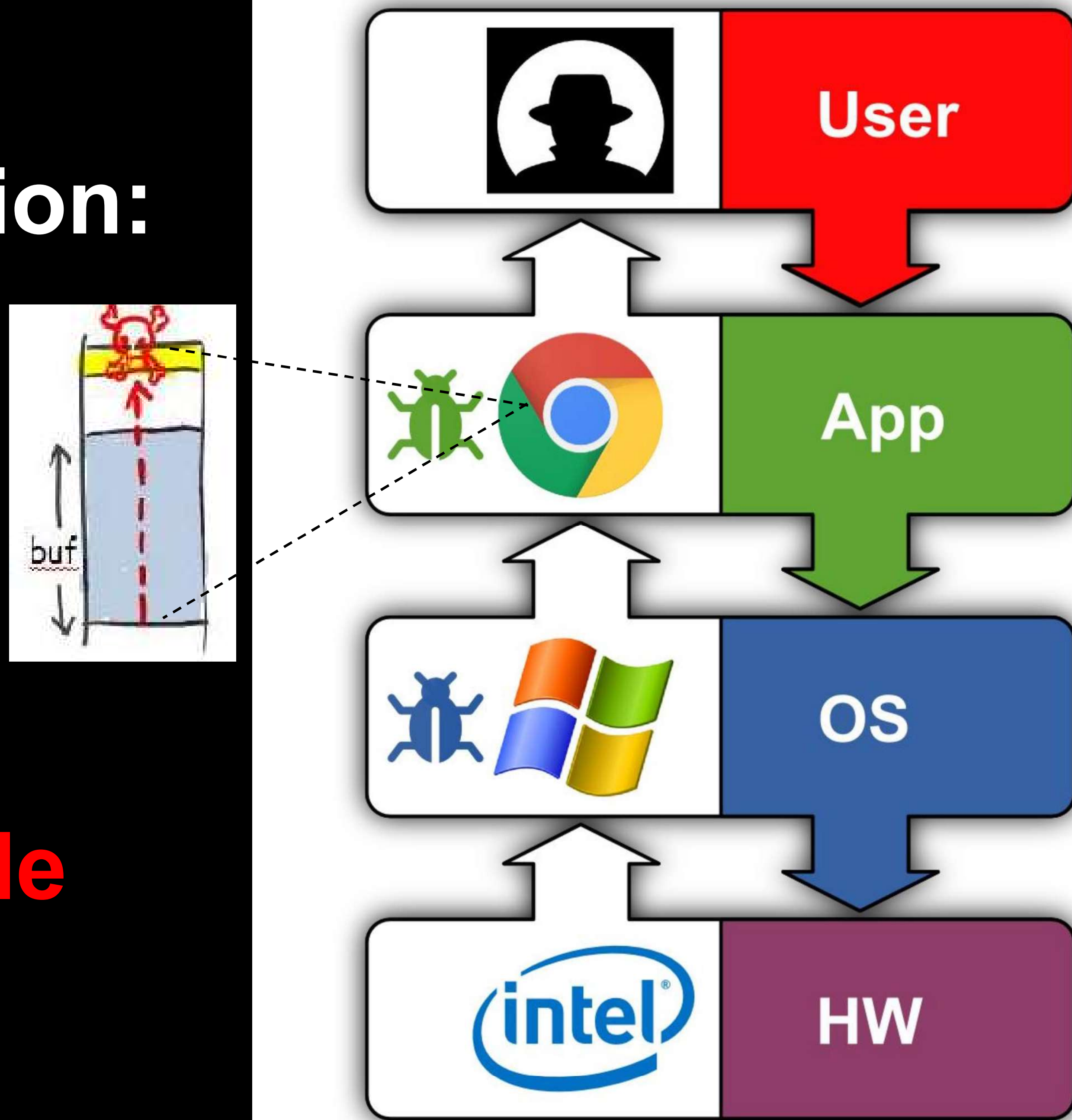
Bugs, Bugs Everywhere!



Software Exploitation:

2014

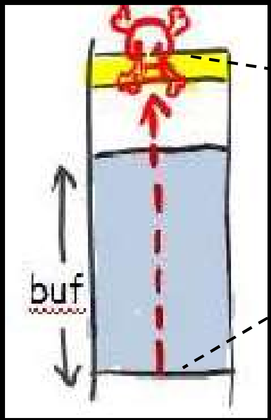
Attacker Exploits Vulnerable Software

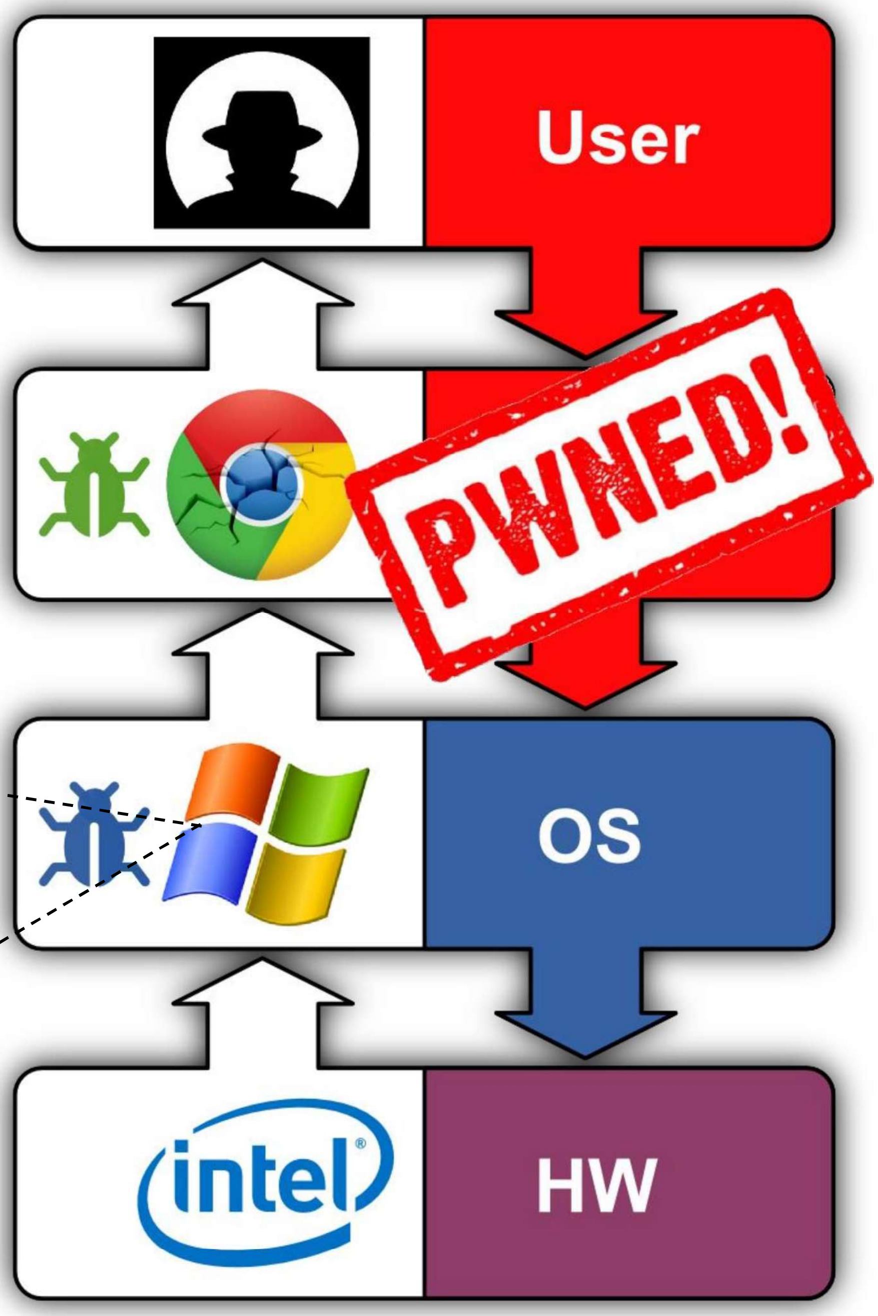


Software Exploitation:

2014

Attacker Owns App

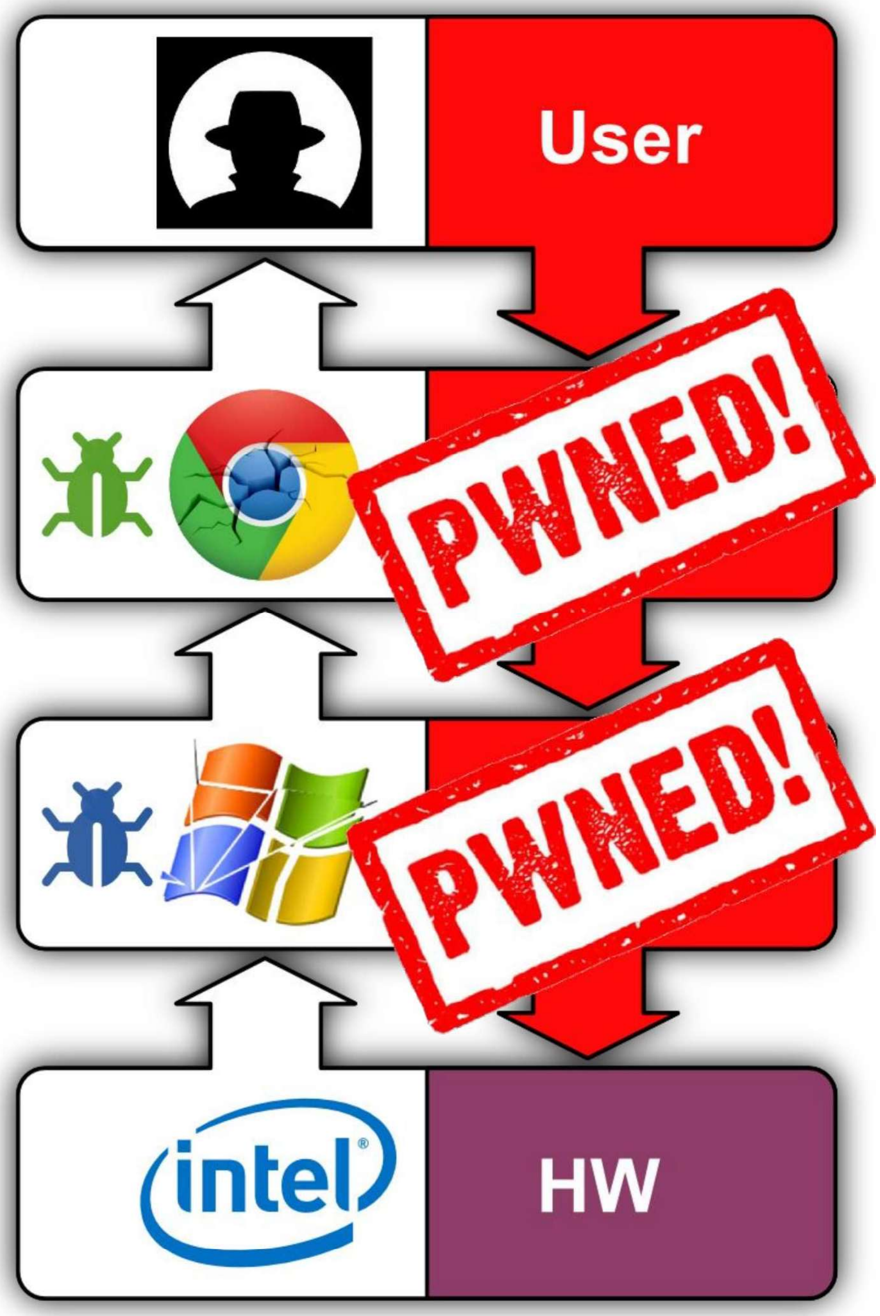




Software Exploitation:

2014

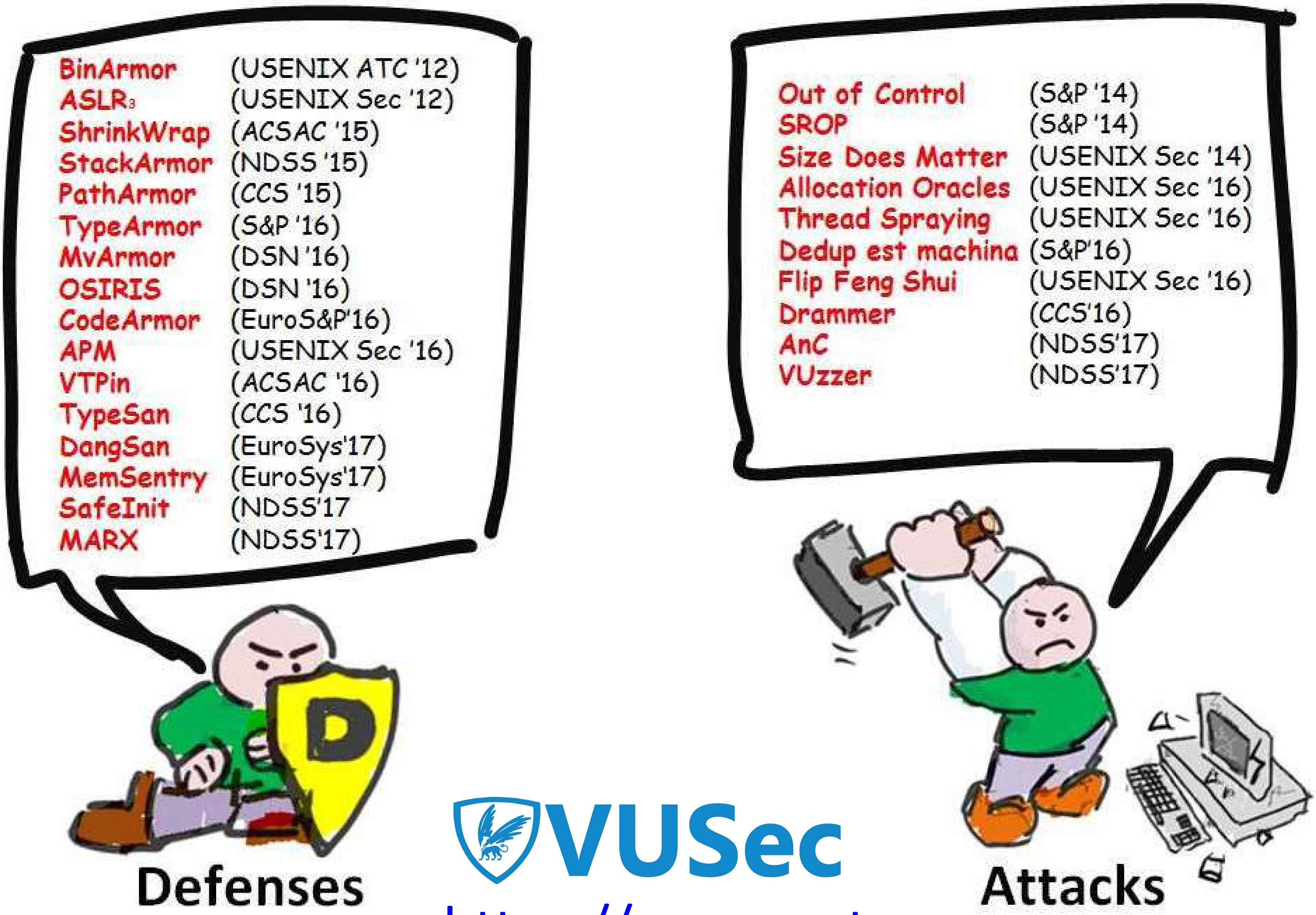
Attacker Owns System



Software Exploitation: 2014

- Systems security problems caused by **bugs**
  - Software and configuration bugs
  - Weak security implementations

- **Impossible** to write software without bugs
  - However, we can mitigate their impact
  - Many defenses proposed by industry and academia



VUSec

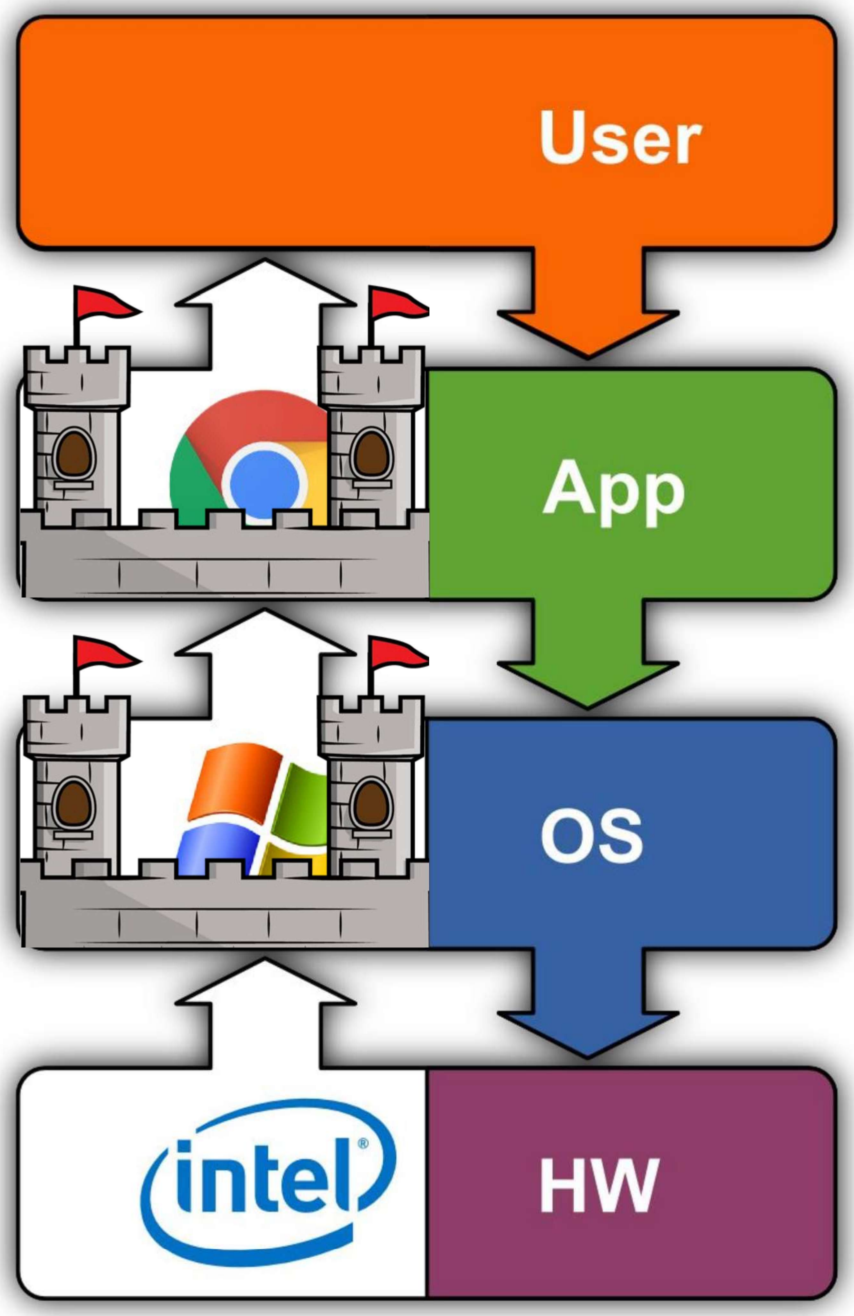
https://vusec.net

Software Exploitation:

2019

Exploits Difficult:

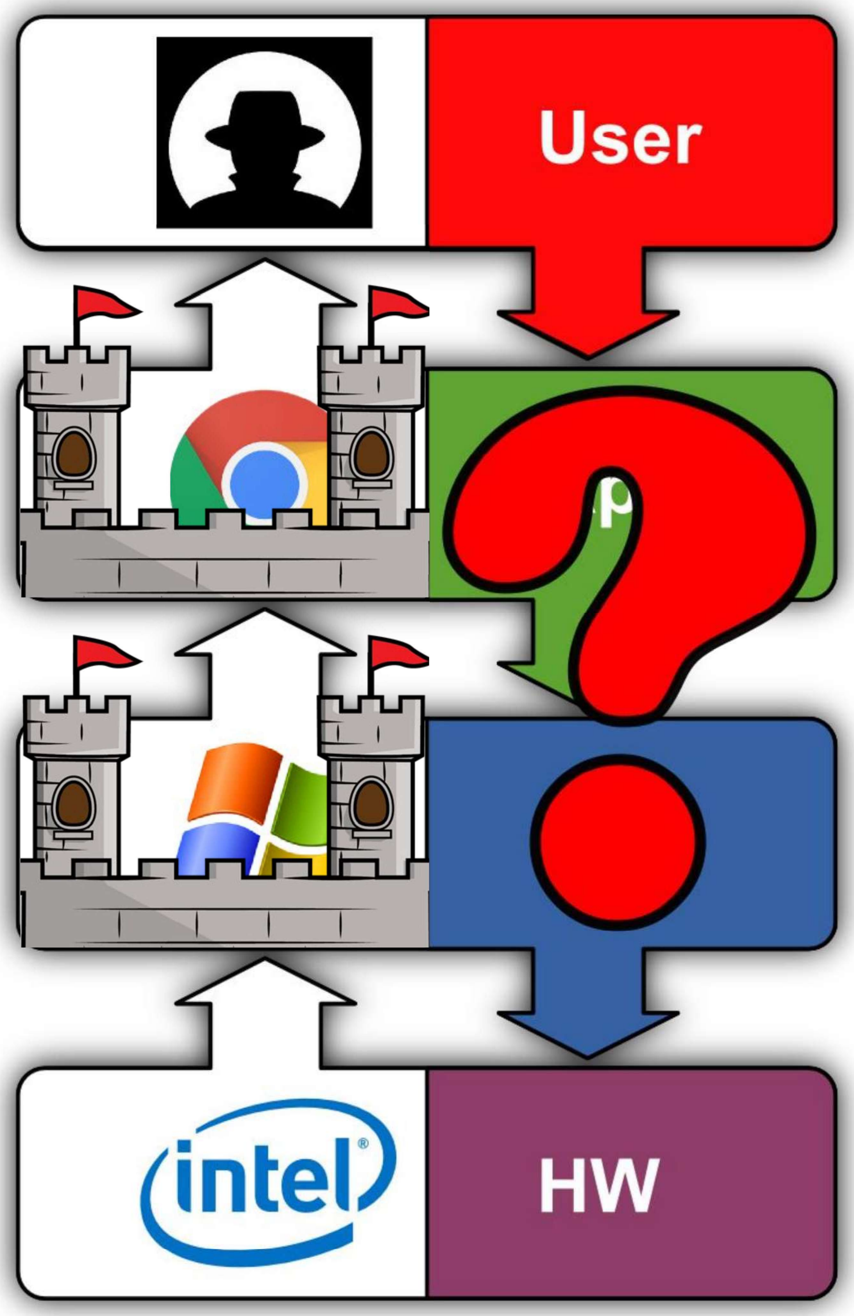
- Mitigations
- Verification



Software Exploitation:

2019

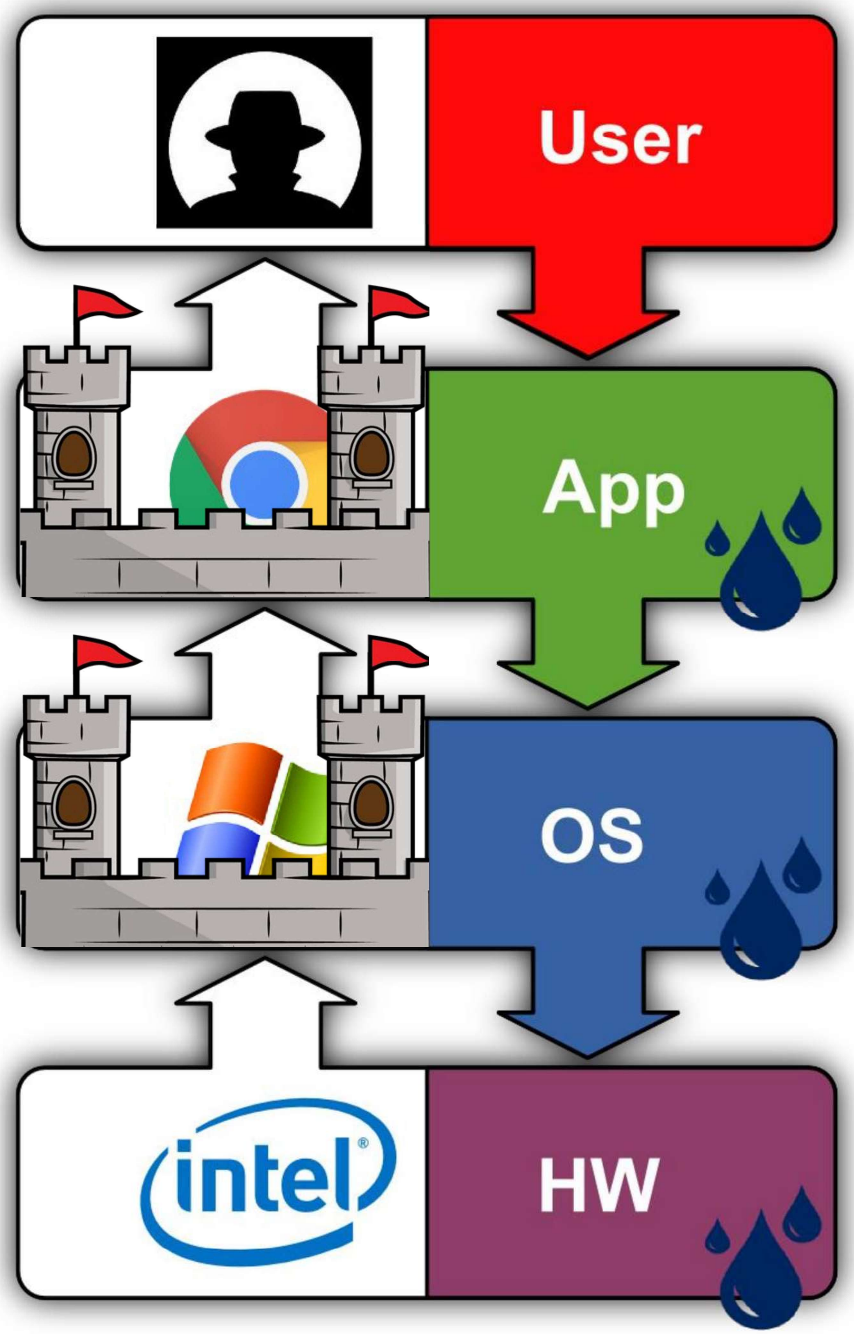
How to Find Memory R/W Primitives?



Software Exploitation:

2019

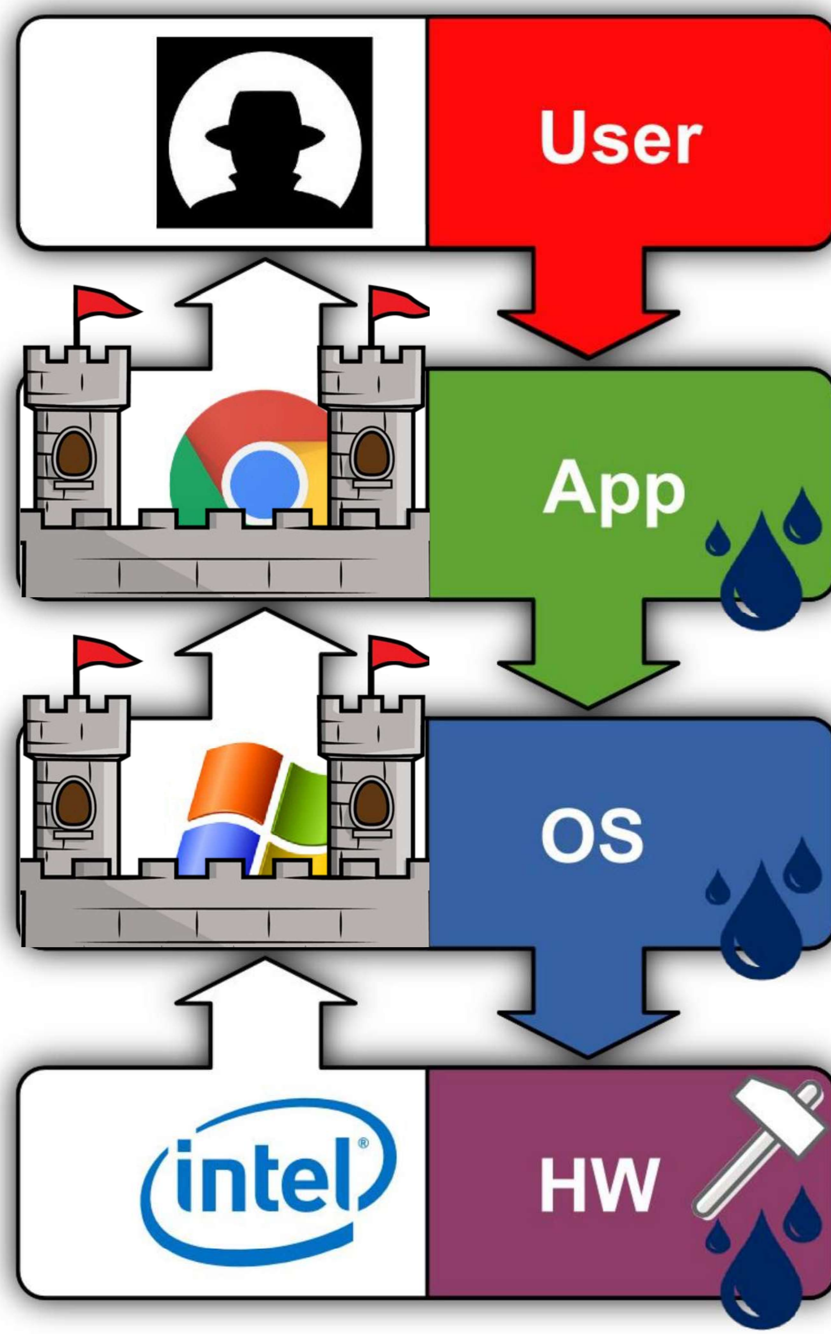
Memory R: Hw/Sw Side Channels



Software Exploitation:

2019

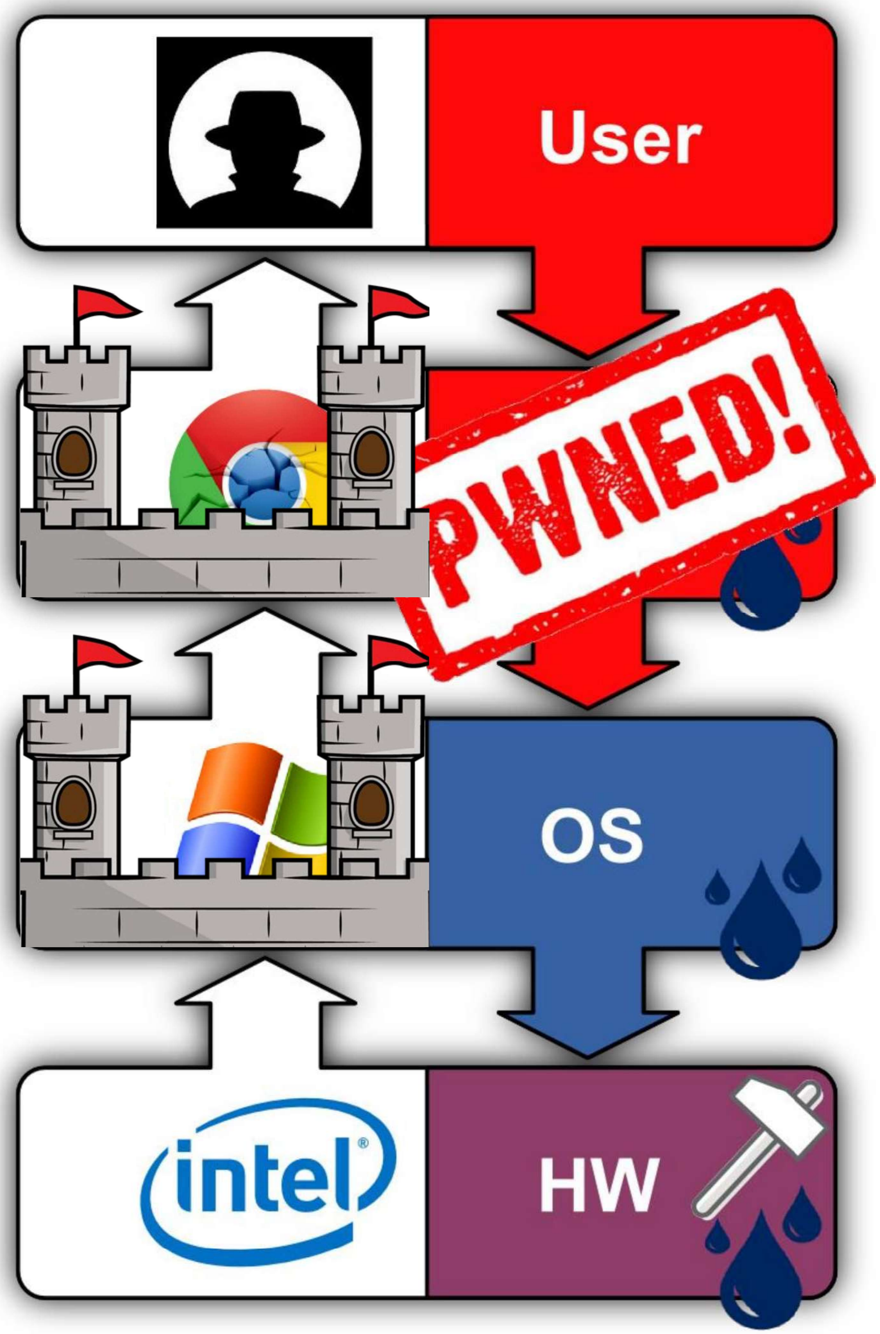
Memory W: Hardware Glitches



Software Exploitation:

2019

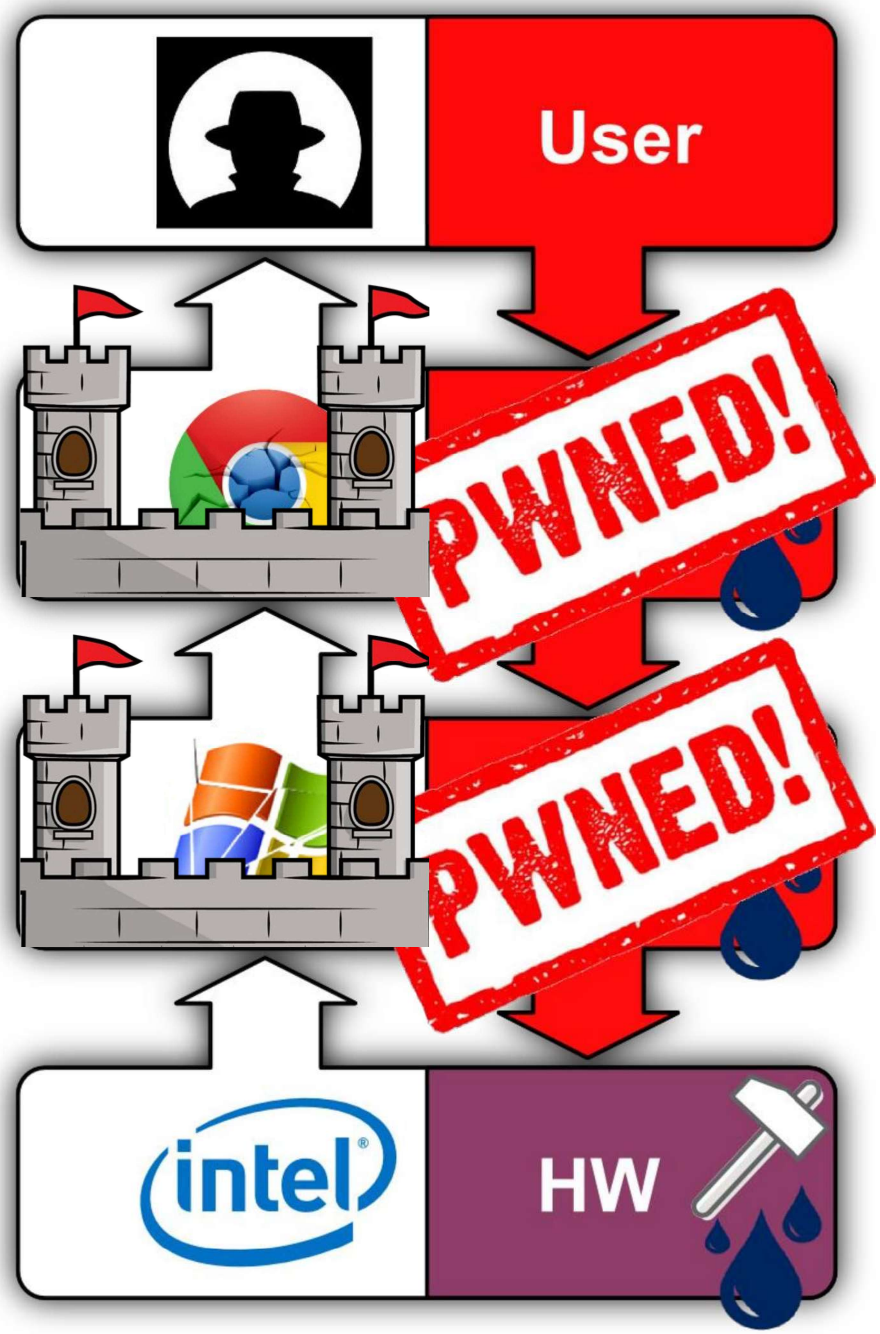
Memory R/W: Back to Reliable Exploits



Software Exploitation:

2019

Memory R/W: Back to Reliable Exploits



Software Exploitation: 2019

- Even if the software is **perfect**...
  - ...with no bugs, well-configured, and latest defenses
  - ...or even formally verified
  - ...it is still **vulnerable**!

- Attackers abuse **properties** of modern hw and sw for reliable exploitation

- We’ll look at **some examples** with different properties

EXAMPLE 1

Bug-free Exploitation in Browsers

Dedup Est Machina

- Published at IEEE S&P 2016
  - with Erik, Kaveh, Herbert
- Won **Pwnie Award** at Black HAT 2016

"*Most Innovative Research*"

- Complete exploit of Microsoft Edge browser on Windows 10 from malicious JavaScript
  - ...without relying on a single software bug

Dedup Est Machina

Memory deduplication
(software side channel)

Dedup Est Machina

Memory deduplication

(software side channel)

+

Rowhammer

(hardware glitch)

Dedup Est Machina

Memory deduplication

(software side channel)

+

Rowhammer

(hardware glitch)

↓

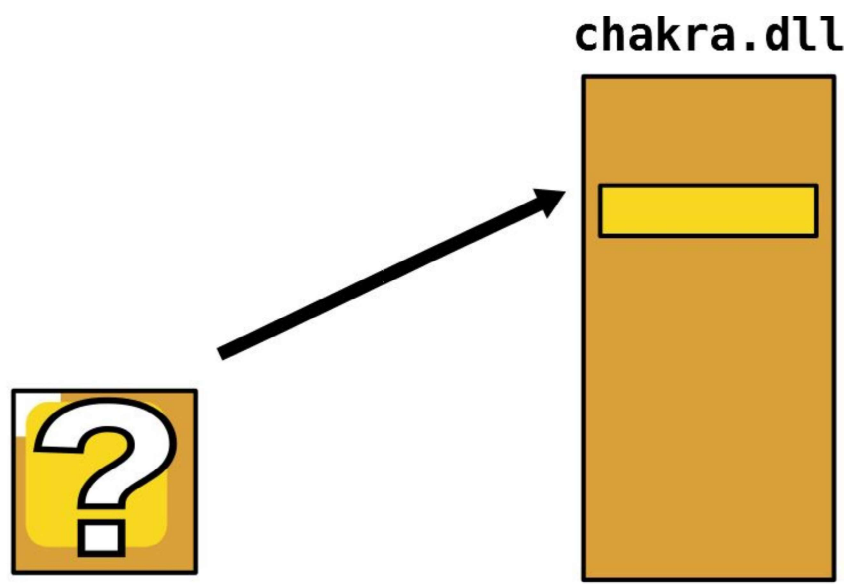
Exploit MS Edge without software bugs

(from JavaScript)

Dedup Est Machina: Overview

Memory deduplication

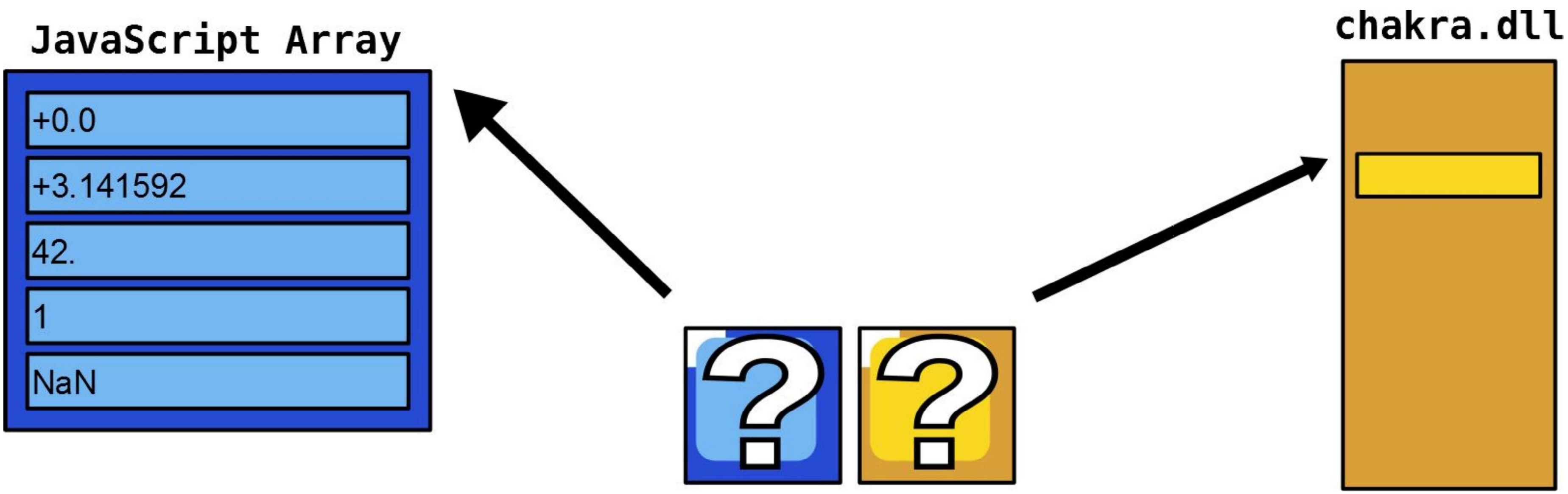
- Leak randomized heap and code pointers



Dedup Est Machina: Overview

Memory deduplication

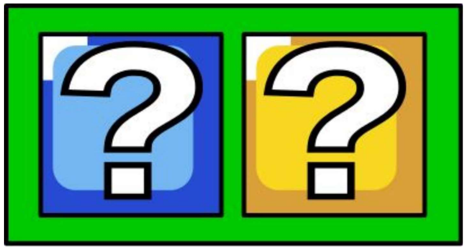
- Leak randomized heap and code pointers



Dedup Est Machina: Overview

Memory deduplication

- Leak randomized heap and code pointers
- Create a fake JavaScript object



Dedup Est Machina: Overview

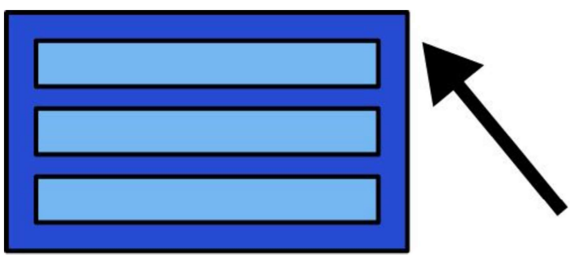
Memory deduplication

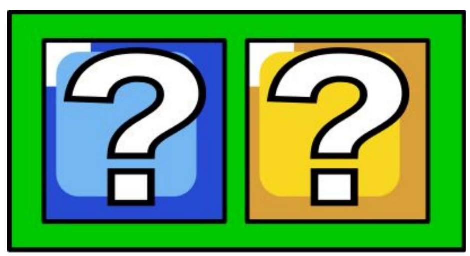
- Leak randomized heap and code pointers
- Create a fake JavaScript object

+

Rowhammer

- Create a reference to our fake object





Dedup Est Machina: Overview

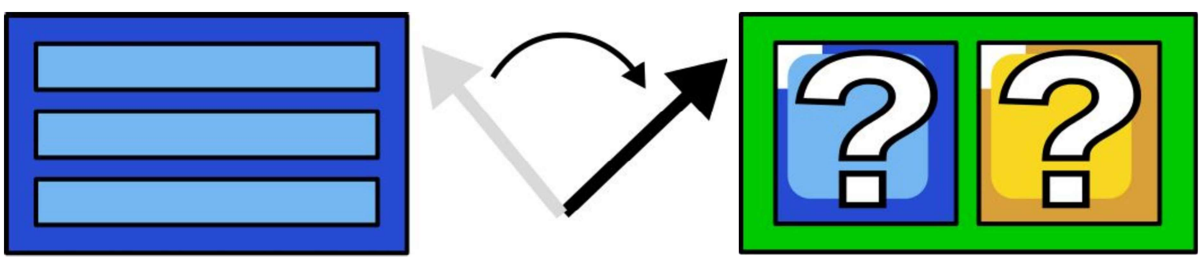
Memory deduplication

- Leak randomized heap and code pointers
- Create a fake JavaScript object

+

Rowhammer

- Create a reference to our fake object



Dedup Est Machina: Overview

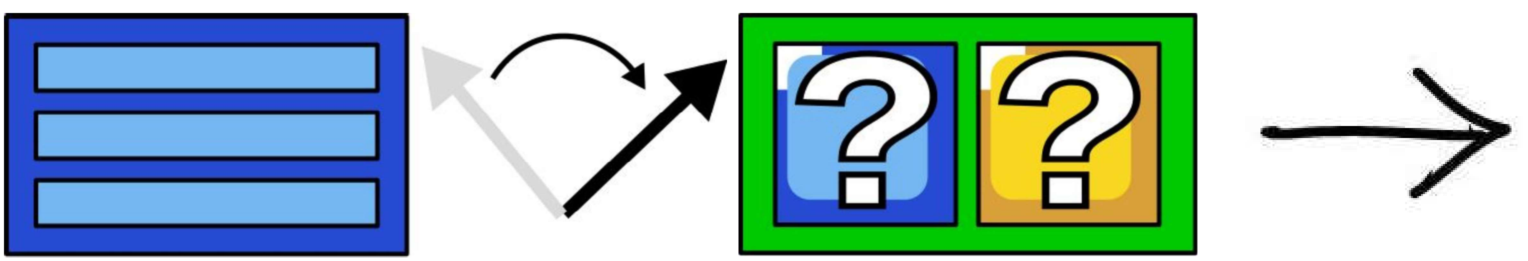
Memory deduplication

- Leak randomized heap and code pointers
- Create a fake JavaScript object

+

Rowhammer

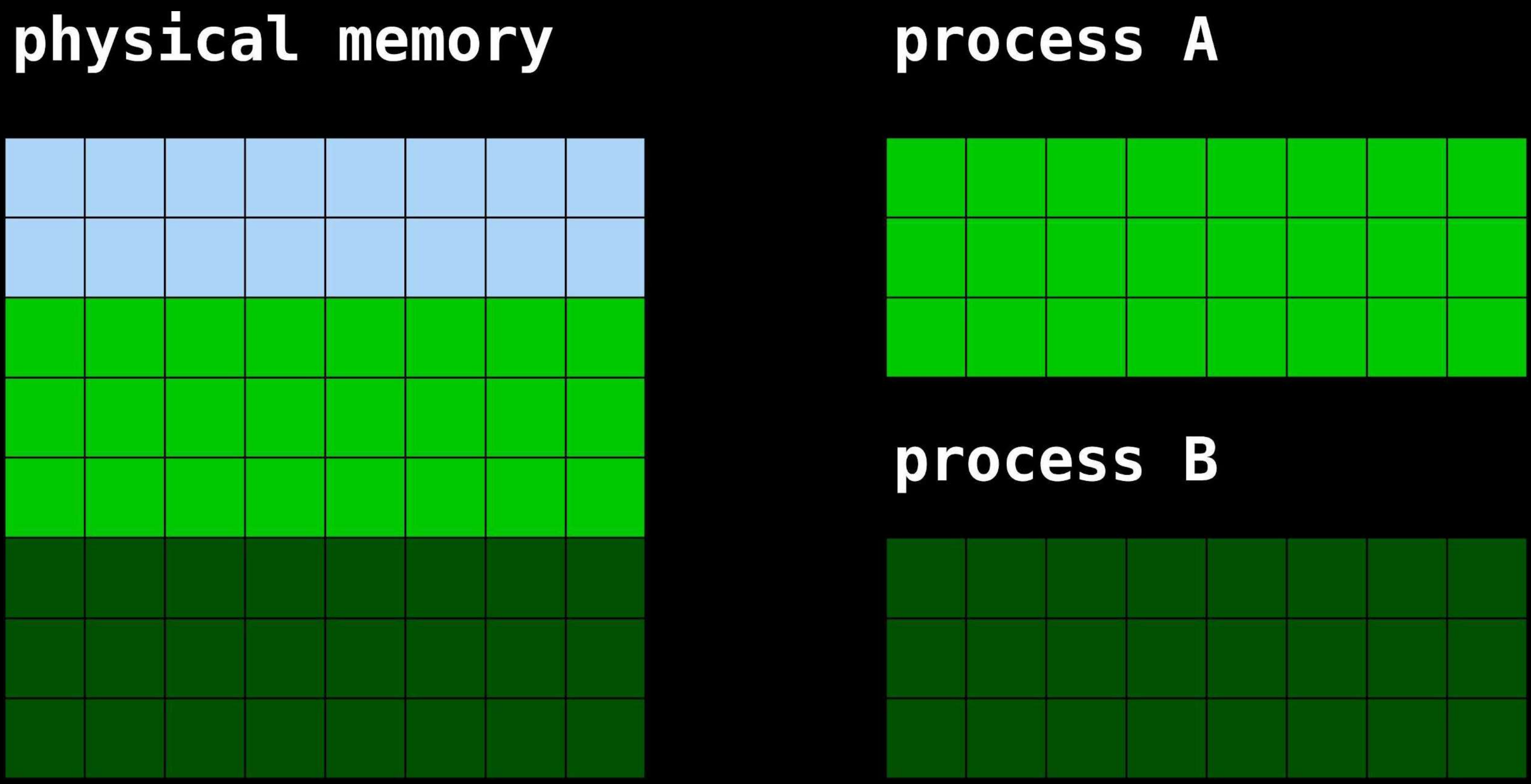
- Create a reference to our fake object



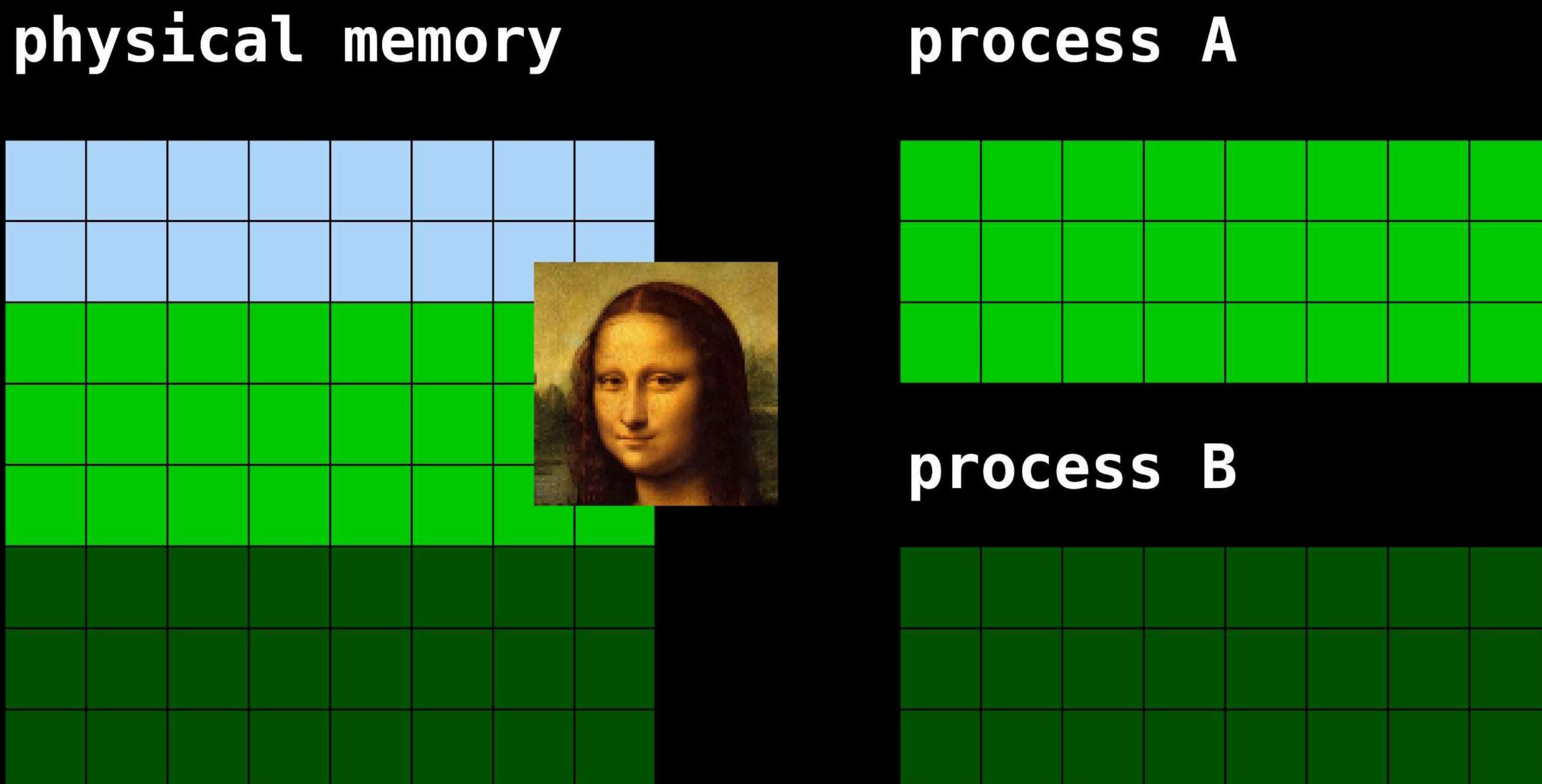
Memory Deduplication

- A strategy to reduce physical memory usage
- Removes duplication in physical memory
- Common in virtualization environments
- Enabled by **default on Windows**
  - Windows 8.1
  - Windows 10

