

Inside the Machine

How Offensive Security is Defining the Way We Compute

Rodrigo Rubira Branco (@BSDaemon)

Chief Security Researcher

STORM (STrategic Offensive Research & Mitigations) Team


Intel Corporation

rodrigo.branco *@* intel.com



**“There is no comfort in
the growth zone and no
growth in the comfort
zone”**

Unknown

A photograph of a man from behind, wearing a black t-shirt and blue jeans. The t-shirt has a humorous text printed on the back. The background shows a residential area with a house and a palm tree under a clear sky.

HEAVEN is where the police
are British, the cooks are Italian,
the mechanics German, the lovers
Brazilian and it is all organized
by the Swiss.

HELL is where the chefs are
British, the mechanics Brazilian,
the lovers Swiss, the police
German, and it is all organized
by the Italians.

DISCLAIMER

I don't speak for my employer. All the opinions and information here are of my responsibility.

Personal:

"If hacking is an art, please excuse me, but I have poetic license"

- BSDaemon

I'm sorry in advance for the excessive usage of generalizations!

FYI: my wife is a lawyer ☺



Rodrigo @ Troopers

My Talks @Troopers

- [KIDS – Kernel Intrusion Detection System \(2008\)](#)
- [Advanced Payload Strategies: “What is new, what works and what is hoax?” \(2009\)](#)
- [Letting your fuzzer knows about target’s internals \(2010\)](#)
- [Dynamic Program Analysis and Software Exploitation: From the crash to the exploit code \(2011\)](#)
- [Into the Darkness: Dissecting Targeted Attacks \(2012\)](#)
- [Behind the Scenes: Security Research \(2013\)](#)
- [Security Analysis of the Computer Architecture: What a software researcher can teach you about your \(2014\)](#)
- [Modern Platform-Supported Rootkits \(2015\)](#)
- [A praise for hackers \(2016\)](#)
- [Blinded Random Block Corruptions \(2017\)](#) -> I hope you still remember, I’m going to discuss it!
- GAP (I must admit that was a sad thing for me) (2018)

DO WHAT YOU LOVE
AND YOU’LL ~~NEVER~~
~~WORK A DAY IN YOUR~~
~~LIFE~~ WORK SUPER
FUCKING HARD ALL
THE TIME WITH NO
SEPARATION OR ANY
BOUNDARIES AND ALSO
TAKE EVERYTHING
EXTREMELY PERSONALLY
@ADAMJK

Objectives

- I'm not here to try to teach the priests how to pray
 - Instead, I'll try to give sermon arguments (PoC || GTFO inspiration)
 - Propose some baseline for certain discussions (on mitigations and exploitability)
 - In this audience, I expect that most of the points themselves are well known, but hope that neglected ones will be called out and the most important ones will be re-enforced

Objectives

- I'm not here to try to teach the priests how to pray
 - Instead, I'll try to give sermon arguments (PoC || GTFO inspiration)
 - Propose some baseline for certain discussions (on mitigations and exploitability)
 - In this audience, I expect that most of the points themselves are well known, but hope that neglected ones will be called out and the most important ones will be re-enforced

Defining the objectives in one image?

Objectives

“The amount of energy necessary to
refute bullshit is an order of
magnitude bigger than to produce it.”

Alberto Brandolini

The attacker when I found a bug



The attacker when I propose a mitigation



Objectives – Halvar's MitiGator [1]



MitiGator: Raising the bar until no more exploits can be seen.

[1] Halvar Flake. "Weird Machines, Exploitability, and provable Non-Exploitability". Oxford Presentation 2018.

