Confidential Computing
Hype, Future and Reality

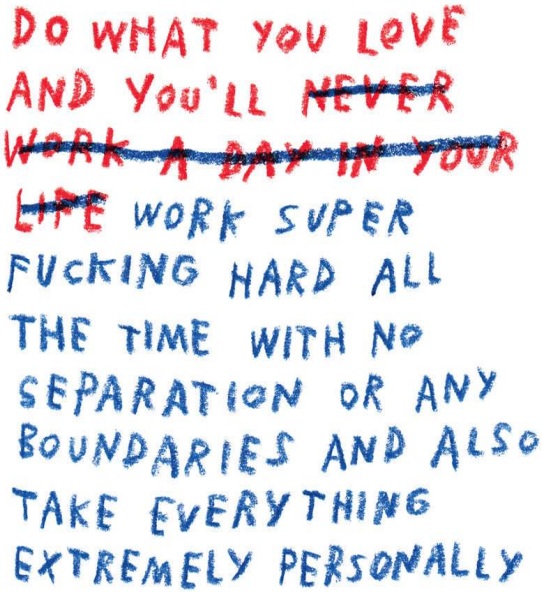
Rodrigo Branco (BSDaemon)
rodrigo @ kernelhacking.com

“If we suppress all discussion, all criticism, proclaiming ‘This is the answer, my friends; man is saved!’ we will doom humanity for a long time to the chains of authority, confined to the limits of our present imagination. It has been done so many times before”.

Richard Phillips Feynman

Disclaimers (1/2)

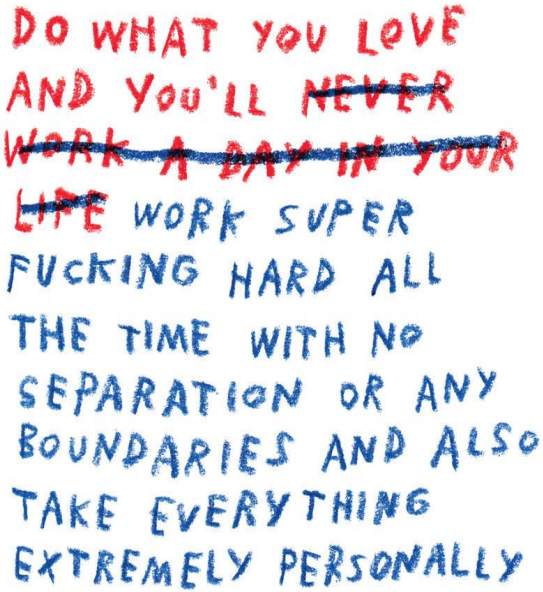
- I’m not speaking for my employer, nor representing past employers
- I believe in technology and the need to developing it
- My main ethical considerations on the specific subject are about readiness and transparency
- Nothing here is intended as personal attacks, therefore I omitted names



Adam J. Kurtz

Disclaimers (2/2)

- I use tons of generalizations knowing very well generalizations are risky and unfair to many - In order to be fast, I hope one alert about it could cover all cases during the talk (yes, I am aware that is not how communication works)
- During the talk, I praise companies, researchers and technologies because I believe in them, not because of any personal benefit



whoami (1/4)

- I’m best described as a failed farmer
- Writing exploits and finding vulnerabilities for **26 years** now
  - Started with CPU bugs by accident, while researching SMM (2007) - **so 15+ years**
  - Worked for Intel, AWS, Google, L3 Harris Trenchant doing CPU/Platform/OS Security (**12+ years**)
    - Tons of horror stories, lots of learnings and lots of great people (my opinions are forged from those experiences and interactions)
    - Jan 2019: https://www.wired.com/story/intel-meltdown-spectre-storm/

whoami (2/4)

- Pertinent to this talk, I was involved in the following (publicly acknowledged) low-level issues (and hundreds more that were silently fixed)
  - **CVE-2023-1998, CVE-2023-00045, CVE-2020-12965, CVE-2020-0543**, CVE-2019-0185, CVE-2019-0155, CVE-2019-14590, CVE-2019-14591, CVE-2019-11089, CVE-2019-11113, CVE-2019-0151, CVE-2019-0152, CVE-2019-0117, CVE-2019-0184, CVE-2019-0155, CVE-2019-14590, CVE-2019-14591, CVE-2019-11089, CVE-2019-11113, **CVE-2018-3693, CVE-2018-12126, CVE-2018-12130, CVE-2018-12127, CVE-2019-11091**, CVE-2019-0115, CVE-2018-12209, CVE-2018-12210, CVE-2018-12211, CVE-2018-12212, CVE-2018-12213, CVE-2018-12214, CVE-2018-12215, CVE-2018-12216, CVE-2018-12217, CVE-2018-3626, CVE-2018-5736, **CVE-2019-0162, CVE-2018-3615, CVE-2018-3620, CVE-2018-3646, CVE-2018-3665, CVE-2018-3639, CVE-2018-3640, CVE-2017-5753, CVE-2017-5754, CVE-2017-5715**

- **20+ patents (covering CFI, side-channels, encrypted memory, etc) that impacted in major Intel features**

whoami (3/4)