Memory Deduplication: Mechanics

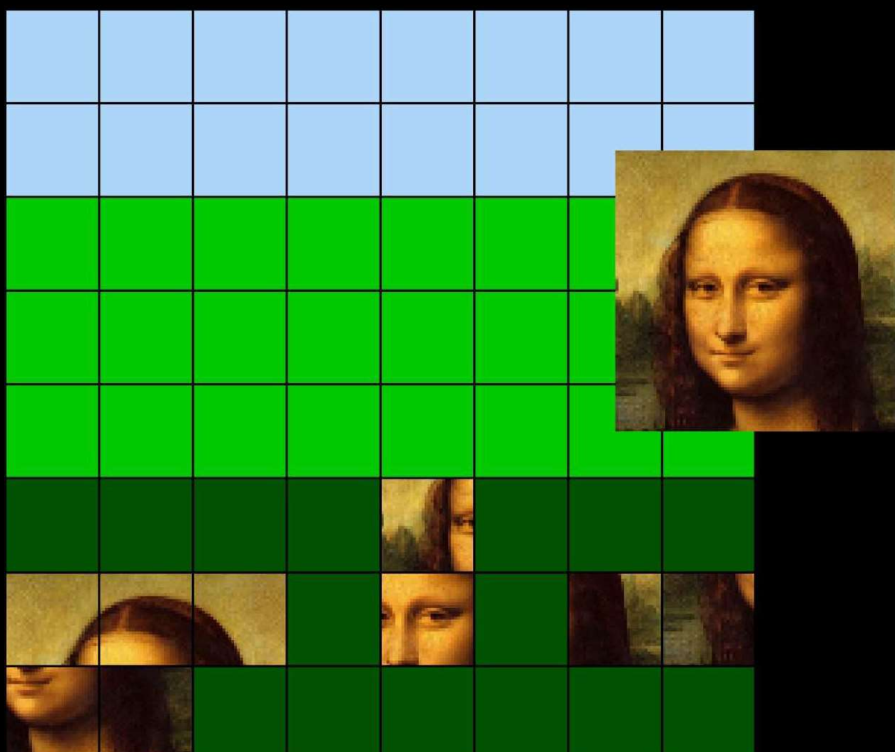


Memory Deduplication: Mechanics

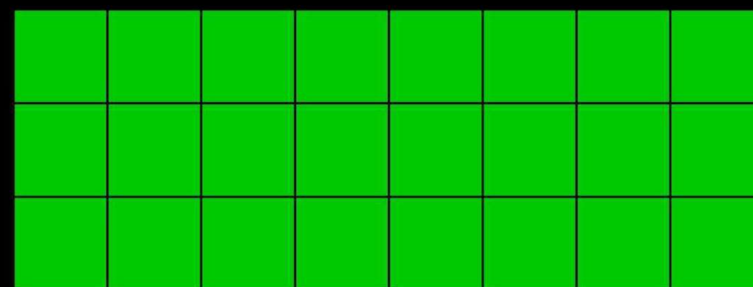


Memory Deduplication: Mechanics

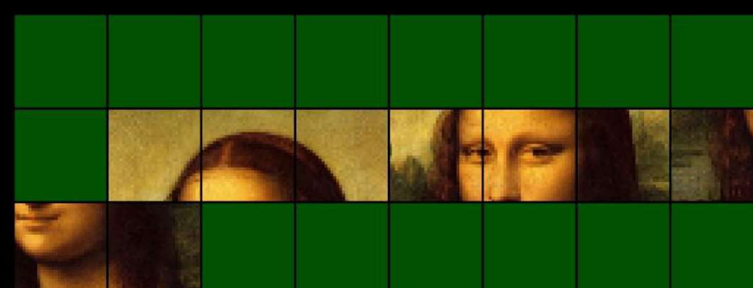
physical memory



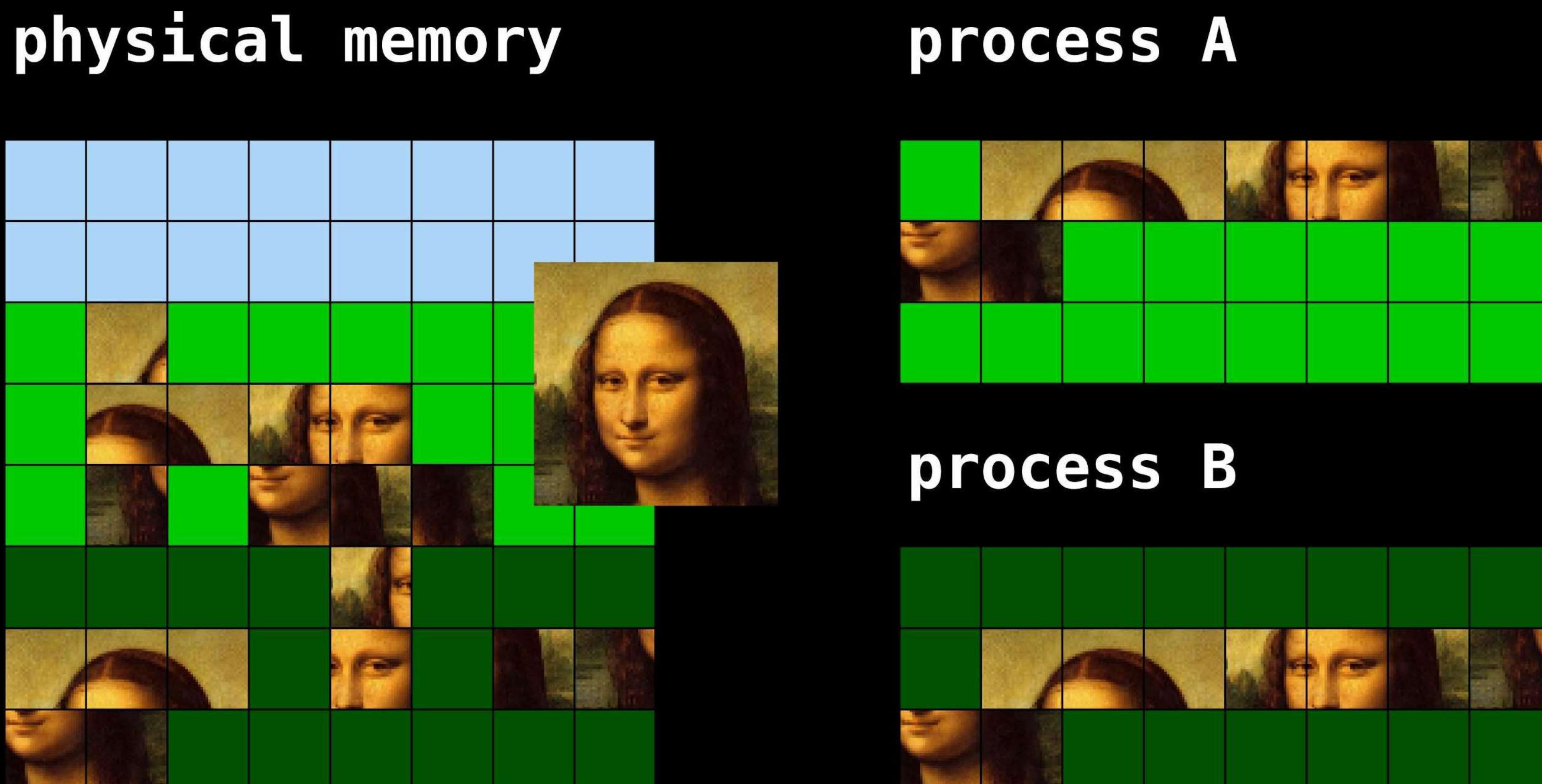
process A



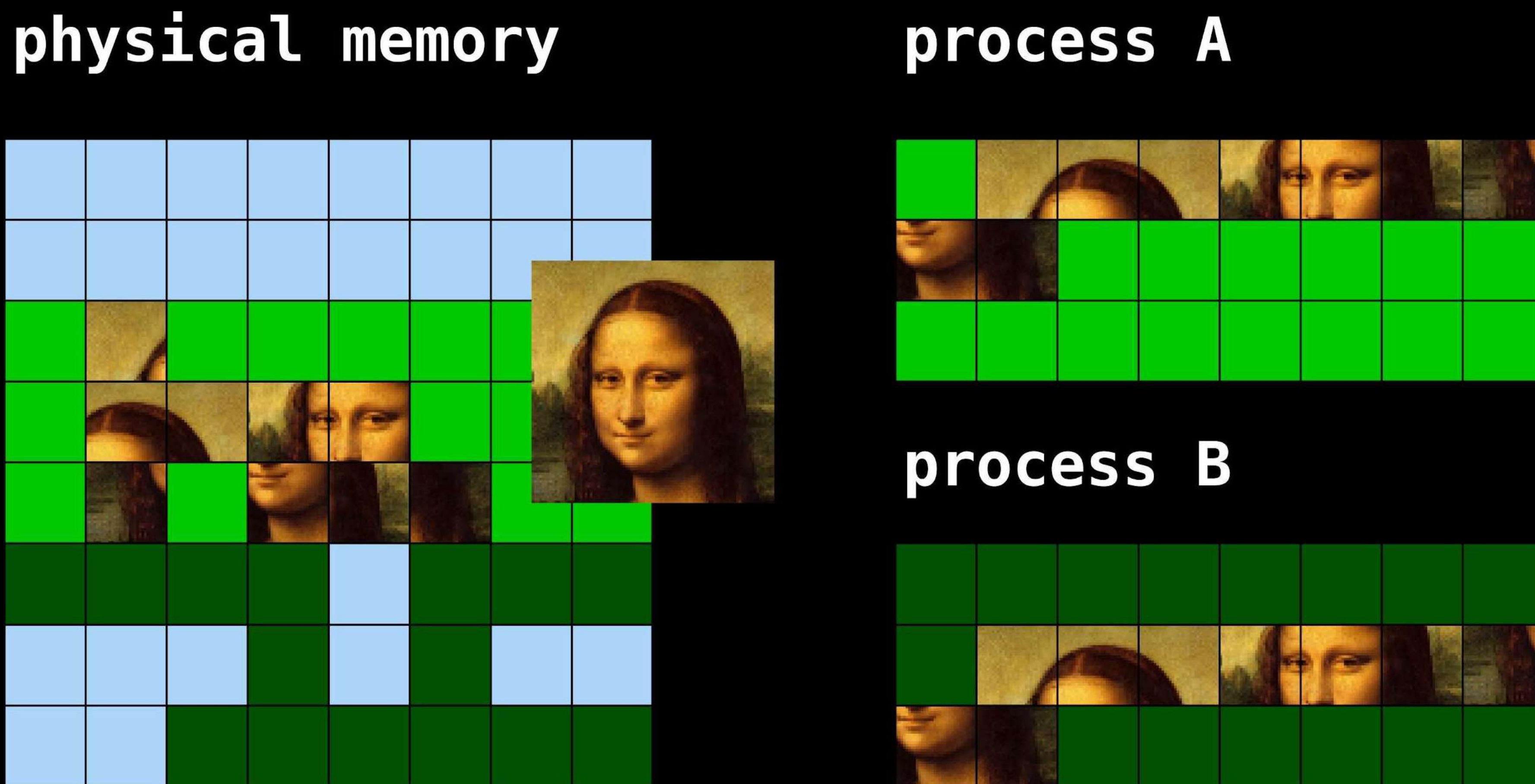
process B



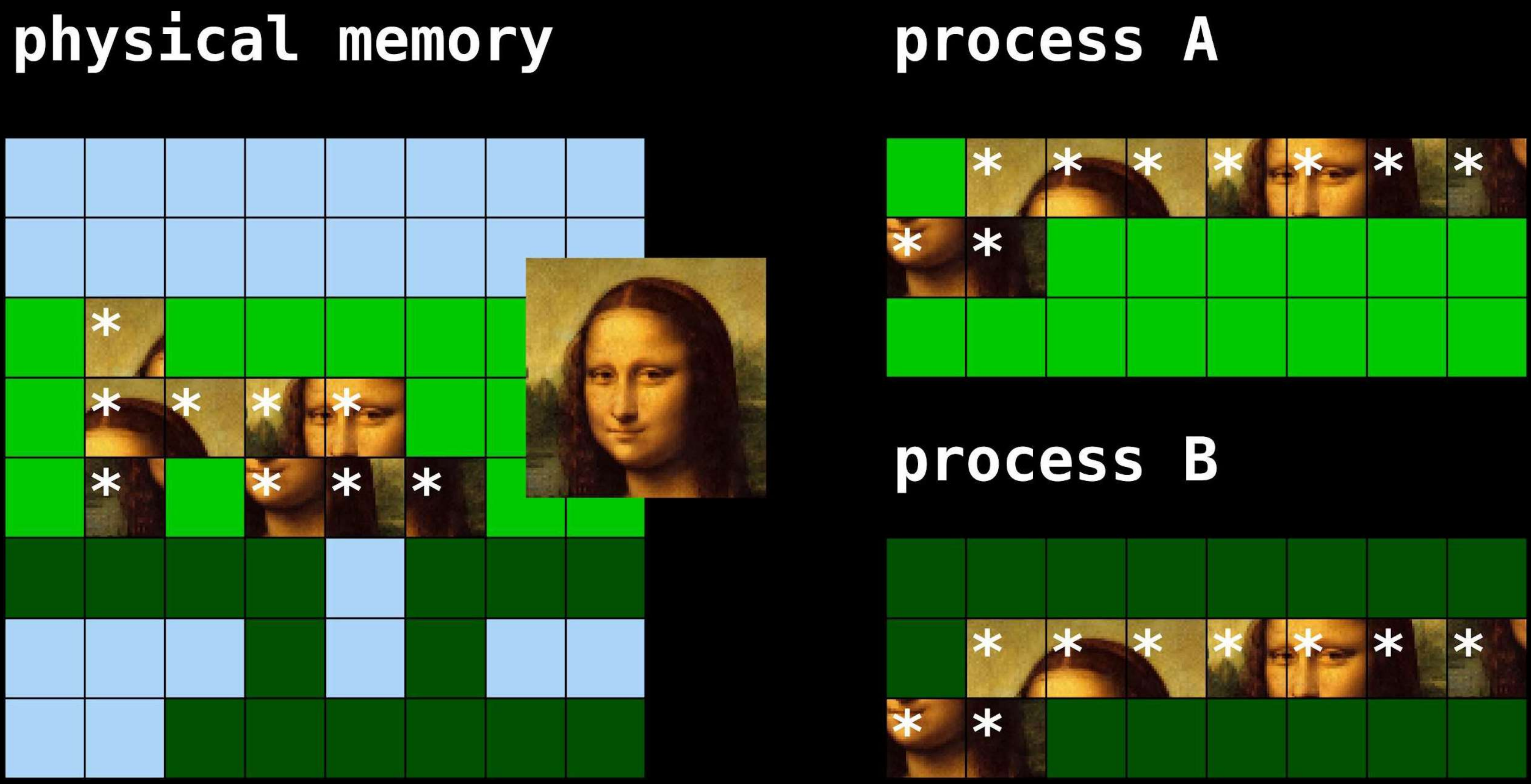
Memory Deduplication: Mechanics



Memory Deduplication: Mechanics



Memory Deduplication: Mechanics



Memory Deduplication: The Problem

- Deduplicated memory is origin-agnostic
- Merges pages across security boundaries
- Attackers can use this as a **side channel**!

Memory Deduplication: Timing Side Channel

normal write

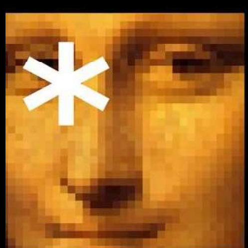
Memory Deduplication: Timing Side Channel

normal write

Memory Deduplication: Timing Side Channel

normal write

copy on write (due to deduplication)



Memory Deduplication: Timing Side Channel

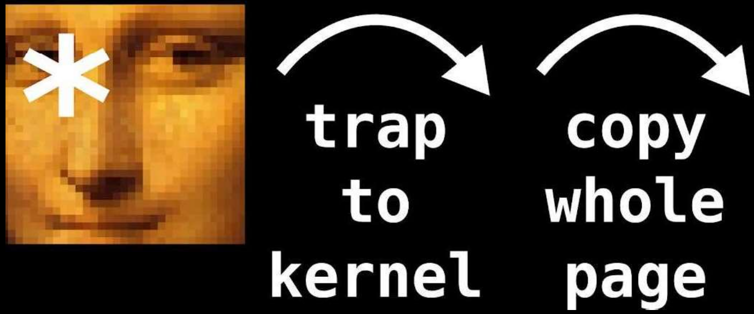
normal write

copy on write (due to deduplication)

Memory Deduplication: Timing Side Channel

normal write

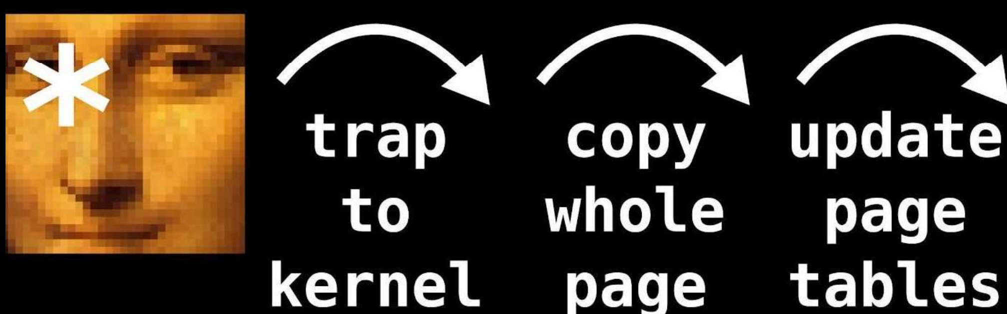
copy on write (due to deduplication)



Memory Deduplication: Timing Side Channel

normal write

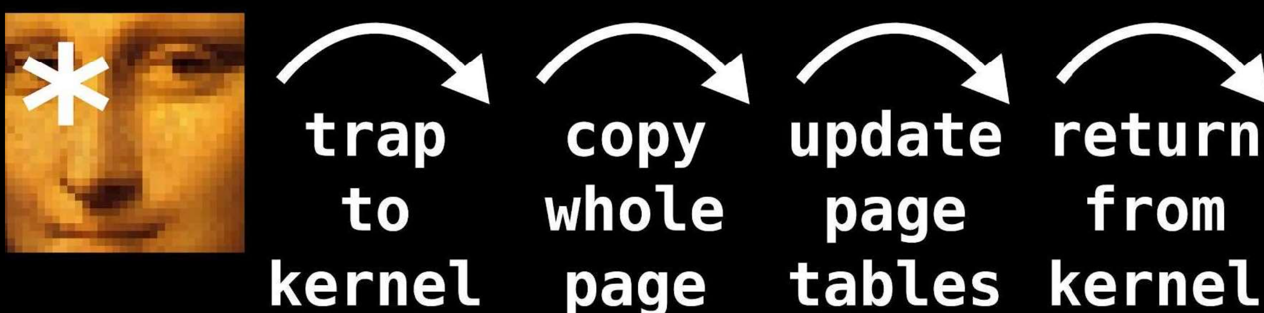
copy on write (due to deduplication)



Memory Deduplication: Timing Side Channel

normal write

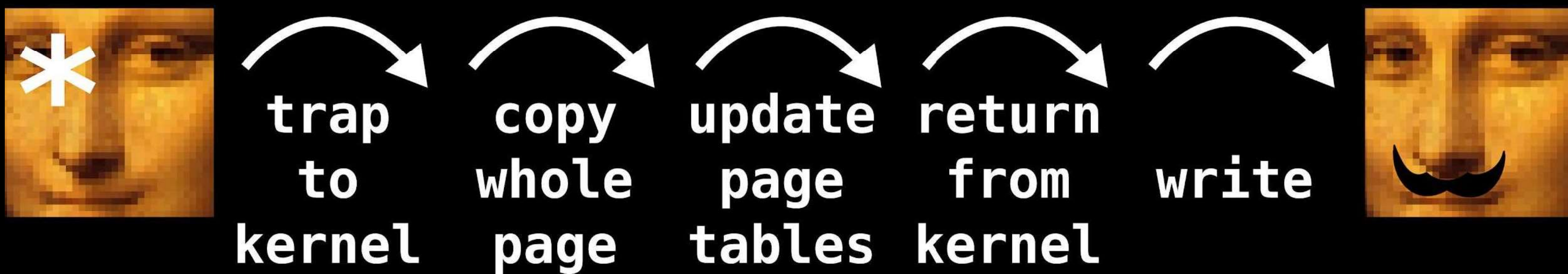
copy on write (due to deduplication)



Memory Deduplication: Timing Side Channel

normal write

copy on write (due to deduplication)



Memory Deduplication: Side-channel Leaks

Attacker can now leak **1 bit** of information (directly from JavaScript and system-wide)

"Does the victim
*process have **this***
page in memory?"

Memory Deduplication: Side-channel Leaks

- Very **coarse-grained.** Still interesting?
    - Is user logged into bank website X?

Memory Deduplication: Software Exploitation

For software exploitation, 1 bit won’t really cut it (e.g., need to leak 64-bit pointers for MS Edge)

*“Can we generalize this to leaking **arbitrary data** like randomized pointers or passwords?”*

Dedup Est Machina: Challenges

Challenge 1:

The secret we want to leak does not span an entire memory page

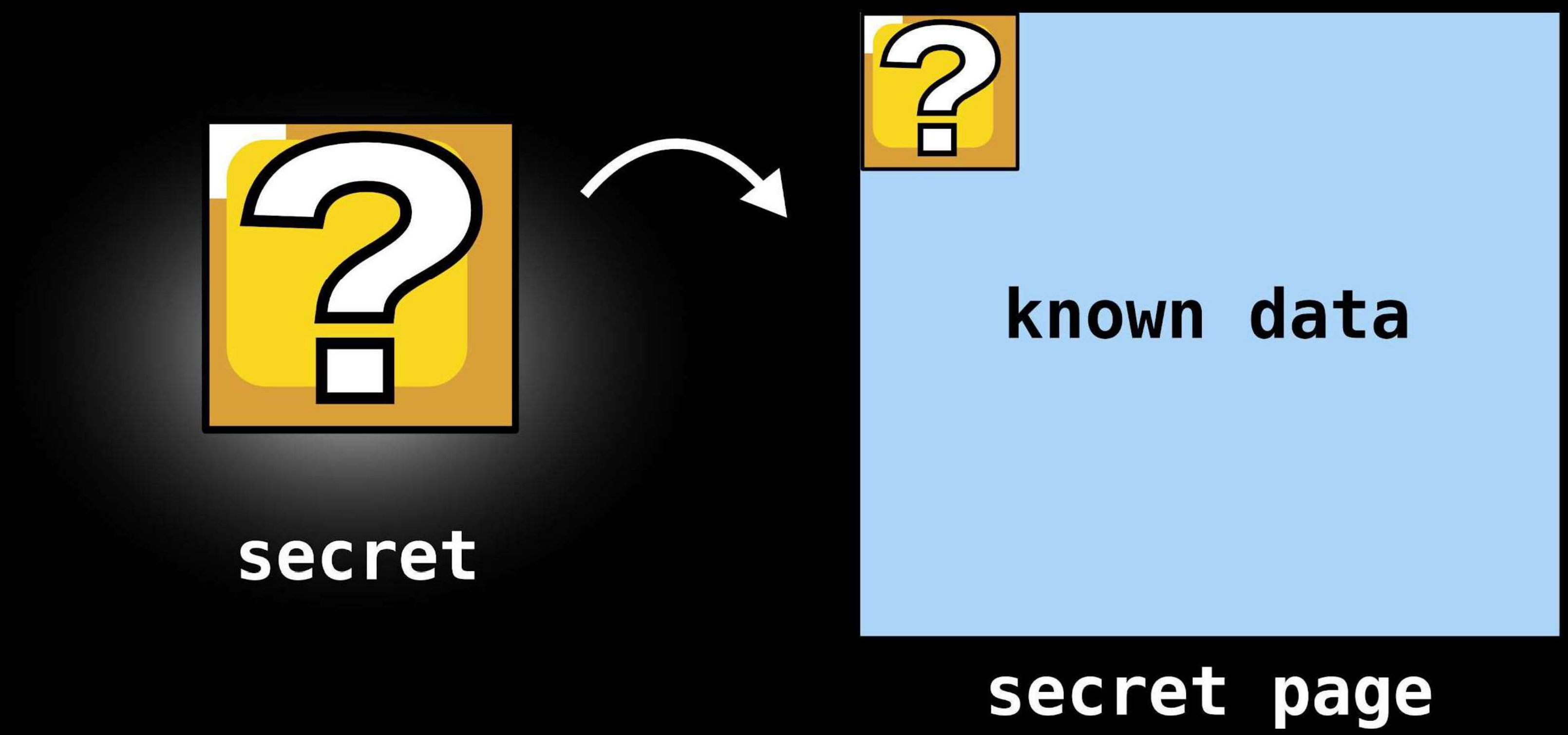
Dedup Est Machina: Challenges

Turning a secret into a page

secret

Dedup Est Machina: Challenges

Turning a secret into a page



Dedup Est Machina: Challenges

Challenge 2:

The secret to leak has too much entropy to leak it all at once

Dedup Est Machina: Challenges

Challenge 2:

The secret to leak has too much entropy to leak it all at once

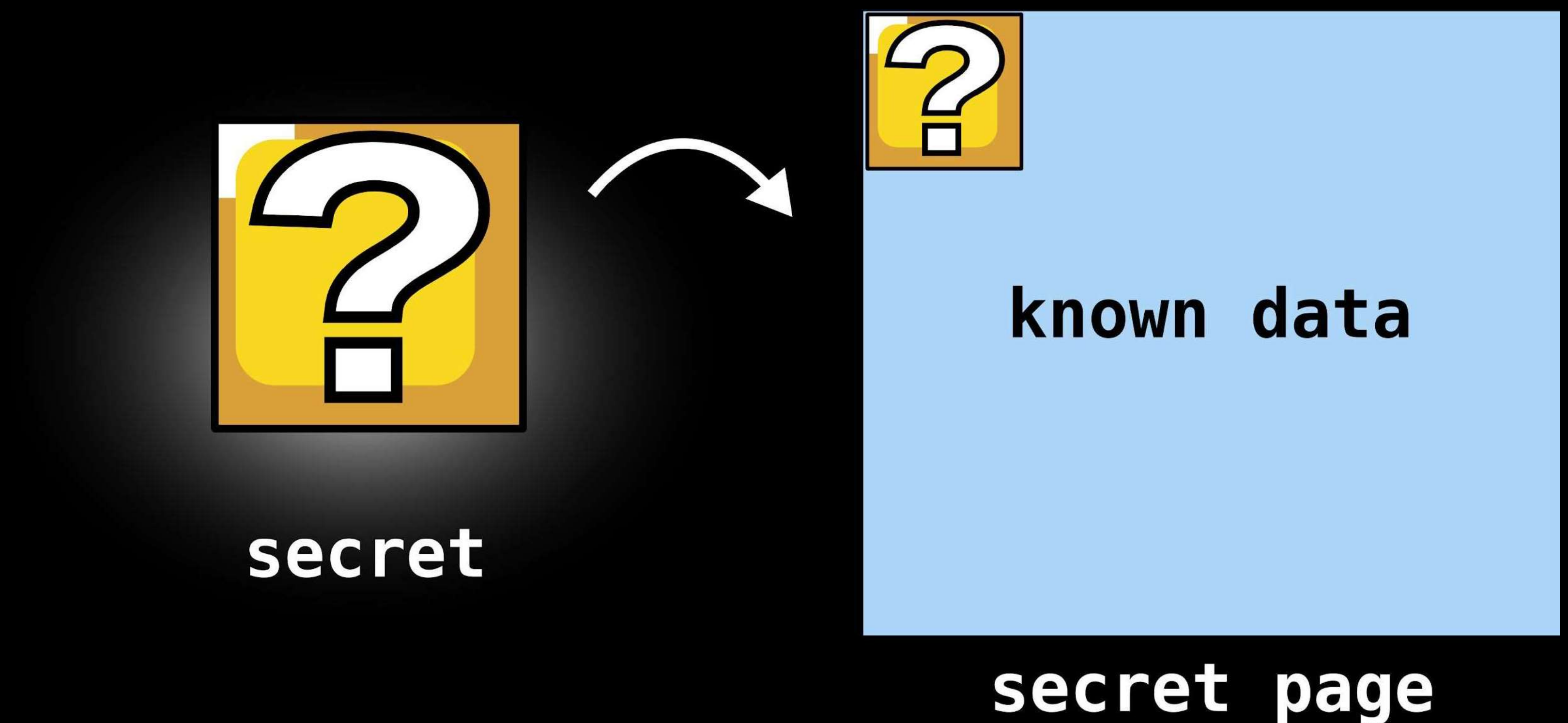
Primitive #1

Primitive #2

Primitive #3

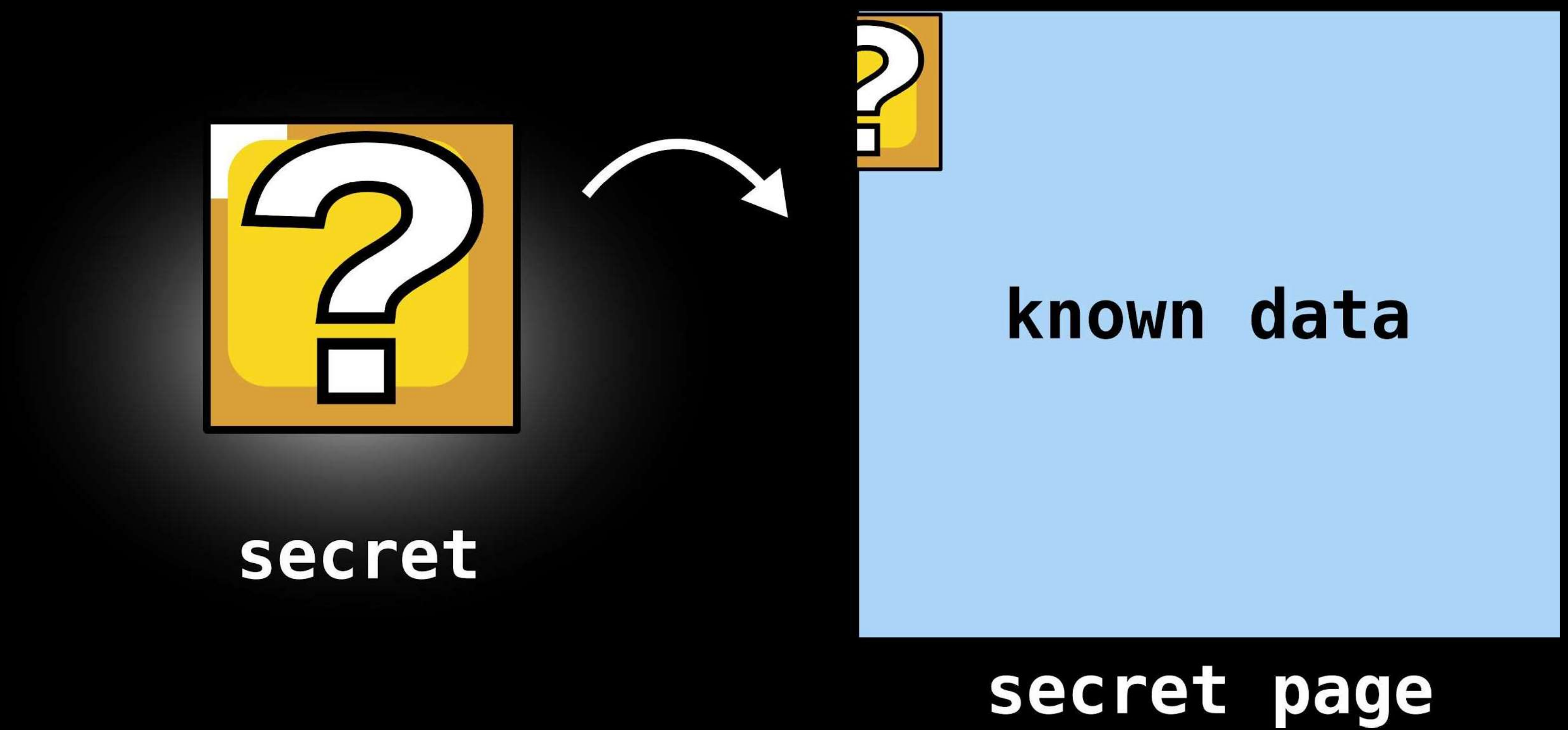
Dedup Est Machina: Primitives

Primitive #1: Alignment Probing



Dedup Est Machina: Primitives

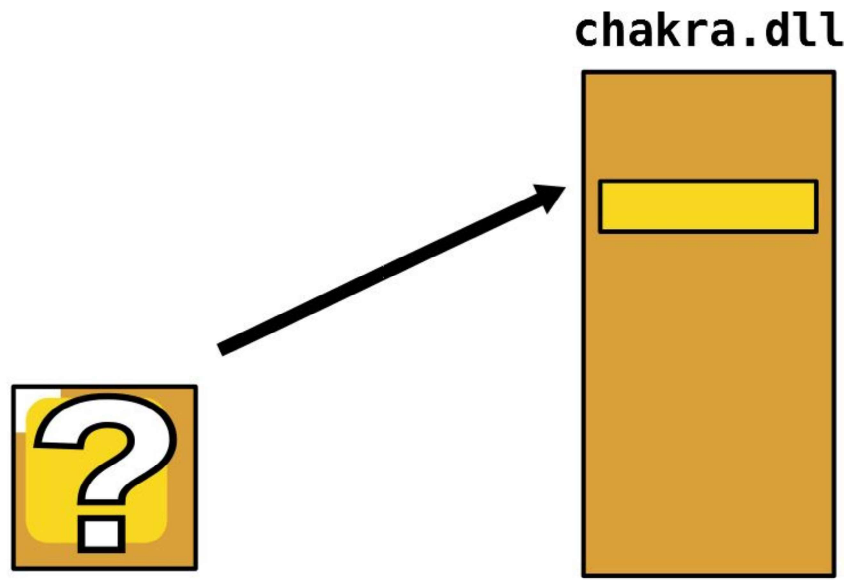
Primitive #1: Alignment Probing



Dedup Est Machina: Overview

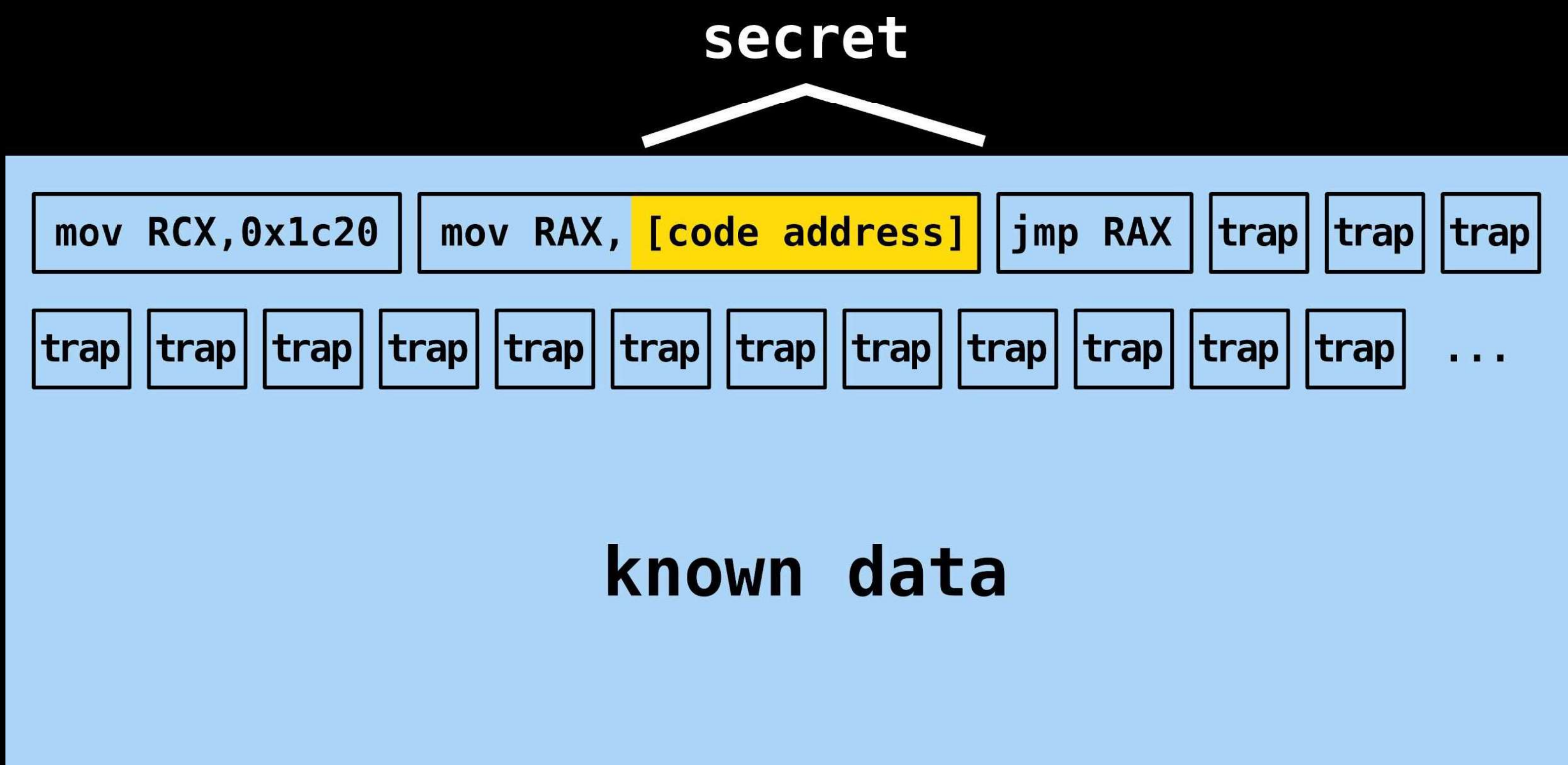
Memory deduplication

- Leak randomized heap and code pointers



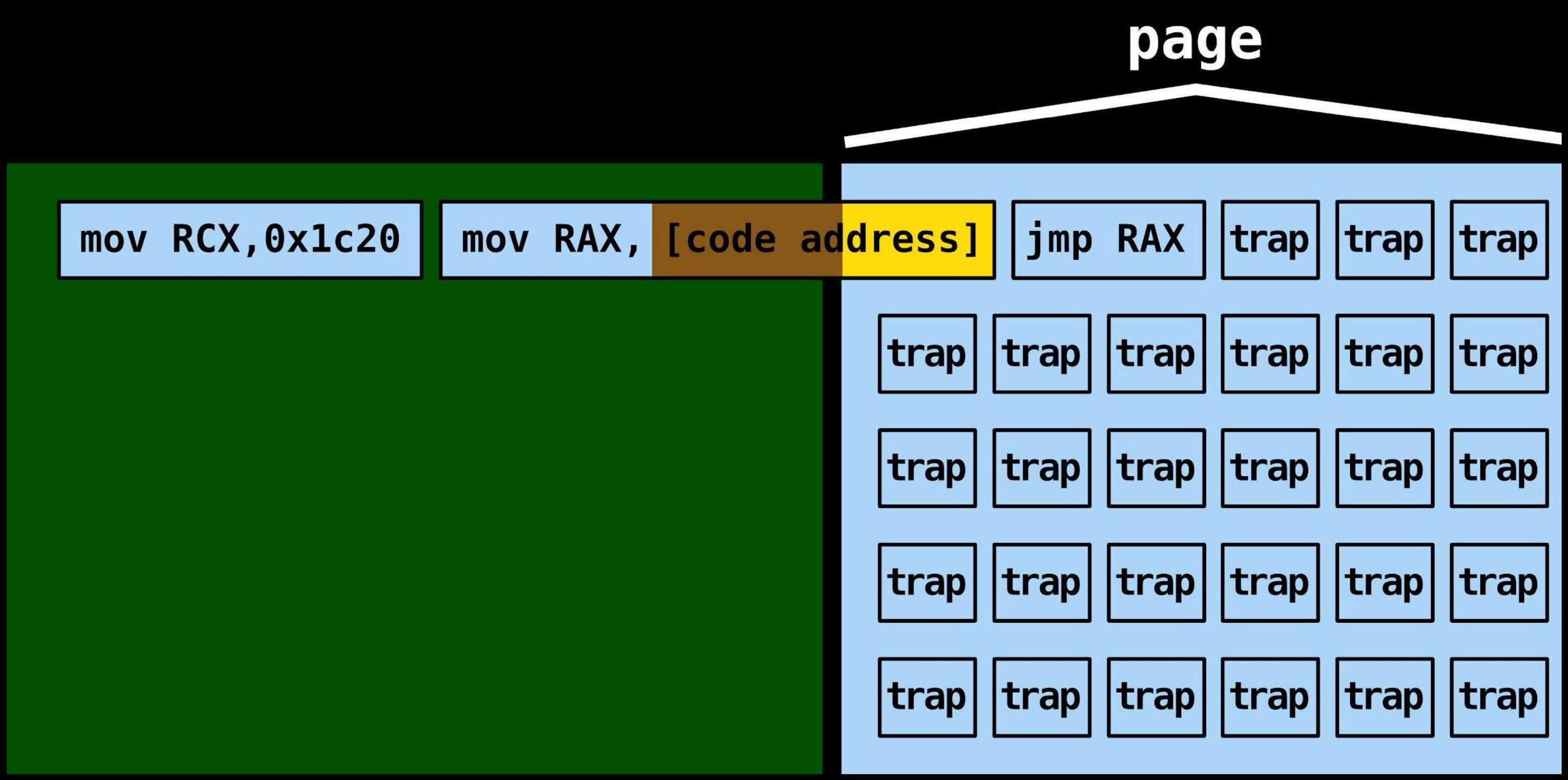
Dedup Est Machina: Leaking Code Pointer (#1)

