DARTMOUTH

Ghostbusting: Mitigating Spectre with Intraprocess Memory Isolation

Ira Ray Jenkins[1], **Prashant Anantharaman**[1], Rebecca Shapiro[2],
J. Peter Brady[1], Sergey Bratus[1], Sean Smith[1]

[1] Dartmouth College
[2] Narf Industries

https://prashant.at
prashant.anantharaman.gr@dartmouth.edu

The Principle of Least Privilege

- It requires that the individual components of a system need to have a minimal set of permissions to perform their functionality.
  - Privilege separation and intraprocess memory isolation are just some of the ways of enforcing this principle.
  - Spectre V1 attack was an example of an intraprocess memory attack where a secret was leaked despite not being accessed by the program at all.
- In our paper:
  - We demonstrate how intraprocess isolation techniques such as Memory Protection Keys (MPKs) and ELF-based Access Control (ELFbac) can be effective in mitigating the Spectre V1 attack.
  - We enforce the policy that a secret after initialization must not be touched.

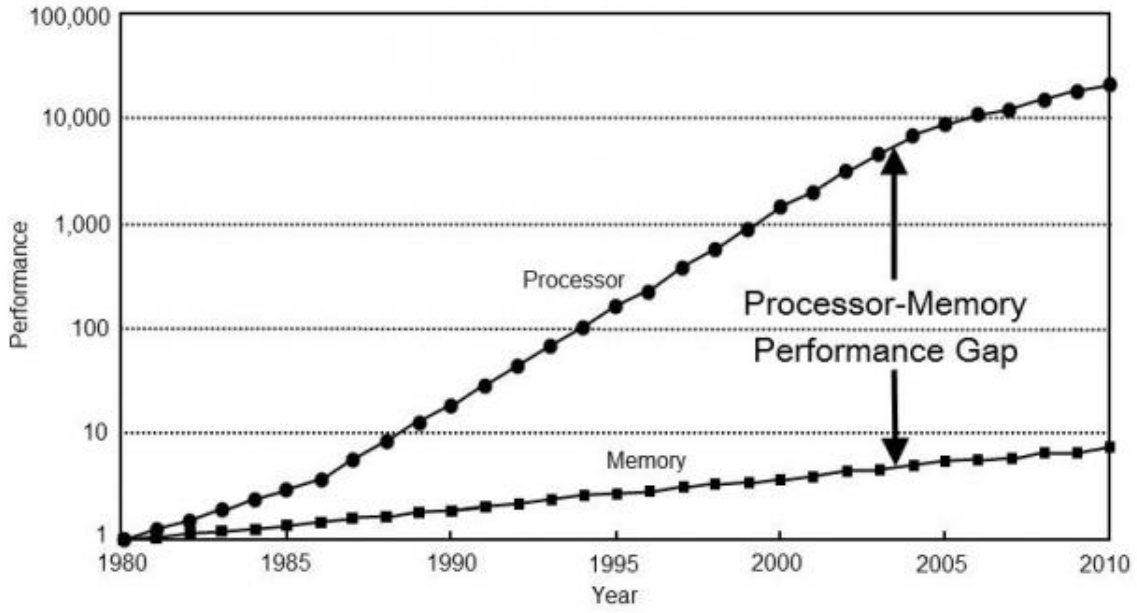
Outline

- **Spectre V1**
- ELF-based Access Control
- Memory Protection Keys
- Evaluation
- Conclusions

Picture: https://en.wikipedia.org/wiki/Spectre_(security_vulnerability)

Speculative Execution

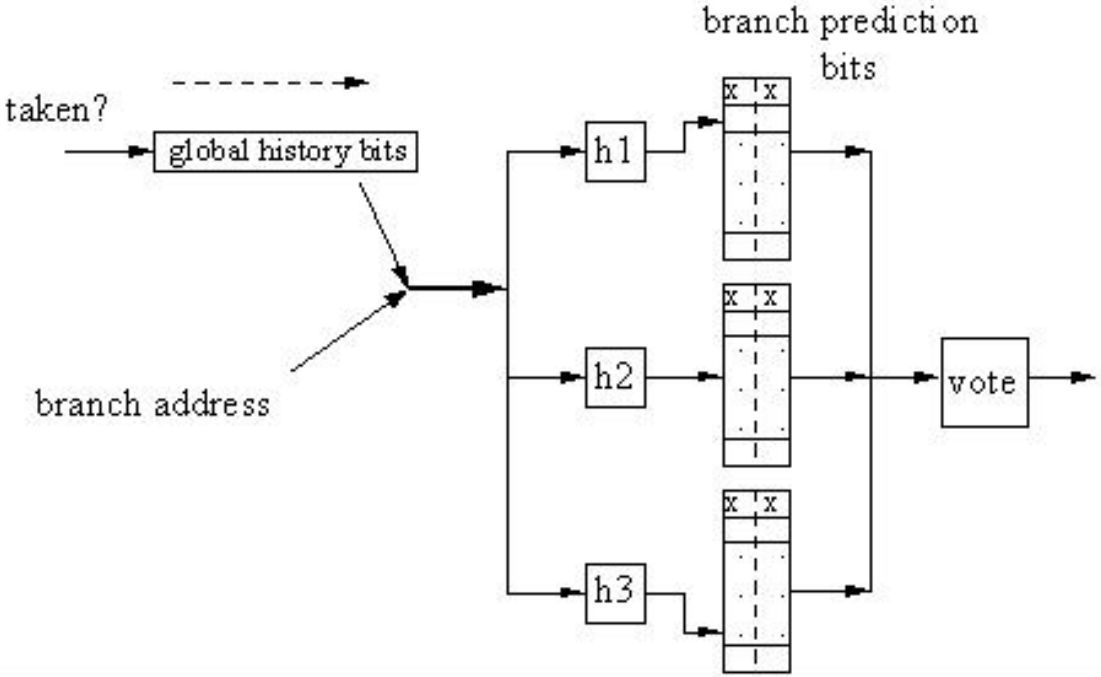
- Instructions within a pipeline are executed out of order.
- The results are later reordered and the dependencies are satisfied to assure semantics are maintained.
- *Speculative execution* predicts the control flow and executes instructions prior to knowing if they are required.