THAT DOES NOT MEAN MUCH

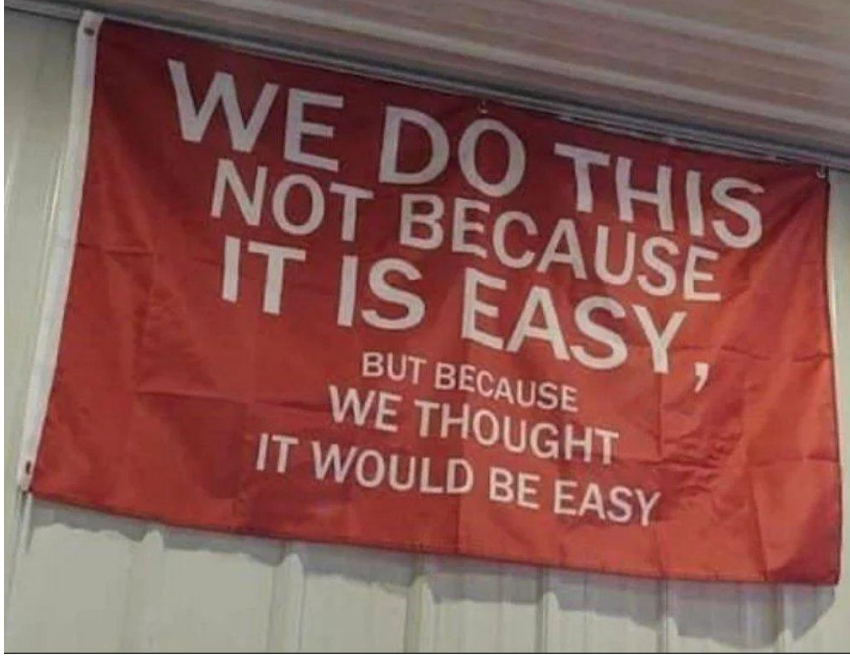
LISTEN TO WHAT I HAVE TO SAY, EVALUATE, CHALLENGE, COME TO YOUR OWN CONCLUSIONS!

MY TIP THOUGH: LOTS OF NUANCES AND REFERENCES FROM BETTER SOURCES, USE THAT AND GO DEEPER!

whoami (4/4)

(This talk is not about the farming failure!)

Objectives



- Have a frank conversation about

Confidential Computing with peers

- Focus on the technical aspects, besides

the unavoidable ‘rant’

- Maybe share some mental framework

for what Confidential Computing is about and what it is not about

- Discuss *current* technological *practical* limitations (some of those are

practical limitations and not theoretical because there are no business ROI even though a solution might exist)

- While this talk has emphasis on Confidential Computing most common usages,

I will do my best to mention general impact outside of confidential computing and outside of the cloud - say, for embedded devices and space

My Take on Security

- Two main challenges in security
  - Assumptions
  - Composition

- I believe it's a rule rather than an exception that the potential for vulnerabilities (and therefore, exploits) is already present at the design stage, and as so, can be anticipated at that stage

- While the details matter for a given exploit to be created, a lot of patterns exist across systems and therefore can be abstracted/generalized
  - That essentially mean that the root cause is not in the specific details, but in a more general aspect of the decisions behind the implementation

What is **Confidential Computing?**

- A marketing term, that is loosely defined **on purpose**

- I rather discuss what does Confidential Computing imply for customers, end-users and overall non-experts?

- AWS has a nice take on Confidential Computing that is beyond jargons

PERCEPTION IS REALITY

I hope we understand each other…

“Now, I have a way of not remembering things when I do something dumb or annoying to people, so I forget what I said that put him out. Whatever it was, I thought I was joking, so I was very surprised by his reaction. I had undoubtedly said some boorish, brash, damn-fool thing, which I therefore can’t remember!”

Richard Phillips Feynman

Can we *NOT* trust our Cloud Provider?

- A big take on Confidential Computing is that the trust is divided, between the cloud provider and the HW manufacturer… **is it?**

- **Your cloud provider is your HW provider… even if we ignore very nuanced bugs [1] [2]**
  - **For example how do you know that there is no memory interposer? [3]**
  - **Or special commands to the memory controller? (none of that is part of ‘attestation’)**

Fragile Knowledge

“I don’t know what’s the matter with people: They don’t learn by understanding; they learn by some other way - by rote, or something. Their knowledge is so fragile!”

…

“So this kind of fragility is, in fact, fairly common, even with more learned people”

Richard Phillips Feynman

But what about the rest of the ‘cloud’ stack?

- Who is providing you the OS images? (is it Microsoft on Azure? are you using Amazon Linux on AWS?)
- Who is providing you migration? instance spawning? event handling? distributed storage and database?
- Where are the keys for remote attestation stored? How do you do the attestation of that system? Or are you using a KMS? Is that KMS in the cloud? Are you using a virtual TPM? If not, how the TPM is even shared between instances?
- Are you instead trying to use multi-cloud? Is the entire SW stack used in trying to do all that multi-cloud a way bigger attack surface than you originally had?

