3 things that Rowhammer taught me

… and their implications for future security research

Who am I?

- Reverse engineer since 1997, vuln development since 1999
- Started reverse-engineering company “zynamics” in 2004
- BinDiff, BinNavi, VxClass, many other contributions
- Acquired by Google in 2011
- Worked on defending Google & large-scale malware analysis until 2015
- Mathematician by trade, hacker by mentality (like to work on boundary between theory and practice)

- Currently on a one-year sabbatical to travel, read, and surf

What is Rowhammer?

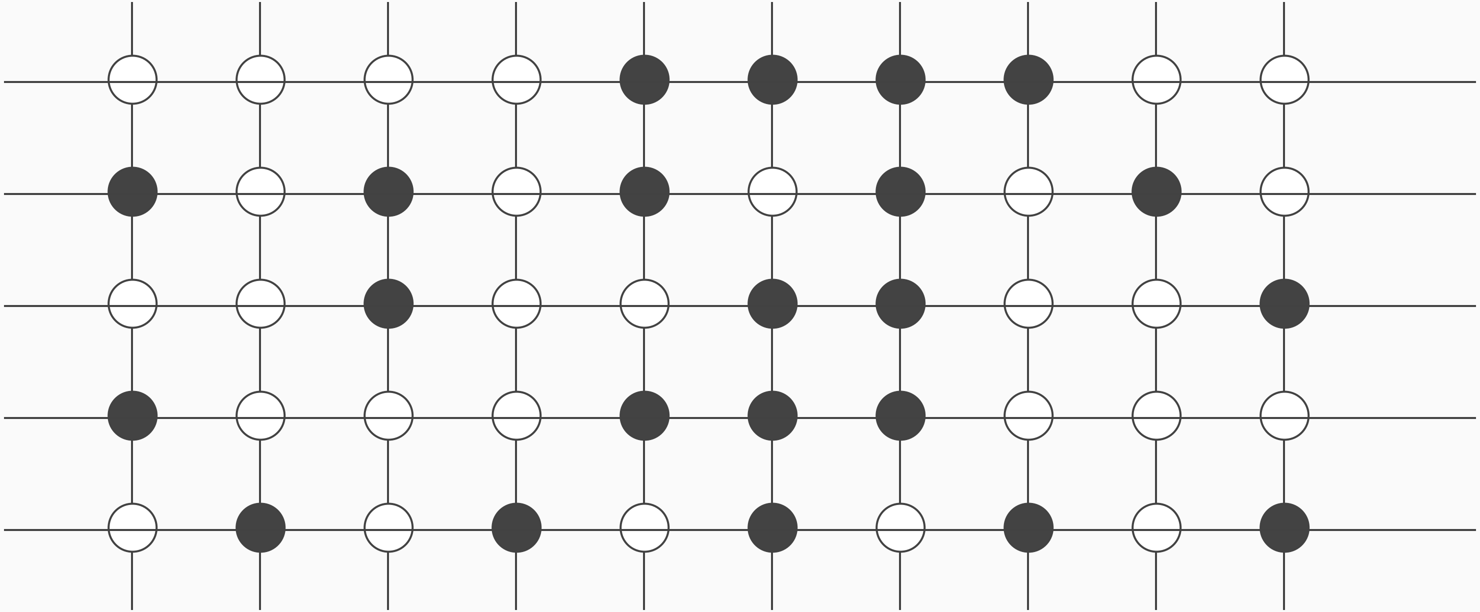
The Rowhammer DRAM bug

Repeated row activations can cause bit flips in adjacent rows

- A fault in many DRAM (DDR3) modules, from 2010 onwards
- Bypasses memory protection: One process can affect others
- All three big DRAM manufacturers shipped memory with this problem
    - A whole generation of machines
    - Particularly bad on Laptops (no ECC)

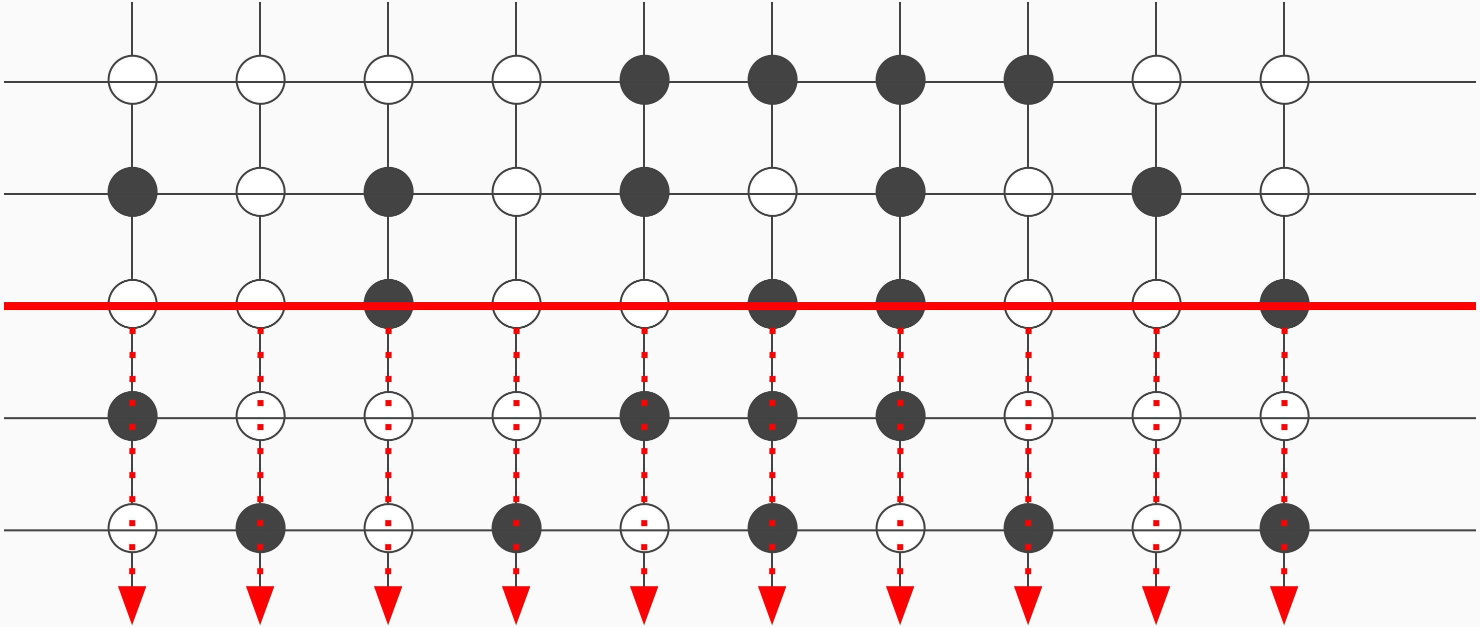
Simplified illustration of Rowhammer issue (hypothesized)

Imagine DRAM as grid of wires, with bits of information stored at the “crossings” as charges