JIT Function Epilogue in MS Edge



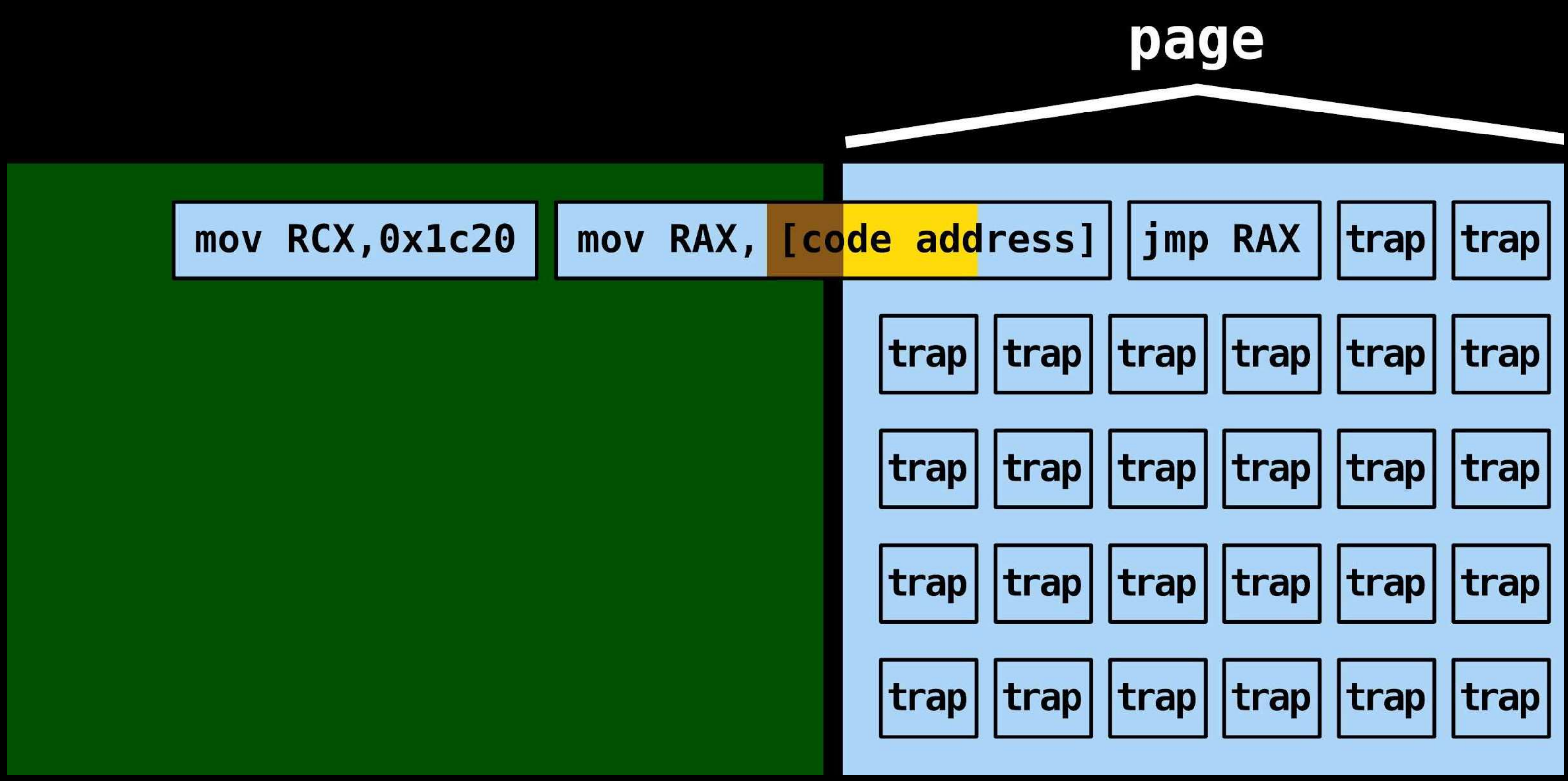
Dedup Est Machina:
Leaking Code Pointer (#1)

JIT Function Epilogue in MS Edge



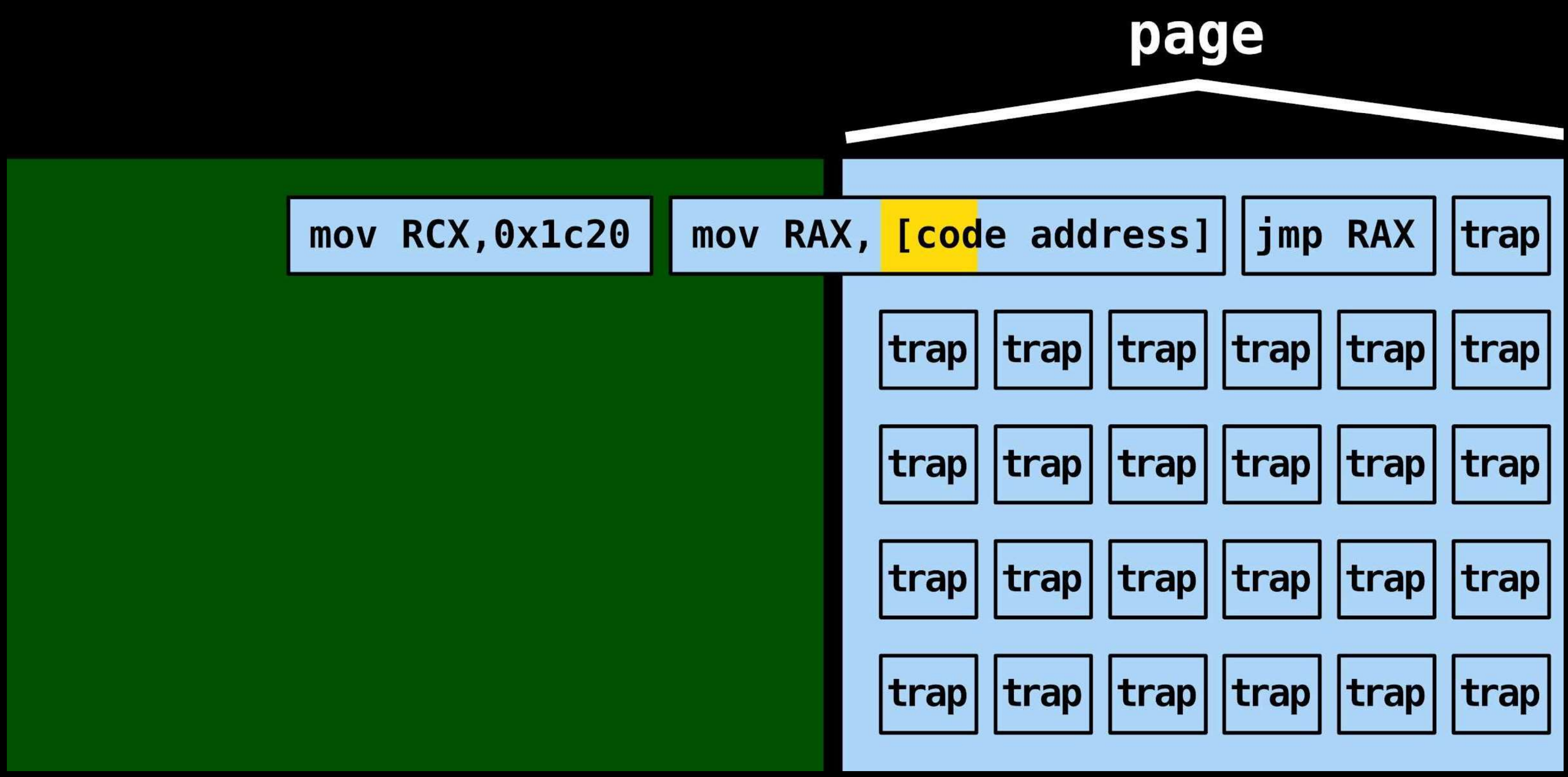
Dedup Est Machina:
Leaking Code Pointer (#1)

JIT Function Epilogue in MS Edge



Dedup Est Machina: Leaking Code Pointer (#1)

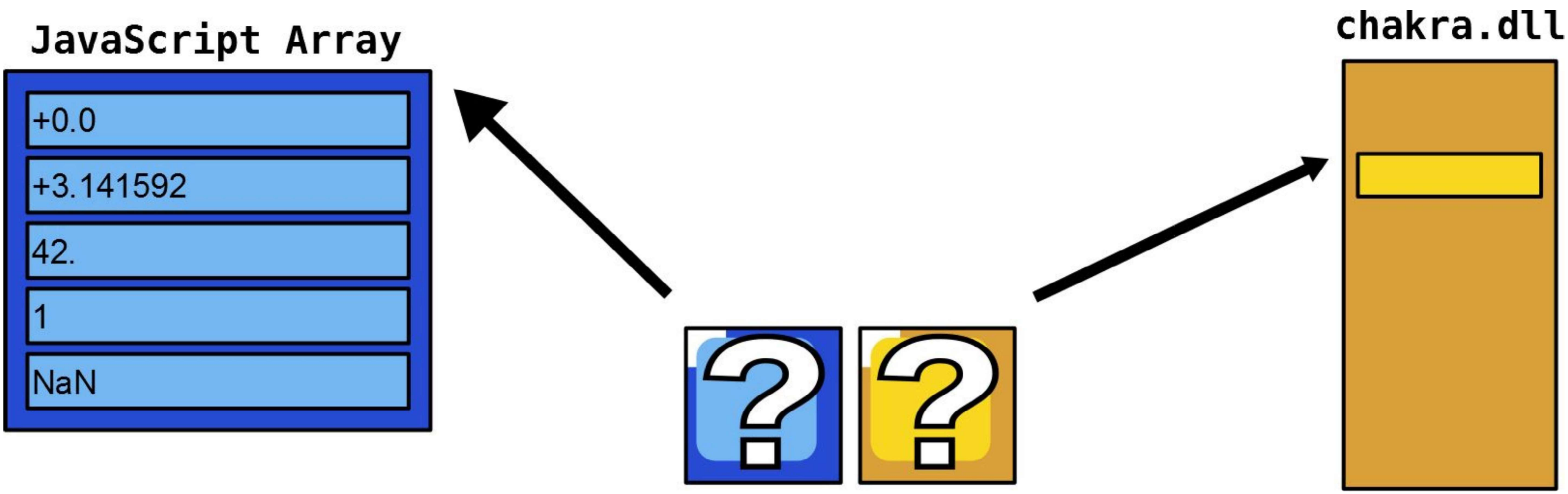
JIT Function Epilogue in MS Edge



Dedup Est Machina: Overview

Memory deduplication

- Leak randomized heap and code pointers



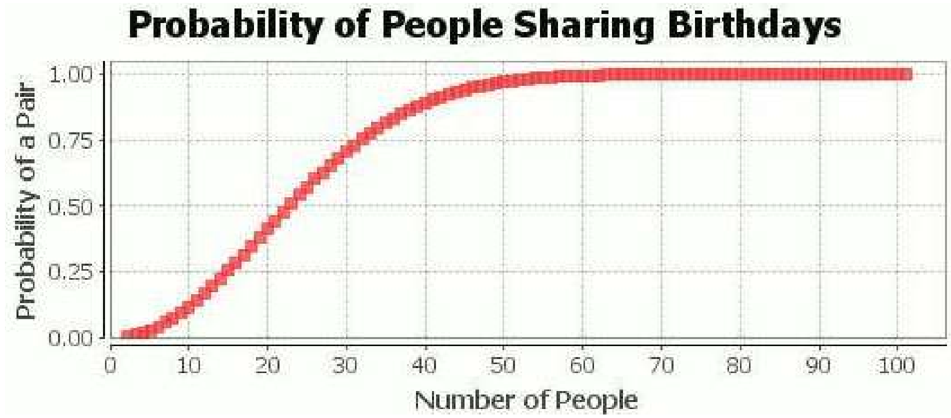
Dedup Est Machina: Leaking Heap Pointer

- Heap pointers are word aligned
  - Alignment probing won't cut it
- Time for the *Birthday HeapSpray* primitive!

"How do we leak a heap pointer
if we can only leak the
*secret **all at once**?"*

Dedup Est Machina: Birthday Paradox

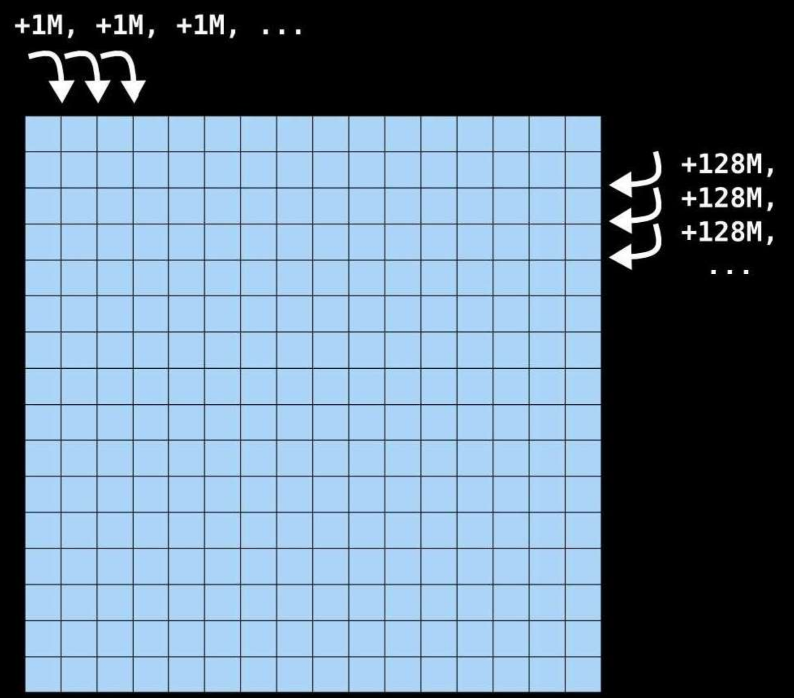
- Only 23 people for a 50% same-birthday chance



- You compare everyone with everyone else
  → **Any match suffices!**

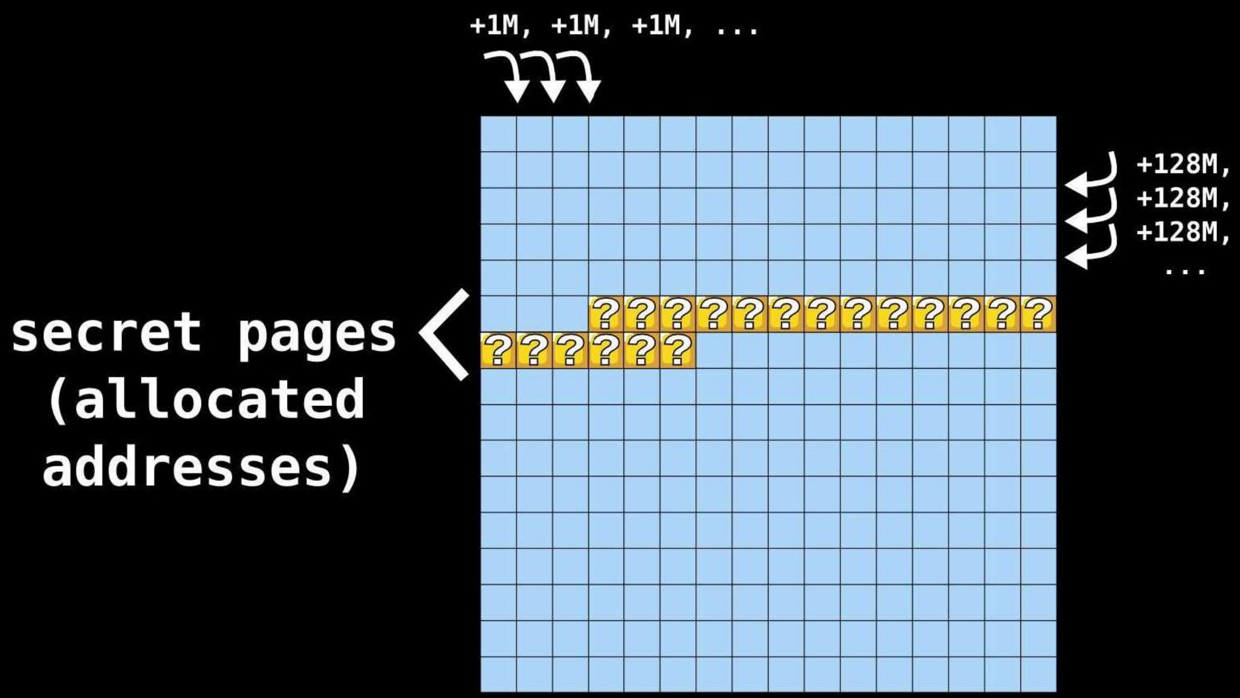
Dedup Est Machina: Leaking Heap Pointer (#3)

Birthday Heapspray



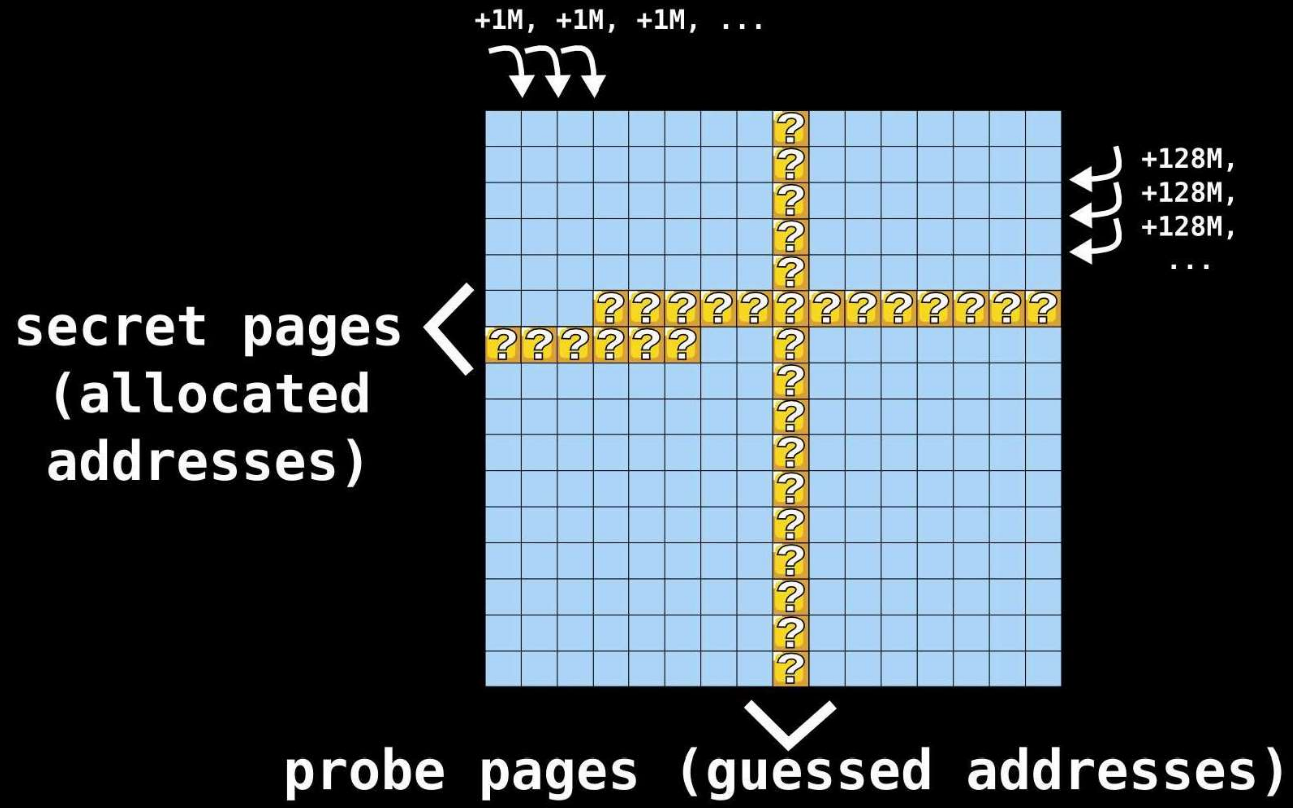
Dedup Est Machina: Leaking Heap Pointer (#3)

Birthday Heapspray



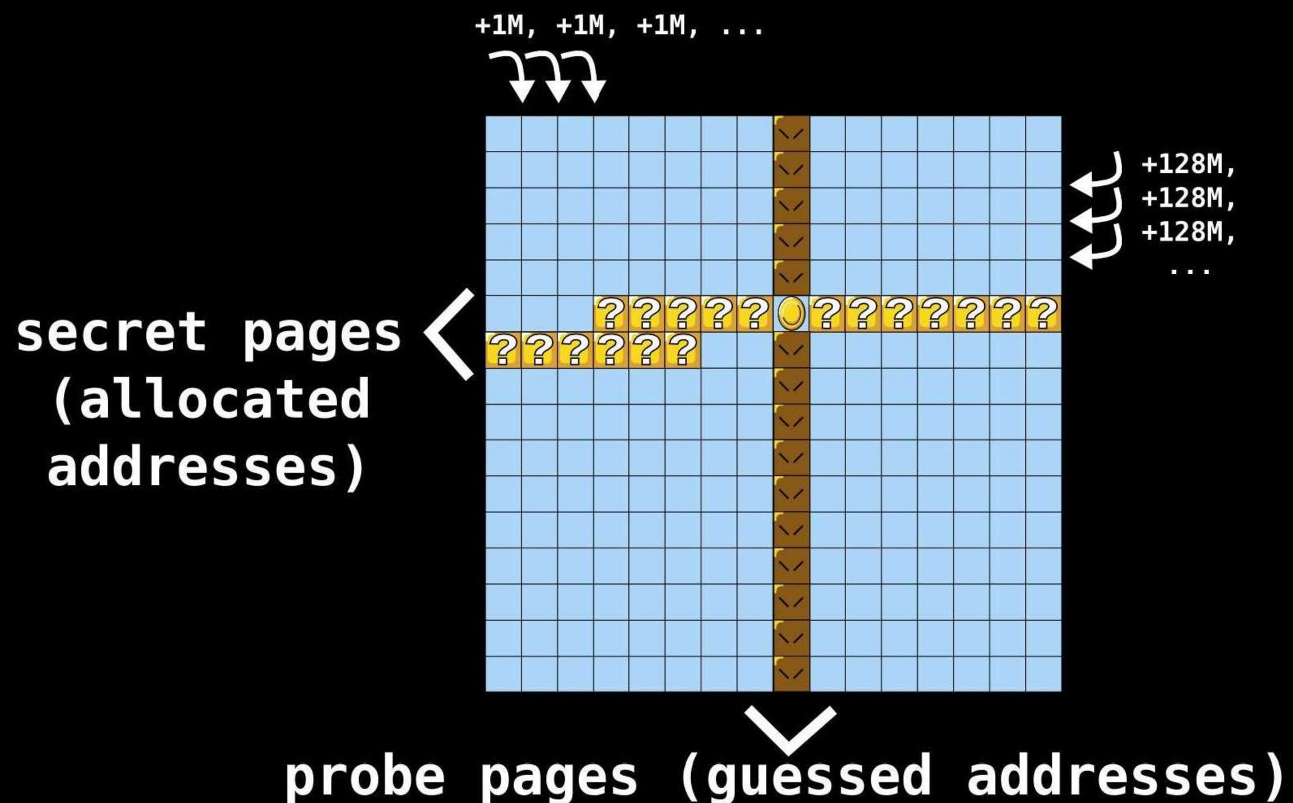
Dedup Est Machina: Leaking Heap Pointer (#3)

Birthday Heapspray



Dedup Est Machina: Leaking Heap Pointer (#3)

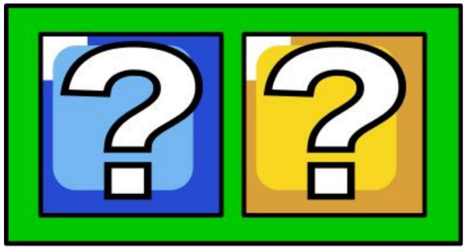
Birthday Heapspray



Dedup Est Machina: Overview

Memory deduplication

- Leak randomized heap and code pointers
- Create a fake JavaScript object

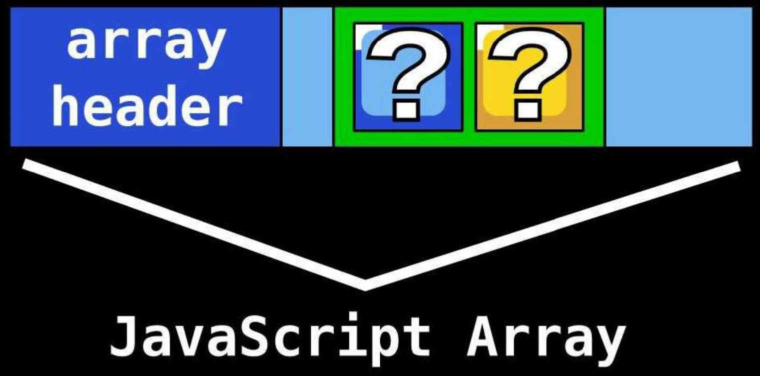


Dedup Est Machina: Creating a Fake Object

Fake JavaScript Uint8Array

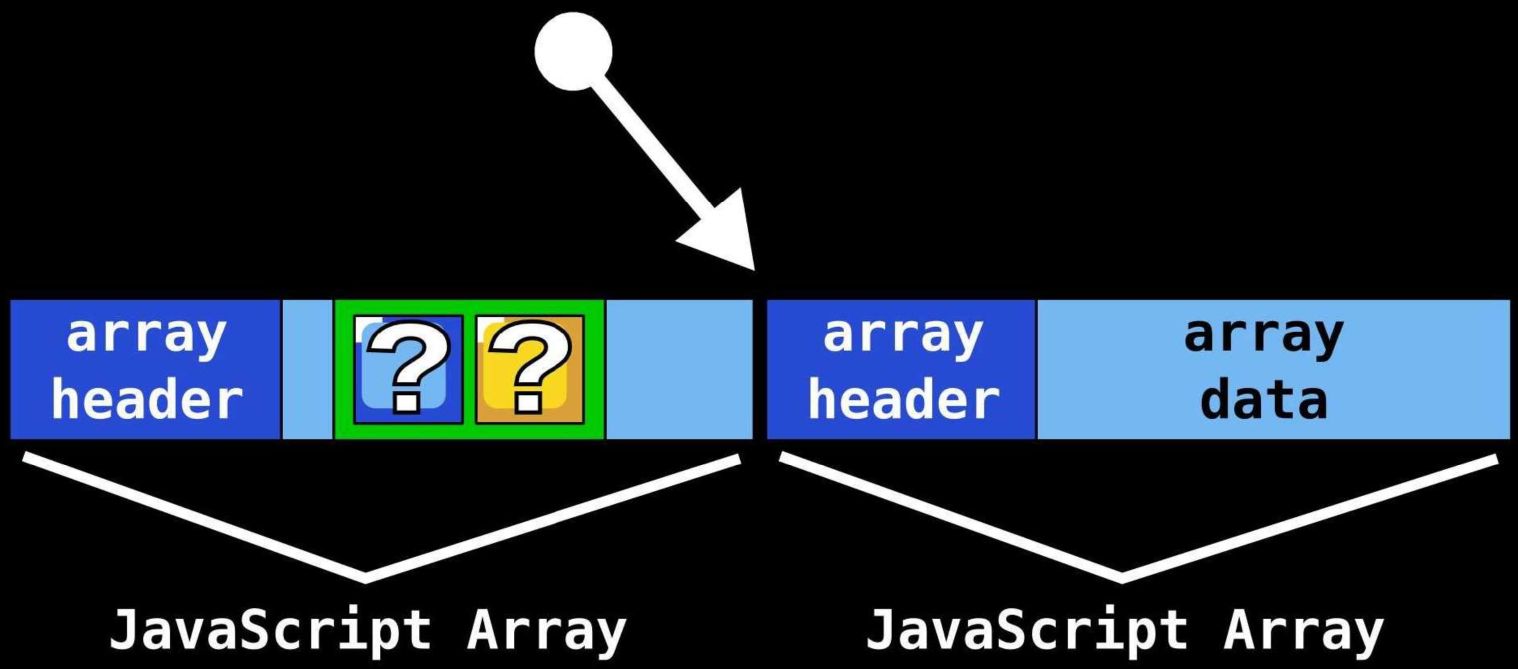
Dedup Est Machina: Creating a Fake Object