But, memory encryption…

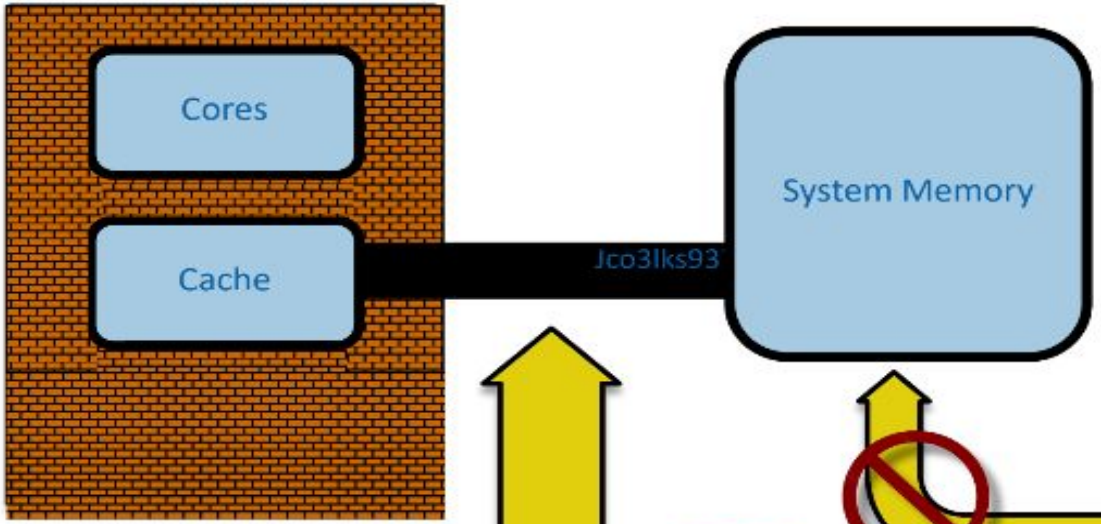
- In fact, what most of folks are calling Confidential Computing is just Memory Encryption (no remote attestation, no worries about build environments, setups, configurations, updates, etc)

- Memory Encryption seems a worth technology, **isn't it?**
  - **It depends, how do you debug instance failures?**
  - **What are you going to use to identify compromises? (now all the years of investment in out-of-instance introspection and the benefits it brings are gone)**
  - **Does Memory Encryption without Authentication (integrity/replay) provide real value? *We will discuss this in a minute***

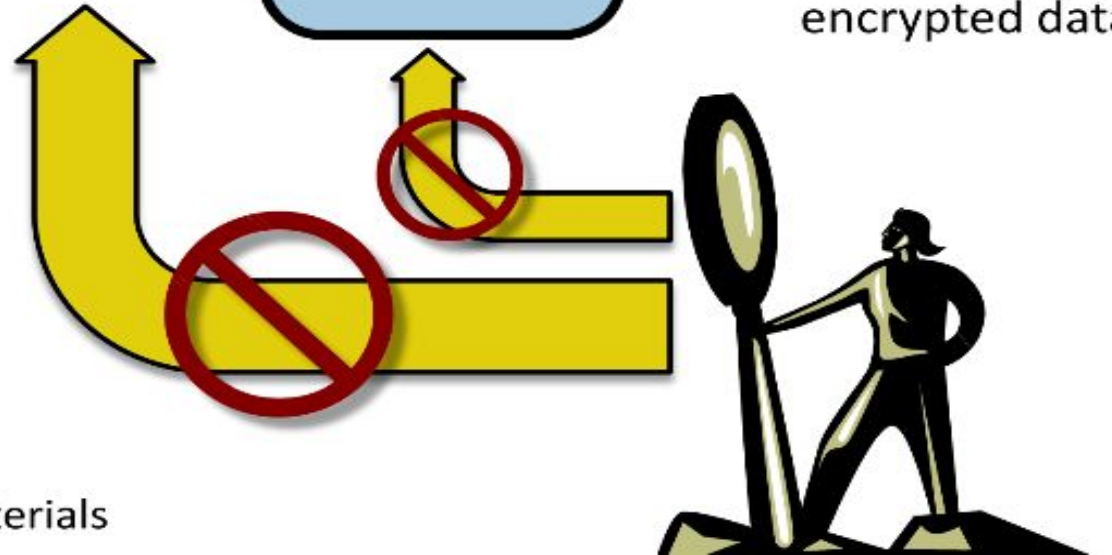
Some Memory Encryption ‘Glossary’

- AMD SME and Intel TME provide a full memory encryption with a single key (usually randomly created by the CPU at boot-time)
- AMD SEV and Intel MKTME essentially provide memory encryption with different keys ‘per-guest’
- Intel SGX (for client) provides memory encryption, integrity and replay protection thru an encryption engine called MEE (Memory Encryption Engine) - it uses a tree (multiple cache lines) rooted in an internal SRAM to the chip as storage of the integrity and replay protection mechanism [1]
  - It has cost and performance implications, even with internal caches
- Intel TDX (server SGX) is falsely advertised as ‘built on the same proven ground as SGX’