Three Points to Take Out (1/3)

- Threat Modeling is a relatively mature element of secure system engineering (where system can be hardware and/or software)
 - We have security objectives, attackers in scope (i.e.: physical presence, physical possession, remote, unprivileged, privileged), threats and features (relevant to demonstrate that the identified threats are addressed)
 - **But: We do not apply the same 'mature' process in defining mitigations (against unknown vulnerabilities/classes of issues)**
 - **Notice: Mitigations should be used/defined on top of usual quality in development, not to 'compensate' for the lack of it**

Three Points to Take Out (1/3)

As a mathematician once **joked** with me
“The Engineer’s Proof by Induction is slightly different:
If it works for $n=1$, $n=2$, $n=3$ it works for all cases”

So here is something to remember:

A new vulnerability mitigation intended to address classes of vulnerabilities should fully mitigate at least 3 real past vulnerabilities to be seriously considered.

Three Points to Take Out (2/3)

- There are not a lot of formalization of exploits (more on this later), which makes exploit writing a mix between deep engineering and art [1]
 - **Understanding a presentation and overall aspects of an exploit instance, is not the same as understanding how vulnerabilities *CAN BE* and *ARE* exploited** (oftentimes, other possible avenues, investigations that did not pan out, other engineering problems that needed to be solved – be it because of the tools that already proportioned it to the researcher, or because the researcher already had the knowledge in his mind - as the real dimensions of time spent are all missed)

[1] Excellent discussion in: “Are Reverse Engineering and Exploit Writing an Art or Science?” at NYUPoly THREADS Conference Panel (Cyber Security Awareness Week), Nov, 2013.

Three Points to Take Out (2/3)

So here is something else to remember:

Those proposing mitigations must understand the problem. Such understanding can be gained through experience writing real exploits (with real limitations and complexities) or by teaming up with someone with that experience.

Disclaimer: Being able to write exploits do not imply liking to do it (I do know developers that work in very good mitigations that do not like to write exploits, but have the full understanding and ability to do so)

The ideas expressed herein represent those of the presenter in his individual capacity.

Three Points to Take Out (3/3)

- When (we) researchers share the knowledge (be it in a presentation, paper, blog, informal conversations), oftentimes (we/they) simplify the problem
 - **Be it because of lack of space/time, lack of perception (sometimes we do not even see the bigger picture), Impostor Syndrome (they know so much about the problem that they believe they do not know enough and end up not discussing many aspects as they feel unfit to do so) – re-enforced by Dunning-Kruger on many in the industry (they know so little that they believe they really understand things) or many others**
- Unfortunately, as a result, there is a misperception of abilities in the industry (as it is believed there are many more people capable of designing proper mitigations than actually there are)
 - That essentially means additional broken mitigations, additional complexity, and additional challenges to actually adopt/propagate the mitigations that do work

Three Points to Take Out (3/3)

- To better describe the relation between Dunning-Kruger and Impostor Syndrome:

**“As the area of our knowledge grows, so too does the
perimeter of our ignorance”**

Neil deGrasse Tyson

Three Points to Take Out (3/3)

So here is the last thing to remember:

Investments in defense should benefit from the investments in offense (be guided by it) [1]. In the mitigations space, implementation details make all the difference and proper architectural definitions are not enough.

[1] Ben Hawkes. “Project Zero – Make Oday Hard”. CanSecWest 2015.

The ideas expressed herein represent those of the presenter in his individual capacity.

What do I miss? (1/2)

- Proper mitigation architecture and design is hard, and it is specially hard when data is not available (or can't be obtained without huge investments)
 - That is why I like so much presentations and posts on Pwn2Own achievements and security research performed (even if they are not releasing something super new, it is more data on what was done)
 - An additional benefit: making sure others can get into the state-of-the art on the field (instead of it become a dyeing art)
 - I would love if additional information on time spent, what was known before (like specific primitives, techniques) and others were also made available (Microsoft more recently makes some of it available, which help understand the rationale behind some of their decisions)

What do I miss? (2/2)

- Example of data that is not easily available: what is the % of memory corruption exploits in Metasploit (or overall, publicly available exploits) that bypass ASLR? DEP? CFG? Canary? Etc...
 - From the ones that do not, which ones would need additional bugs? (versus ones that the exploit is just a PoC but could already have bypassed the mitigation with the obtained primitives?)
- While it looks like only a data collection problem (from what is already available), it does require a lot of knowledge to dig deeper to properly answer the ‘was it possible anyway?’ question
 - “The fallacy of the defense in depth” (Short blog post I’ve made, in Portuguese only, in 2014)