Source: https://www.extremetech.com/computing/261792-what-is-speculative-execution

Branch Prediction

- Dynamic Branch Predictors use:
  - Single bit: simply storing the last branch taken.
  - Multi bit: Pattern History Tables (PHTs)
- PHTs store the history of the branches taken to allow future branches to be predicted.
- Neural Networks have also been designed to predict branches.



http://web.engr.oregonstate.edu/~benl/Projects/branch_pred/

Speculative Execution Attacks

- In 2018, CVE-2017-5753 introduced “Spectre”
- CPUs MUST flush pipeline when miss-speculation occurs.
- Flushing does occur for the pipeline, but not for the caches and microarchitectural *effects* remain after the transient instructions.

Apple issues Meltdown and Spectre patches for older versions of its Mac operating system — and you should install them right away (AAPL, INTC)

TECH

This new vulnerability affecting Intel processors sounds pretty scary

By Andy Meek @aemeek
August 14th, 2018 at 6:32 PM

#TRENDING

PRIVACY AND SECURITY

Intel's Never-Ending Spectre Saga Continues to Be a Hot Mess

Alex Cranz
1/29/18 1:25PM • Filed to: INTEL

57 1

Spectre V1 PoC

- There's a speculative bypass of the bounds check.
- The underlying technique for V1 is to exploit the branch prediction by poisoning the PHT to mispredict this conditional branch.
  - Train the CPU with valid values for x
  - Give a bad *x* value.
  - CPU speculates and caches an "index" into array2.
  - Use timing side channel to recover "secret" from array2.
- The program, however, never touches the "secret"

Attacker controls x

```
void victim_function(size_t x) {
  if (x < array1_size) {
          temp &= array2[array1[x] * 512];
  }
}
```

Ways people have mitigated it in commercial software

Without /Qspectre

```
1  ?example@@YAEHHPAH0@Z PROC
2   mov ecx, DWORD PTR _index$[esp-4]
3   cmp ecx, DWORD PTR _length$[esp-4]
4   jge SHORT $LN4@example
5   mov eax, DWORD PTR _array$[esp-4]
6  ; no lfence here
7   mov dl, BYTE PTR [eax+ecx*4]
8   mov eax, DWORD PTR _array2$[esp-4]
9   movzx ecx, dl
10  shl ecx, 8
11  mov al, BYTE PTR [ecx+eax]
12 $LN4@example:
```

With /Qspectre

```
1  ?example@@YAEHHPAH0@Z PROC
2   mov ecx, DWORD PTR _index$[esp-4]
3   cmp ecx, DWORD PTR _length$[esp-4]
4   jge SHORT $LN4@example
5   mov eax, DWORD PTR _array$[esp-4]
6   lfence
7   mov dl, BYTE PTR [eax+ecx*4]
8   mov eax, DWORD PTR _array2$[esp-4]
9   movzx ecx, dl
10  shl ecx, 8
11  mov al, BYTE PTR [ecx+eax]
12 $LN4@example:
```

What about Linux?

12 arch/x86/kvm/vmx.c

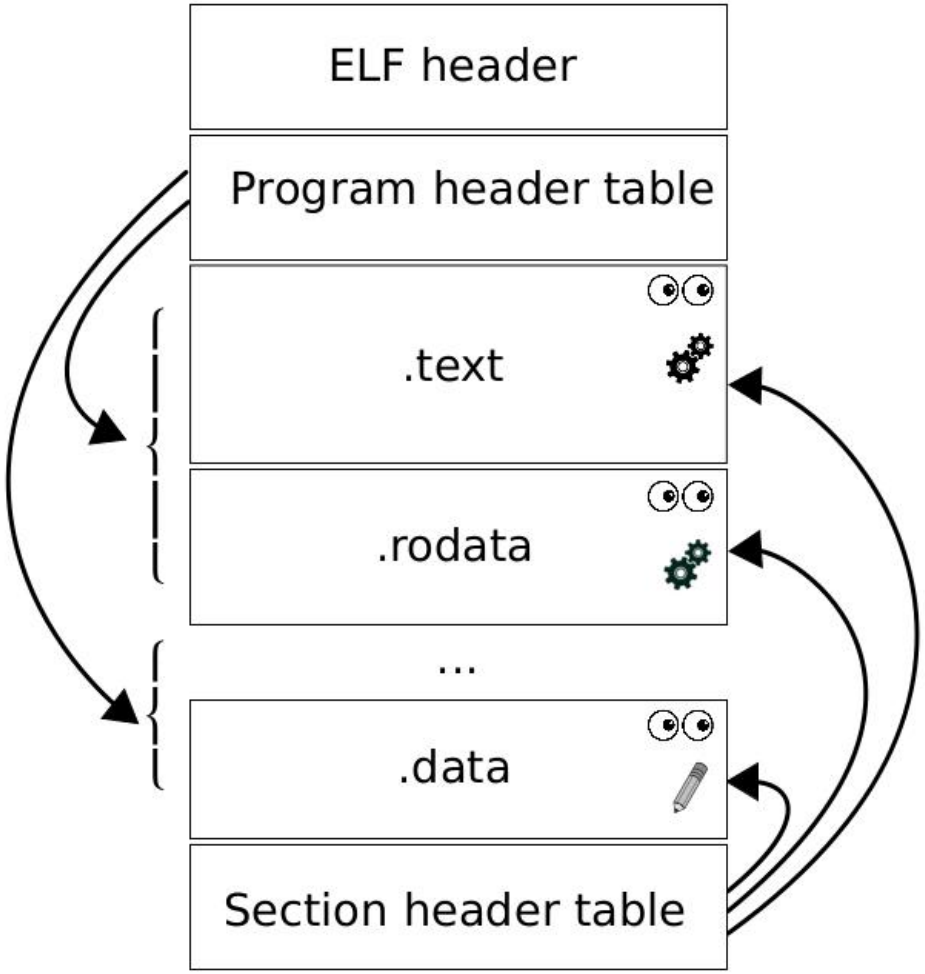
```
@@ -884,8 +884,16 @@ static inline short vmcs_field_to_offset(unsigned long field)
 {
        BUILD_BUG_ON(ARRAY_SIZE(vmcs_field_to_offset_table) > SHRT_MAX);

-       if (field >= ARRAY_SIZE(vmcs_field_to_offset_table) ||
-           vmcs_field_to_offset_table[field] == 0)
+       if (field >= ARRAY_SIZE(vmcs_field_to_offset_table))
+               return -ENOENT;
+
+       /*
+        * FIXME: Mitigation for CVE-2017-5753.  To be replaced with a
+        * generic mechanism.
+        */
+       asm("lfence");
+
+       if (vmcs_field_to_offset_table[field] == 0)
                return -ENOENT;

        return vmcs_field_to_offset_table[field];
```