Fake JavaScript Uint8Array



Dedup Est Machina: Creating a Fake Object

Fake JavaScript Uint8Array



Dedup Est Machina: Overview

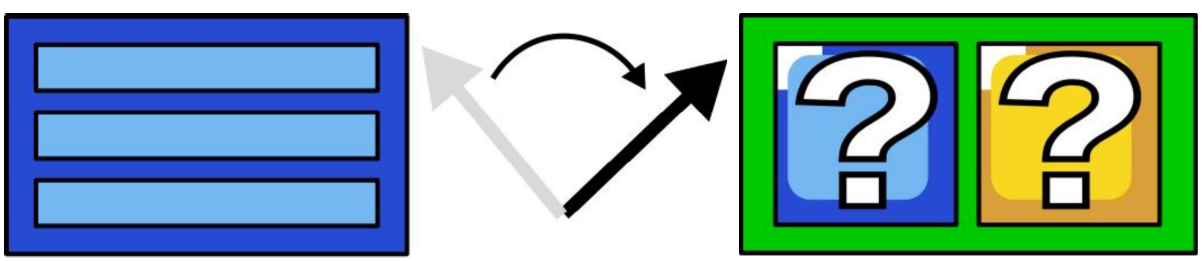
Memory deduplication

- Leak randomized heap and code pointers
- Create a fake JavaScript object

+

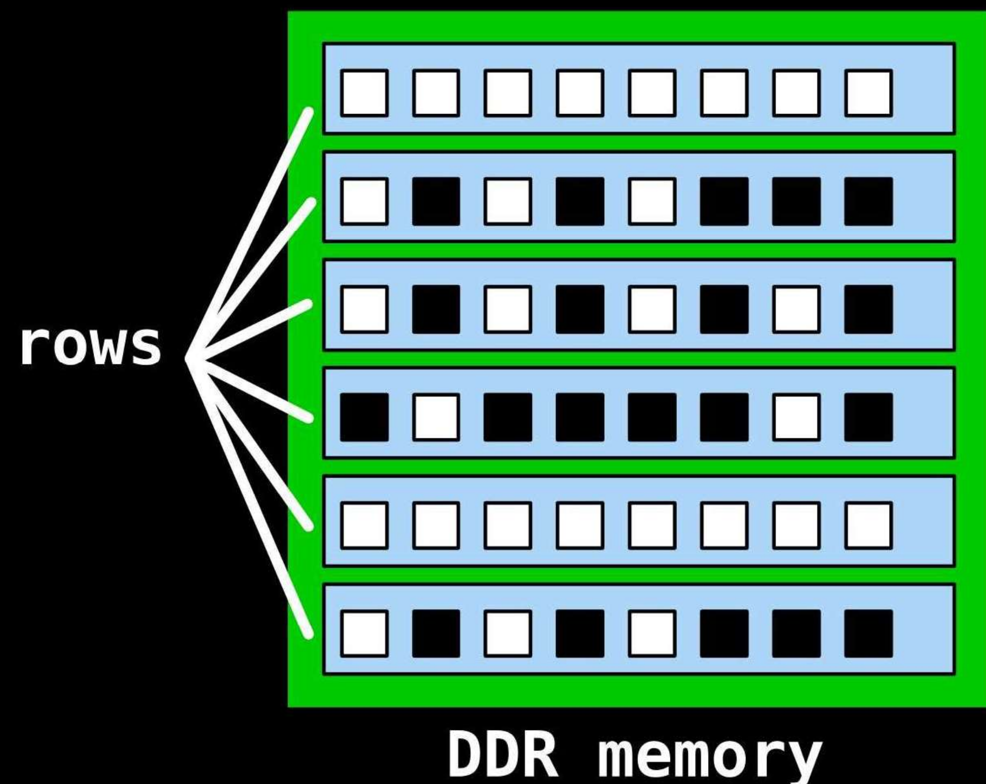
Rowhammer

- Create a reference to our fake object



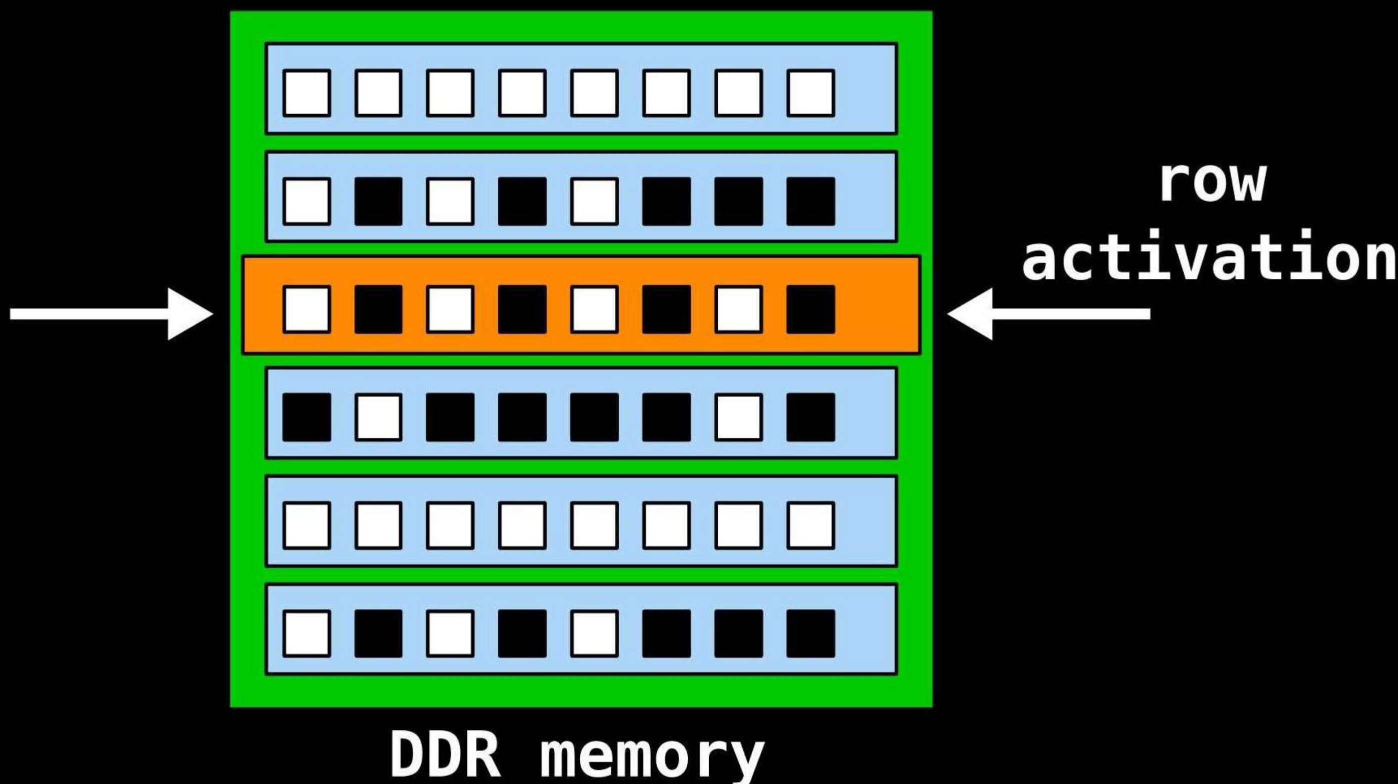
Dedup Est Machina: Referencing the Fake Object

Rowhammer



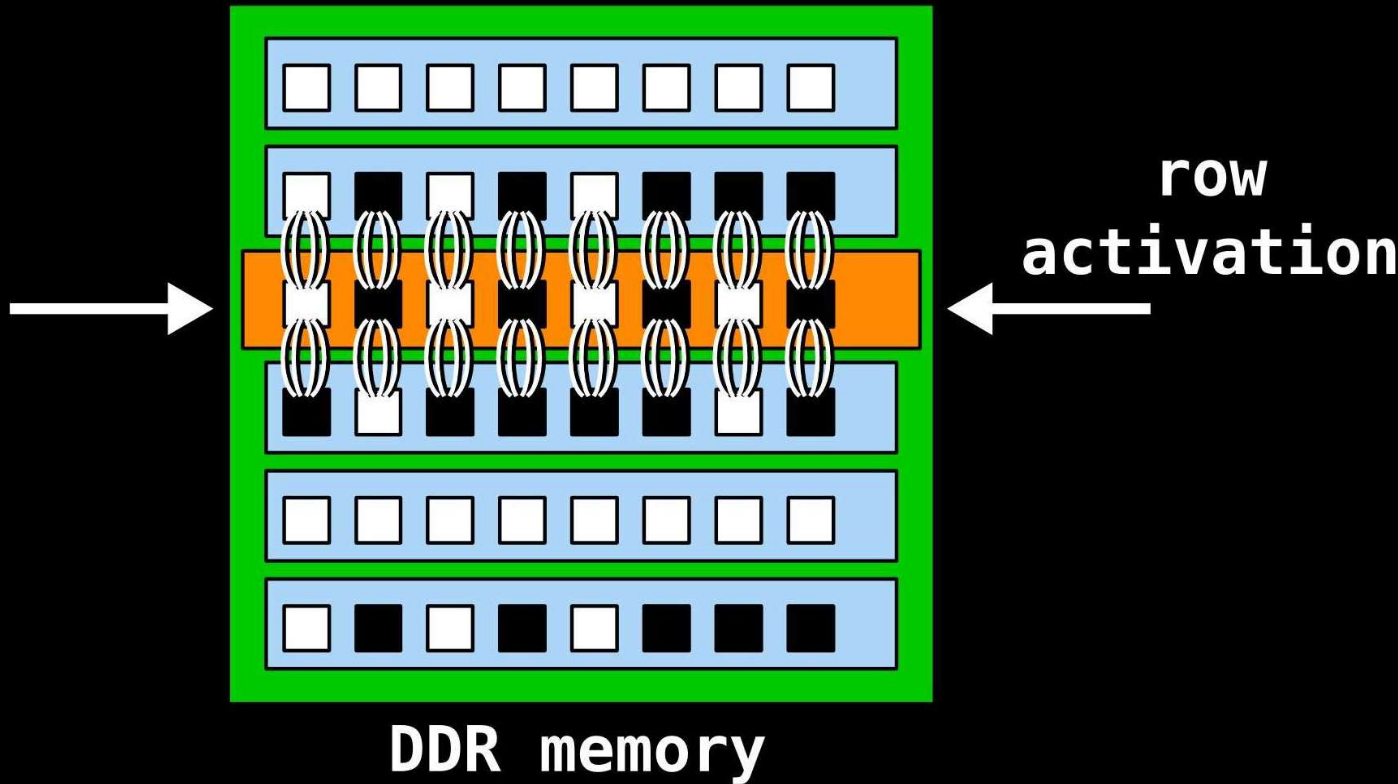
Dedup Est Machina: Referencing the Fake Object

Rowhammer



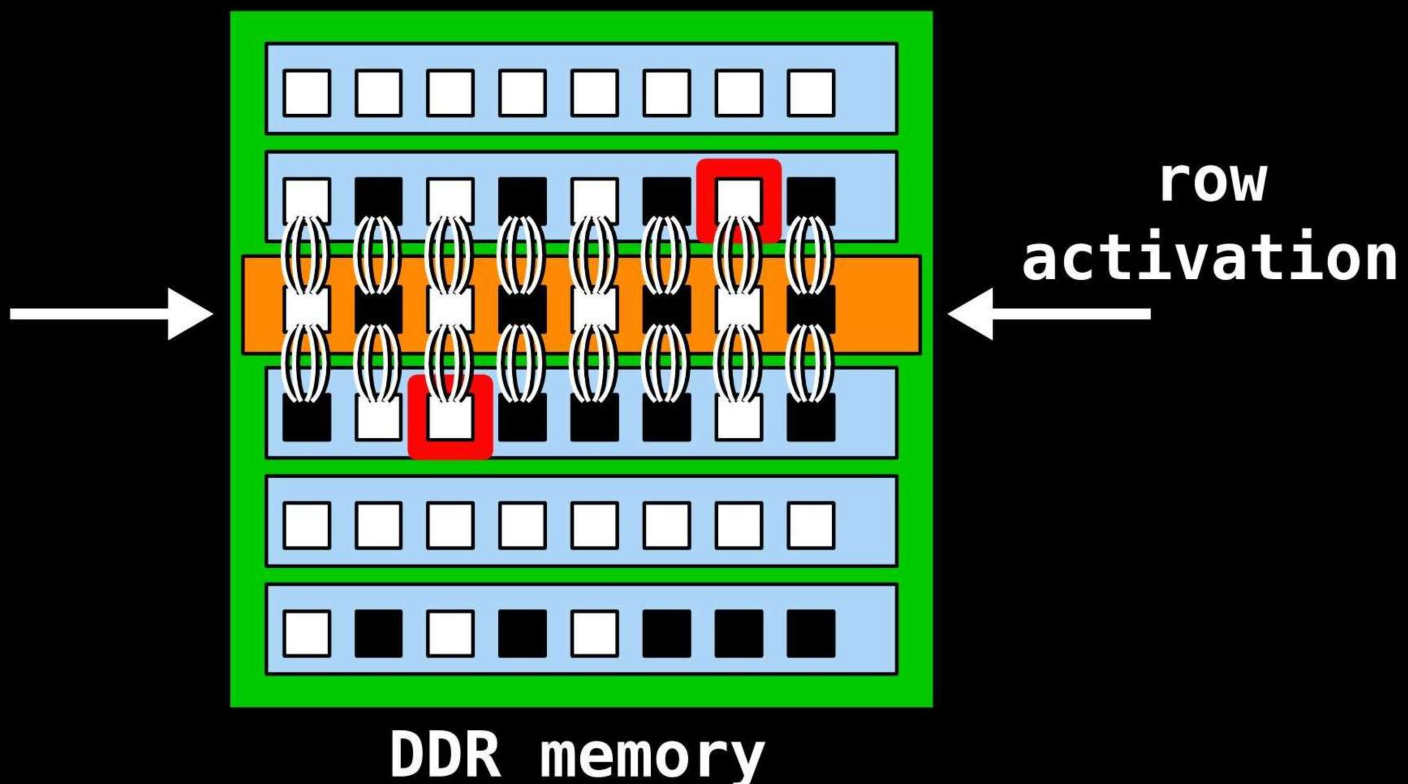
Dedup Est Machina: Referencing the Fake Object

Rowhammer



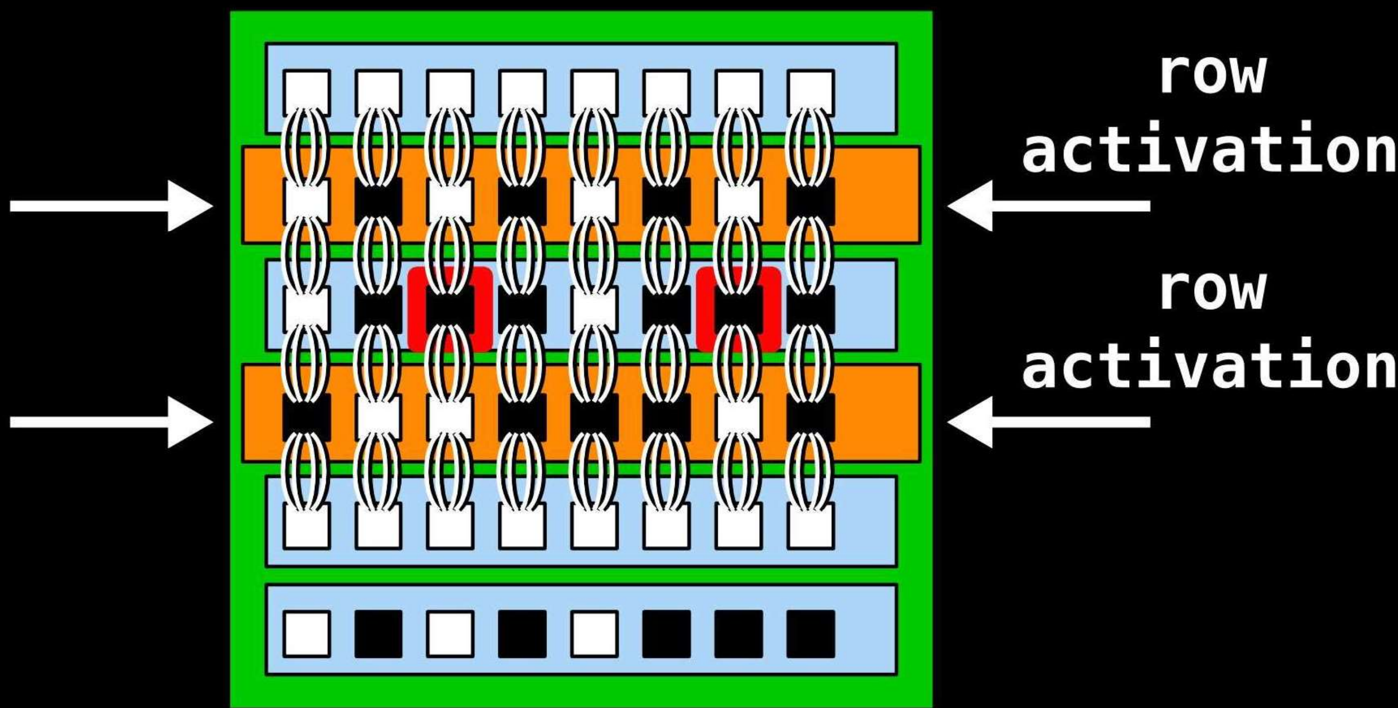
Dedup Est Machina: Referencing the Fake Object

Rowhammer



Dedup Est Machina: Referencing the Fake Object

Double-sided Rowhammer

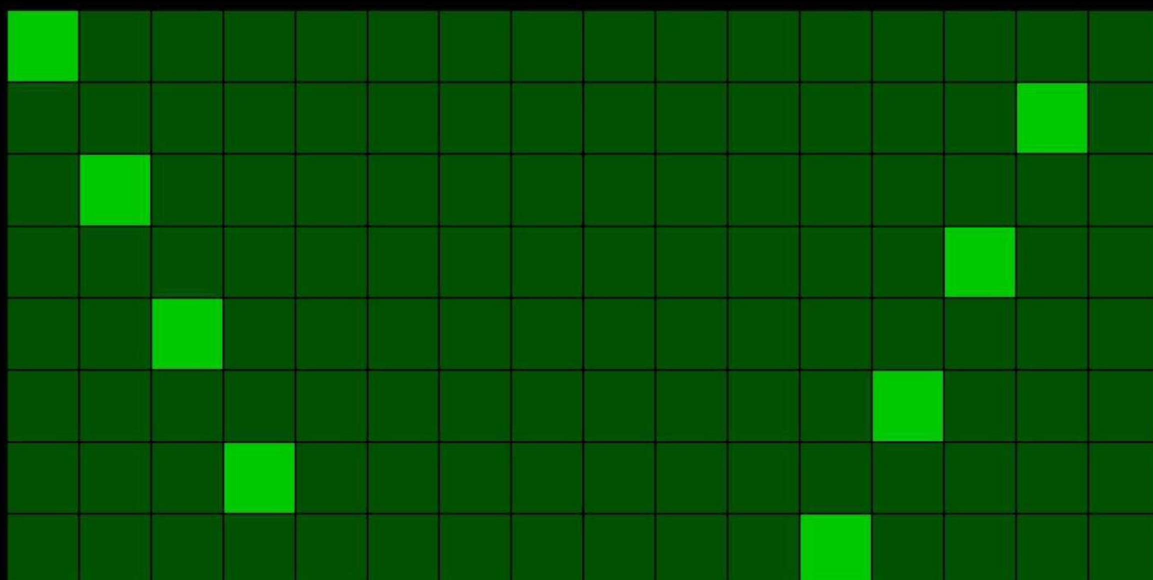


DDR memory

Dedup Est Machina: Referencing the Fake Object

Double-sided Rowhammer

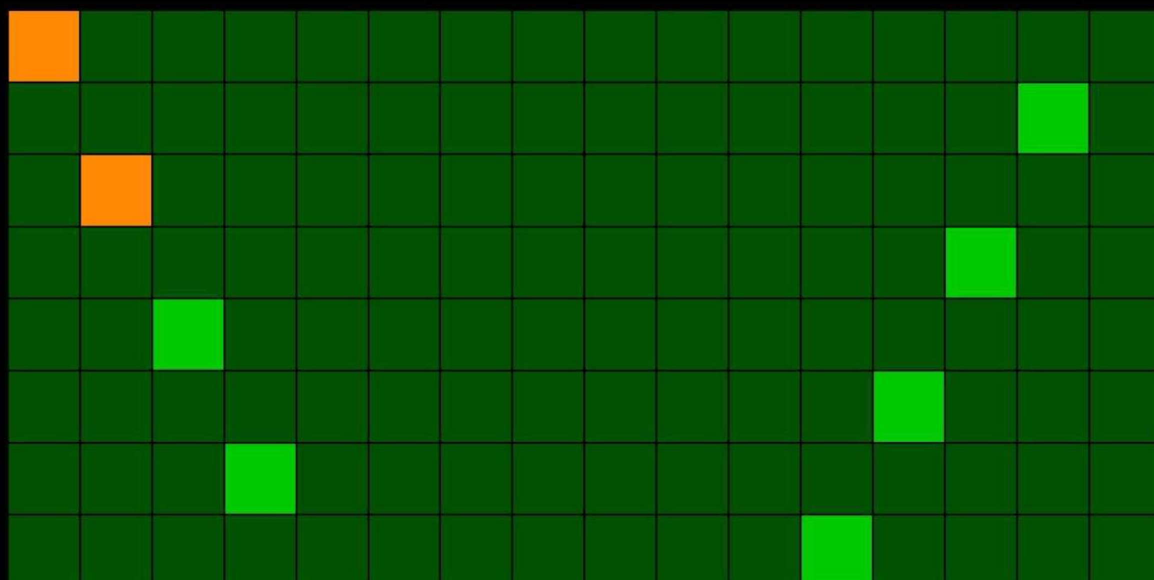
physical memory



Dedup Est Machina: Referencing the Fake Object

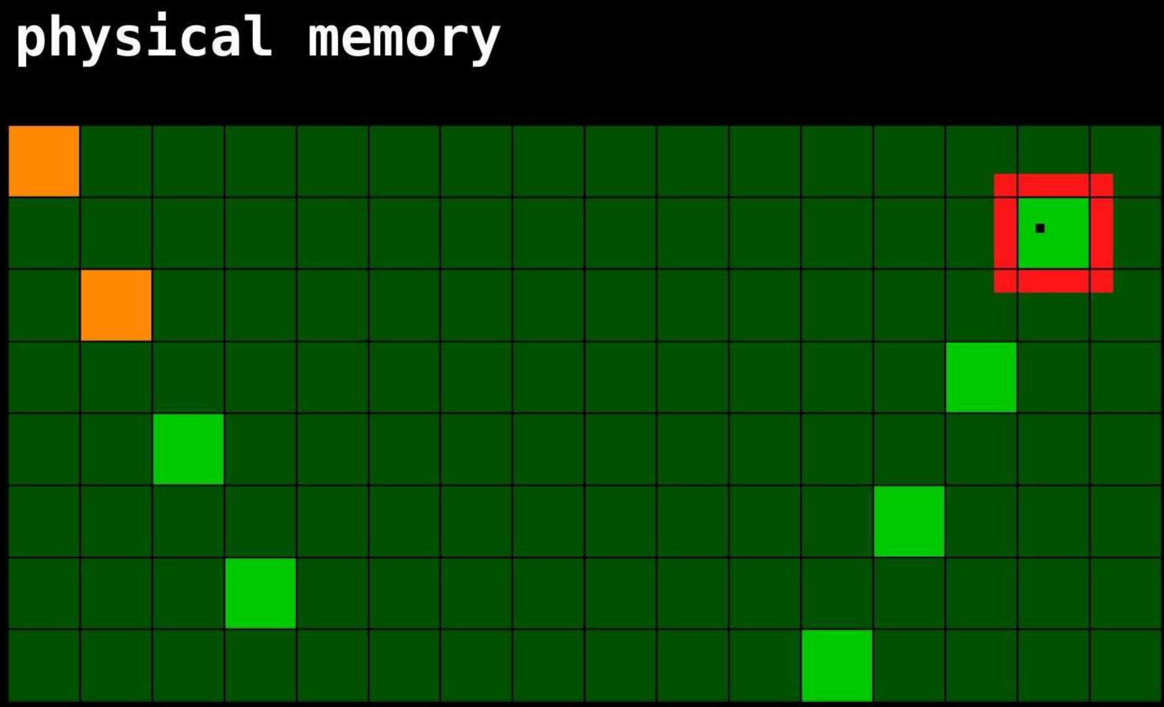
Double-sided Rowhammer

physical memory



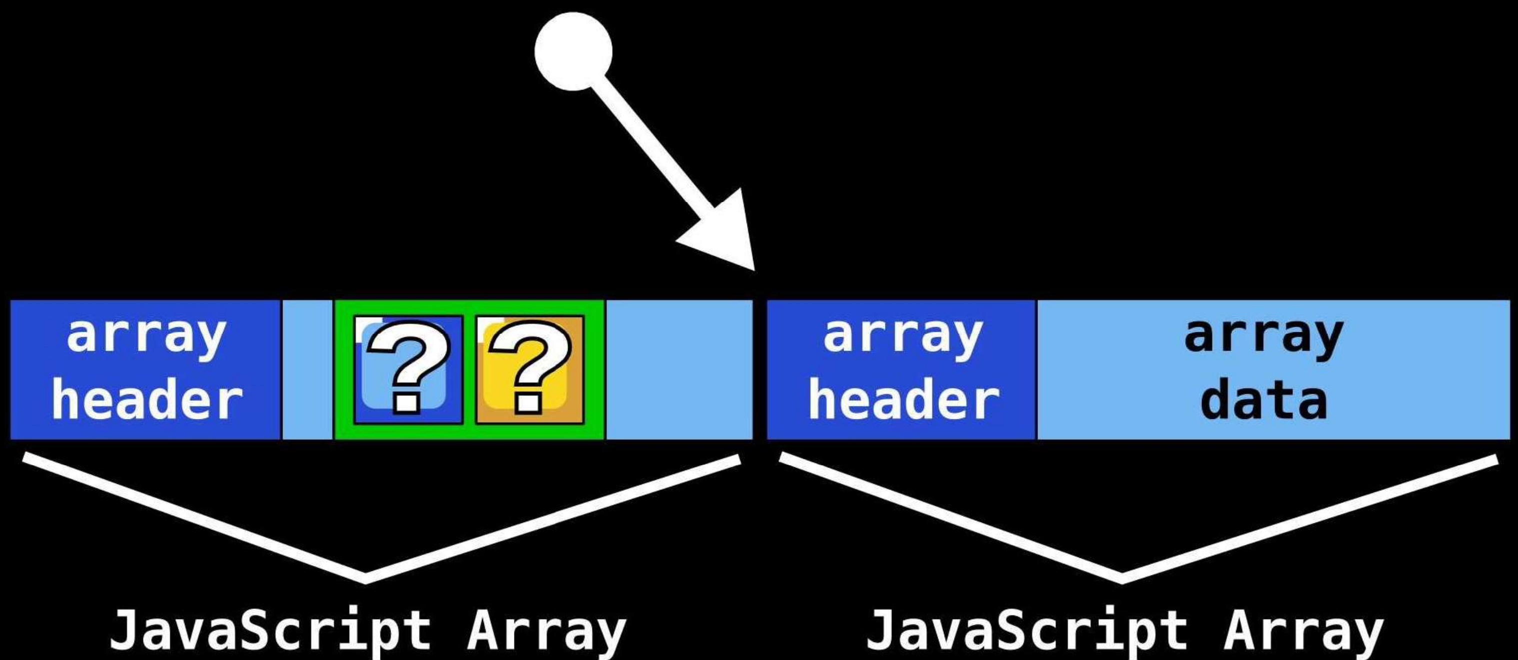
Dedup Est Machina: Referencing the Fake Object

Double-sided Rowhammer



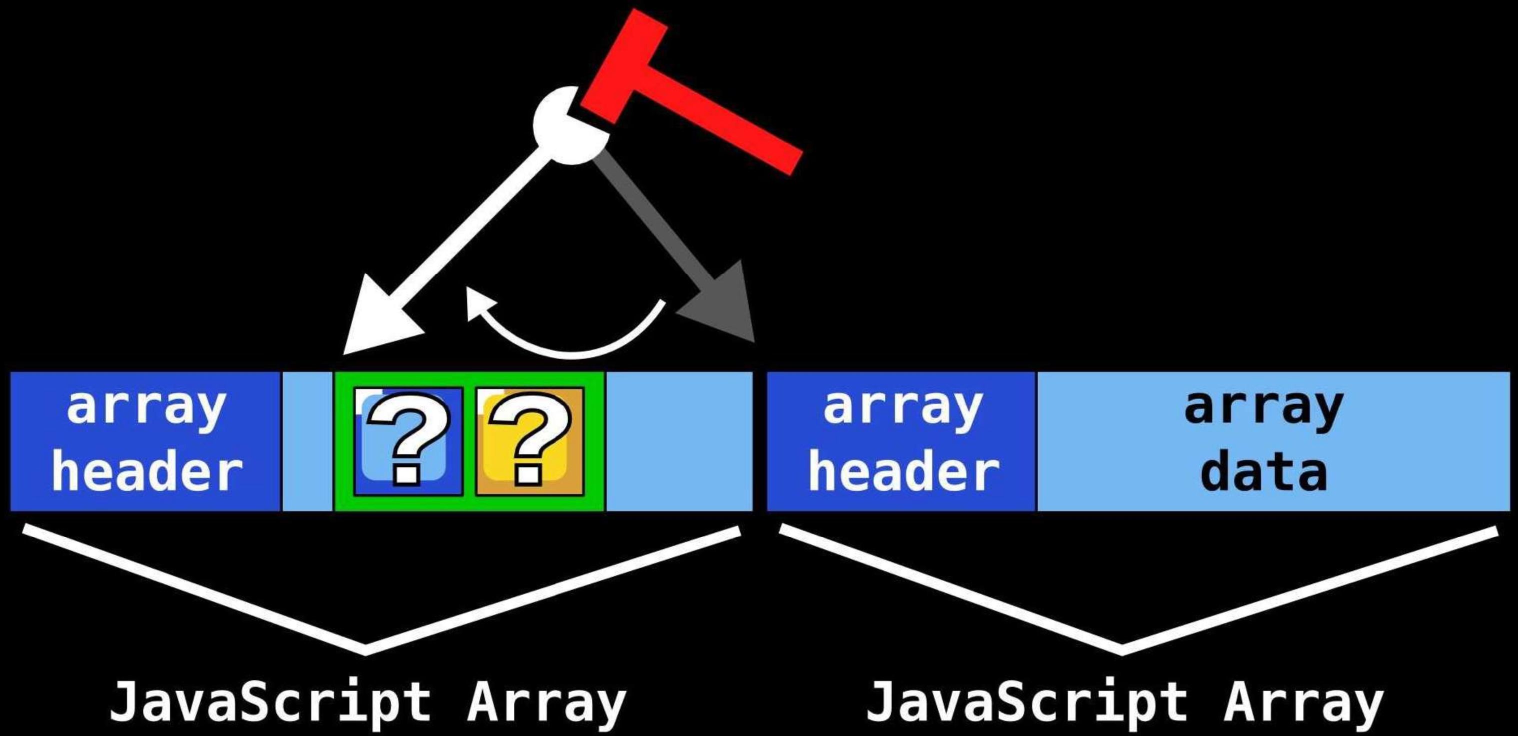
Dedup Est Machina: Referencing the Fake Object

Pointer Pivoting



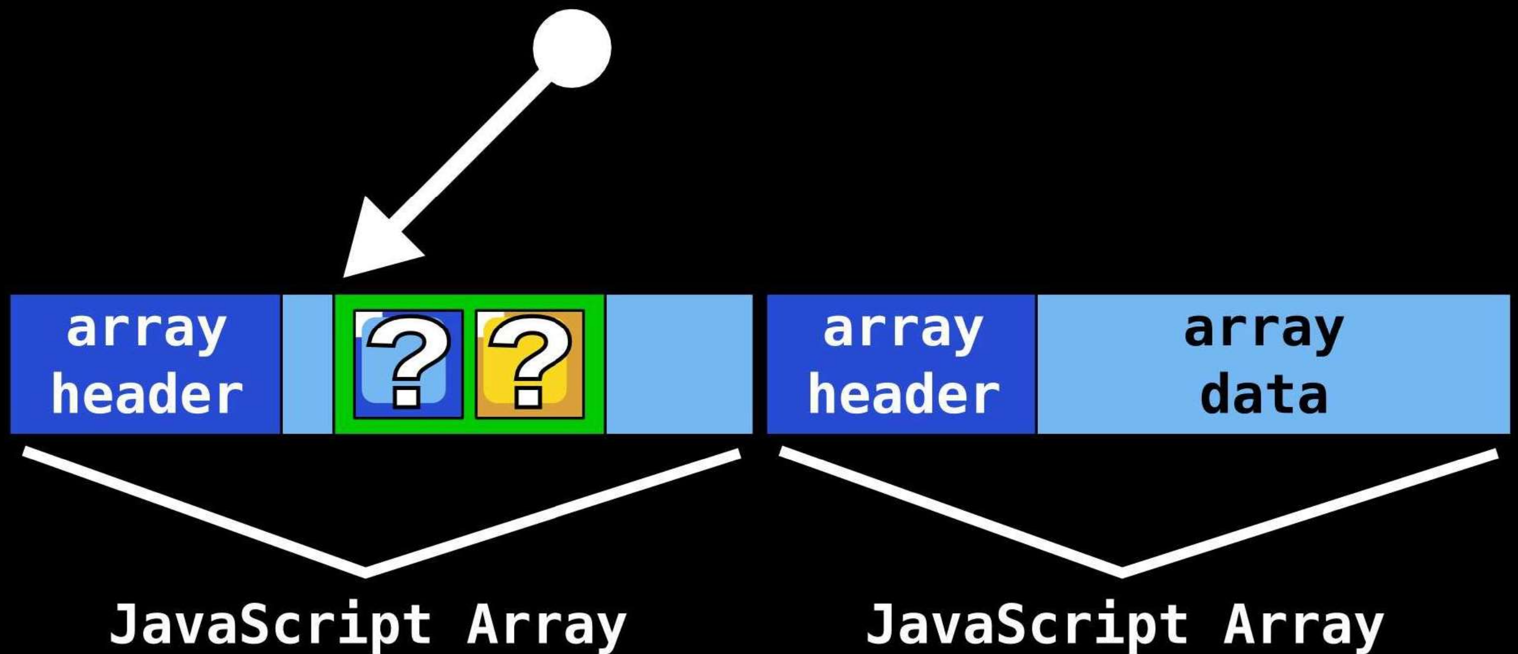
Dedup Est Machina: Referencing the Fake Object

Pointer Pivoting



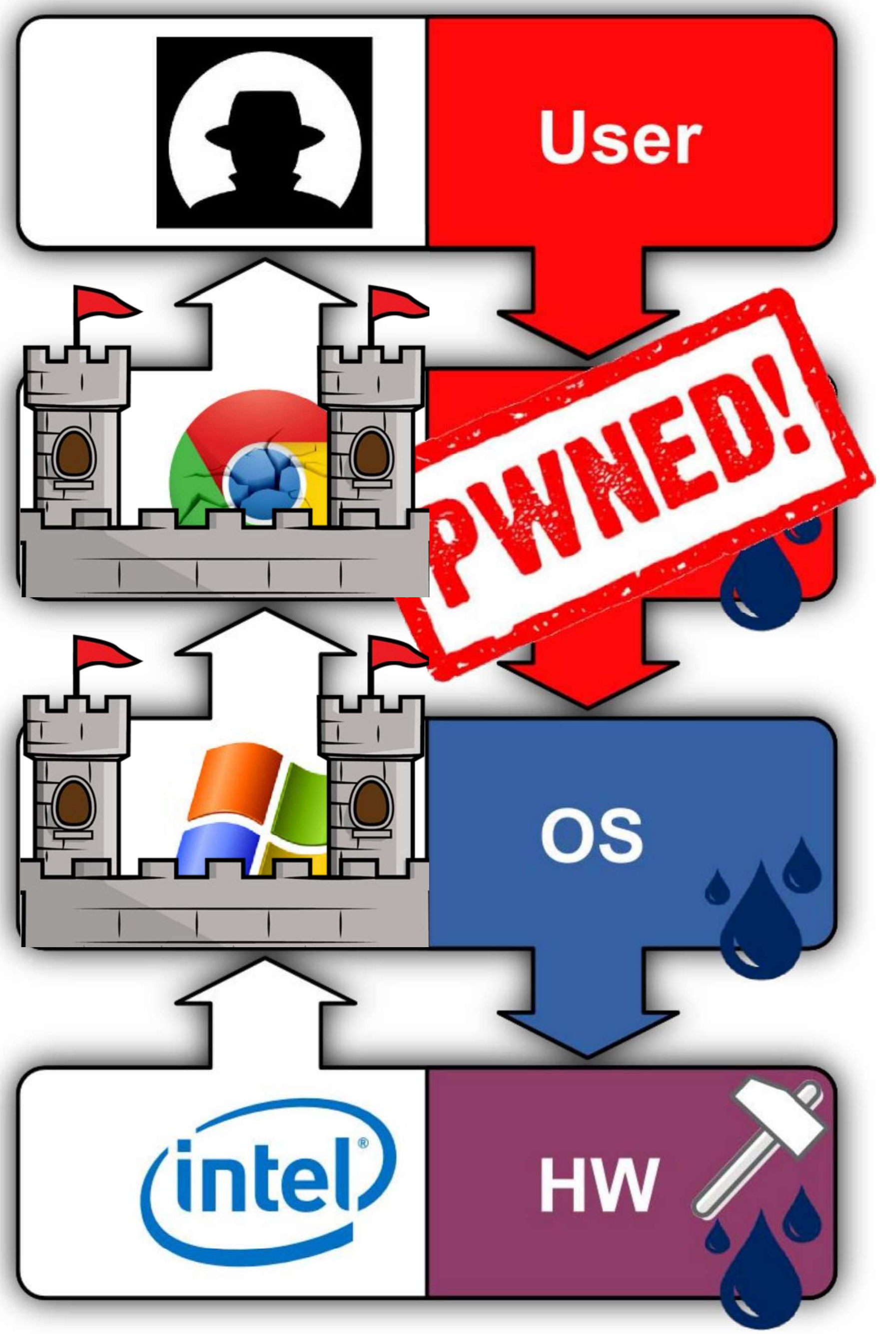
Dedup Est Machina: Referencing the Fake Object

Pointer Pivoting



Dedup Est Machina:

Can One Attack the Full System?