Kudos and hope more folks understand

- Katie Moussouris, for, between so many things, her work on the realities/dynamics/economics behind Bug Bounties
 - Recommended reading: “The wolves of vuln street – the first system dynamics model of the Oday market” (<https://www.hackerone.com/blog/the-wolves-of-vuln-street>)
 - And the more recently work (that IMHO, was inspired by Katie’s): “On Bounties and Boffings” by Trail of Bits (<https://blog.trailofbits.com/2019/01/14/on-bounties-and-boffins/>)
- Bug bounties do not have the scalability that many believe, and I would argue that while the numbers are already deceptive (on who actually get real pay-outs), they are even worse in reality
 - Many companies do not have the maturity to do the proper triaging and are likely paying for bugs that they should not

The Importance of Formalization

- In the past years we've been witnessing an exceptionally important trend in computer security towards formalizing offensive work
 - That is essential to finally migrate from a mix art/engineering form to purely engineering (with all the needed talent that engineering requires)
 - It is also a key element in bringing the academia towards the state-of-the-art and with that, scale offense (currently a big challenge)
- From that trend, terms like weird machine appeared, LangSec gained much more traction/support
 - Opening doors for everyone to understand that data is what drives code, that parsers are recognizers and creating new concepts on secure development
 - But it also demonstrated that even some basic assumptions are in conflict with what was taught by defense

LangSec x Secure Development

- In current secure development guidance, it is a 'rule' that input is checked as close as possible to the use of the input, which makes 'shotgun' parsing acceptable – while LangSec proposes the recognition to happen at the entry point
- In current development guidance, the rule of intrinsically secure functions seem to make sense (after all, many vulnerabilities arise from copy+paste code from one project to another – without copying the input handling of the parameters – or by adding new functionality – and as so, new paths to vulnerable functions). But once again, due to real life restrictions (development time and performance of having duplicated checks), recognizing everything **and** adding specific checks is hard in practice. Somehow more guidance/discussions are still needed even in basic areas

More formalizations

- In recent years, we've also seen the work by Julien Vanegue on heap modeling for exploit writing [1]
 - Which helps rationalizing between different mitigations
 - And move us from art to science state in the subject (improving toolset, enabling real/better automation)
- And more recently, Halvar's paper [2] providing a theoretical (mathematical) framework to define exploitation (of memory corruptions)

[1] Julien Vanegue; "Heap Models for Exploit Systems"; IEEE S&P LangSec Workshop; 2015

[2] Halvar Flake; "Weird machines, exploitability, and provable unexploitability"; 2018

Not a comprehensive list

- Complexity has been identified in many areas, an example that I like
"a modern compressor is a bit like a compiler. The compressed data is a kind of program in bytecode, and the decompressor is just an interpreter that runs that bytecode"
Link: <http://cbloomrants.blogspot.com>
- Obviously, my intent is not to be comprehensive, but to demonstrate an important trend
 - There were many earlier tries on defining exploit writing, like the work by Gerardo Richarte [1]
 - And the conscious effort by Microsoft on using the understanding on exploits to drive mitigation strategies, shared in multiple presentations along the years - See [2] for one that includes HW in the threat scope
 - The ROP definition for example, was very well known by practitioners (exploit writers), but only got widely understood once properly defined (in an academic paper) [3]

[1] Gerardo Richarte; "About exploit writing"; 2002

[2] David Weston; "Hardening with Hardware"; BlueHat IL 2018

[3] H. Shacham; "The geometry of innocent flesh on the bone: return-into-libc without function calls (on the x86)". CCS 2007

Once upon a time...

- ... A professor of mathematics (Nei Soma)
- In a computer engineering grad school (Airforce Technology Institute – ITA in Brazil)
- Had a student that had as job title: “Exploit writer”
- So, the professor had the idea: **Define your work using set theory**

What now?