Outline

- Spectre V1
- **ELF-based Access Control**
- Memory Protection Keys
- Evaluation
- Conclusions

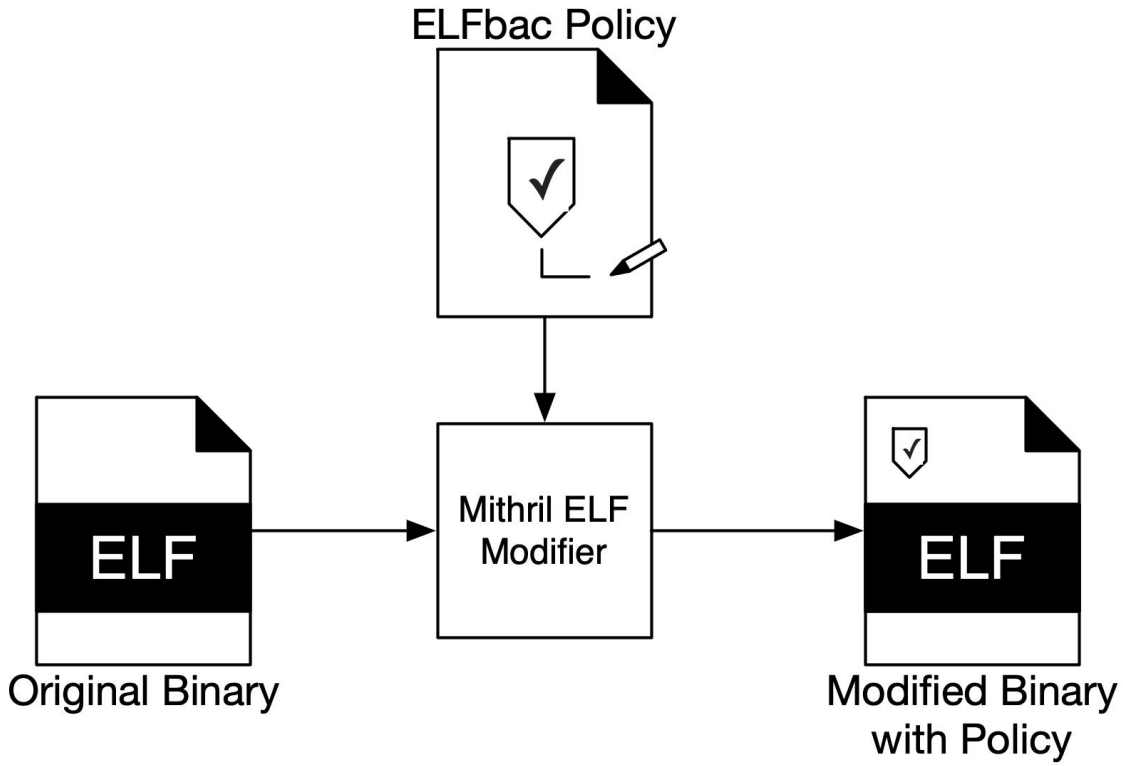
ELF-based Access Control

- ELFbac uses *policy-infused* binaries.
- ELF binaries contain sections and segments.
  - Sections define semantically distinct units of a program: code, data, metadata, etc.
  - Segments group sections.
  - They define the permissions of the memory sections.
- What if we can enforce permissions on the sections instead of the segments? Fine-grained access control in binaries.