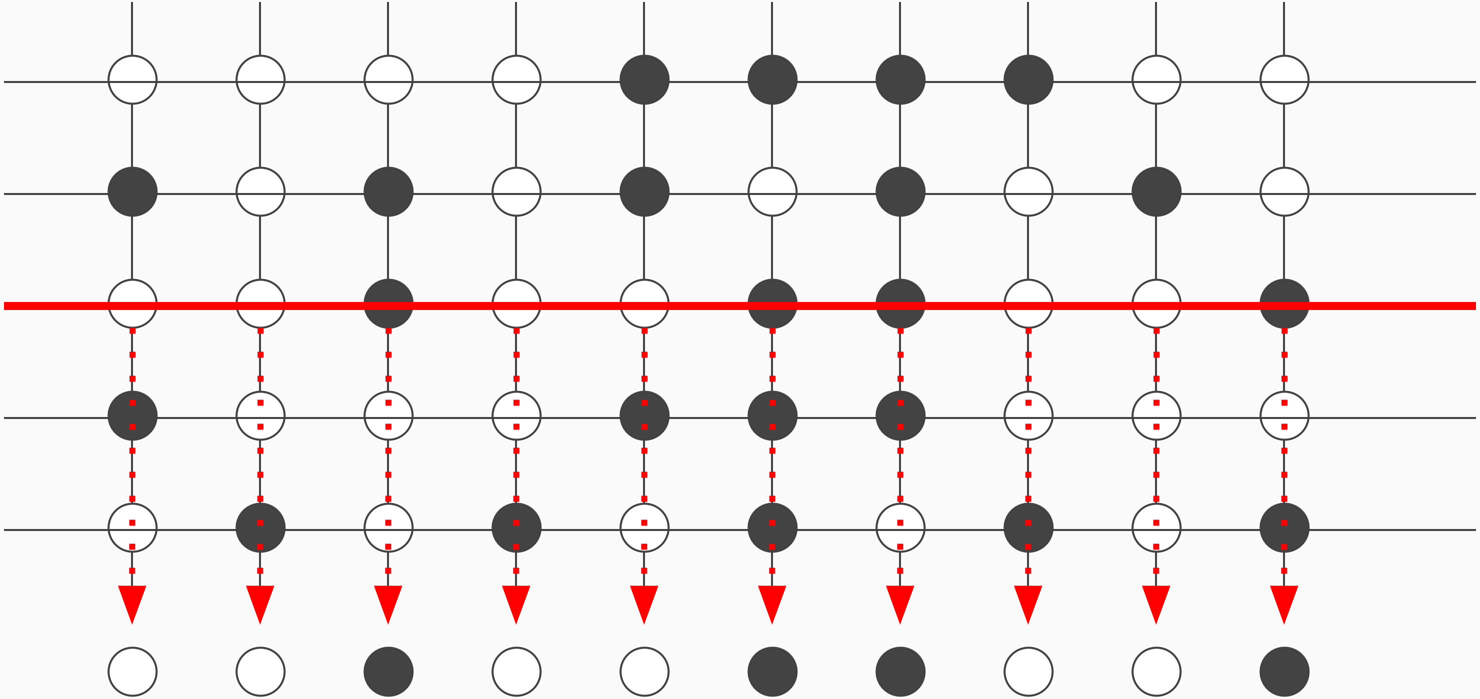
Simplified illustration of Rowhammer issue (hypothesized)

Apply current to a “row” so that the values can be read out



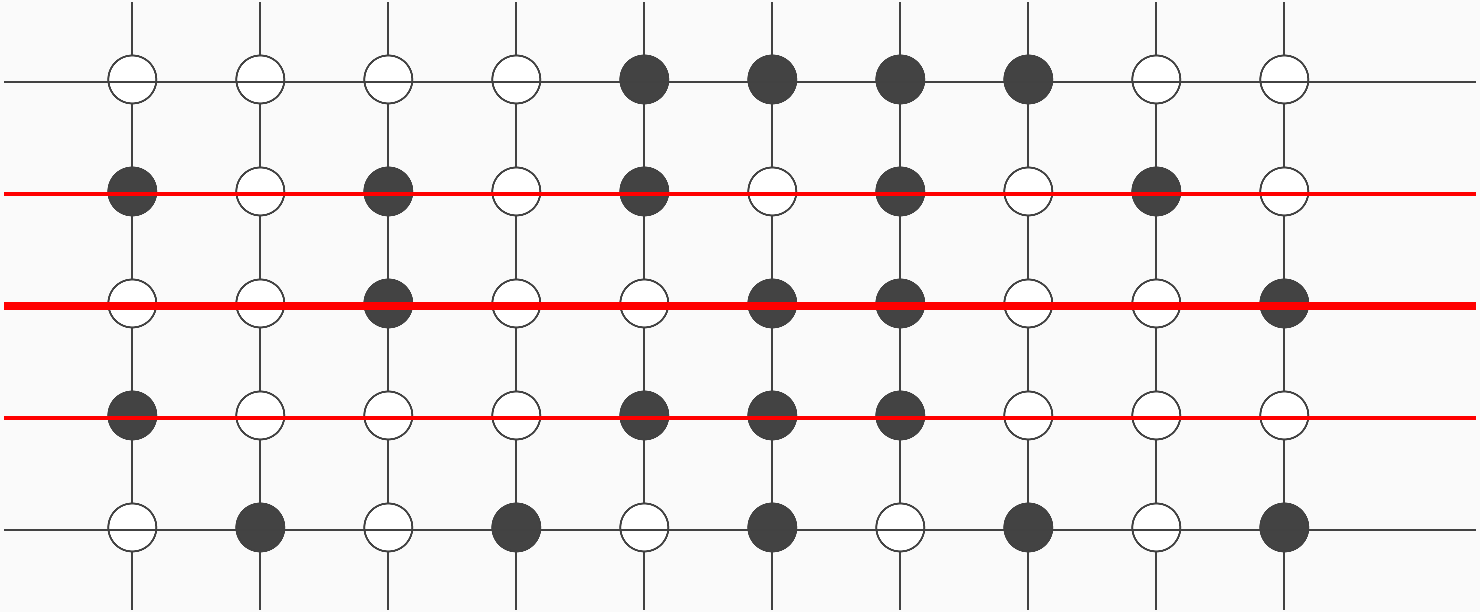
Simplified illustration of Rowhammer issue (hypothesized)

Apply current to a “row” so that the values can be read out



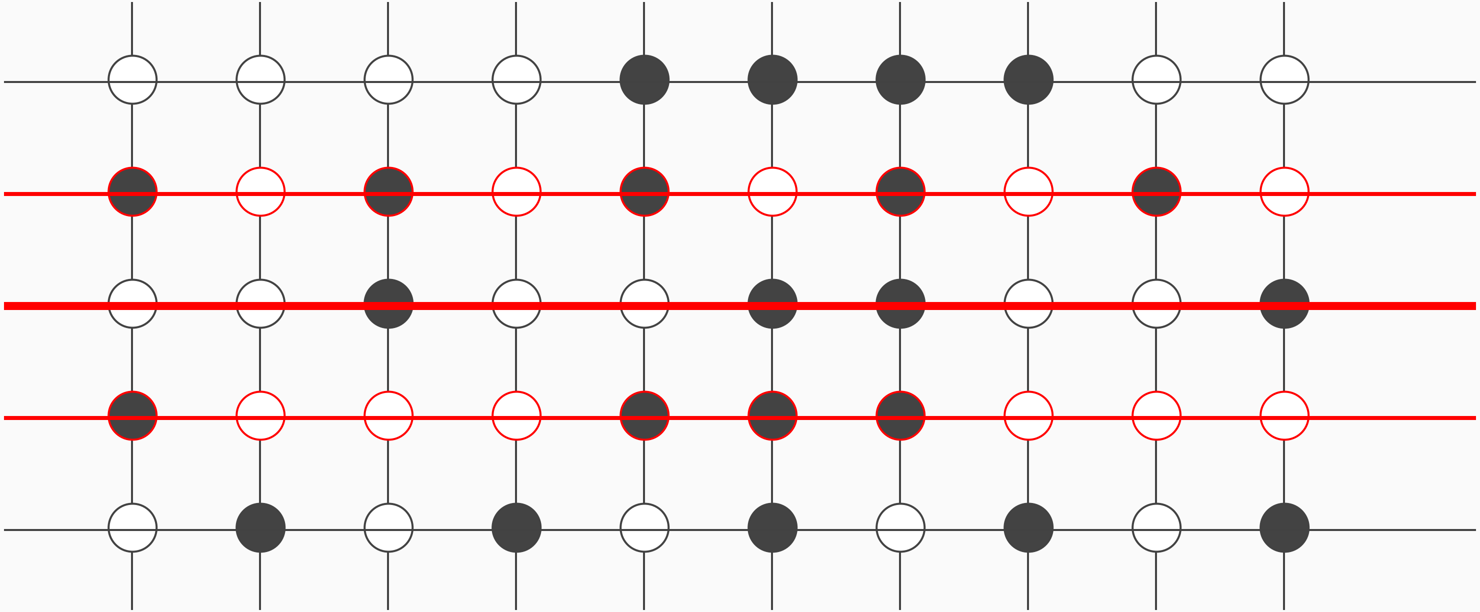
Simplified illustration of Rowhammer issue (hypothesized)

Due to DRAM density, a tiny bit of charge leaks to neighboring “rows”