- The work itself got lost, but never fully forgotten
 - I kept using the ideas in discussions
 - Until I finally met some **motivated individuals** that really incentivized (and worked with me) to better elaborate and put on paper the idea
 - A full paper is available “A Mathematical Modeling of Exploitations and Mitigation Techniques Using Set Theory”: <http://spw18.langsec.org/papers/Kawakami-Exploit-modeling-using-set-theory.pdf>
 - For anyone interested in the actual formulas and details
- Here are their names:
 - Kekai Hu
 - Ke Sun
 - Henrique Kawakami

Set Theory Mitigation and Exploit Modeling

A quick glance

- Exploit Primitives (EP) - attack ability that can be achieved from a security vulnerability
 - Each primitive is composed of two elements: type and property
 - Type: read, write, execute
 - 5 major primitive properties (as defined by PaX Team [1]):
 - Arbitrary Addresses (AA), Arbitrary Content (AC), Arbitrary Operation (AO), Arbitrary Number of Times (AN) and Arbitrary Time (AT)
 - **Limitation: We did not model partial.**
- Exploit Objective (EO) is the ultimate goal of an attacker against the analyzed/modeled system (E.g: Attacker wants administrative access in a remote system).
 - Represents the concept of chained vulnerabilities used by one exploit (un-ting the relation of one exploit to one vulnerability, that is the classical view by non exploit writers)
- Exploit Condition (EC) is the minimal required combination of EP in a system to achieve an exploit objective (EO)
- Exploit Difficulty (ED): we propose the usage of Big O notation to define ED and measure mitigation quality

[1] PaX Team; “RAP: RIP ROP”; Hackers 2 Hackers Conference (H2HC); 2015

Probabilistic x Deterministic Mitigation

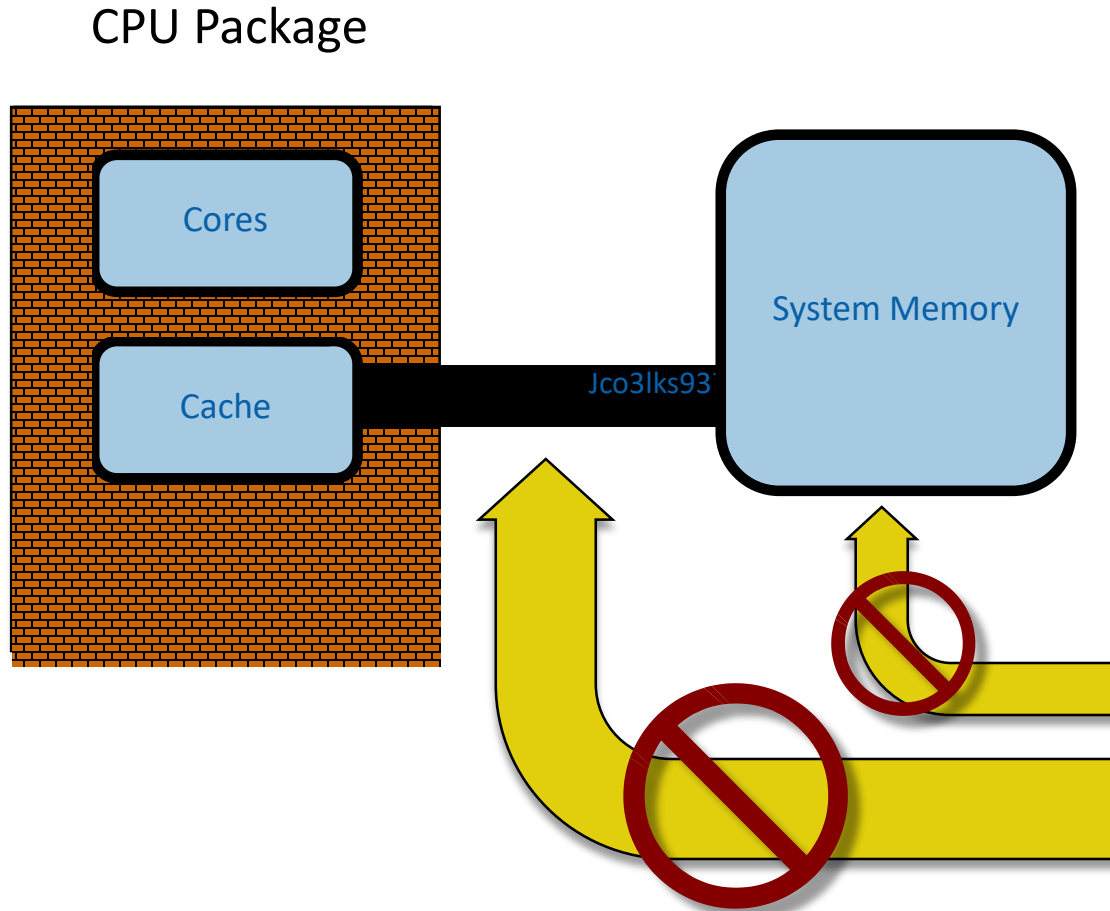
- Deterministic mitigation (DM) fully eliminate (or with an un-computable order of chances probabilistically eliminate) at least one EP (**emphasis: there is no 1:1 relation between vulnerability and primitives, and vulnerability and exploits**)
 - Properly applied encryption for **at rest data** [1], for example, can offer DM (if we assume the cipher indeed requires un-existent amounts of computation to be broken). It adds $O(\infty)$ complexity
 - * if the data is not at rest (like in the DRAM encryption case), encryption (without integrity) adds non-determinism for the attacker that can be bypassed [1]
- Probabilistic mitigation (PM) is a mitigation that **significantly** increases ED
 - *AND* reduces the successful rate of its target EP
 - It adds $O(n)$ complexity