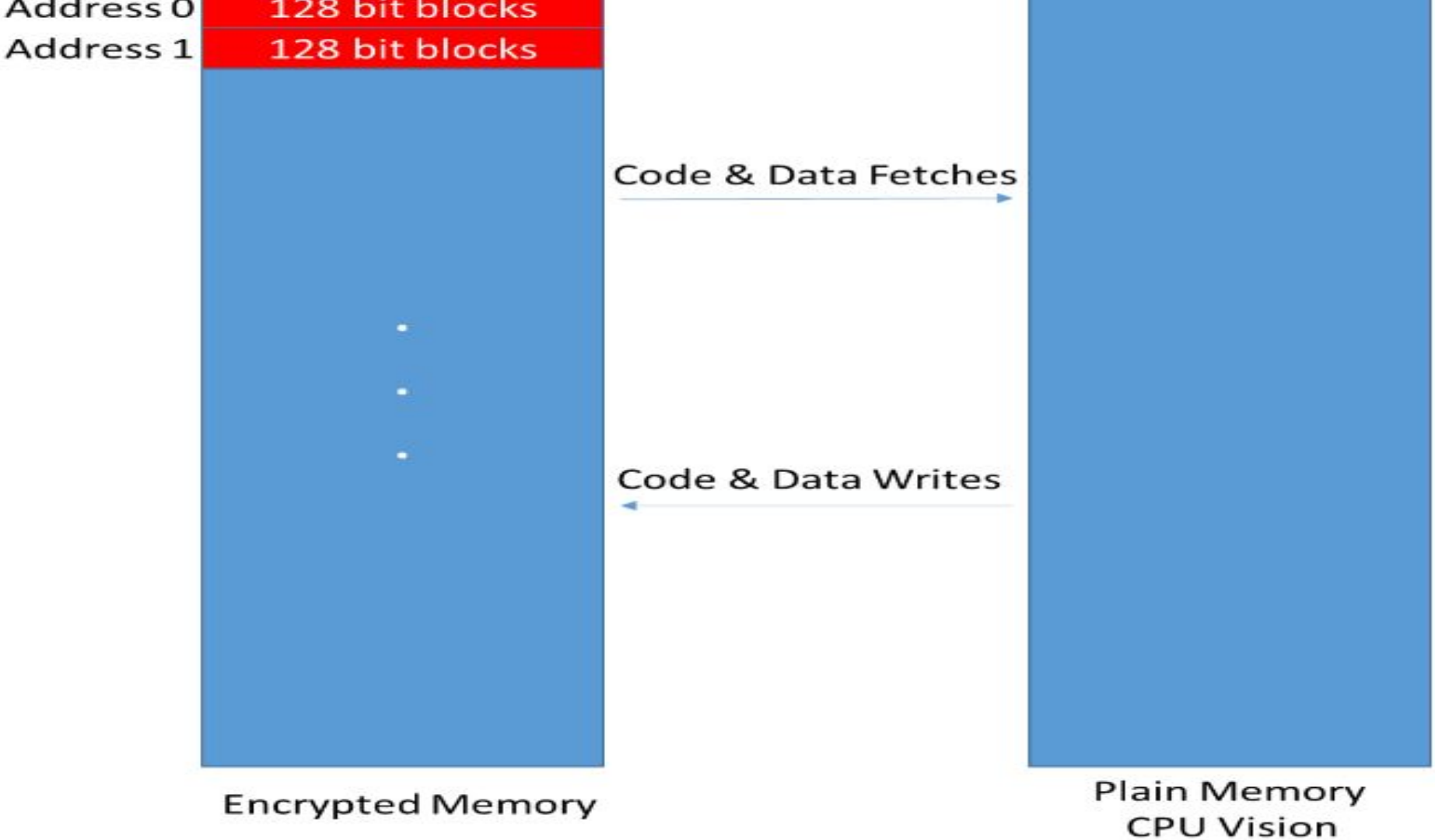
CPU Package



1. Security perimeter is the CPU package boundary
2. Data and code unencrypted inside CPU package
3. Data and code outside CPU package is encrypted
4. External memory reads and bus snoops see only encrypted data



Taken from Intel's SGX materials


Address 0
128 bit blocks
Address 1
128 bit blocks
Code & Data Fetches
Code & Data Writes
Encrypted Memory
Plain Memory
CPU Vision

SGX x TDX: The hard truth (1/3)

- TDX does not use MEE, instead it has a different encryption mechanism (that does not provide integrity/replay protection)
- Server parts use a different MPH, which is a key HW component (likely the biggest HW change) for SGX (which tracks enclave pages, memory accesses, etc for the access control mechanisms)
- Server parts have different decoding logic (which tracks ranges and memory accesses decoding for different parts of the platform)

SGX x TDX: The hard truth (2/3)

- Server parts have a very different debugging and unlocking HW logic (which prevent against a variety of HW attacks with physical presence)
- Server parts support CPU hotplugging and multi-core (so while in Client you can have one key fused in the CPU for the part, in servers each CPU will have their own key and need to negotiate a platform key)
- The below ISA (so microcode and XuCode) for server has server-specific flows (that only execute on server parts), to deal with reliability, hotplugging and many other server-specific capabilities

SGX x TDX: The hard truth (3/3)

- Initialization logic is very different in servers, so are the ACMs and things like mcheck (that guarantees such initialization is in a safe state)
- **So what is really re-used/same? The above-ISA architectural enclaves! That is just SW that uses the actual instructions to implement things like quoting**

Can we trust the CPU manufacturers?