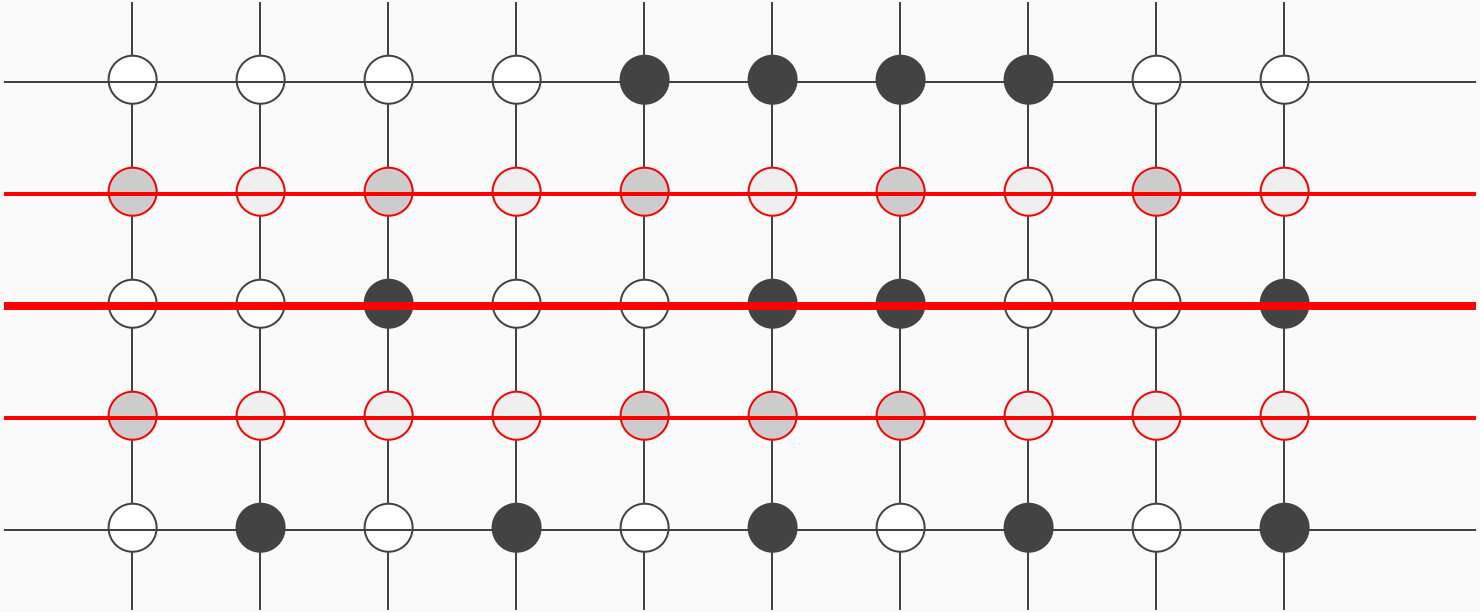
Simplified illustration of Rowhammer issue (hypothesized)

This leads to tiny bits of charge leaking out of the neighboring cells



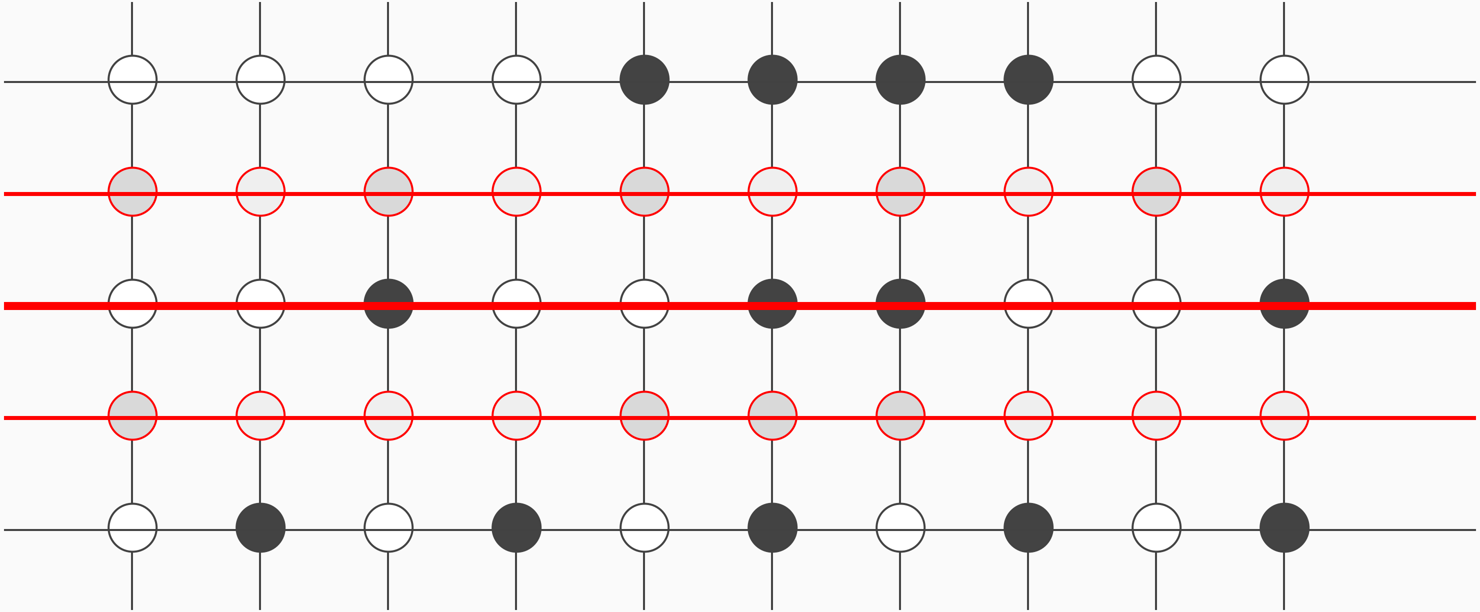
Simplified illustration of Rowhammer issue (hypothesized)

Repeat often enough, and the neighboring bits can “flip”



Simplified illustration of Rowhammer issue (hypothesized)

Repeat often enough, and the neighboring bits can “flip”



Exploiting Rowhammer

Rowhammer was deemed *non-exploitable* and only a *reliability issue*

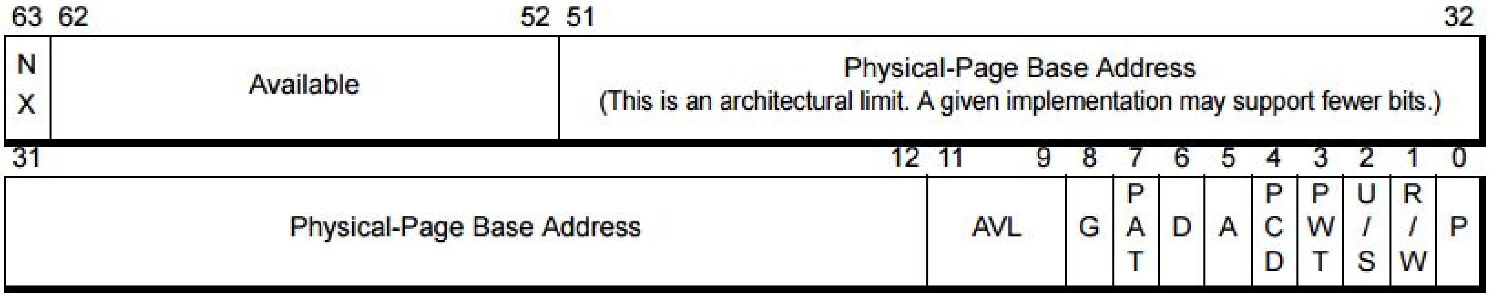
- We found a way to abuse it generically for local privilege escalation on x86
- Idea works on all major OSes (Windows / Linux / OSX)
- We implemented only the Linux variant

(“We” == Mark Seaborn and me)

Exploiting Rowhammer: How to exploit a random bit flip

How can a random bitflip be exploited?

- Page tables are 4k pages containing arrays of 512 “PTE”s (page table entries
- Each PTE is 64 bit and looks like this:



63	62 … 52	51 … 32
NX	Available	Physical-Page Base Address (This is an architectural limit. A given implementation may support fewer bits.)