[1] Gueron, Shay; Branco, Rodrigo. “Blinded Random Corruption Attacks”. IEEE HOST 2016

New Primitive? Non-obliviousness

- The ability of an attacker to observe modifications and accesses are happening (but not the ability to see/control their values)
 - Relevant in the new world of memory encryption
 - Practitioners are aware of it, even if not explicitly named (e.g.: included in infoleak definition)
- An attacker that has:
 - read (of encrypted content) is non-oblivious to memory changes (new, previously the attacker would have an infoleak of an address or a read of content)
 - write (before crypto operation, without knowing the key) causes a random flip into the memory (non-controlled write, at any address was previously defined)
 - Interesting though: even 1 bit flip will cause now block-sized (and within block boundaries), uncontrolled write
- Discussed in Troopers 2017 (work with Shay Gueron)
 - For a scenario without this primitive, we are essentially talking about cold boot
 - Memory encryption is a deterministic solution against cold boot attacks

1 image, 1000 words?



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

Technologies? Disclaimer

- The list in the next slide are of technologies that use some kind of memory protection (with or without authentication, different encryption modes, etc)
- **We did not necessarily look into those technologies, therefore we are not claiming they are vulnerable, we are not saying they are comparable or even that they have similar purposes. The list is not comprehensive either. It is not showing in chronological order of creation or any kind of order. We just using them as examples of real cases of memory encryption technologies**

Technologies?