Dedup Est Machina: System-wide Exploitation

- Deduplication is enabled system-wide
- We can leak secrets from other processes
- Say arbitrarily long passwords
    - E.g., 30-byte password hashes in nginx
- System-wide Rowhammer is more involved
    - We don’t “own” other processes’ physical memory
- We’ll look at this in our **next example**

Dedup Est Machina: Impact

- We shared our MS Edge exploit with Microsoft. They addressed this issue in MS-16-093 (CVE-2016-3272) by disabling memory deduplication on Windows 10
- Disable it on legacy systems (Powershell)

```
> Disable-MMAgent -PageCombining
```

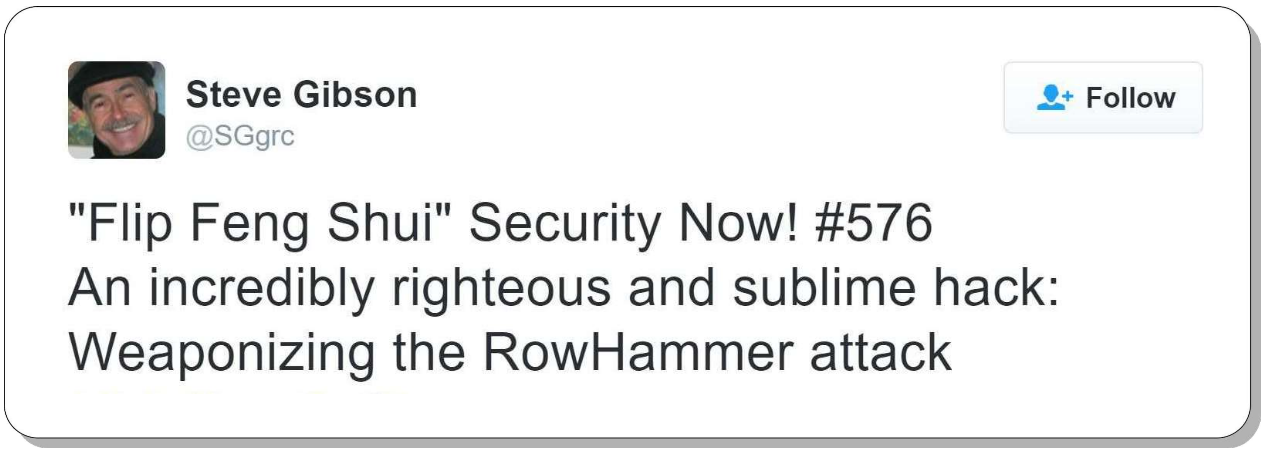
- https://vusec.net/projects/dedup-est-machina

EXAMPLE 2

Bug-free Exploitation in Clouds

Flip Feng Shui

- Published at USENIX Security 2016
  - with Ben, Kaveh, Erik, Herbert, and Bart (KU Leuven)
- Much media attention



- System-wide exploit in public KVM clouds
  - ...without relying on a single software bug

Flip Feng Shui: Overview

Rowhammer

(hardware glitch)

Flip Feng Shui: Overview

Rowhammer

(hardware glitch)

+

Memory deduplication

(physical memory massaging primitive)

Flip Feng Shui: Overview

Rowhammer

(hardware glitch)

+

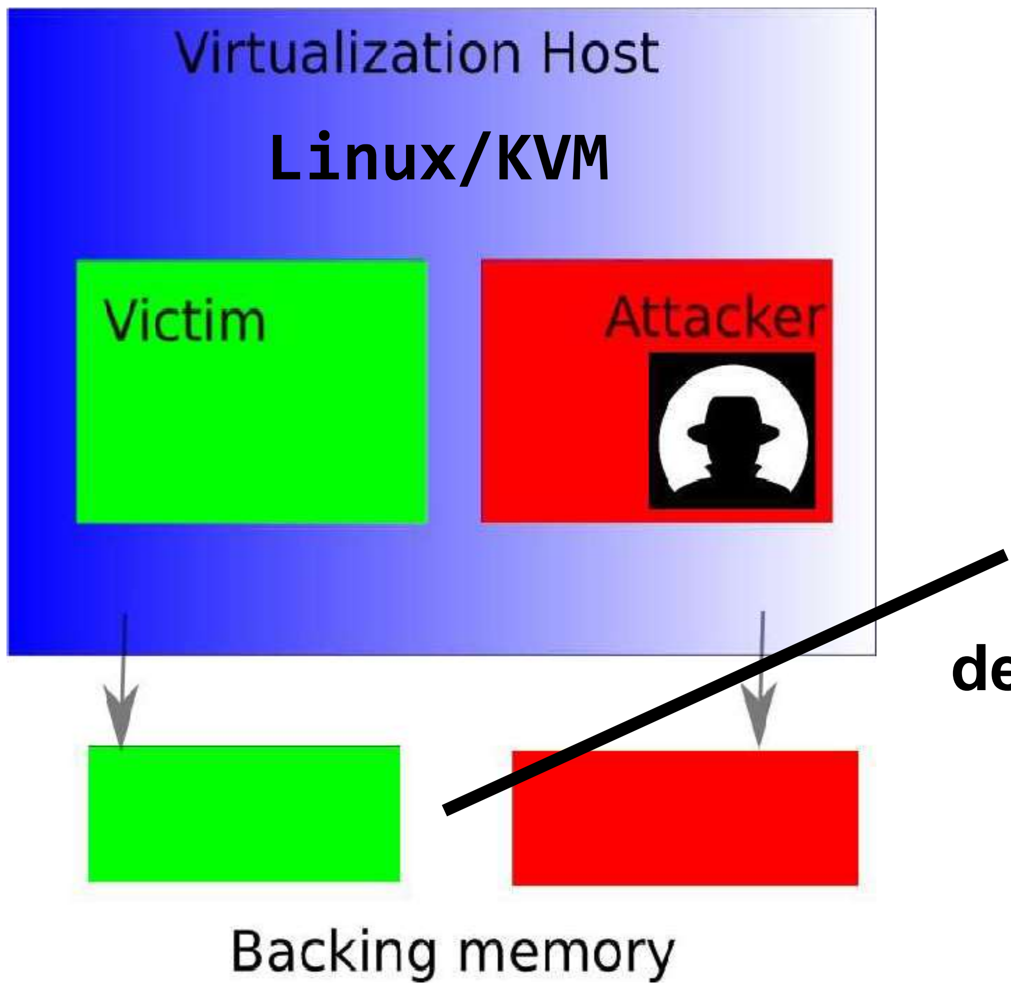
Memory deduplication

(physical memory massaging primitive)

↓

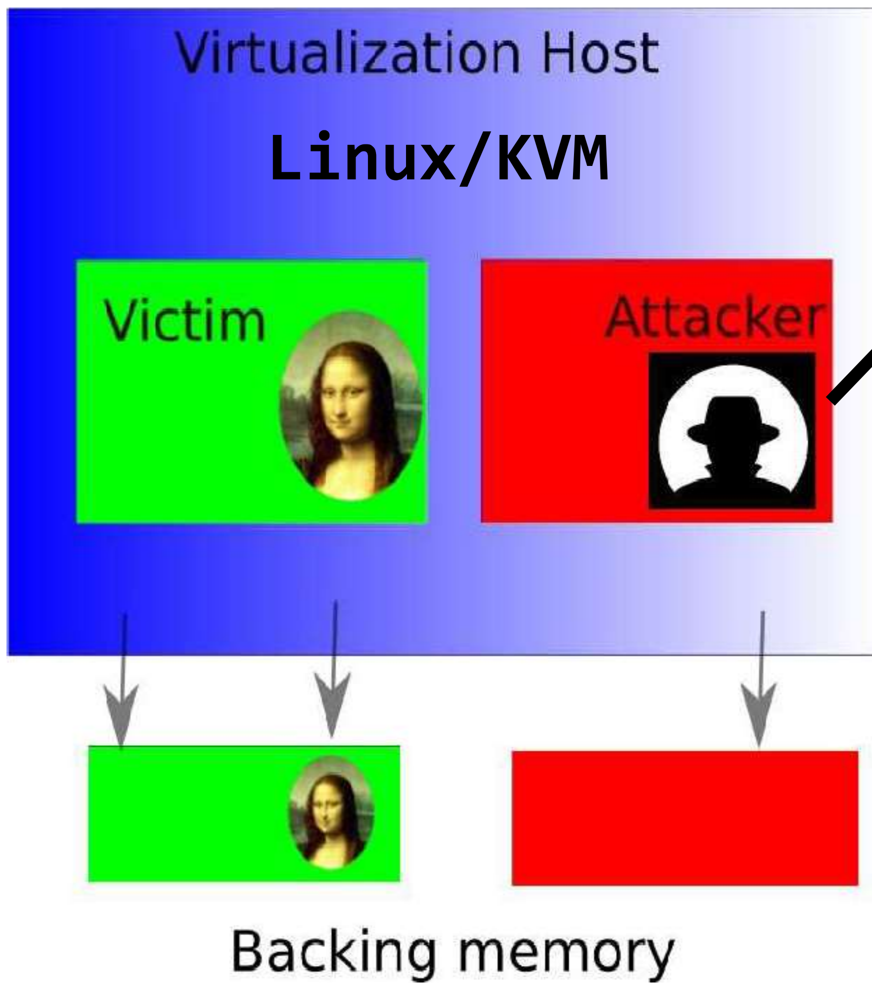
Cross-VM compromise in public Linux/KVM clouds without software bugs

Flip Feng Shui: Attacker's Goals



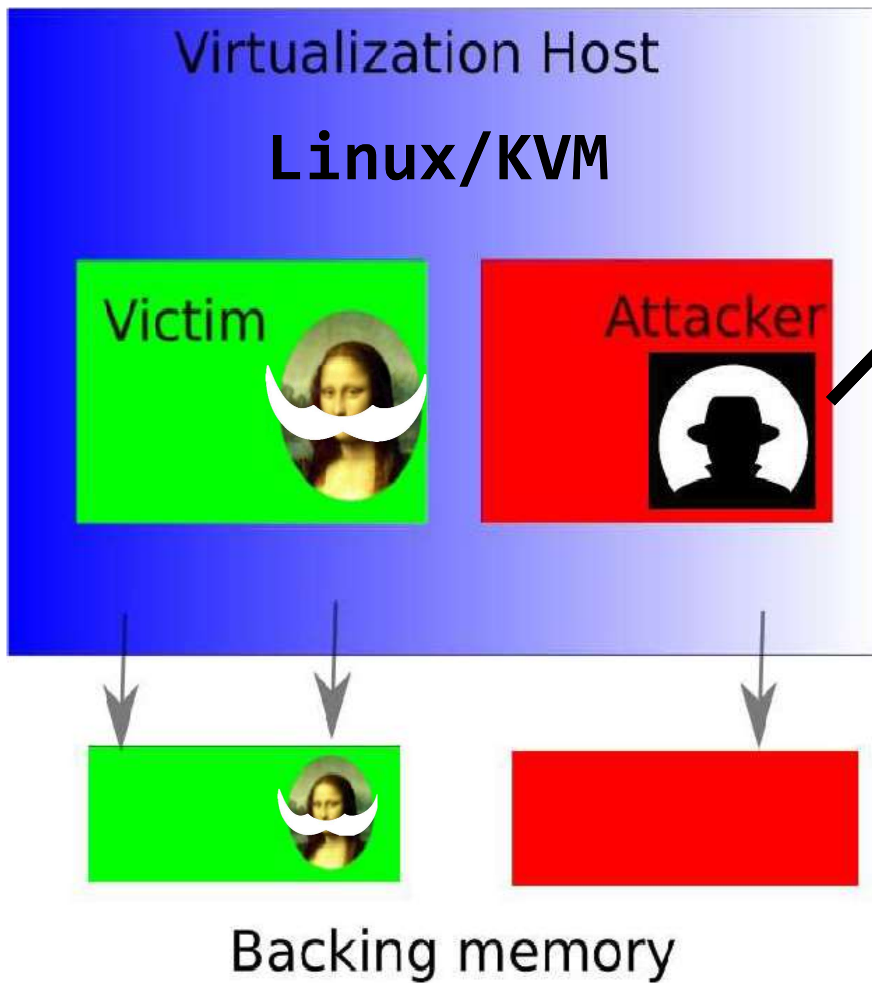
KSM: cross-VM memory deduplication

Flip Feng Shui: Attacker's Goals



Target sensitive memory page in victim VM's memory

Flip Feng Shui: Attacker's Goals



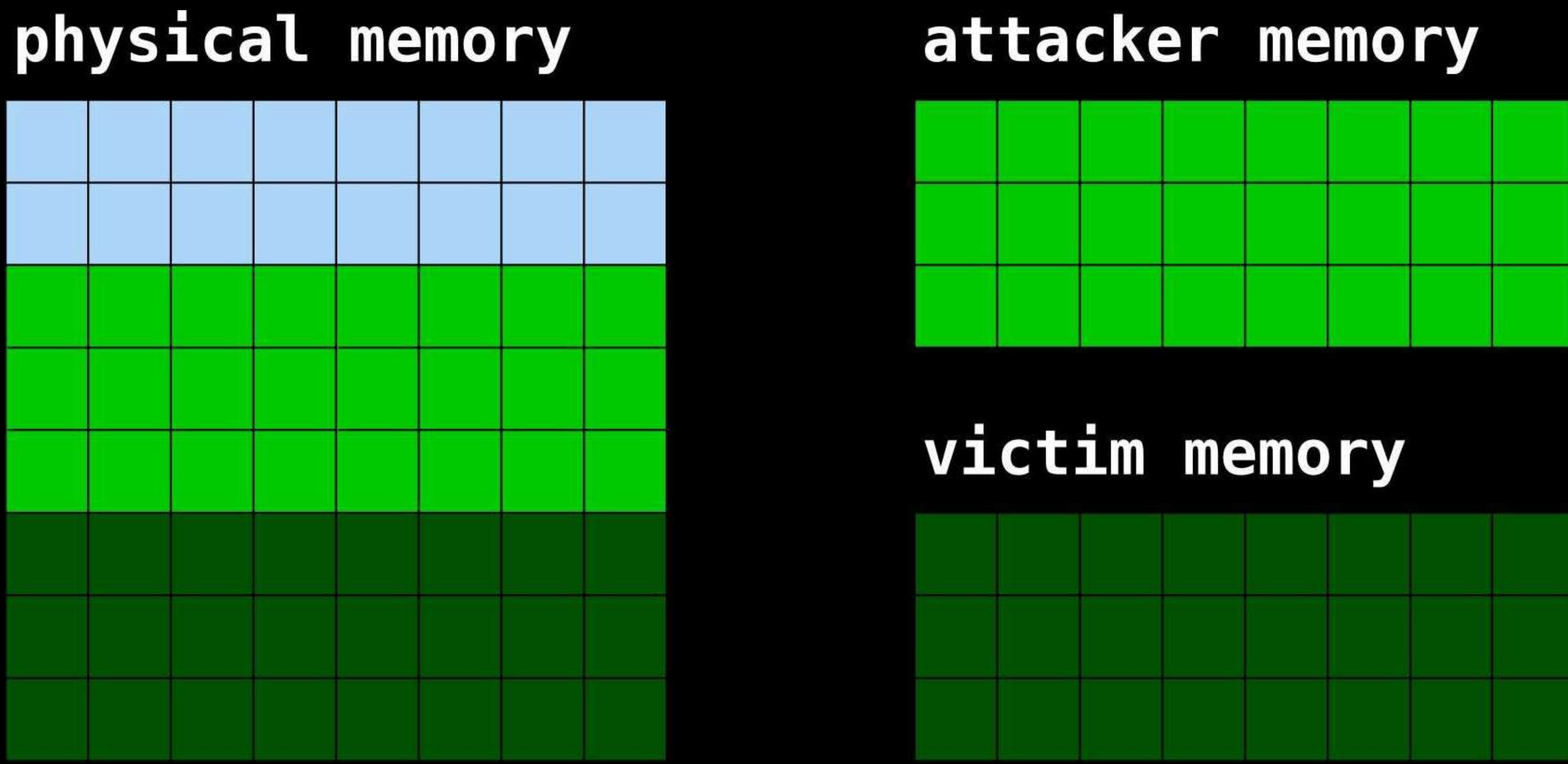
Corrupt sensitive page to subvert victim VM

Flip Feng Shui: Mechanics

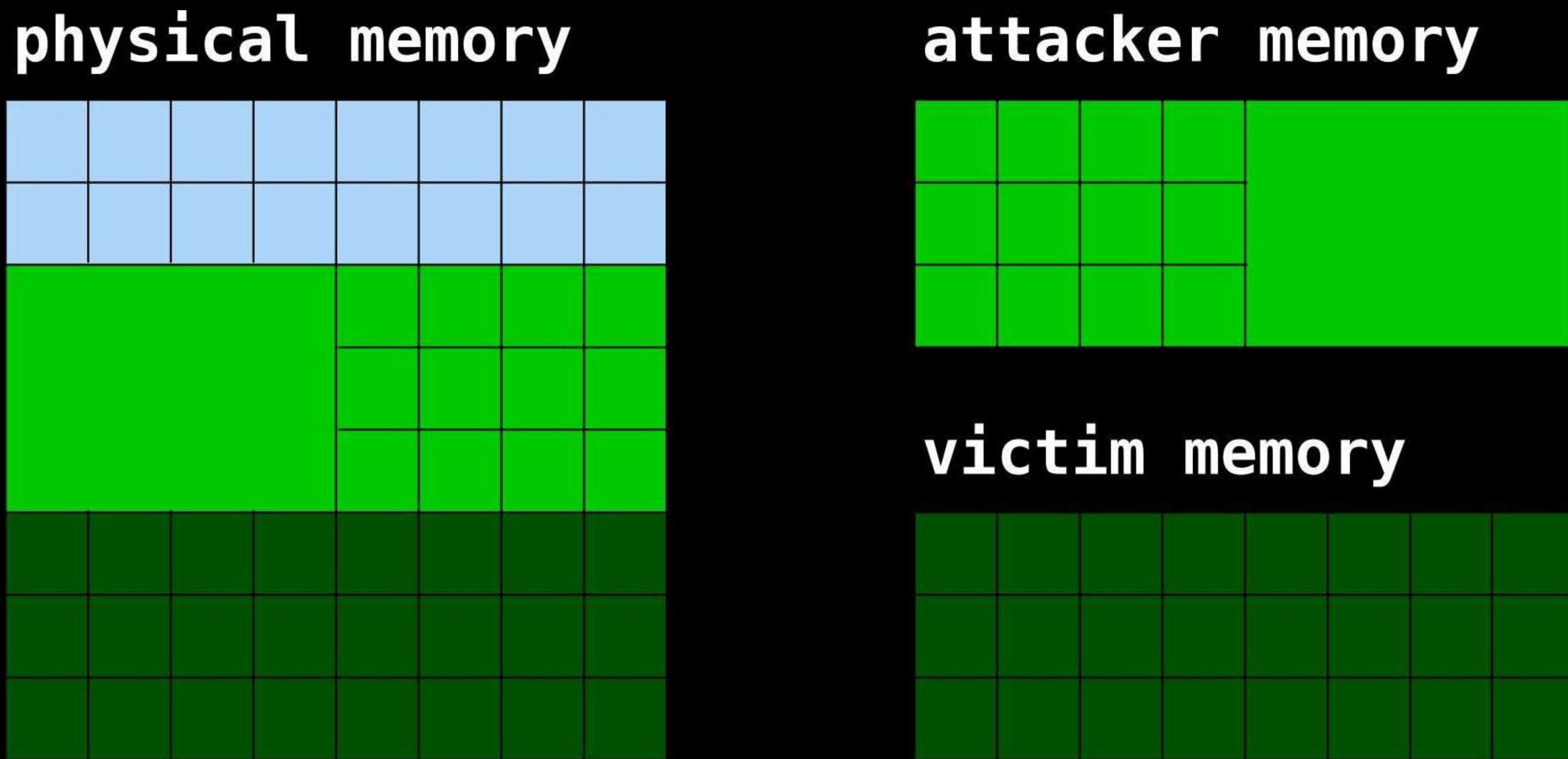
Step 1:

The attacker needs to find a vulnerable physical page to flip bits at a given sensitive offset