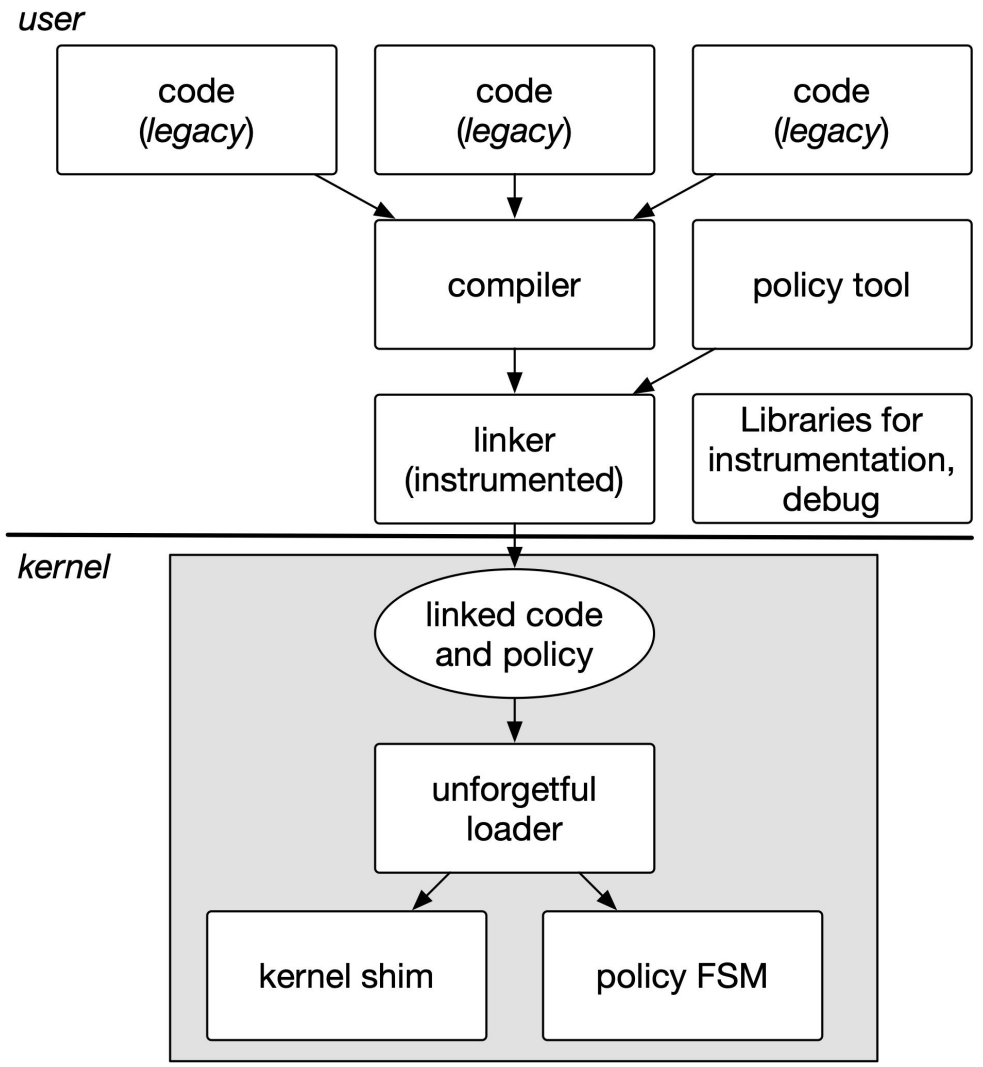
Injecting the policy

- We isolate the global data such as the “secret” in the case of Spectre into a separate section using the __attribute__ gcc syntax.
- The policy is described in a domain-specific language based on Ruby.
- The policy gets added to the binary as a separate ELF section.