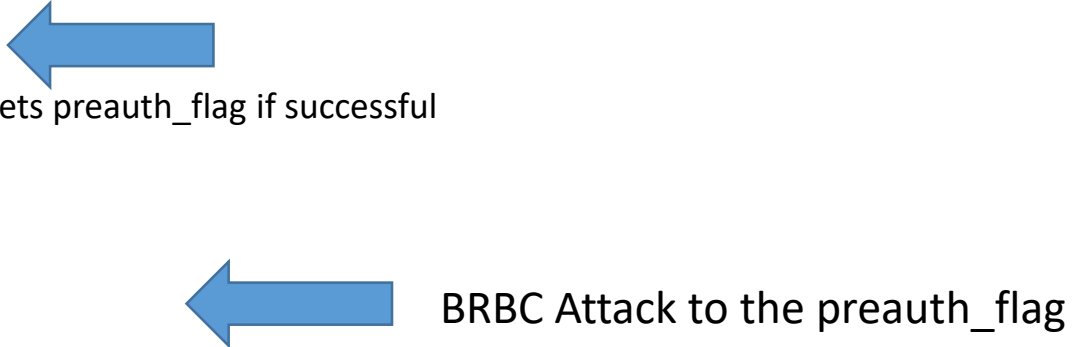
- Xbox
- Nintendo 3DS Security Processor (**PoC** || **GTFO 14**)
 - They discuss the possibilities of attacking encrypted code and the likelihood for the random corruption to create a valid (good for the attacker) opcode
 - Article: How likely are random bytes to be a NOP sled on ARM? By Niek Timmers and Albert Spruyt
- Apple Secure Enclave Processor
 - See the talk Demystifying the Secure Enclave Processor @ Black Hat 2016 by Tarjei Mandt and cia
- Intel MEE (Memory Encryption Engine), TME (Total Memory Encryption) and MKTME (Multi-Key TME)
- AMD SEV (Secure Encrypted Virtualization) and SME (Secure Memory Encryption)

Becoming “root” on a locked system with a BRBC attack (back into Troopers 2017)

```
global var1...varn
global preauth_flag
global preauth_related
code_logic() {
    if (preauth_enabled) {
        call_preauth_mechanism() -> sets preauth_flag if successful
    }
repeat_auth:
    if (preauth_flag) goto auth_ok;

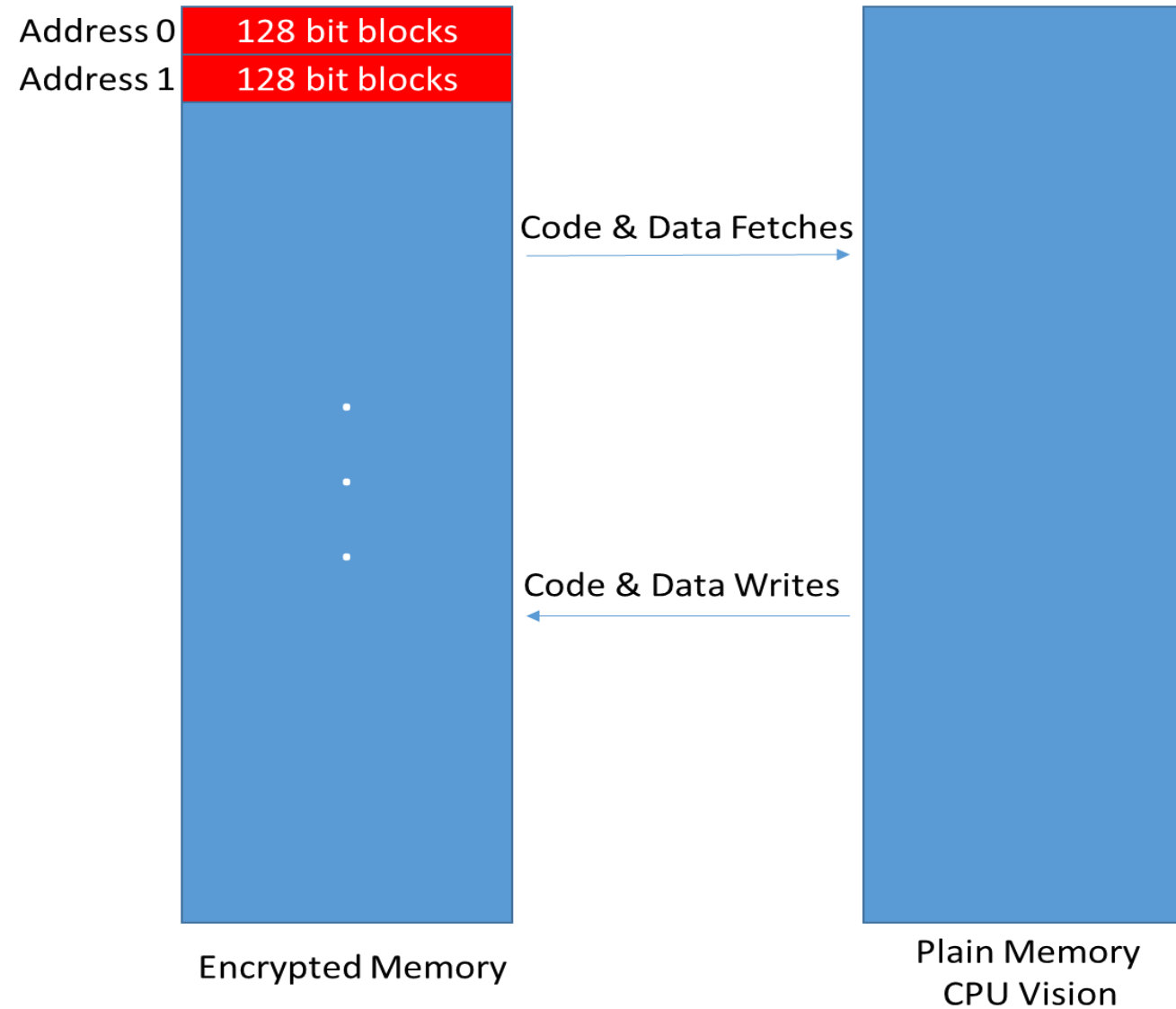
    authentication_logic();

auth_ok:
    return;
}
```