31 … 12	11 … 9	8	7	6	5	4	3	2	1	0
Physical-Page Base Address	AVL	G	PAT	D	A	PCD	PWT	U/S	R/W	P

Figure 5-21. 4-Kbyte PTE—Long Mode

- Permission bit - 2% chance
- Physical page base address (20 bits) - 31% chance

Exploiting Rowhammer: How to exploit a random bit flip

Basic strategy: Spray most of physical memory with page tables

- `mmap()` the same file repeatedly (and writeable)
- e.g. To fill 4GB of physical memory, `mmap()` 2048GB of virtual address space
  - Works because we have a 64-bit virtual address space
  - Works because major OS have no bounds on page tables

Helps ensure two things:

- That a bit-flipped physical page number will point to a page table
- That a random bit flip will hit a PTE

Exploiting Rowhammer: How to exploit a random bit flip

Begin “hammering” memory

Periodically check for bit flip (indirectly):

- Scan virtual address space for a page that now points elsewhere
- If found, does the page look like a page table?
    - If so, deduce which address it controls
        - Modify it
        - Scan virtual address space for a second page that now points elsewhere

Can be used to exploit *completely random bit flips* (e.g. from cosmic rays)

Lessons:

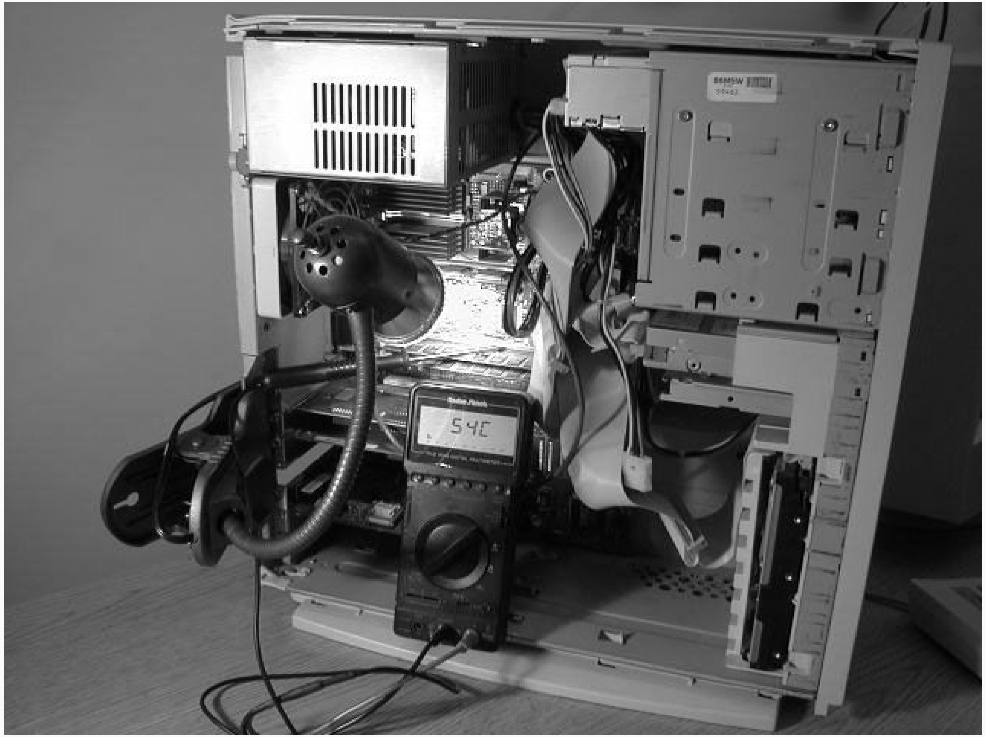
One bitflip is enough

… at least when you don’t need 100% reliability

One bitflip is enough

2003 paper: “Using Memory Errors to Attack a Virtual Machine”

Connected Heat lamp to RAM, induced bit flips



2015 Rowhammer.

Common thread: The most random and obscure issues can be exploited.

One bit is enough.

Deterministic computing is a good abstraction - but PREAM

… “physics rules everything around me”

Deterministic computing is “just” an abstraction

- We think about computers as deterministic machines
- In reality, they are “probabilistically deterministic”
    - Deterministic most of the time
    - Probability of non-determinism in normal operation is made vanishingly low (“reliability”)

- What about the probability of non-determinism in worst-case operation?
    - Hardware is not engineered that way - probability of failure much higher

Deterministic computing is “just” an abstraction

- Example: Flash wear leveling
- Example: Speedpath analysis and yield maximization in CPU design
- Hardware vendors are incentivized to sell chips “just at the border of working”
- Bigger safety margins mean drastically reduced profit margins

Deterministic computing is “just” an abstraction: Cloud

- Cloud computing changes risk calculous

- Even if only 1 in 5000 CPUs can be made to misbehave, Cloud providers are at risk

- Attackers can rent CPUs until they get lucky

“Impenetrable defense” is not a good concept ...

… because the chip at hand may not implement the spec all the time

“Impenetrable defense” is not a good concept

- Realizing that computers are only average-case deterministic means that “secure enclaves” may be a doomed concept

- A secure enclave (even with formally verified software) is verified against specified semantics of a chip

- Your particular chip may only honor those semantics 99.999% of the time

- The 0.001 may be enough for the attacker

“Impenetrable defense” is not a good concept

- Trend in recent years has been toward more opaque blocks in computers
- Intel Management Engine, Trustzone, Intel SGX etc.
- Trending concept is “hard, trusted, inviolable small cores” that are completely opaque and non-inspectable
- Once an attacker manages to get in, there is no way to tell, and no way to get him out

Implications?

The three lessons from Rowhammer imply three interesting areas of research

Implications for fundamental security research

- The three lessons give importance to (at least) three exciting topics for research - one each in:
    - Theoretical computer science
    - Borders between Electrical engineering, Statistics, Physics
    - IT Systems Engineering (Hardware, Software, Ecosystems)

Theoretical Computer Science

A proper theory of exploitation

A proper theory of exploitation

- Theoretical understanding of exploitation is weak
- Misjudgements happen even by seasoned professionals ("a random bit flip will not be exploitable")

- No good theoretical framework exists to analyze
  - ... what issues are "exploitable"
  - … what issues are "not exploitable"
  - … where the boundary between them lies
  - … what facilitates exploitability

- Result: Misjudgements, bogus "mitigations", etc.

A proper theory of exploitation: Suggestion

- Assume a tiny toy CPU
  - Finite array of memory cells, a few registers
  - PEEK, POKE, ADD, Jcc, RECV, SEND instructions
  - Read/write memory, add values, recv value, send value
  - Finite (but large) sequence of attacker-provided inputs
- Programmer “emulates” a finite state machine on this CPU - with a formally describable set of valid states
- Attacker gets to choose an arbitrary valid state as starting point

- Now assume corruption of a random bit - what is the probability that the attacker can now reach **any** possible state of the toy CPU given large enough input sequence?

A proper theory of exploitation: Suggestion

- A proper theory of exploitation would help understand many things:
  - Distinguish useful from not-useful mitigations
  - Help identify at what level of complexity of the emulated finite-state machine single-bit corruptions become exploitable
  - Put “exploitation” on sound footing in academic circles, and bring it out of obscurity / “magic” territory

I hope to eventually get around to working on this

Electrical Engineering, Statistics, Physics

Worst-case analysis of chips, wear-and-tear, “achieving nondeterminism”

Speed binning, worst-case paths, malicious input

- With cloud computing, virtualization, sandboxing and ubiquitous JIT, the instruction stream of the CPU is now malicious input

- CPUs are “speed binned” at end of production cycle
- IC manufacturers will optimize for maximum yield without “random” failures (speed path testing, crosstalk, leakage)

- What about “malicious inputs” (input code designed to cause pessimal behavior in the IC) ?

Speed binning, worst-case paths, malicious input

EE community performs research on speed- and voltage-binning all the time - but with reliability focus.

Security Research is needed on ...

- Identifying critical speed paths on silicon, identifying critical areas for crosstalk on silicon
- How to identify input that exercises worst-possible paths
- Can the critical path be made more critical by intentionally causing transistor degradation? Transistor aging is a thing, and solid-state is not always solid.

It may become economical to rent 1000 CPUs for 1 year if I can get one of them to degrade enough to compromise the cloud provider.

Speed binning, worst-case paths, malicious input

- This looks like an extremely fruitful area of research!
  - IC manufacturers can’t afford to test every chip for very long.
  - Attackers can profit from small fractions of CPUs being faulty - even if those faults occur very rarely
  - IC manufacturers can’t be arbitrarily careful (direct impact on profitability).

Research will need collaboration between security experts and people familiar with VLSI design, process-induced variations in IC production, and yield-optimization.

Research may require budget (hardware investigation is expensive)

Would love to investigate this, but need an EE collaborator :-)

Speed binning, worst-case paths, malicious input

Side warning: I expect the IC industry to aggressively lobby against this research being performed out in the open.