- SGX threat model specifically excluded ‘side-channels’ (we know how well it worked to try to tell attackers that they can’t use something against the system)
  - Interestingly, SGX was the first wide-spread usage of a threat model in which the OS itself is considered an attacker against a less-privileged entity (the enclave)
- AMD SEV came later, but made even worst mistakes
  - Crypto bugs
  - Registers were not protected at all (literally the hypervisor could just change or read registers)
  - They wrote that integrity was not needed, since any memory ‘changes’ would just cause a crash
  - They ignored side-channels too
  - They had all the original bugs in non-memory-encrypted systems (like TLB and other cache-related issues)
- And we did not even discuss DMA, VM ownership of devices, migration, etc…

I wonder why…

“I wonder why. I wonder why.
I wonder why I wonder.
I wonder why I wonder why.
I wonder why I wonder!”

Richard Phillips Feynman

Do we even need integrity?

- Arguably we want confidentiality, so do we even need integrity?
- I guess that is all old news now, but the answer is obviously yes (integrity failures do lead to full confidentially problems)
- Memory encryption without integrity (which should include replay protection) is only good against cold boot attacks
- Shay Gueron and I published a few general cases (that are technology agnostic since they use characteristics of the SW and not implementation bugs) to demonstrate how memory encryption without integrity can be broken [1] [2] [3]

Discovery…

“It would have been a fantastic and vital discovery if I had been a good biologist. But I wasn’t a good biologist”. … “We were there at the right place, we were doing the right things, but I was doing things as an amateur - stupid and sloppy”

Richard Phillips Feynman

What does encrypting the memory changes for an attacker?

- The attacker is not able to read the memory contents directly
- Attacker changes to the memory result in ‘unpredictable’ (random) changes of entire blocks (memory encryption schemes use block ciphers)

Both benefits from the security perspective assume the attacker has the ability to read and write memory, somehow (otherwise, there is no need for encryption in the first place)

But memory is not like a stored file

- Memory is in-use, has context, imply all kinds of SW stacks
- Memory encryption also does not create obliviousness: The attacker is still able to observe that memory areas (blocks) are changing (albeit not knowing the specific contents of those changes)

- So we can say that the attacker is:
  - **Blinded**: Does not have the plaintext value for any (**most**?) memory locations
  - And can only cause a **Random (Block) Corruption** (the attacker has no control over the plaintext that would be ‘consumed’ by the system after a modification of an encrypted memory area (and no matter the size of the modification, the impact causes an entire block to change)
  - But the change is in a **controlled** location (block-aligned)

Non-obviousness

- There is really an uncommon **primitive** to discuss (arguably it only exists because of memory encryption)
- Why it is important?
  - Let’s imagine a scenario of the malicious (cloud/system) administrator
  - They can use different means to read/write VM/guest memory (like debug stubs, HW JTAG ports, etc)
  - Memory encryption means that such individual now is prevented from accessing that VM (and its secrets - assuming all other building blocks work properly, such as the image encryption, attestation, etc)
  - That administrator though is still able to interact with the system via its valid ways (like ssh to the system, browsing a page it hosts, etc). Maybe even interacting with the login prompt of that system
  - And observe the encrypted memory changing (so while the admin is not able to know the contents, they can analyze memory changes versus interactions with the system to learn about locality!)

Brilliant People

“What bothered me was, I thought he must have done the calculation. I only realized later that a man like Wheeler could immediately see all that stuff when you give him the problem. I had to calculate, but he could see”

Richard Phillips Feynman

Example target : if (!0) data-only attacks

```
global var1…varn
global preauth_flag
global preauth_related
code_logic() {
 if (preauth_enabled) call_preauth_mechanism() // if successful, sets preauth_flag
 repeat_auth:
        if (preauth_flag) goto auth_ok
        authentication_logic(); // goes to repeat_auth in case of failure
 auth_ok:  return true;
}
```

TOCTOU (Time-of-use/Time-of-check) Race Condition

- While the block is corrupted, in most systems, the adjacent block is not even used (given that pre-auth is not even enabled)
- The preauth_flag just checks for !0, thus we do not need to control the exact value
- But still, how do we win the race?
  - Simple…
  - The system stops in the authentication_logic waiting for the password!
  - We corrupt the flag, and type an invalid password
  - Because the authentication fails, the logic is repeated, but this time the preauth_flag was corrupted and we login without the password

SIMPLE TO DEMONSTRATE, REAL CASE, AFFECTED ALL LINUX SYSTEMS

(notice that the pre-authentication mechanism does not need to be enabled)

But, but… you are Brazilian, so you are just lucky!