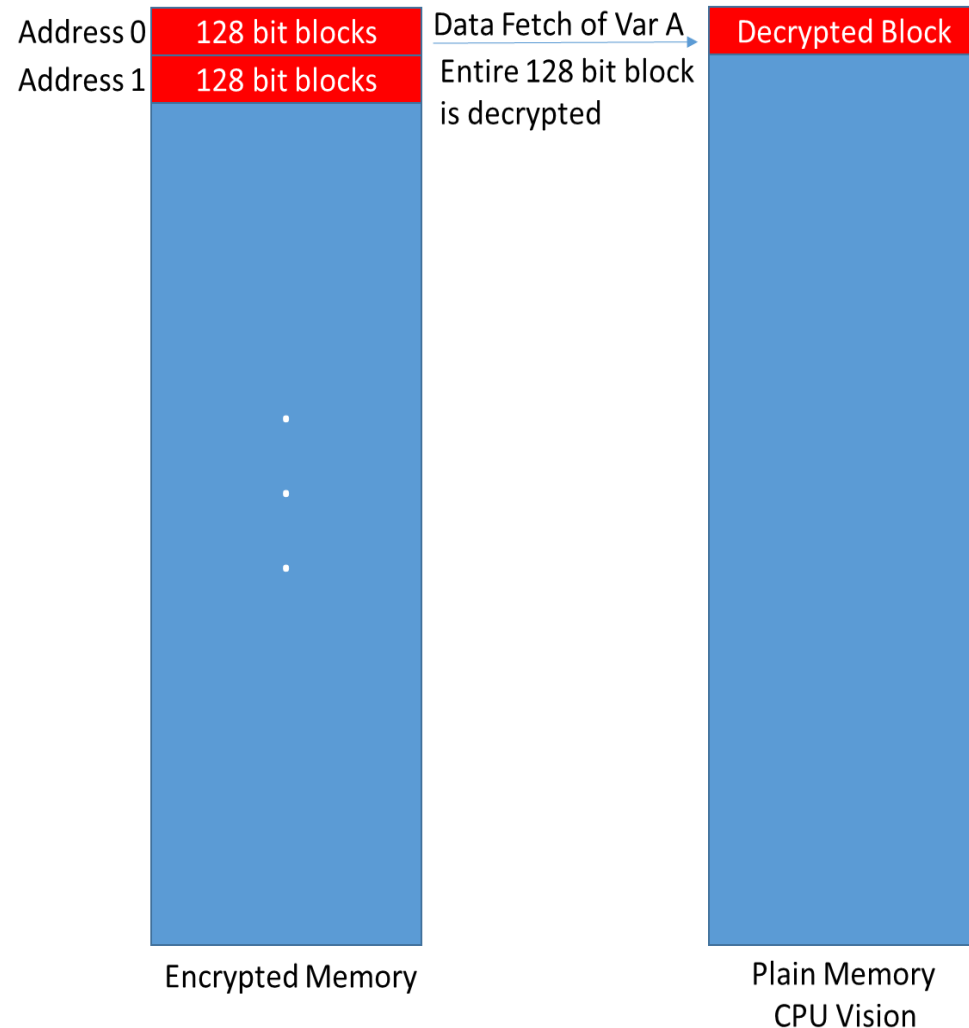


BRBC Attack to the preauth_flag

Memory, as encrypted blocks



Corruption within a block, randomly corrupt everything in that block



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)

Introducing: Feasible Brute-force-based BRBC

- 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)
 - The primitives again: **Arbitrary Addresses** (AA), Partially **Arbitrary Content** (AC), Arbitrary Operation (AO), **Arbitrary Number of Times** (AN) and **Arbitrary Time** (AT), **Non-Obliviouness** (NO)
- 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 quote by *Noir*

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;
```

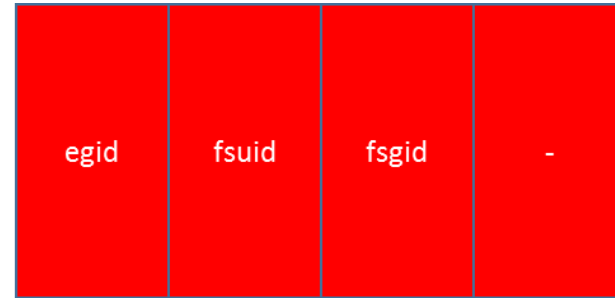
Is alignment inside the target block an issue?

No, since the adjacent elements do not matter for the attack

Option 1
eid is last
element
of a block



Block 1



Block 2

Affects: gid,
suid, sgid
(everything
before it)

Option 2
eid is
middle
element of a
block



Block 1



Block 2

Affects: sgid,
egid, fsuid
(max 2 before
and two after
it)

Option 3
eid is first
element of a
block



Block 1



Block 2

Affects: egid,
fsuid, fsgid
(everything
after it)

Do we have a winner candidate?

Premisse	Does it 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

Who is the founder of this little Hacker?



What about limitations?

- We need to be able to locate such a data structure in memory (we are blind to the memory contents)
 - There are a lot of challenges to being able to do that (and system pressure might affect the ability of doing it)