It is an incredibly competitive and secretive industry that adheres to the motto “only the paranoid survive”.

Don’t go there unless you are willing to pick a fight.

“IT systems engineering” (Hardware, Software, Ecosystem)

Building inspectable (and hence defendable) systems

Building inspectable (and hence defendable) systems

- The real world has the concept of “ownership” and “possession”
- I may be in possession of a rental car, but the car company has “ownership”.

- IT systems have a third dimension: “Control”
- I can have ownership and possession of an IT system, but I may not have control over it

In today’s systems, **it is impossible to establish** who is in control. Hardware enclaves try to prevent loss-of-control, but this cannot be 100% -- remember PREAM.

Building inspectable (and hence defendable) systems

- If we want defendable systems, we need to build systems where it is possible for the person with ownership and possession to establish control.
- This implies:
    - Need to engineer systems to provide non-updateable and tamper-evident hardware paths to calculate & display checksums over code
    - Need to engineer systems so that every place that may contain code can be inspected
    - Need to engineer & build a public ledger infrastructure (similar to Certificate Transparency, but with working Gossip protocols) for signed code

Building inspectable (and hence defendable) systems

- The current approach of allowing (and even furthering) opacity makes our problems worse, not better

- In a world where all signing keys can be either stolen or coerced, any code signature scheme without publicly inspectable ledger needs to be rejected

- Possibility must exist to determine origin of all code within a machine

Building inspectable (and hence defendable) systems

- If you can’t establish who is in control of a machine, all security discussions devolve into scholasticism

Summary :-)

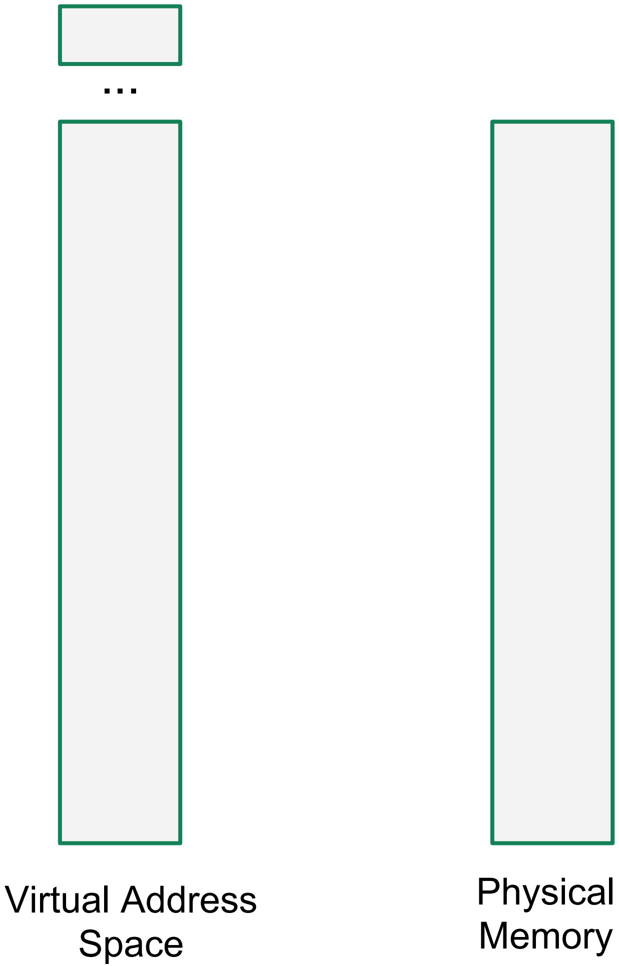
Summary :-)

- Rowhammer is an interesting issue, but it is the implications that are most interesting
- Computer security is full of extremely hard and extremely interesting questions
- Collaboration across disciplines (EE, Theoretical CS, product design) will be crucial

- If you find any of these topics interesting, please contact me at thomas.dullien (at) gmail.com
- I am particularly happy about any information from people with EE/VLSI/Chip-design/yield-optimization background

Questions?

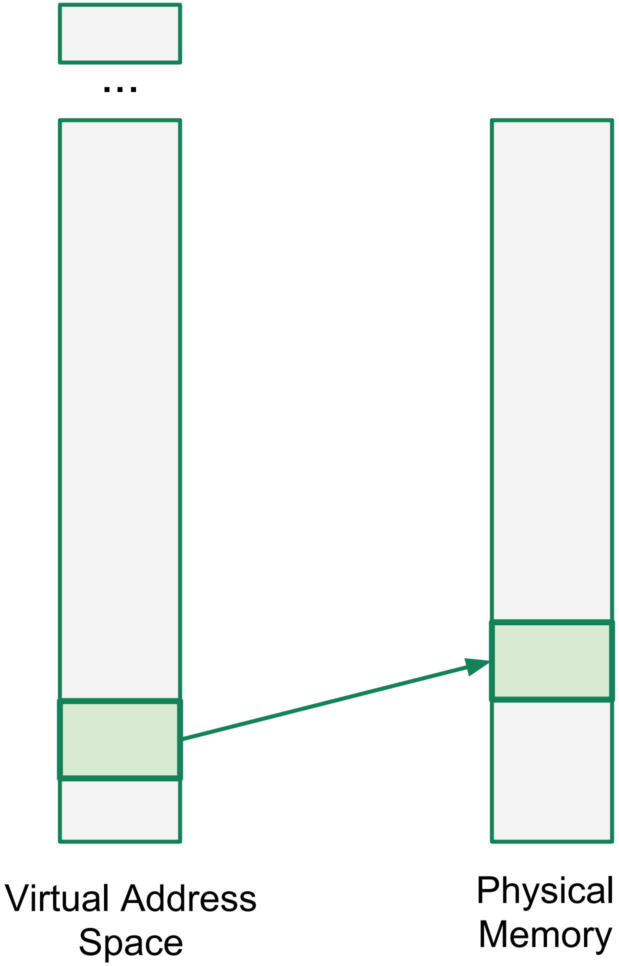
Appendix: Illustration of Rowhammer PTE attack



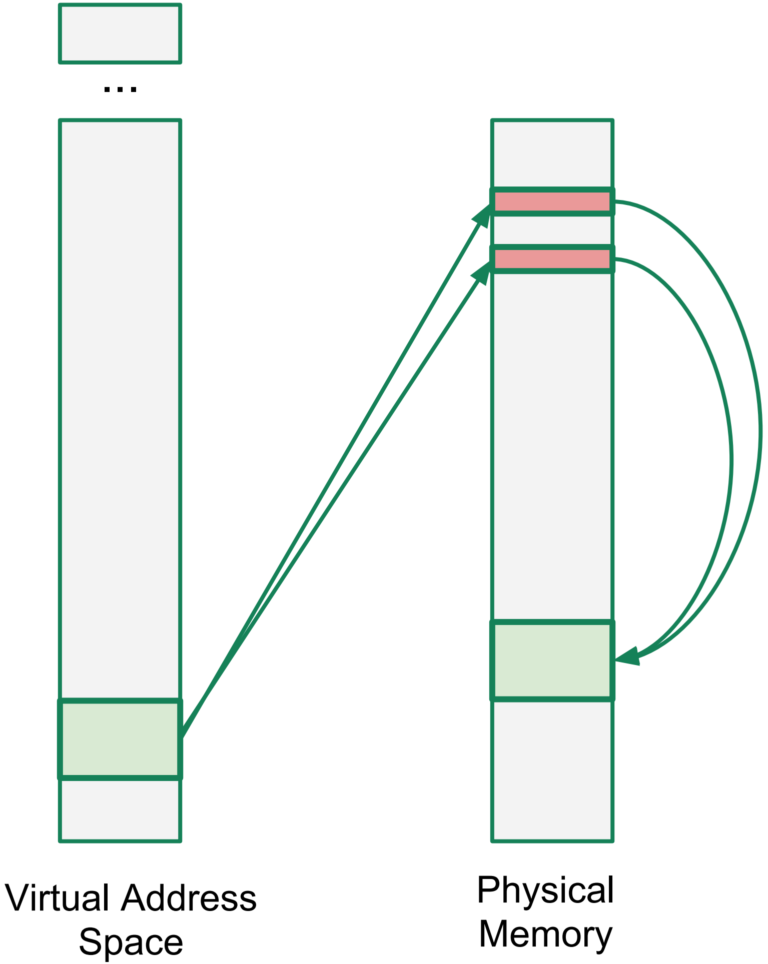
...