- Indeed the code logic here is terrible (it could do a 1==auth_flag instead of !auth_flag, it could make sure that the auth_flag was only used when/if preauth was even enabled, etc, etc, etc)

- In truth though, you are facing *ALL SOFTWARE* in the world (it is a general purpose technology after all!)
  - It is bound to exist many more cases of logic like that in which a limited, uncontrolled (value) corruption of a block is still beneficial to an attacker

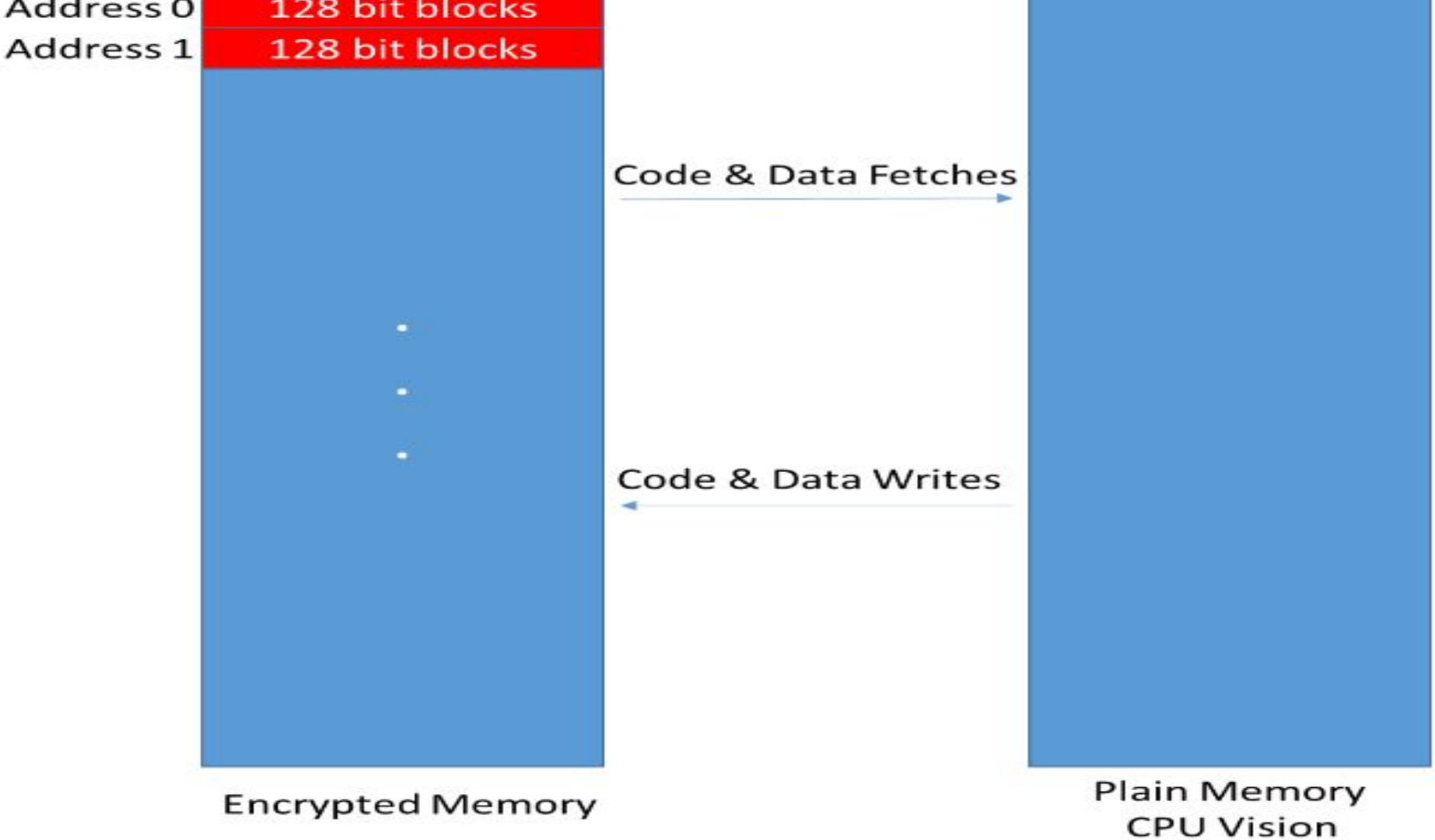
Yeah, I will just use the compiler to remove all != logic