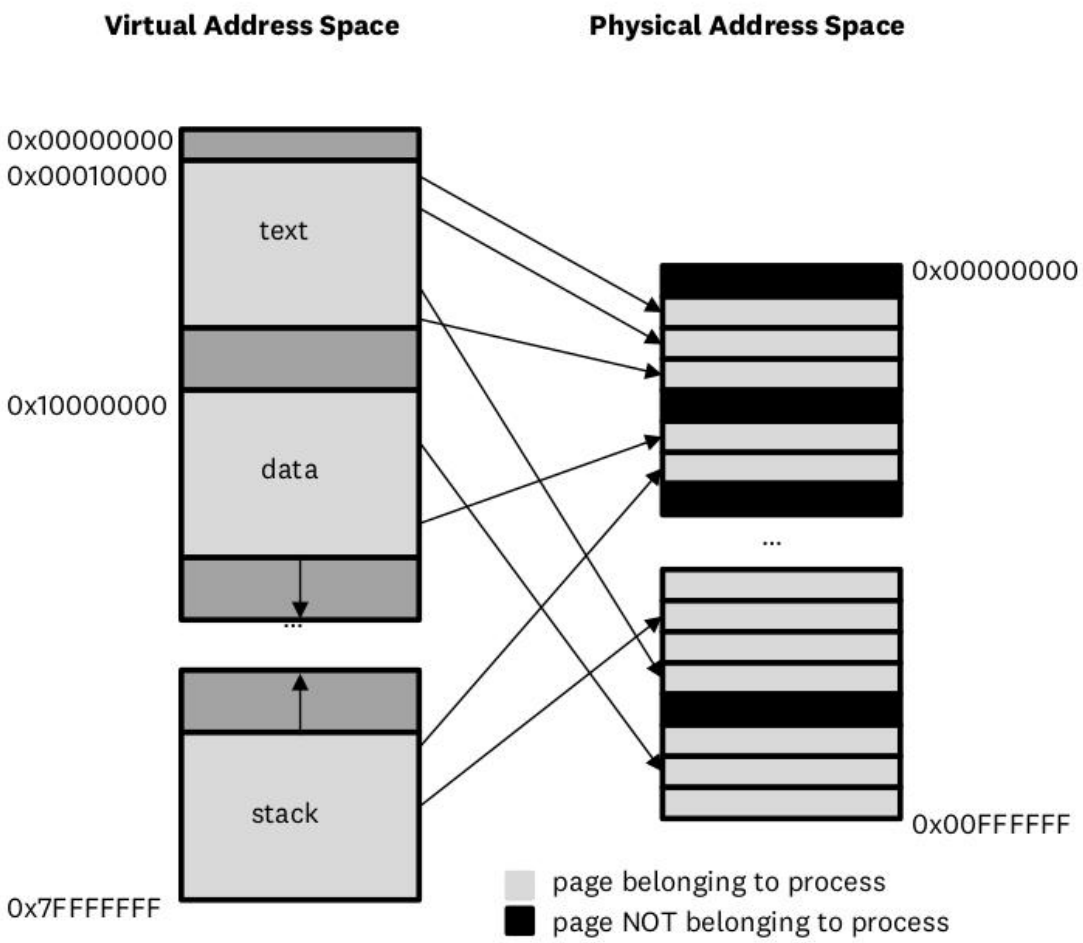


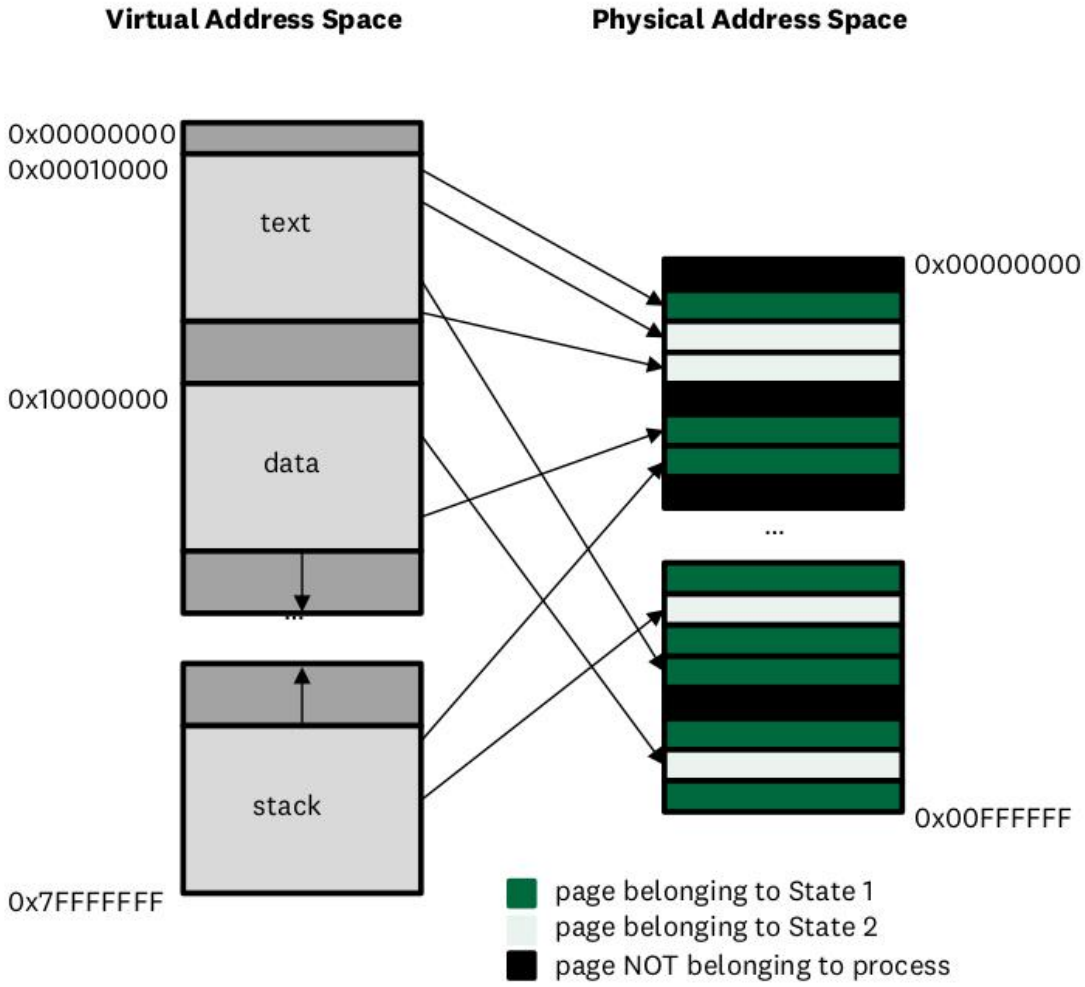
How does ELFbac enforce policy?

- The loader is policy aware.
- The kernel enforces the policy:
  - All the pages are unmapped. At each new access, there is a page fault and the permissions are checked.
  - During a state transition, the TLBs are flushed to invalidate all the entries and the cache.
- So what does the program's address space look like?