Virtual Address Space

Physical Memory

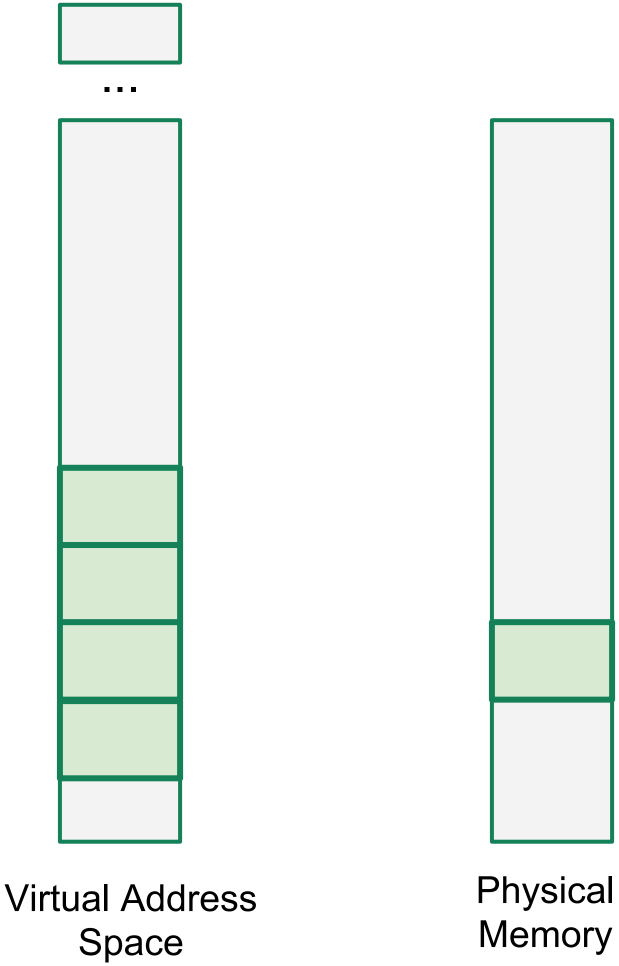


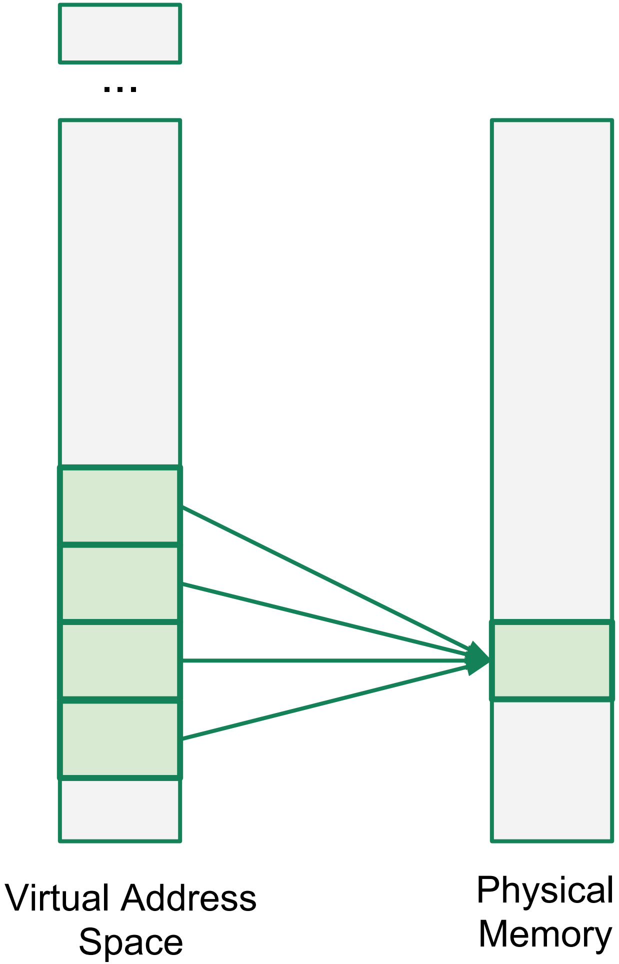
What happens when we map a file with read-write permissions?



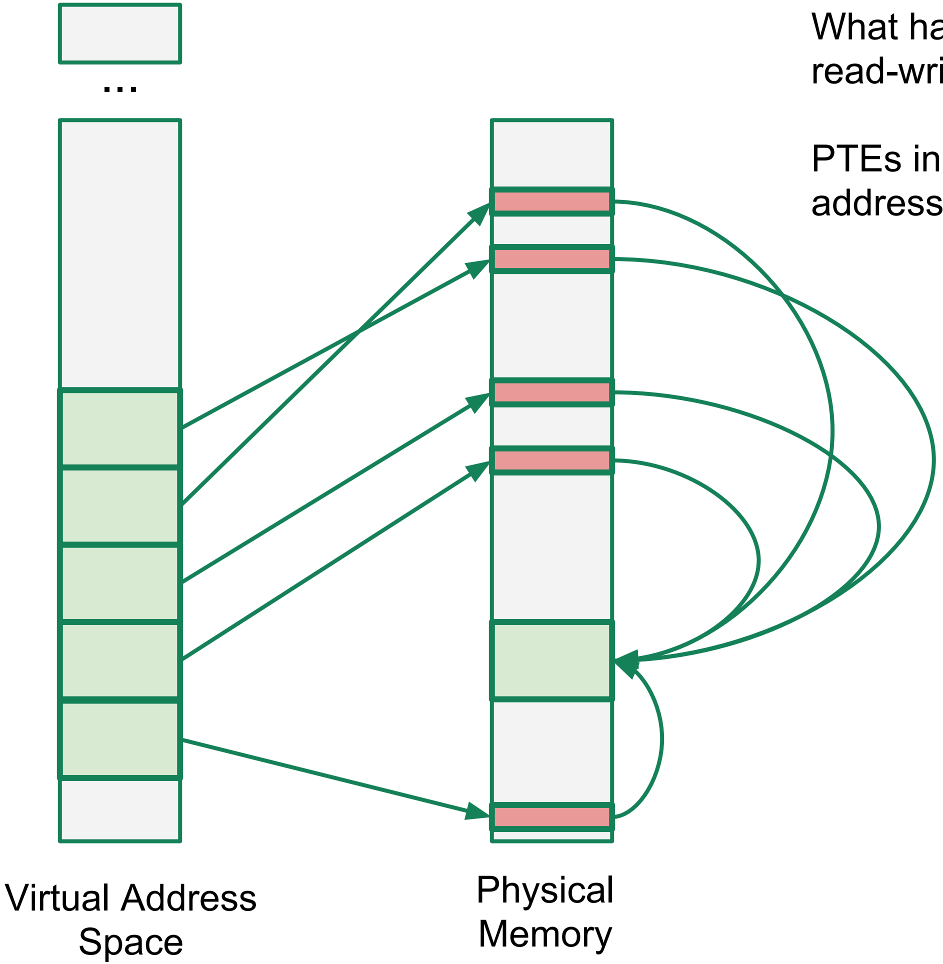
What happens when we map a file with read-write permissions? Indirection via page tables.

What happens when we repeatedly map a file with read-write permissions?



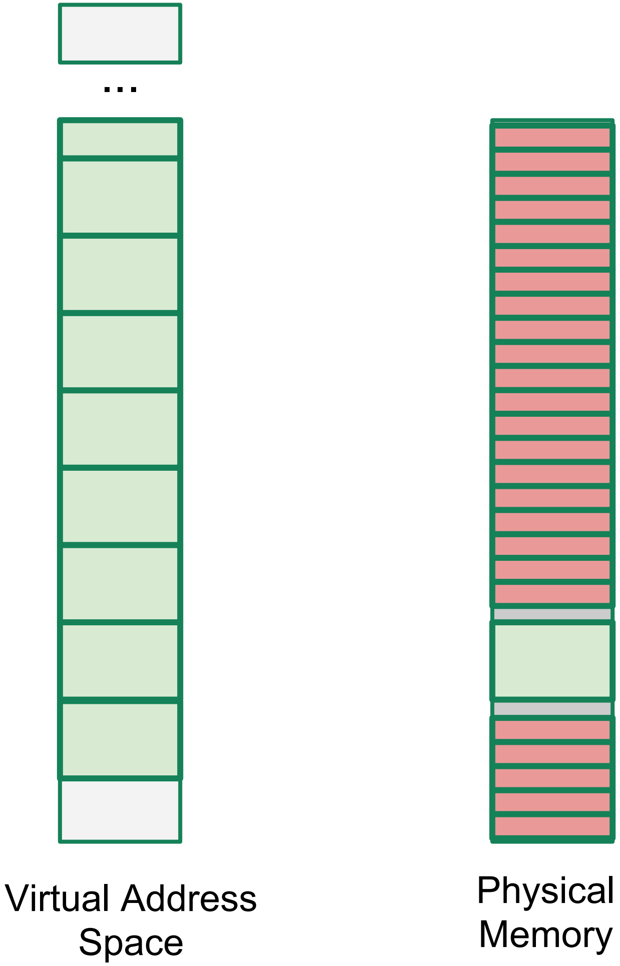


What happens when we repeatedly map a file with read-write permissions?