- For now, our PoC requires a process running on the target (with no privileges)
 - We elevate that process' privilege
 - The requirement for such a process is exactly to avoid the limitation (given we have a process we control, we use such a process to spawn multiple child process and create a predictable memory layout and pattern that we can identify – that is how we locate the correct structure in memory)
 - We are studying other possibilities (like for server-side process that anyhow spawn child processes, and others)

Three Points to Take Out (My conclusions!)

- A new vulnerability mitigation intended to address classes of vulnerabilities should fully mitigate at least 3 real past vulnerabilities to be seriously considered
- Those proposing mitigations must understand the problem. Such understanding can be gained through experience writing real exploits (with real limitations and complexities) or by teaming up with someone with that experience
 - Careful with new primitives (primitive replacement policy anyone?)
- Investments in defense should benefit from the investments in offense (be guided by it). In the mitigation's space, implementation details make all the difference and proper architectural definitions are not enough

The ideas expressed herein represent those of the presenter in his individual capacity.

The end!! Really is !?

Rodrigo Rubira Branco (@BSDaemon)

Chief Security Researcher

STORM (STrategic Offensive Research & Mitigations) Team

Intel Corporation

rodrigo.branco *@* intel.com



**“There is no comfort in
the growth zone and no
growth in the comfort
zone”**

Unknown