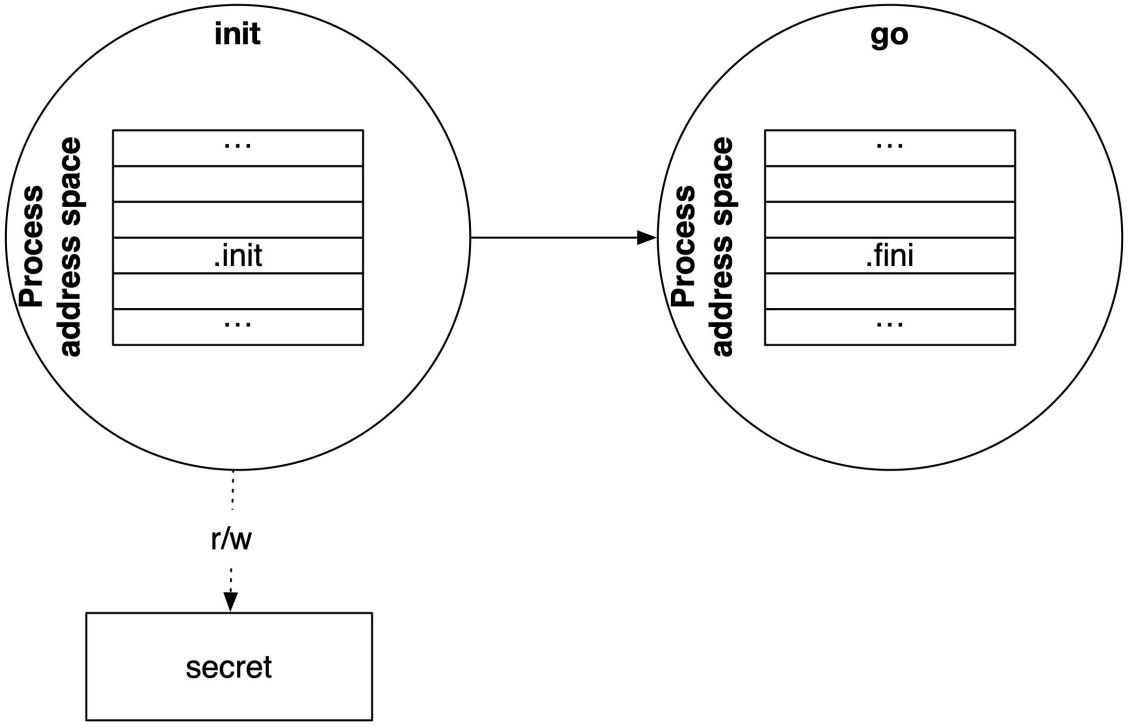
Process view vs. Kernel View



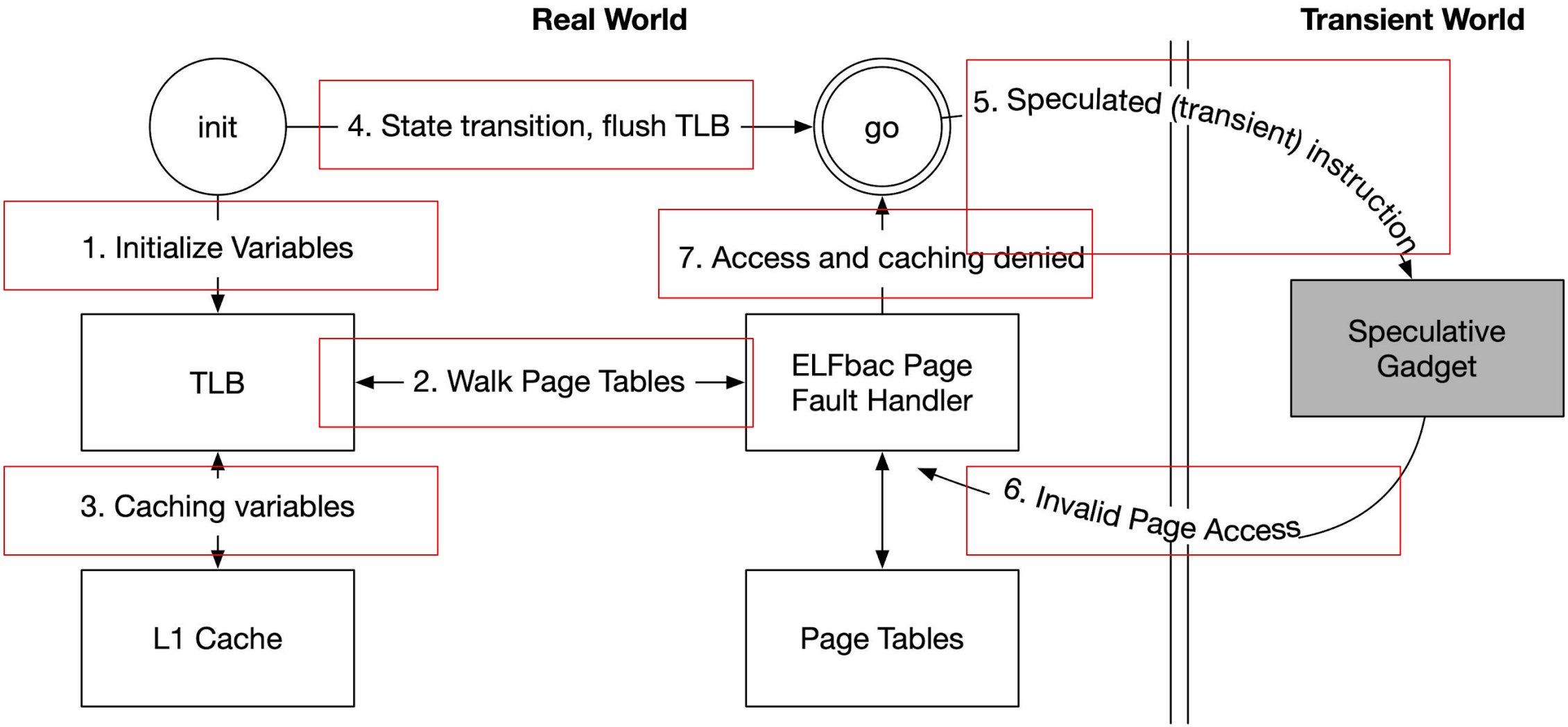


ELFbac policy vs. Spectre

- In the V1 PoC, apart from initialization, the rest of the program does not touch the variable “secret”
- We divide the program into two states.
  - Only the *init* state has access to the secret.
  - The program transitions to the *go* immediately after initialization of all the globals.



ELFbac policy vs. Spectre (contd.)



Limitations in this approach

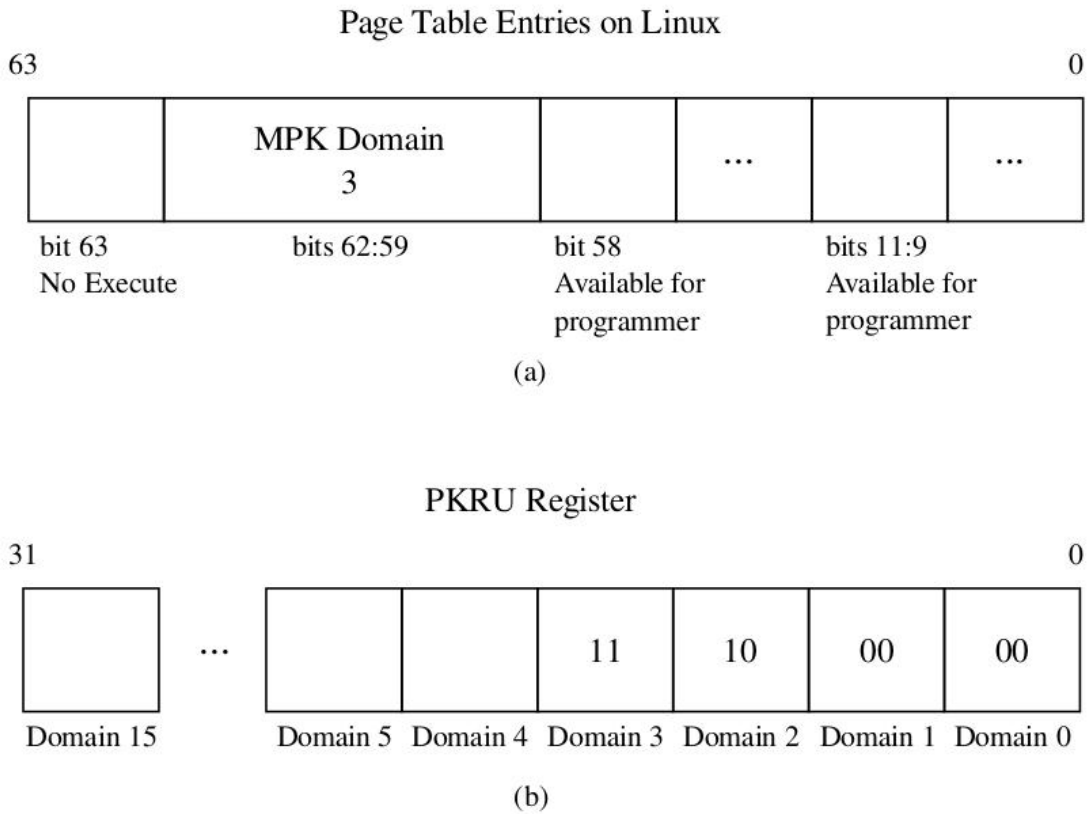
- Since there is a TLB flush on state transitions, there is a huge performance hit.
- The performance hit gets compounded because the pages are lazily loaded.
- The number of page faults is much higher as we'll see in the evaluation.
- Maybe there is another way to enforce intraprocess isolation?