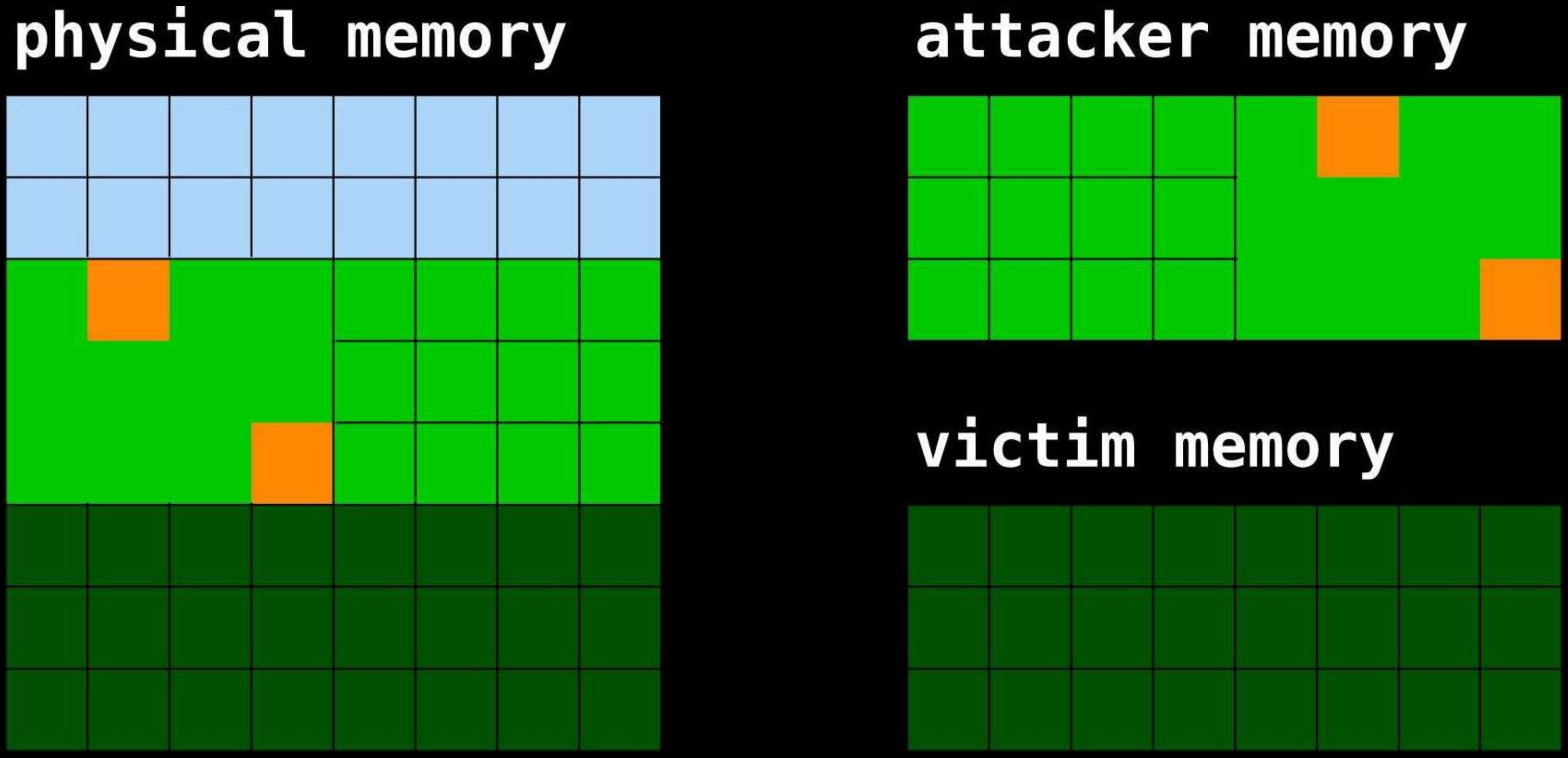
Flip Feng Shui: Templating



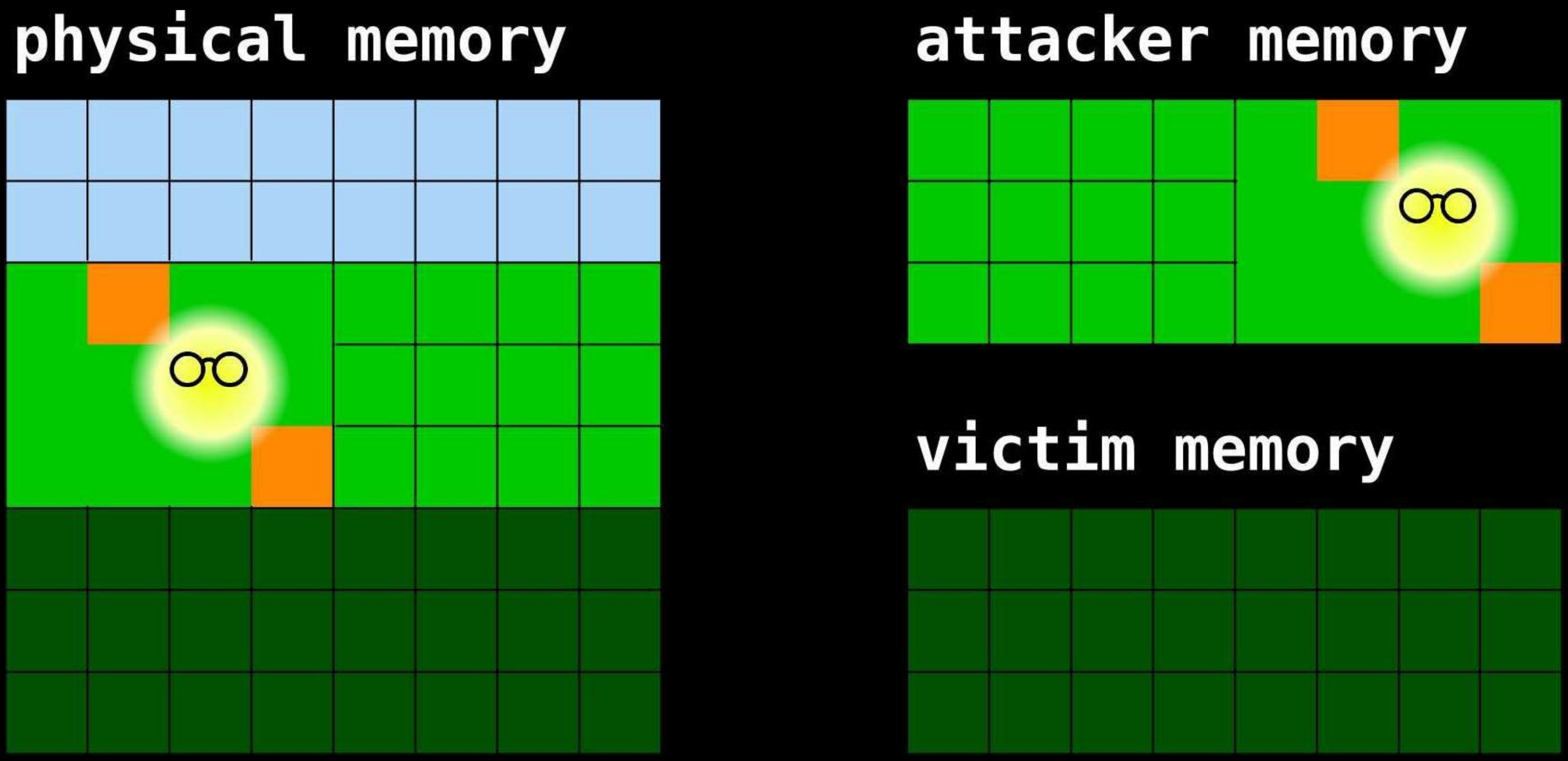
Flip Feng Shui: Templating



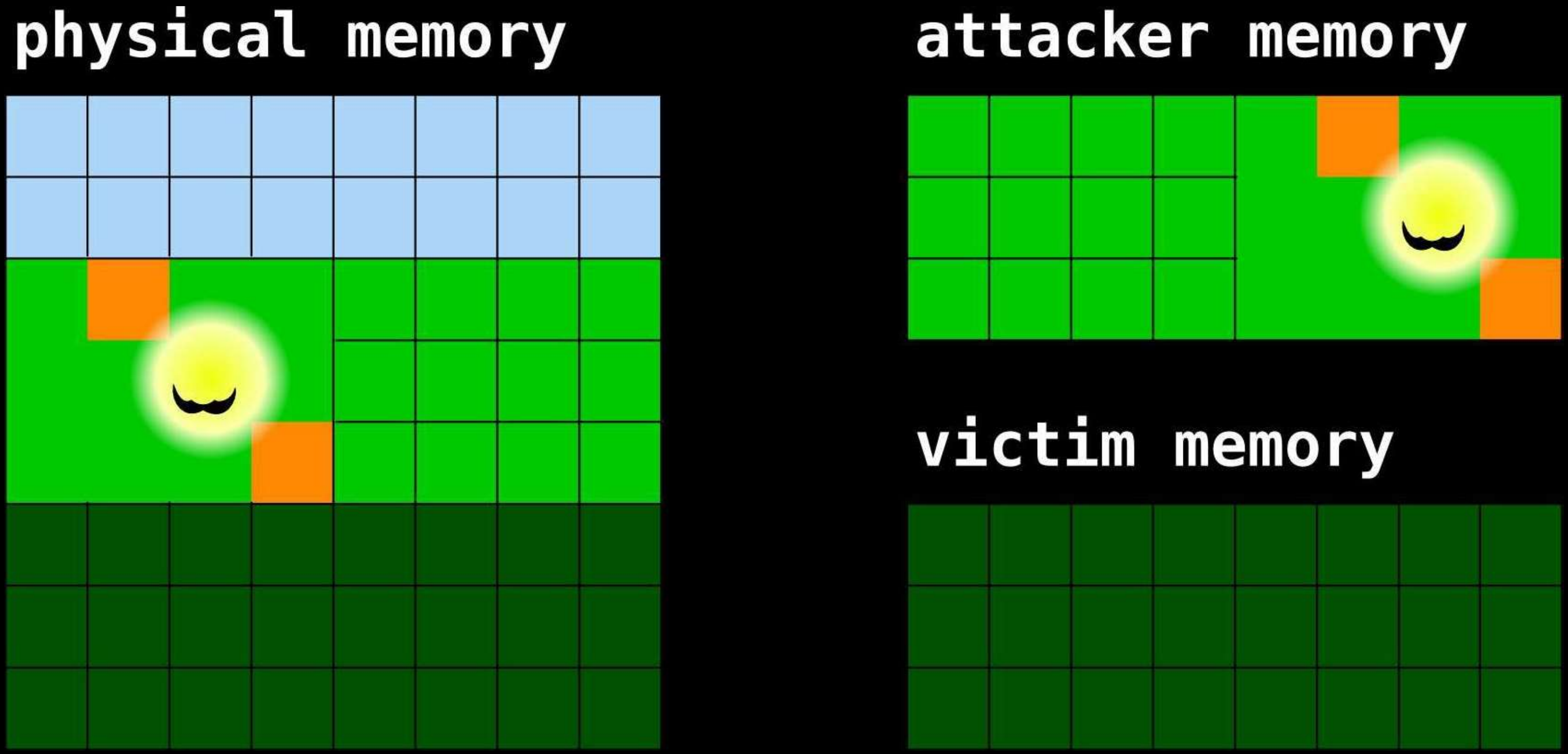
Flip Feng Shui: Templating



Flip Feng Shui: Templating



Flip Feng Shui: Templating

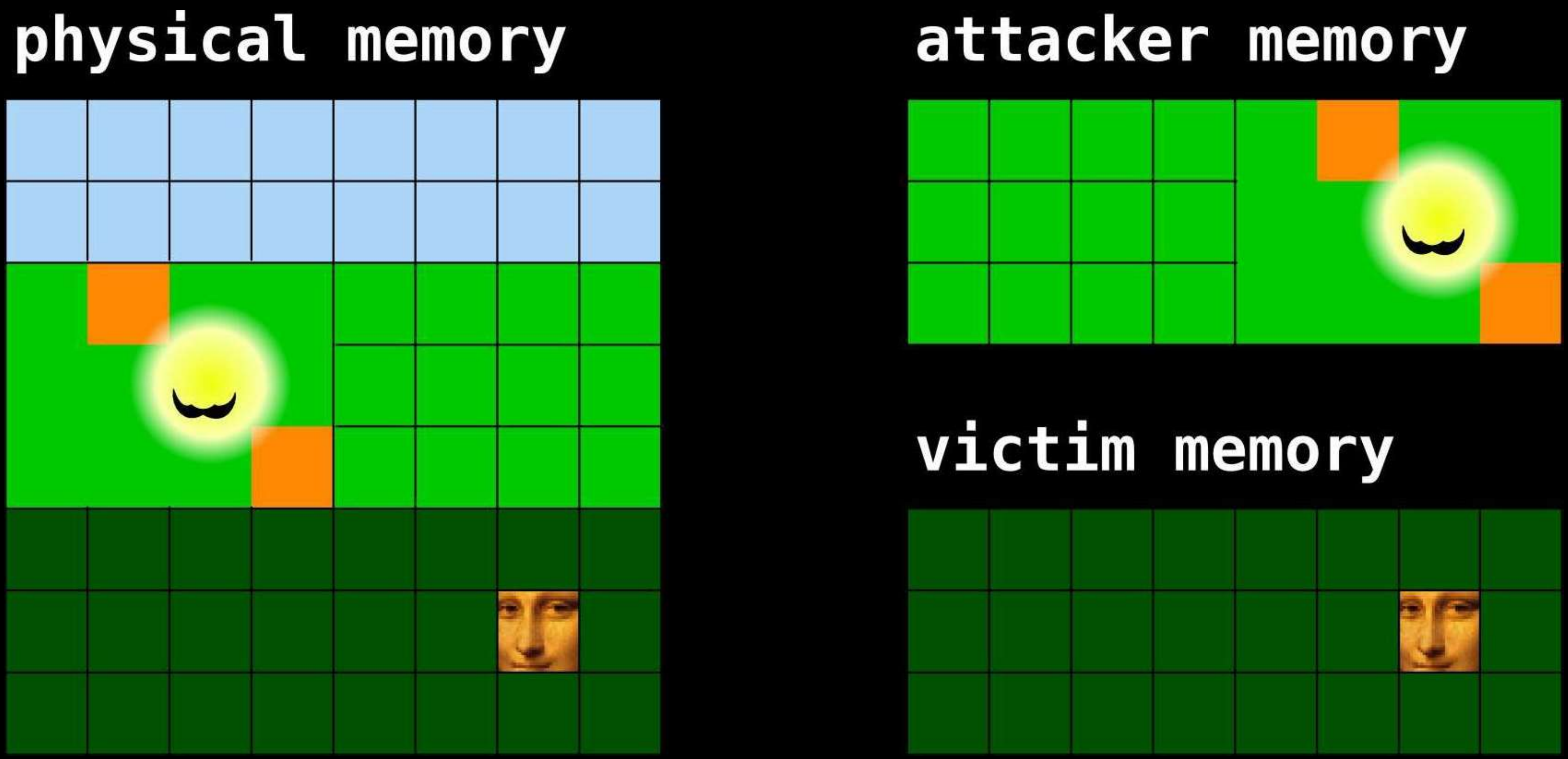


Flip Feng Shui: Mechanics

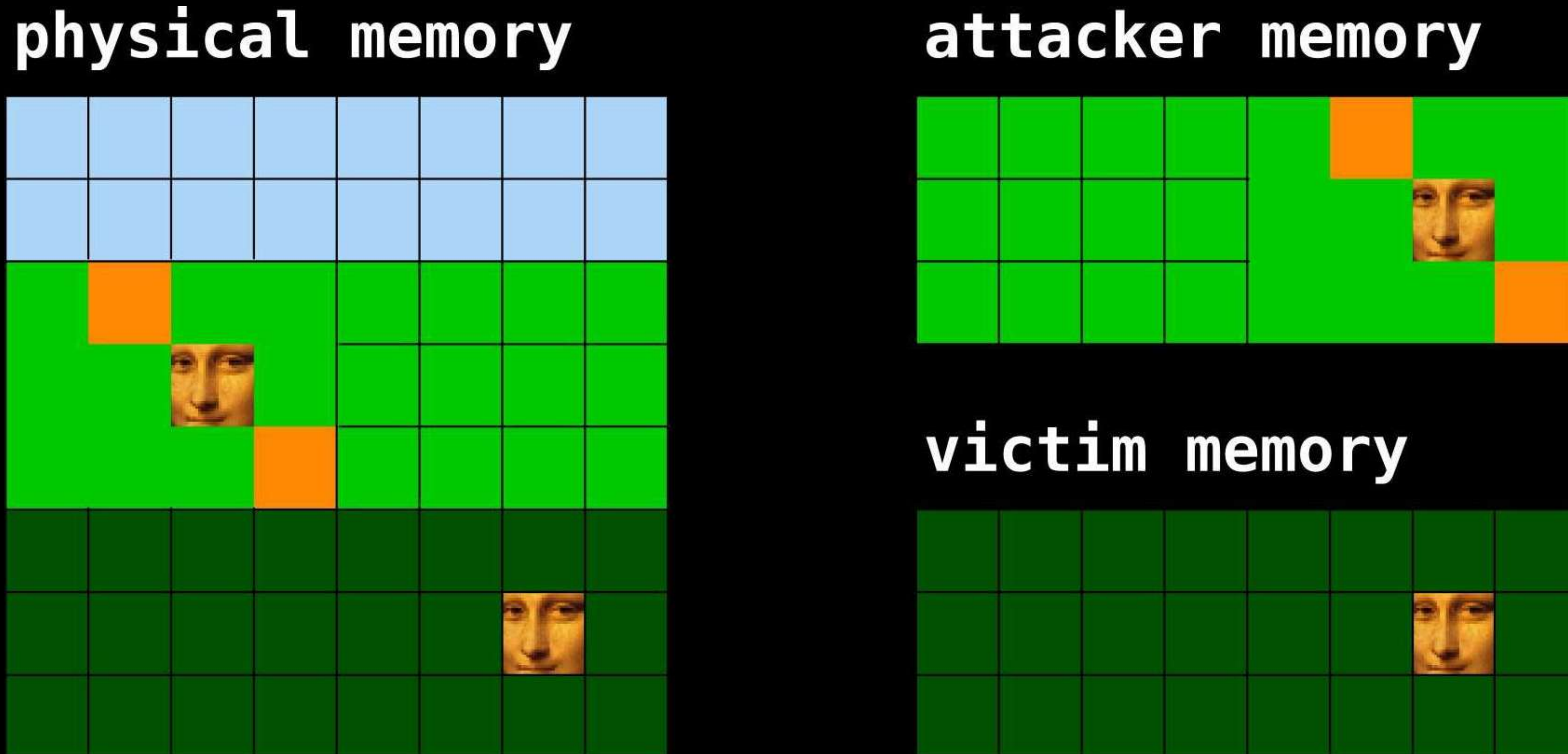
Step 2:

The attacker needs to force the system to map the victim page into the vulnerable template

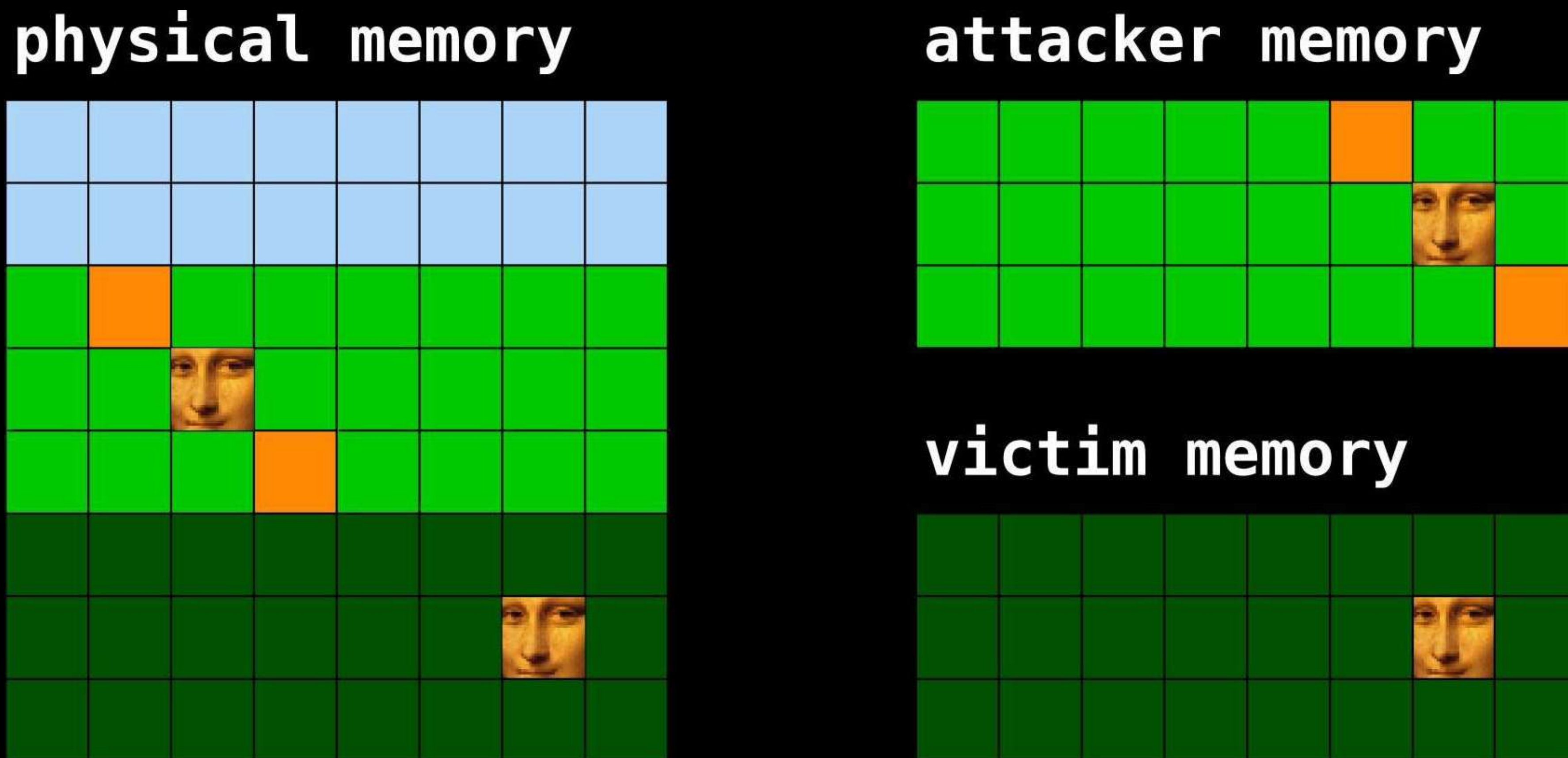
Flip Feng Shui: Physical Memory Massaging



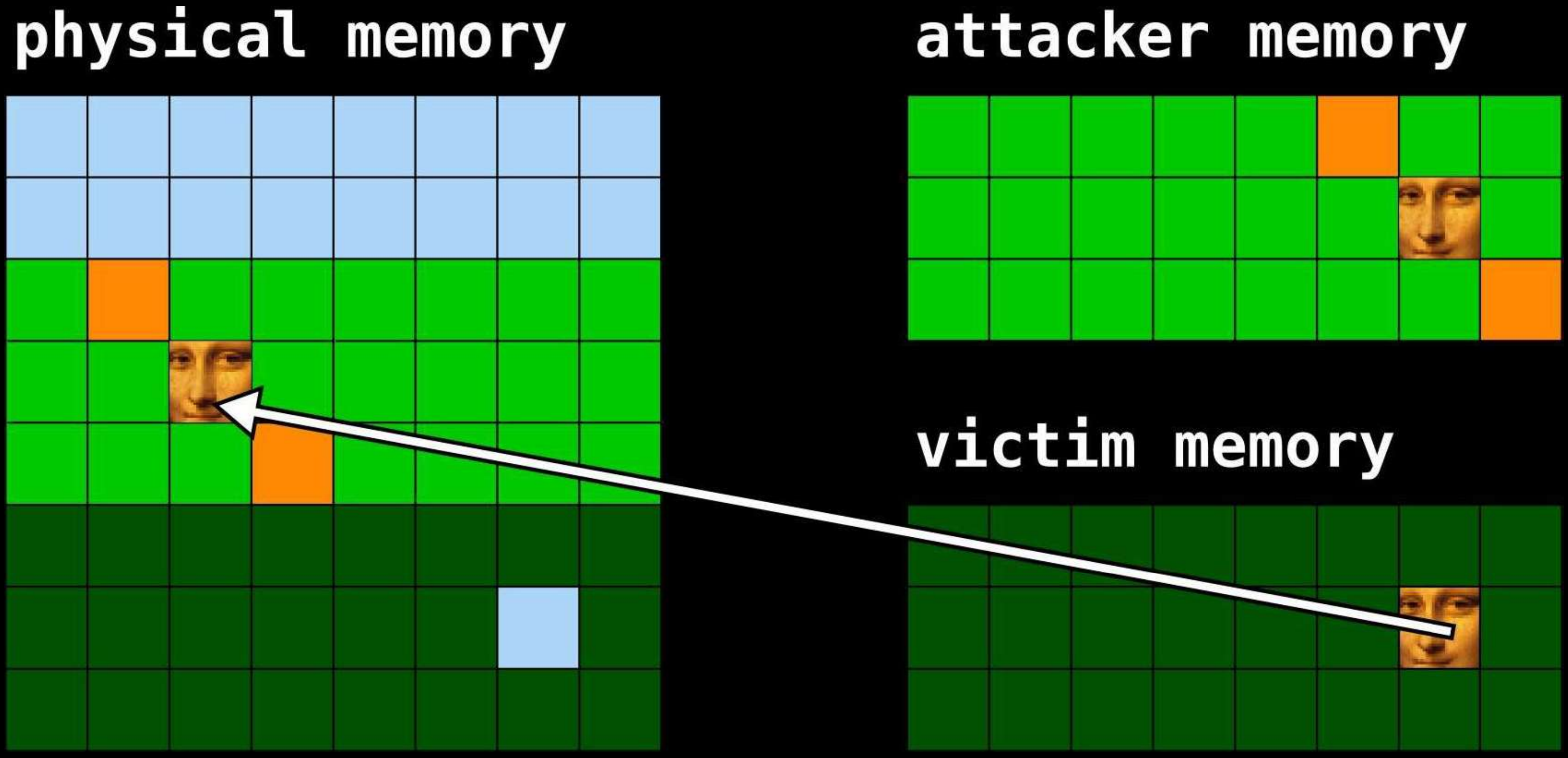
Flip Feng Shui: Physical Memory Massaging



Flip Feng Shui:
Physical Memory Massaging



Flip Feng Shui: Physical Memory Massaging

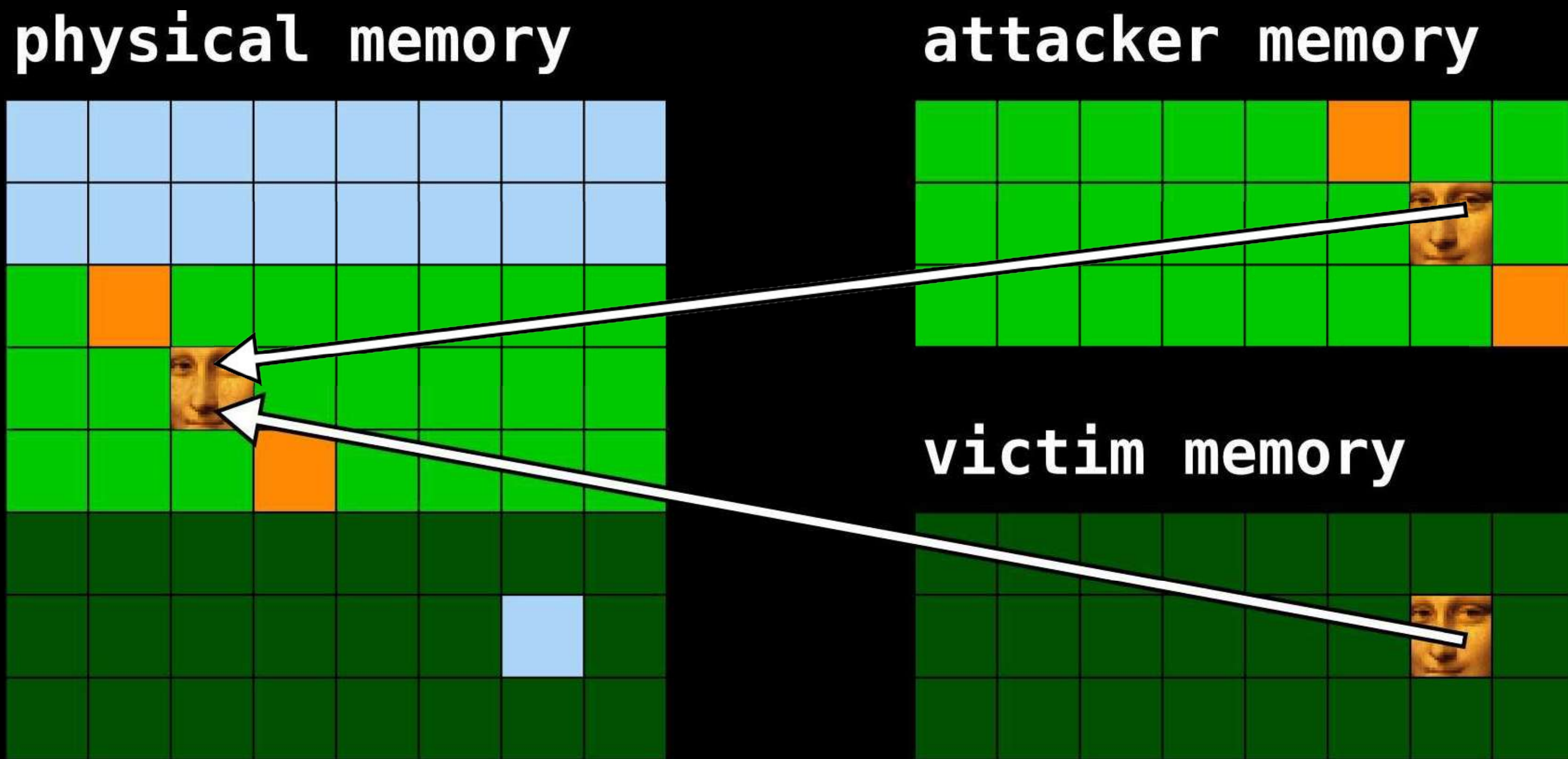


Flip Feng Shui: Mechanics

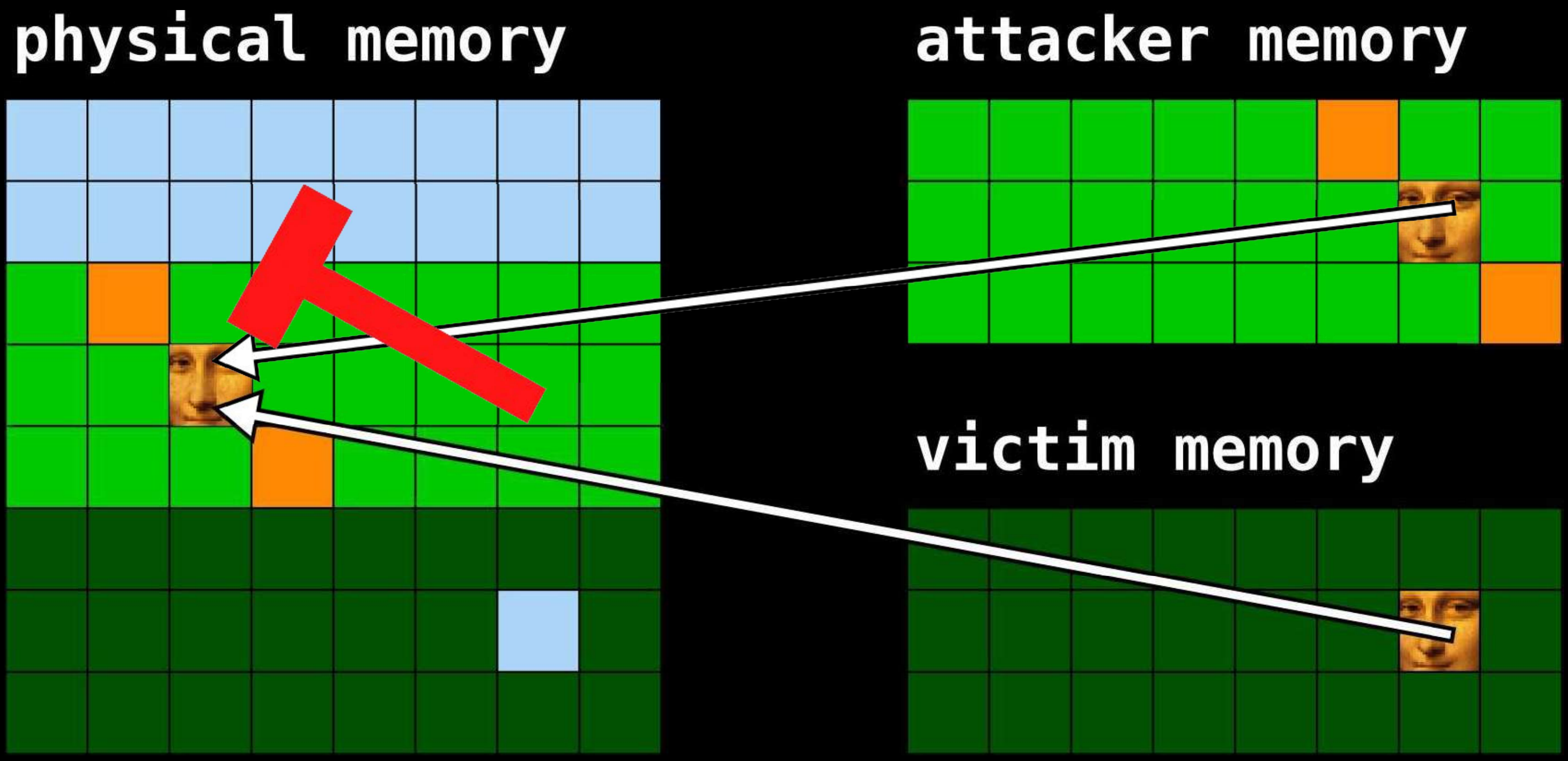
Step 3:

The attacker needs to flip the bit at the sensitive offset in the vulnerable template

Flip Feng Shui: Exploitation

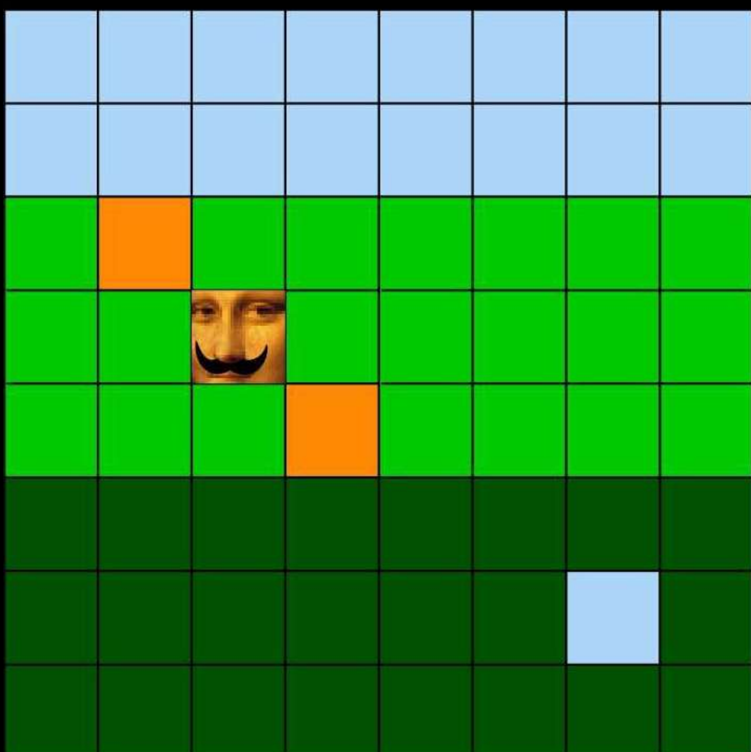


Flip Feng Shui: Exploitation

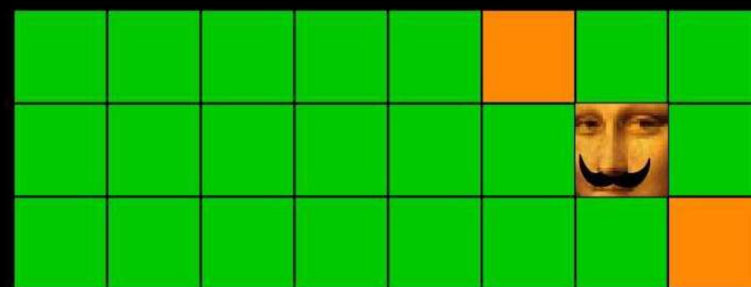


Flip Feng Shui: Exploitation

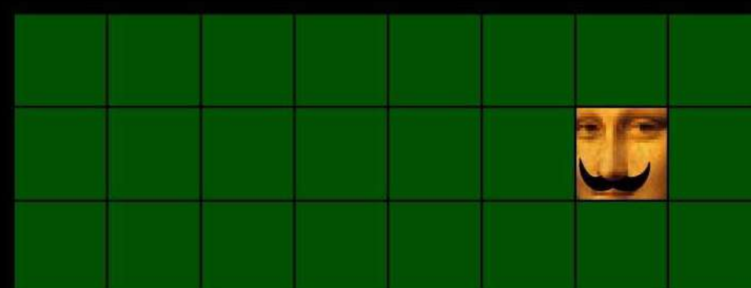
physical memory



attacker memory



victim memory



Flip Feng Shui: Finding a Victim Page

The attacker wants a **victim page**:

- containing security-sensitive data
  - Corruption should result in cross-VM compromise
- with predictable content
  - For memory deduplication to map it into attacker VM
- with ideally many sensitive offsets
  - Easier to find useful templates

Flip Feng Shui:
Finding a Victim Page

How about **public cryptographic keys**?