What happens when we repeatedly map a file with read-write permissions?

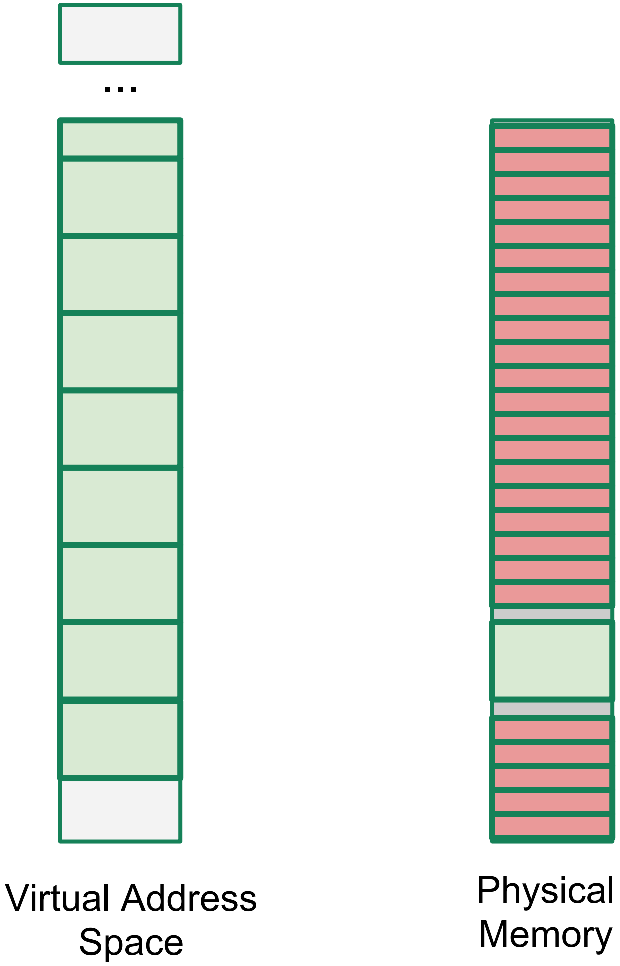
PTEs in physical memory help resolve virtual addresses to physical pages.



What happens when we repeatedly map a file with read-write permissions?

PTEs in physical memory help resolve virtual addresses to physical pages.

We can fill physical memory with PTEs.

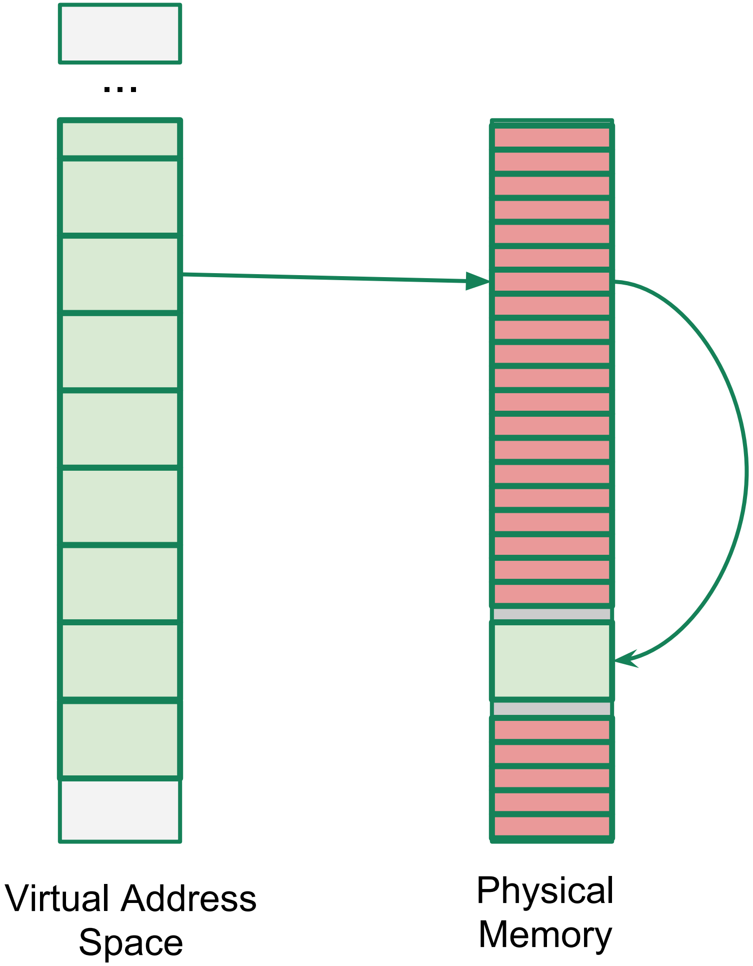


What happens when we repeatedly map a file with read-write permissions?

PTEs in physical memory help resolve virtual addresses to physical pages.

We can fill physical memory with PTEs.

Each of them points to pages in the same physical file mapping.



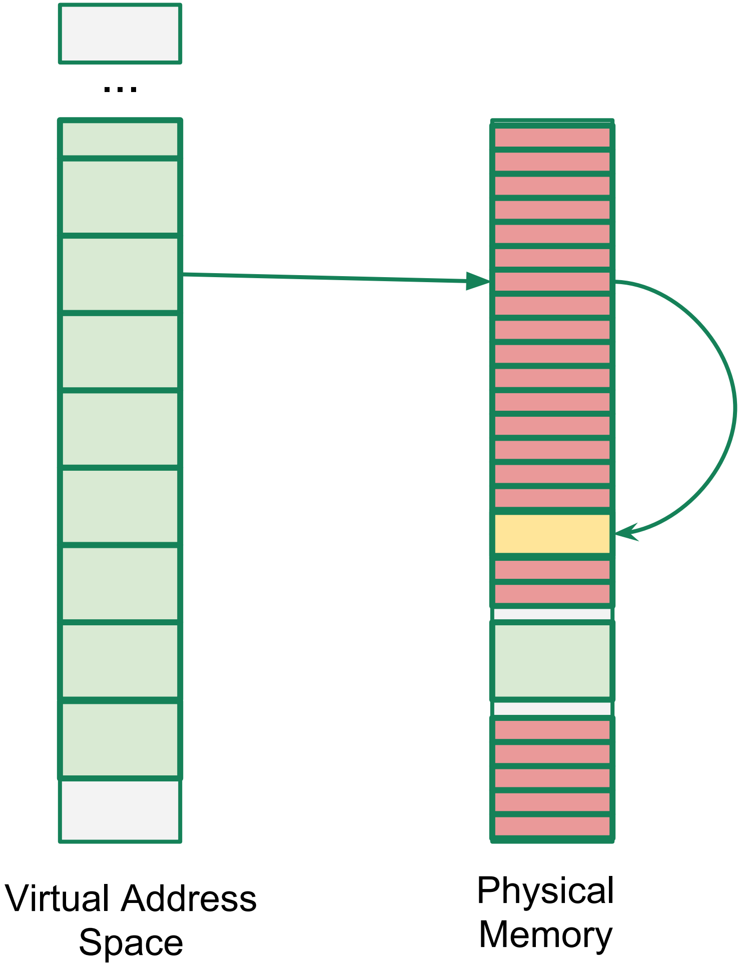
What happens when we repeatedly map a file with read-write permissions?

PTEs in physical memory help resolve virtual addresses to physical pages.

We can fill physical memory with PTEs.

Each of them points to pages in the same physical file mapping.

If a bit in the right place in the PTE flips ...



What happens when we repeatedly map a file with read-write permissions?