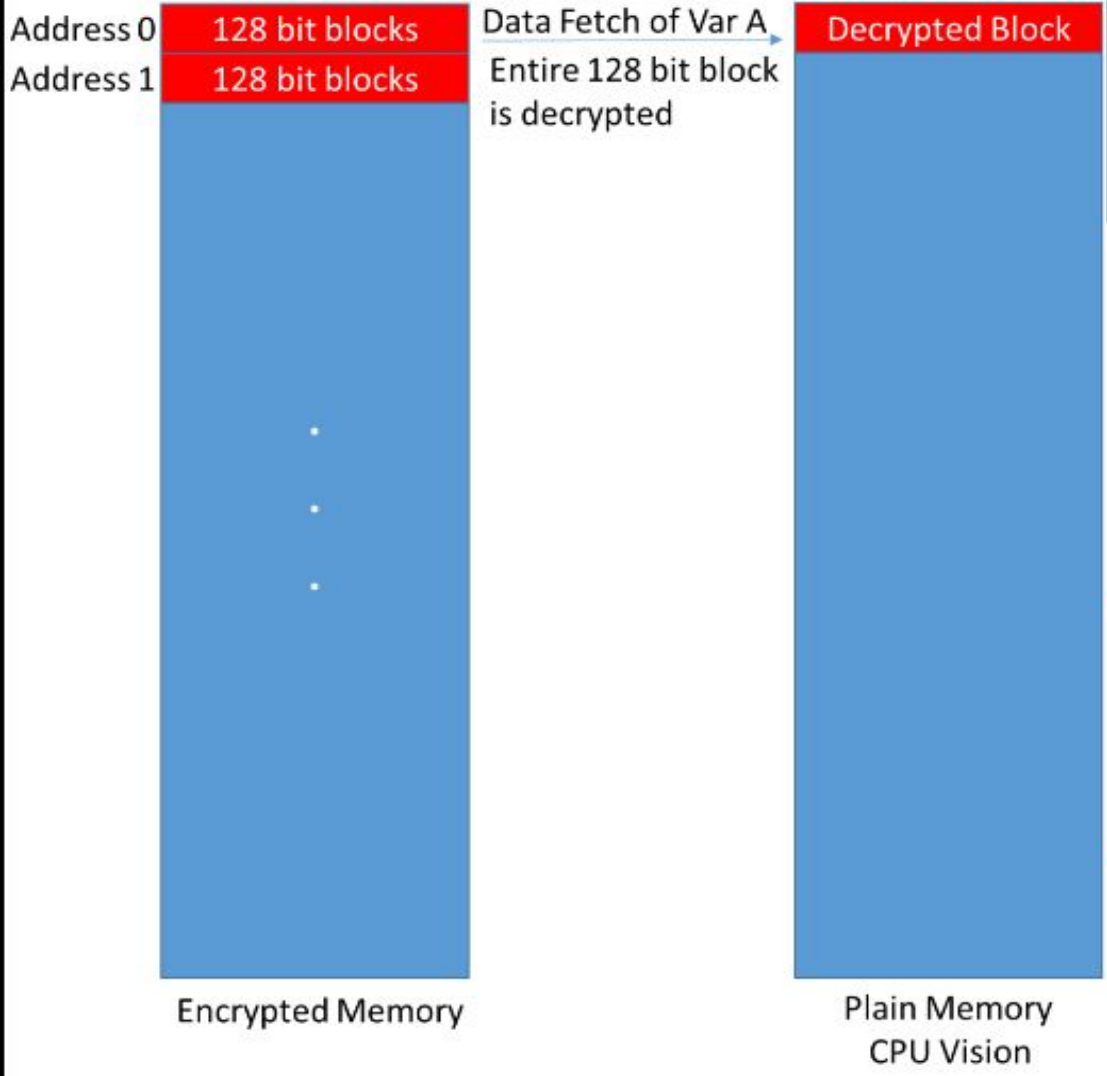
- I have bad news :)

- The primitive you assume the attacker has (arbitrary uncontrolled-value write, at arbitrary times in arbitrary locations without obliviousness) is quite powerful
  - It is literally like in every single memory location we could have data changing arbitrarily
  - SW can't really deter that power (we need integrity)

- Maybe in very controlled setups (and together with many other mitigation mechanisms and a lot of obscurity and prayers) this would not be enough for a determined attacker…

Here is another example to consider…


Address 0
128 bit blocks
Address 1
128 bit blocks
Code & Data Fetches
Code & Data Writes
Encrypted Memory
Plain Memory
CPU Vision



Showing addresses as block locations, to facilitate the discussion.

Var A:

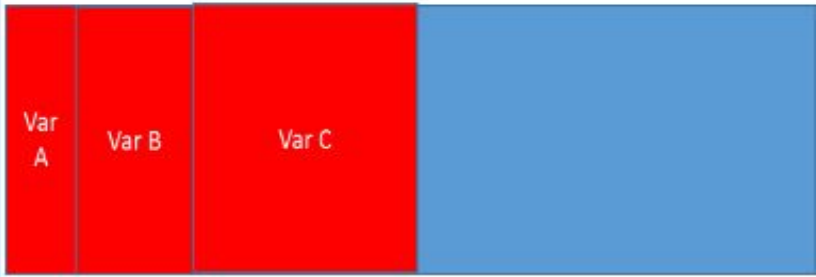
For example, an unsigned char, an 8 bits type

Var B:

For example, an unsigned short, 16 bits type

Var C:

For example, an unsigned int, 32 bits type



Decrypted Block

Corrupting var A, also corrupts Var B and C (bits corrupted randomly/independently as a property of the block cipher)

Yeah, obviously… so what?

If we find a way to brute-force a block that has this characteristics

- Many different data elements, with different sizes
- One of those elements being of interest, and small enough to be fully brute-forced (like a 32 bit integer)

• And for which we are able to tell if we somehow have a value we want

- In which the other elements, if changed, do not affect our interests as an attacker
- And for which any value would not affect the system stability (meaning: we can repeat the corruption as many times as we want)

• Then we are able to

- Have a fully controlled memory overwrite! (we just need to brute-force the element of interest til it randomly has the value of our interest!)

Can we make a pie with so many ingredients? *

- Linux Kernel manages processes using a data structure named task_struct

- Such struct has lots of elements necessary to store the process information, such as memory areas, opened files, privileges and so on

- For privileges, it uses a pointer to another data structure, which is the credentials... Having a look at it, we have something quite interesting

* Homage to a famous quote by Noir

Sounded like impossible?
A bit on the Linux Kernel...