- Public keys are not secret, thus predictable
- Arbitrary corruption weakens their security

Flip Feng Shui: OpenSSH Attack

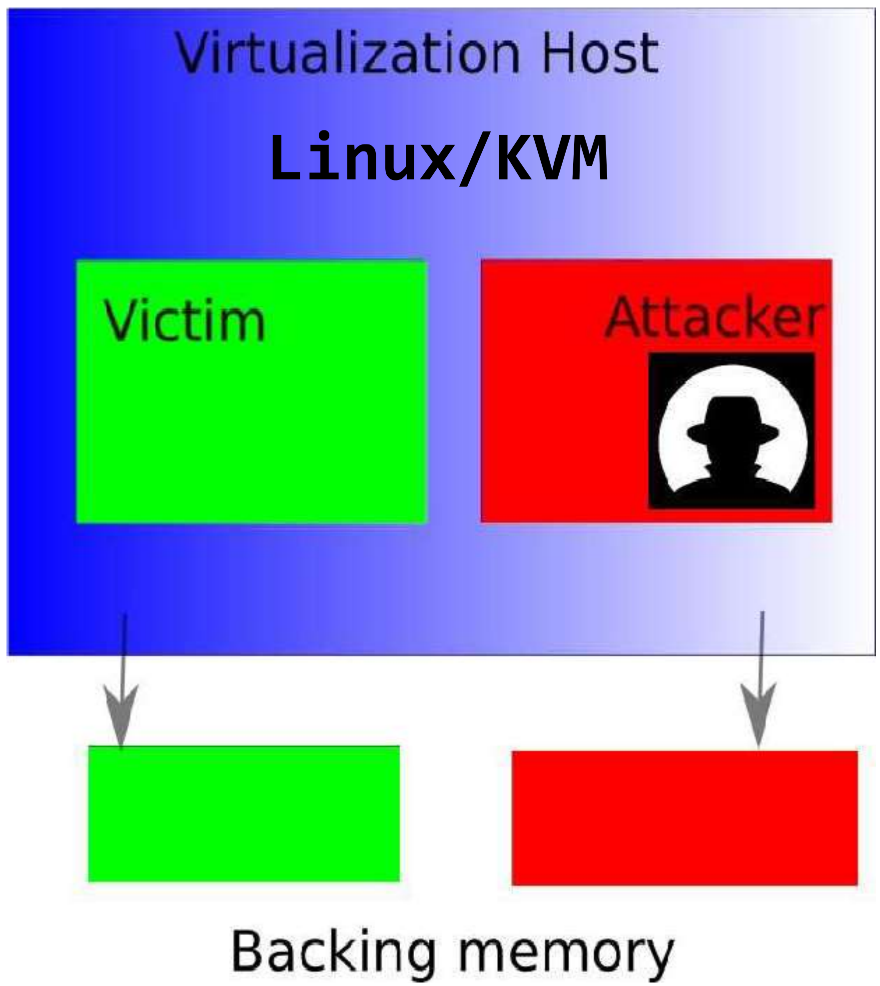
How about **public cryptographic keys**?

- Public keys are not secret, thus predictable
- Arbitrary corruption weakens their security

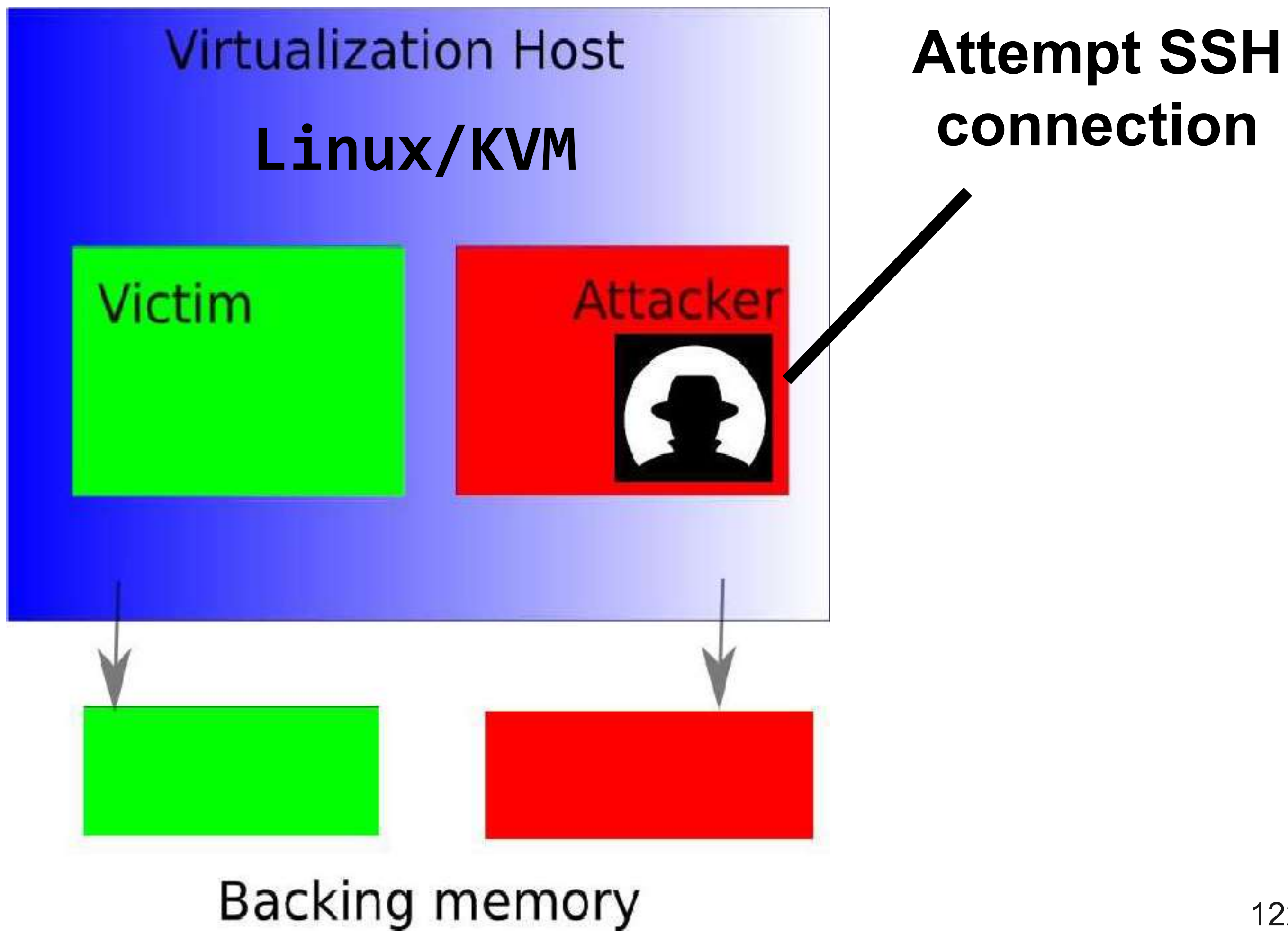
Target OpenSSH’s `~/.ssh/authorized_keys` to SSH to victim VM and login as administrator


```
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQDMUensMjWvw+d4SLKCVcP0MR
3n2PsSohXBroW/qOcUXB8NFH1bWXUORC/uSPnAnWH1QYeuIP5UNnkBXWpDGgjm
WTbrUfA4tqW1BBwjii4qIUWcBGql1dBUvqWsWbZ86/NY2fsKLtLDkk1eFhcJmN
FXnYkRs3J21BGS7JdUnDd9ue0x2Nk/aSp2GODzAXwDPhwQNw4LQ8/xZTkn5Djq
IAAXBpa+qaqTMdKNItOi/IVLoR/7BqgVslt3tbgZmew4IsmUFQMCwKdxBk5TxA
agAjCmwmh+gRt0/tb6tDKzvVCNcHc4968VPXJYK2+Hr/RdYloYSLoIV/DQcTIy
yYzhUV5v test@source
```

Flip Feng Shui: OpenSSH Attack



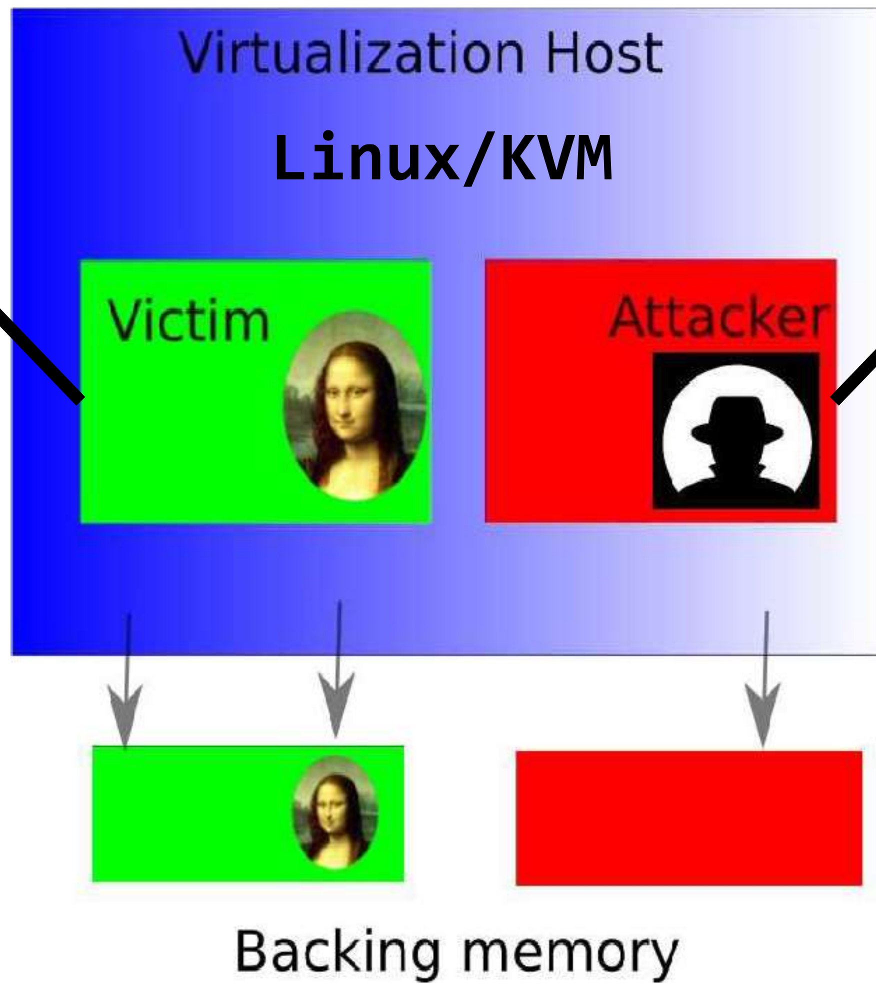
Flip Feng Shui: OpenSSH Attack



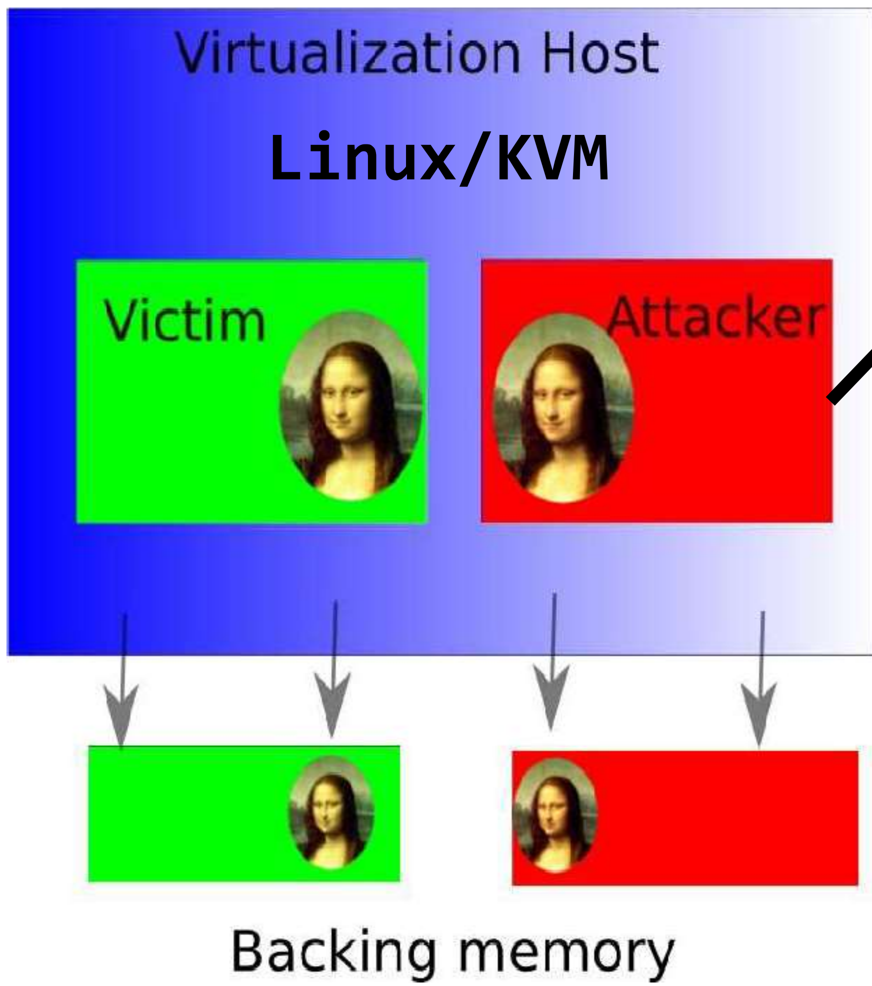
Flip Feng Shui: OpenSSH Attack

Check authorized_keys

Attempt SSH connection

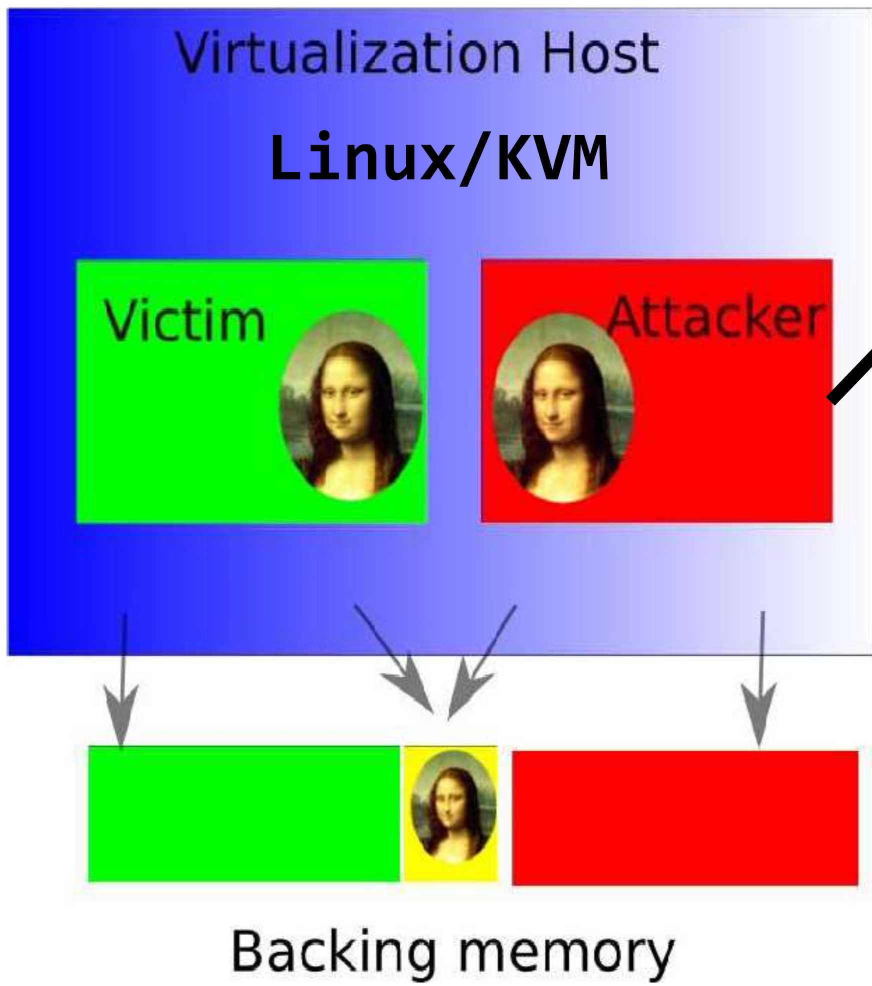


Flip Feng Shui: OpenSSH Attack



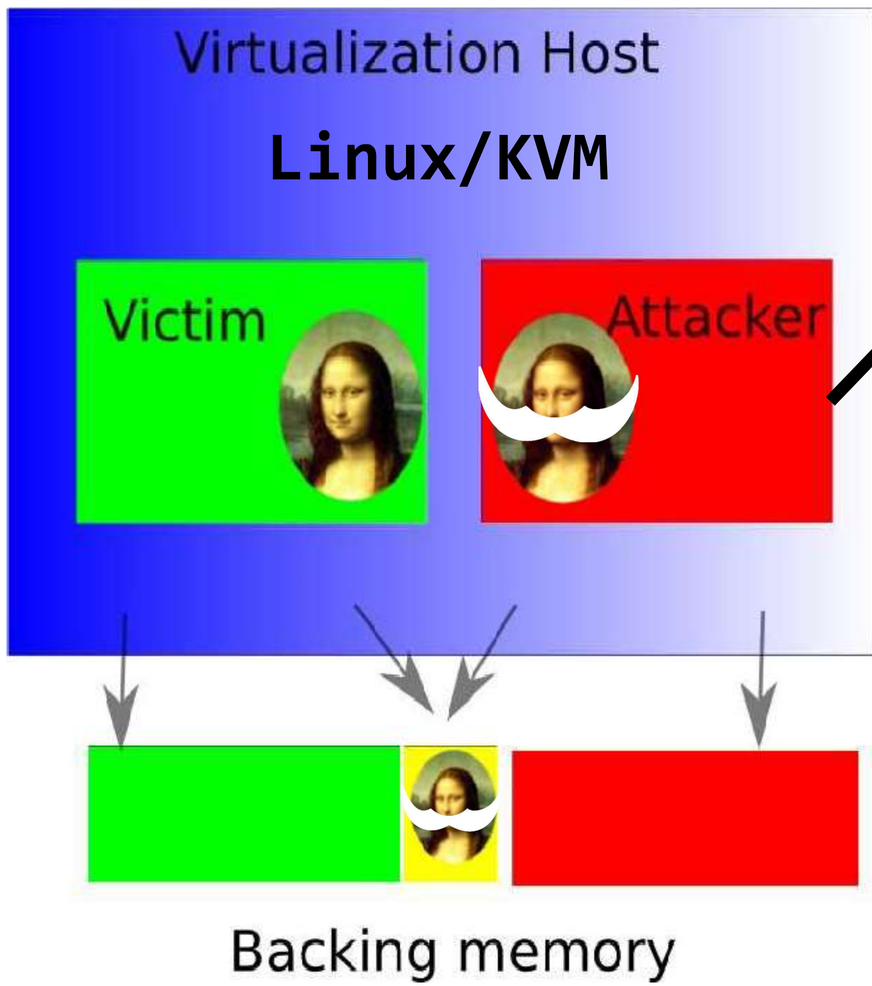
Craft victim page content in vulnerable template

Flip Feng Shui: OpenSSH Attack



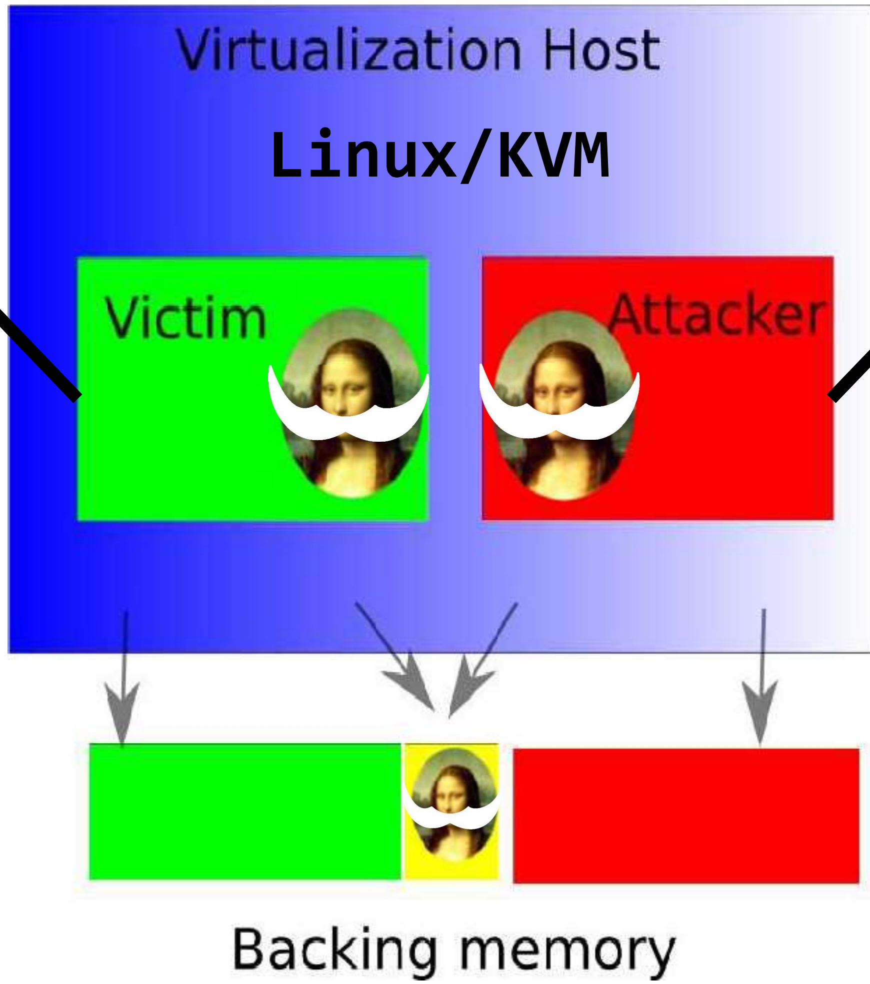
Dedup moves the victim page to the vulnerable template

Flip Feng Shui: OpenSSH Attack



Flip Feng Shui: OpenSSH Attack

Changes are reflected in the victim page



It's Hammer time!

Flip Feng Shui: OpenSSH Attack

A bit flip in a **public RSA key**...

- Results in a weak key one can factorize
- Easy to reconstruct the new private key
- We do this in minutes and login to the VM!


```
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQDMUensMjWvw+d4SLKCVcP0MR
3n2PsSohXBroW/qOcUXB8NFHlbWXUORC/uSPnAnWH1QYeuIP5UNnkBXWpDGgjm
WTbrUfA4tqW1BBwjii4qIUWcBGql1dBUvqWsWbZ86/NY2fsKLtLDkk1eFhcJmN
FXnYkRs3J21BGS7JdUnDd9ue0x2Nk/aSp2GODzAXwDPhwQNw4LQ8/xZTkn5Djq
IAAXBpa+qaqTMdKNItOi/IVLoR/7BqgVslt3tbgZmew4IsmUFQMCwKdxBk5TxA
agAjCmwmh+gRt0/tb6tDKzvVCNcHc4968VPXJYK2+Hr/RdYloYSLoIV/DQcTIy
yYzhUV5v test@source
```

Flip Feng Shui: Impact

- Notified:
  - Red Hat, Oracle, Xen, VMware, Debian, Ubuntu, OpenSSH, GnuPG, hosting companies

- GnuPG "*included hw bit flips in their threat model*"

gpgv: Tweak default options for extra security.

```
author     NIIBE Yutaka <gniibe@fsij.org>
           Fri, 8 Jul 2016 20:20:02 -0500 (10:20 +0900)
committer  NIIBE Yutaka <gniibe@fsij.org>
           Fri, 8 Jul 2016 20:20:02 -0500 (10:20 +0900)
commit     e32c575e0f3704e7563048eea6d26844bdfc494b
```

- https://vusec.net/projects/flip-feng-shui

Mitigations

“Can we just disable memory deduplication and buy better DRAM?”

Yes, you really should, but...

Mitigations

No dedup and want phys mem massaging?

- Enter **Drammer** [CCS ‘16]:
  - Deterministic RH root exploit on ARM/Android
  - User-level DMA mem to flip bits and massage mem

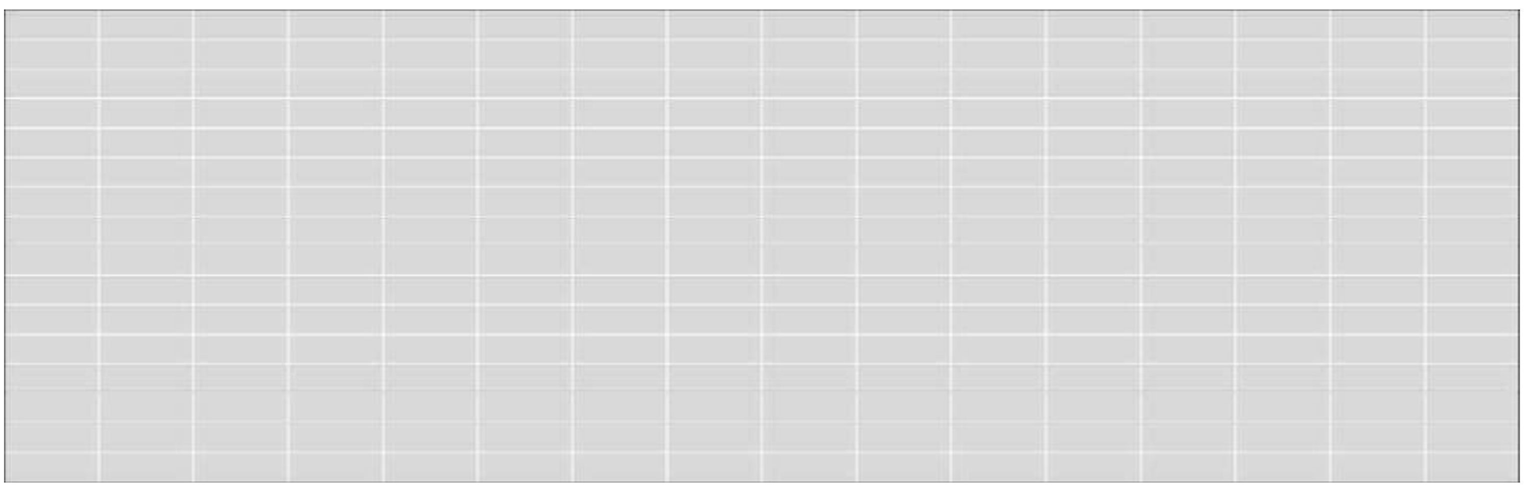
“Android Security Reward”

- More recently:
  - JS: **Glitch** [S&P ‘18]
  - Remote: **Throwhammer**
- https://vusec.net/projects/drammer

Mitigations

No dedup and want phys mem massaging?

- Enter **Drammer** [CCS ‘16]:
  - Deterministic RH root exploit on ARM/Android
  - User-level DMA mem to flip bits and massage mem
- Physical memory massaging approach:



Mitigations

No dedup and want phys mem massaging?

- Enter **Drammer** [CCS '16]:
  - Deterministic RH root exploit on ARM/Android
  - User-level DMA mem to flip bits and massage mem
- Physical memory massaging approach:

Exhaust all physical memory

Mitigations

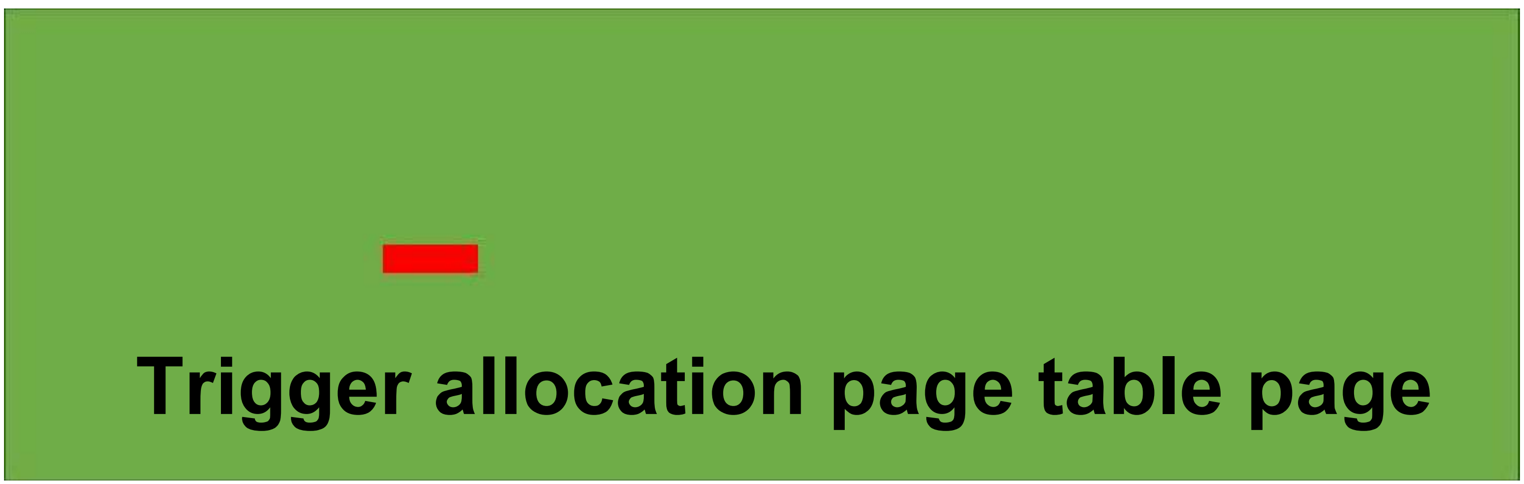
No dedup and want phys mem massaging?

- Enter **Drammer** [CCS ‘16]:
  - Deterministic RH root exploit on ARM/Android
  - User-level DMA mem to flip bits and massage mem
- Physical memory massaging approach:

Mitigations

No dedup and want phys mem massaging?

- Enter **Drammer** [CCS '16]:
  - Deterministic RH root exploit on ARM/Android
  - User-level DMA mem to flip bits and massage mem
- Physical memory massaging approach:



Mitigations

No dedup and want to leak pointers?

- Enter **AnC** [NDSS ‘17]:
  - Leak heap and code pointers from JS in seconds
  - Cache attack on the MMU

- 4 CVEs (Intel, AMD, ARM)
- Tested on 22 microarchs
- Apple: WebKit mitigation

- https://vusec.net/projects/anc

Mitigations

No Rowhammer?

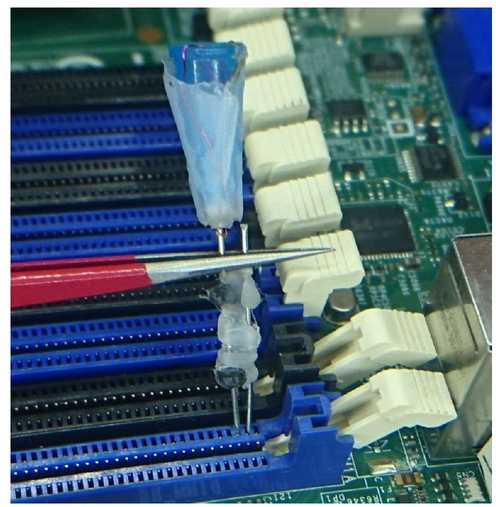
- Not so fast
  - Rowhammer exploits fundamental DRAM properties
- Originally on x86, we found flips on ARM
  - See *Drammer* paper
- Discovered on DDR3, still there on DDR4
  - Despite targeted hardware countermeasures

...What About ECC?

Mitigations

ECC memory?

- Enter **ECCploit** [S&P '19]:
  - Reversed commodity (SECDED-like) ECC functions
  - First reliable ECC-aware Rowhammer exploit
- Approach:
  - Side-channel corrections
  - Induce individual bit flips via Rowhammer
  - Combine flips to bypass reversed ECC functions



- https://vusec.net/projects/eccploit

Mitigations

No dedup and no Rowhammer?

- Other primitives will come along (e.g., AnC)

Expect:

- More hw/sw properties you didn't know about
- More **side channels** (Meltdown, Spectre…)
- More **hardware glitches** (SSD bit flips…)
- A **radical change** in the way we think about sys security and "reasonable" threat models

Mitigations

Recipe moving forward:

- More “open” hardware
- Systems-level solutions (hw, sw, hw+sw)

Some examples:

- VUSion [SOSP’17]
  - Secure, constant-time memory deduplication
- ZebRAM [OSDI’18]
  - OS design for comprehensive Rowhammer protection

Conclusion

- Software security is getting better
- But hw and sw are getting extremely complex
- Potentially huge unexplored attack surface
- Attackers can subvert even “perfect” software
- Need a deep understanding of hw/sw stack

VUSec

https://vusec.net