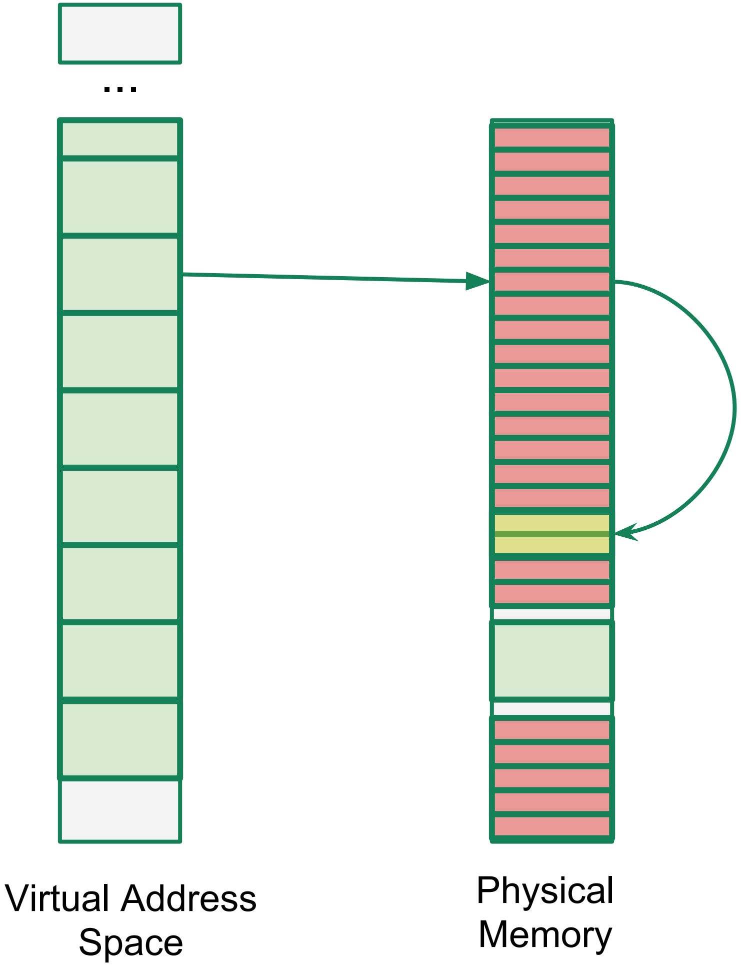
PTEs in physical memory help resolve virtual addresses to physical pages.

We can fill physical memory with PTEs.

Each of them points to pages in the same physical file mapping.

If a bit in the right place in the PTE flips …

… the corresponding virtual address now points to a wrong physical page - with RW access.



What happens when we repeatedly map a file with read-write permissions?

PTEs in physical memory help resolve virtual addresses to physical pages.

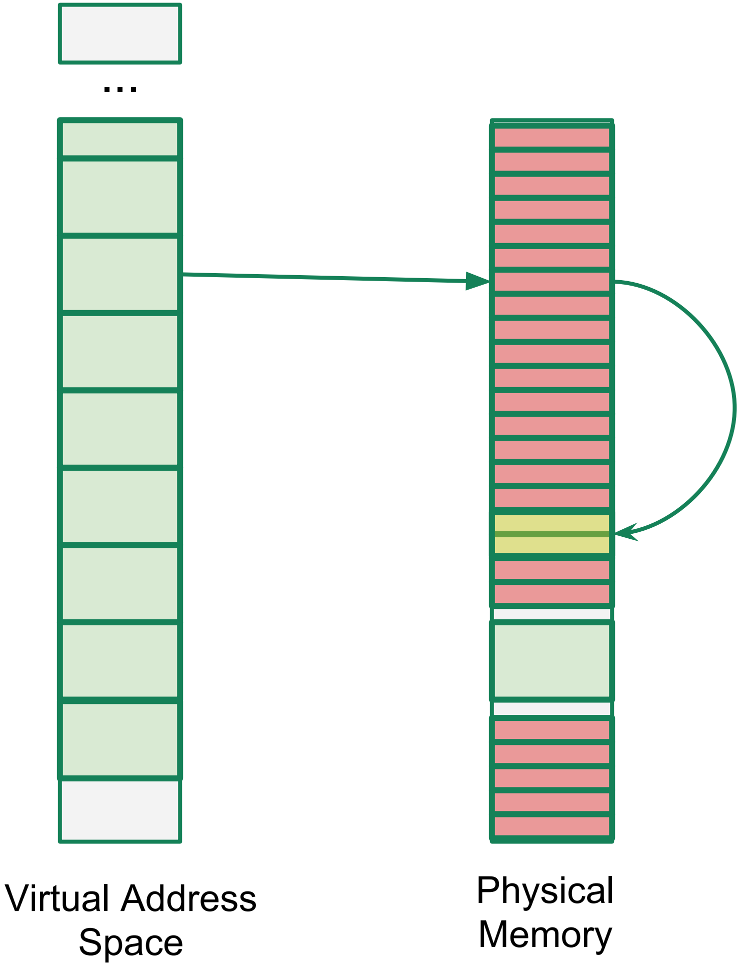
We can fill physical memory with PTEs.

Each of them points to pages in the same physical file mapping.

If a bit in the right place in the PTE flips …

… the corresponding virtual address now points to a wrong physical page - with RW access.

Chances are this wrong page contains a page table itself.



What happens when we repeatedly map a file with read-write permissions?

PTEs in physical memory help resolve virtual addresses to physical pages.

We can fill physical memory with PTEs.

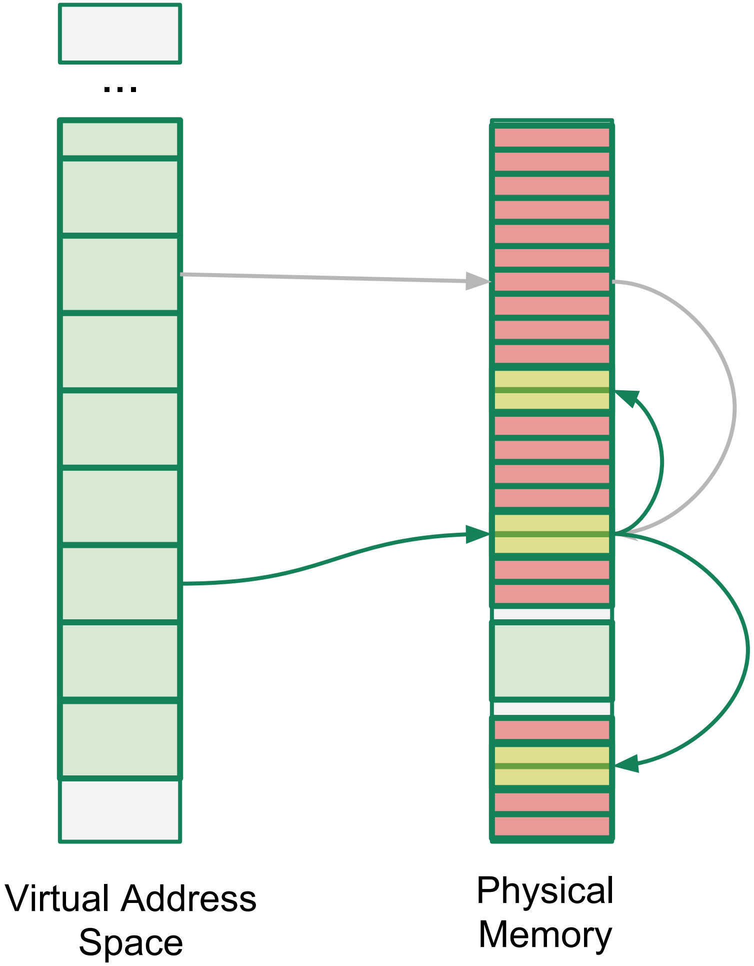
Each of them points to pages in the same physical file mapping.

If a bit in the right place in the PTE flips …

… the corresponding virtual address now points to a wrong physical page - with RW access.

Chances are this wrong page contains a page table itself.

An attacker that can read / write page tables …



What happens when we repeatedly map a file with read-write permissions?

PTEs in physical memory help resolve virtual addresses to physical pages.

We can fill physical memory with PTEs.

Each of them points to pages in the same physical file mapping.

If a bit in the right place in the PTE flips …

… the corresponding virtual address now points to a wrong physical page - with RW access.

Chances are this wrong page contains a page table itself.

An attacker that can read / write page tables can use that to map **any** memory read-write.