Following the task_struct of a process (to find our target), we see there is a process credentials entry, which is a structure that has many elements, of interest we have:

```
kuid_t      uid;     /* real UID of the task */
kgid_t      gid;     /* real GID of the task */
kuid_t      suid;    /* saved UID of the task */
kgid_t      sgid;    /* saved GID of the task */
kuid_t      euid;    /* effective UID of the task */
kgid_t      egid;    /* effective GID of the task */
kuid_t      fsuid;   /* UID for VFS ops */
kgid_t      fsgid;   /* GID for VFS ops */
```

Notice that kuid_t and kgid_t are typedef's to __kernel_uid32_t and __kernel_gid32_t, which in turn:

```
typedef unsigned int   __kernel_uid32_t;
typedef unsigned int   __kernel_gid32_t;
```

Premise	Satisfy?
Many different data elements, with different sizes	Yes
One of those elements being of interest, and small enough to be fully brute-forced (like a 32 bit integer)	Yes (euid is our target)
And for which we are able to tell if we somehow have a value we want	Yes (our target process has elevated privileges)
In which the other elements, if changed, do not affect our interests as an attacker	Yes (we can change other elements if needed with the privileges)
And for which any value would not affect the system stability (meaning: we can repeat the corruption as many times as we want)	Yes

What a waste of time, there is integrity…

- AMD ‘integrity’ is provided via a table that sits, in-memory
  - Access to that table is controlled
  - If you trust memory access controls are enough, why do you need memory encryption?
  - Their integrity protection does not provide replay protection
- Intel has technology leveraging ECC (hum, Shay Gueron and I are the inventors for that patent)
  - It also does not provide replay protection
  - It is unknown how it affects reliability in practice, at large scale
- Homomorphic encryption is an entire field in itself (that might have many of the same problems)
  - Intel did include (work by Shay Gueron again) new instructions that made it practical to use for certain operations (like adding encrypted fields on a table)
  - Still limited applicability, not for general system use

Other excuses

- There is an asymmetry in technology: Marketing will just change their arguments when you debunk one
- After memory encryption properties were proven ‘not enough’, new arguments were added to justify its existence (like the poor’s man integrity and that it prevents other classes of bugs - like a buffer overflow)
  - Shallow conversations with non-tech folks might make those convincing!
- For those other classes of bugs (in which, for example a hypervisor would otherwise overwrite guest memory) you do not need memory encryption at all (just unmap guest memory, as ASI feature does for the Linux Kernel, as AWS does for their hypervisor and as grsecurity does in KERNSEAL)

Is it Difference in SPACE? Redundancy as Authentication

- Space-deployed systems have computation redundancy (which albeit different from encryption with integrity and replay protection, does provide a ‘similar’ result)
- Interestingly, a HW redundancy check would detect the attacker’s memory flip, and would potentially revert it (the two systems that match would prevail over the one that the flip occurred)
- Unfortunately, while the attacker ‘flips’ have a random consequence after decryption, their location and timing are controlled. And thus can be made in all redundant elements simultaneously and equally

It is not All Lost Though

- But redundancy can be used to make the attack statistically way more hard (potentially deterministically)
  - Making the possibility of accessing all redundant systems simultaneously could be a design goal

- Another idea is to use a shifter/adder in the address fetcher, literally randomly shifting memory locations between the different redundant systems (as in: Address 0 of System 1 matches Address 0+X of System 2 and address 0+Y of System 3)
  - Now, the attacker would have to do the monitoring of the system to discover the random shift
  - So if for every memory change multiple changes were performed, and with a large shift, the attack would be harder
  - Note: This is not a completely formed idea, prototyping and tests are needed to make it feasible and impactful

The experts…

“And I remembered, when I saw this article again, looking at that curve and thinking, That does not prove anything!” …” Since then I never pay any attention to anything by experts. I calculate everything myself”

Richard Phillips Feynman

Future - as if I could foresee it (1/2)

- The rise of confidential computing will bring new reliability problems (and hide even further compromises) - a lot of technology has yet to be developed - anyone jumping into using it should not lie to themselves they are doing it for security reasons :)

- Cloud providers are a national security risk
  - Too many secrets concentrated in one technology set
  - Research on attacks leveraging multi-tech capabilities are still non-existent
  - They can do way more than standalone companies, but savings at scale, problems in scaling talent, and others also create new challenges

Future - as if I could foresee it (2/2)

- Supply Chain security and hardware inspection specifically will become a major issue/topic

- BIOS/UEFI (and other platform firmware) will become more and more targets
  - Binarly and LVFS responded to part of my ask years ago (Troopers Keynote) to have a way to do industry-wide searches
  - We still need to fulfill the second half, which was for users/companies all over to upload their images for comparison with images on systems in other regions

Conclusions

- There is no confidential computing in a general purpose platform in 2023 -> you do have to trust the hardware/infrastructure provider

- Memory encryption is an interesting building block, but it brings with it other challenges that have to be considered

- I want to see the world continue developing the technology, I see more (immediate) importance for it in the desktop market than in the server market

Questions !?

Rodrigo Branco (BSDaemon)
rodrigo @ kernelhacking.com

“Of course I realized what it was: They could not DO it!”. … “It was a kind of one-upmanship, where nobody knows what is going on, and they’d put the other one down as if they did know. They all fake that they know, and if one student admits for a moment that something is confusing by asking a question, the others take a high-handed attitude, acting as if it is not confusing at all”.

Richard Phillips Feynman