Outline

- Spectre V1
- ELF-based Access Control
- **Memory Protection Keys**
- Evaluation
- Conclusions

Memory Protection Keys

- Since the permission enforcement happens via page faults and TLB flushes, this does incur a huge overhead.
- Page table entries on Linux include 4 bits reserved for the security domain or state in which this page would be accessible.
- The PKRU register stores 2 bits for each state or security domain: read and write permissions for the domain.



Memory Protection Keys (contd.)

- We implemented the same state machine as earlier.
  - Init state where initialization is allowed.
  - *Go* state where access to the secret is revoked.
- We revoked permissions to the secret after it was initialized.
- WRPKRU is a user-land instruction.

```
int real_prot = PROT_NONE;

int pkey = pkey_alloc (0,
PKEY_DISABLE_WRITE);

int ret = pkey_mprotect(secret ,
getpagesize (), real_prot , pkey);
```

Outline

- Spectre V1
- ELF-based Access Control
- Memory Protection Keys
- **Evaluation**
- Conclusions

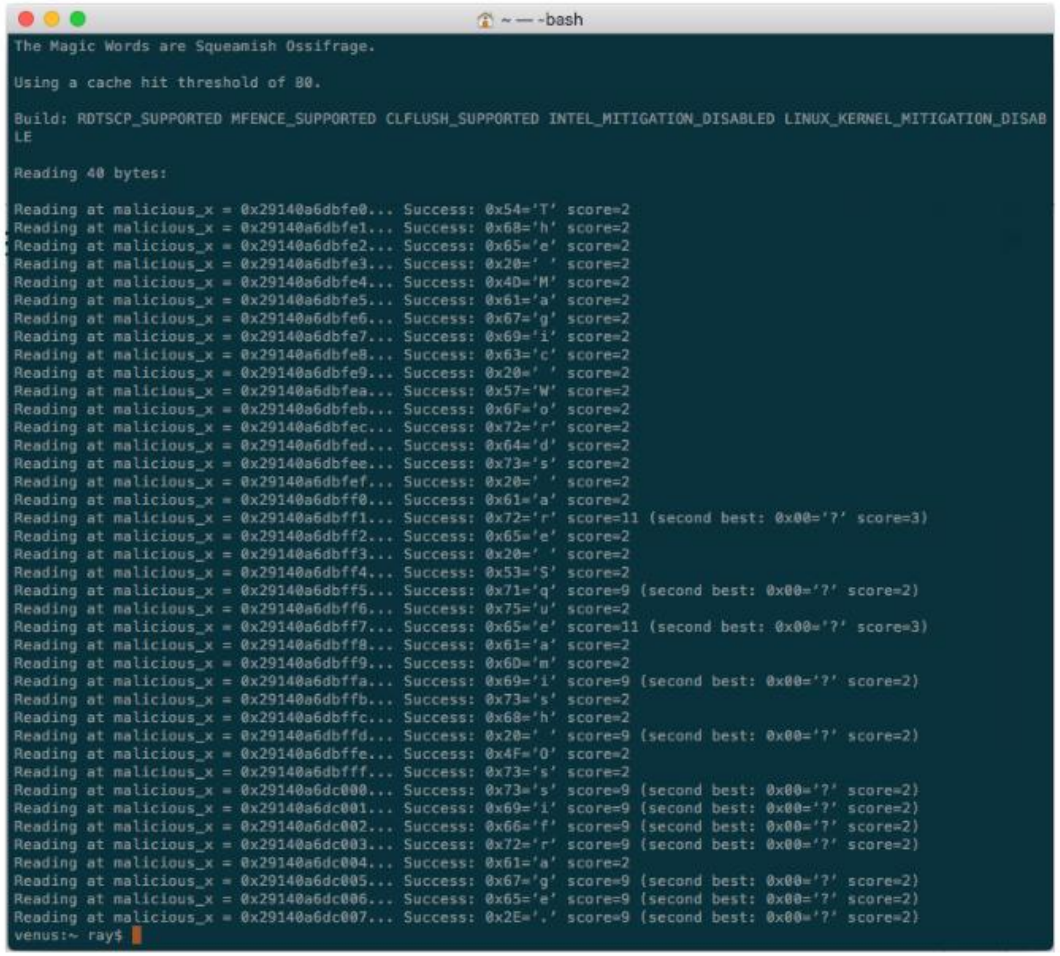
Evaluation

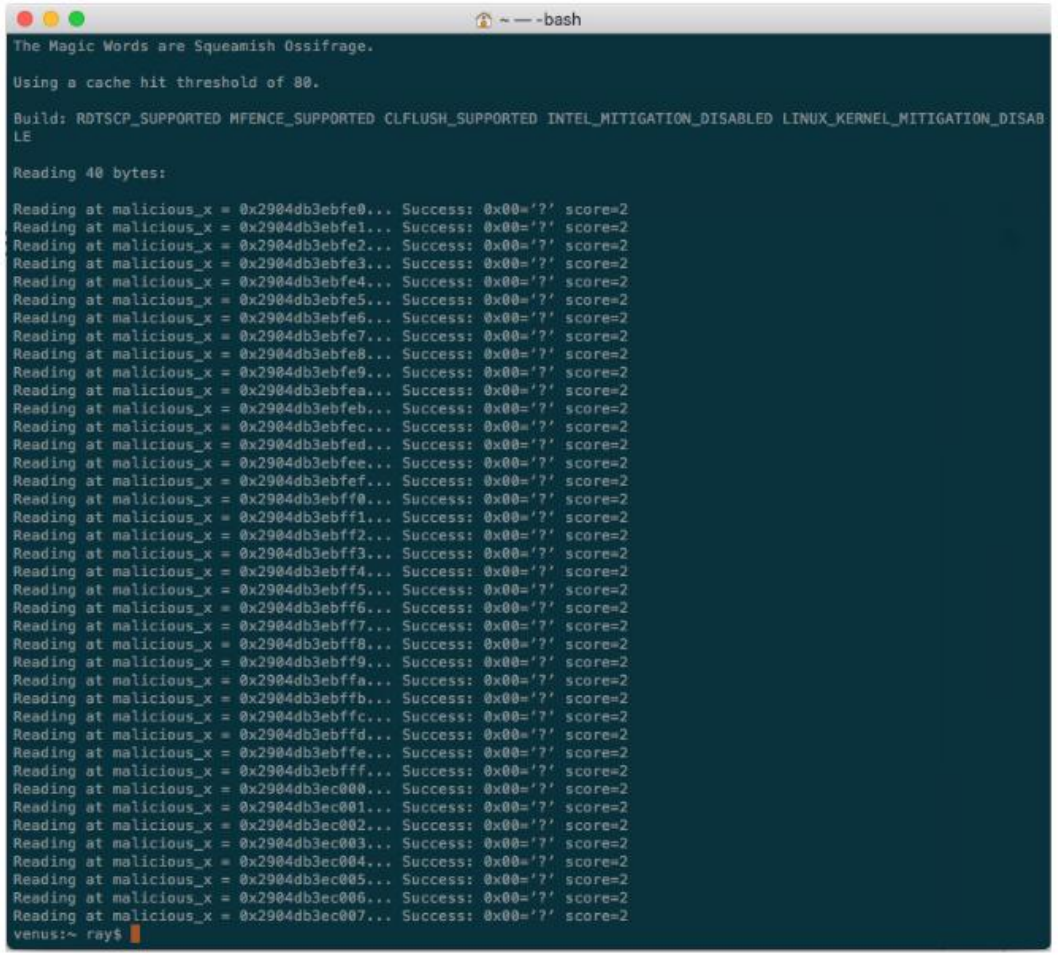
- Is intraprocess memory isolation effective against SpectreV1？
- What is the programmer effort required to build a policy for ELFbac and to modify the existing source code？
- How does ELFbac compare in terms of programmer effort to other mitigation techniques against Spectre V1？
- What is the performance impact due to ELFbac and MPKs in comparison to other mitigations？

ELFbac and MPKs vs. Spectre V1

- We built two policies for ELFbac: one allowing Spectre V1 PoC to execute, and another to disallow it.
- We also built two modifications of our MPK implementation to again allow and disallow the attack.
- In both cases, when the protections are turned on, we found that the secret was not found since the speculative branch is unable to access the secrets.

Demo of the PoC





Programmer Effort

- Using serializing instructions such as *lfence* would only include adding one line of code.
- However, we would need to identify every instance of code that can be speculatively executed and add an *lfence*.
- The process of building the right ELFbac policy involves a lot of trial and error.

	LoC added for ELFbac	LoC added for MPKs
Original Spectre V1 PoC	3	5
Policy code in DSL	33	0

Performance

- We performed our ELFbac experiments on an Intel Xeon E31245 3.30 GHz processor with four cores and 4GB RAM running a modified ELFbac kernel and Loader.
- MPK experiments were done on an Intel Xeon Platinum 8168 instance on Microsoft Azure Cloud with support for MPKs with one core and 2GB RAM.

	Page Faults	Context Switches	Time Elapsed	State Transitions
Original Spectre PoC	170	88	0.01s	NA
`lfence` solution	170	89	0.02s	NA
Spectre V1 exploit with ELFbac Policy 1	304	86	0.01s	0
Spectre V1 exploit with ELFbac Policy 2	320	92	1.31s	1
Spectre V1 mitigation with ELFbac Policy 2	320	98	1.36s	1
Spectre Allowed with MPKs	92	83	0.02s	NA
Spectre V1 mitigation with MPKs	92	83	0.01s	NA

Discussion and Conclusions

- Our work using ELFbac and MPKs are isolated to Intraprocess memory attacks such as Spectre V1.
  - SpectreRSB and Spectre 1.1 are also intraprocess memory attacks and *could* be mitigated using the same technique.
  - SpectreRSB attacks exploiting multiple processes and the Intel SGX, however, are not in the scope of ELFbac that targets intraprocess memory attacks.
- ELFbac does need some speed enhancements. We are working on a version of ELFbac that uses MPKs for intraprocess isolation.
- Neither ELFbac nor MPKs mitigate vulnerabilities entirely, but isolate them and make life harder for attackers.

Thank You

https://prashant.at
prashant.anantharaman.gr@dartmouth.edu

Ray Jenkins: jenkins@cs.dartmouth.edu
Rebecca Shapiro: bx@narfindustries.com
Sean Smith: sws@cs.dartmouth.edu
Sergey Bratus: sergey@cs.dartmouth.edu
J. Peter Brady: jpb@cs.dartmouth.edu