# Comprehensive Kernel Safety in the Spectre Era: Mitigations and Performance Evaluation (Extended Version)

Davide Davoli, Martin Avanzini, and Tamara Rezk Inria, Université Côte d'Azur

November 28, 2024

#### Abstract

The efficacy of address space layout randomization has been formally demonstrated in a shared-memory model by Abadi et al., contingent on specific assumptions about victim programs. However, modern operating systems, implementing layout randomization in the kernel, diverge from these assumptions and operate on a separate memory model with communication through system calls. In this work, we relax Abadi et al.'s language assumptions while demonstrating that layout randomization offers a comparable safety guarantee in a system with memory separation. However, in practice, speculative execution and side-channels are recognized threats to layout randomization. We show that kernel safety cannot be restored for attackers capable of using side-channels and speculative execution, and introduce enforcement mechanisms that can guarantee speculative kernel safety for safe system calls in the Spectre era. We implement two suitable mechanisms and we use them to compile the Linux kernel in order to evaluate their performance overhead.

# 1 Introduction

Memory safety violations on kernel memory can result in serious ramifications for security, such as e.g. arbitrary code execution, privilege escalation, or information leakage. In order to mitigate safety violations, operating systems—such as Linux—employ address space layout randomization [27, 40, 61, 55, 26, 56]. This protection measure can prevent attacks that depend on knowledge of specific data or function location, as it introduces randomization of these addresses.

On the one hand, the efficacy of layout randomization has been formally demonstrated in Abadi et al.'s line of work [2, 1, 4], as a protective measure within a shared-memory model between the attacker and the victim. These results, however, are contingent on specific assumptions regarding victim programs, notably the absence of pointer arithmetic, introspection, or indirect jumps. These precise constraints shaped a controlled environment where memory safety could be enforced effectively via layout randomization. However, operating systems employing layout randomization on kernel (a.k.a. KASLR in Linux e.g. [27]) diverge from these assumptions. Notably, they operate on a separate memory model, wherein, kernel code—acting as the victim—resides on kernel memory, while user code—acting as the potential attacker—resides in user space. The interaction between the two occurs through a limited set of functions provided via system calls [68]. In the operating system's realm, system calls may be written in C and assembly code, further deviating from the restricted conditions outlined by Abadi et al. This introduces a distinction not only in the expressiveness of victim code considered but also in the underlying memory model.

Hence, our first research question emerges: can we relax the language assumptions proposed by Abadi et al. [2, 1, 4] while concurrently demonstrating that layout randomization offers a comparable safety guarantee in a system with memory separation? We affirmatively respond to this question by

showcasing that layout randomization probabilistically ensures kernel safety within a classic attacker model, where users of an operating system execute without privileges and victims can feature pointer arithmetic, introspection, and indirect jumps.

On the other hand, in the current state-of-the-art of security, often referred to as the Spectre era, speculative execution and side-channels are well known to be effective vectors for compromising layout randomization [34, 50, 35, 51, 45, 14]. Indeed, our first result neglects the impact of speculative execution and side-channels. Recognizing this limitation, our second research question arises: can we restore a similar safety result in the Spectre era?

In this regard, we formally acknowledge that by relying solely on layout randomization it is not possible to restore kernel safety. We then introduce a new condition, called *speculative layout non-interference* akin to speculative constant-time [16], which intuitively asserts that victims should not unintentionally leak information on the kernel's layout through side-channels. Our research formally demonstrates that under this assumption, the system is safe, and perhaps surprisingly, without the necessity of layout randomization. Later, we show that speculative layout non-interference is not a necessary requirement, and this motivates us to study how safety can be enforced without requiring that property.

Our third contribution is to show that kernels can be protected even without requiring speculative layout non-interference. Following the approach of other similar works in this field [71, 22], we do so by relating safety in the classic execution model to the speculative one. Specifically, we show that a kernel that is safe against classic attackers can be protected against speculative attackers by applying specific program transformations. However, the initial safety requirement that we impose on the kernel cannot be granted solely by layout randomization. Finally, we show the soundness of three such transformations, and we implement them in order to evaluate their overhead in terms of performance.

This marks the first formal step toward strengthening kernel safety in the presence of speculative and side-channel vulnerabilities, and the surpassing of layout randomization as a system level protection mechanism.

In summary, our contributions are:

- We formally demonstrate the effectiveness of layout randomization to provide kernel safety for a classic operating system scenario, with system calls offered as interfaces to attackers and different privilege execution modes, as well as kernel and user memory separation.
- We empower the attackers of our first scenario to execute side-channel attacks and control
  speculative execution. We demonstrate that kernel safety is not maintained under this more
  potent attacker model, and we subsequently present a sufficient condition to ensure kernel safety.
- We show that it is possible to enforce safety against speculative attackers on a system that enjoys weaker security guarantees by the application of a program transformation.
- We implement these instrumentations, and we measure their overhead on the Linux kernel. The experimental evaluation shows that they impose low performance overheads on computationally heavy user-space tasks.

This paper is an extended version of the ACM CCS conference paper [24] on kernels' safety in presence of speculative execution. The additional material in this work, compared to the conference version, is the following:

- We model indirect branch prediction, which allows us to consider attacks related to Spectre v2, making the attack model stronger than in the conference paper (Section 6).
- We review state-of-art mitigations against speculative execution attacks, showing that none of them can easily be adopted to enforce kernel's safety in presence of speculative execution (Section 7).

```
int buf[K+1][H];
int recv(socket* s, size_t idx) {
  if (valid(s, idx)) return buf[*s][idx];
  return 0;
}

void send(socket* s, size_t idx, int msg) {
  if (valid(s, idx)) {
    buf[*s][idx] = msg;
    if (buf[K][0] != NULL) (*buf[K][0])(s, idx);
}
}
```

Figure 1: System Calls vulnerable to memory corruption

- We extend the transformation given in the conference version of this paper [24, Section 7] to protect against indirect branch speculation.
- We define two additional transformations. The first one is an optimized version of the transformation given in the conference version of this paper [24, Section 7], and the other one blocks all forms of *kernel-space* speculation taken in account in this work (Section 8).
- We implement the transformations as LLVM passes, and we offer them as open source [23].
- We evaluate the performance overhead of these transformations using the SPEC® CPU 2017 [13] and UnixBench [53] that are state-of-art benchmarks to evaluate the performance of computationally heavy user-space and kernel-space tasks, respectively (Section 9).

The paper is structured as follows: in Section 2 we give an overview of the contributions of this paper, motivated by some concrete examples. In Section 3, we introduce our execution model by giving its language and semantics; in Section 4, we establish threat models. Section 5 is devoted to showing that layout randomization is an effective protection measure for attacks that do not rely on speculative execution and side-channel observations. In Section 6 we first extend the model of Section 3 to encompass time-channel info leaks and speculative execution, then we show that layout randomization is not a viable protection mechanism in this scenario. State-of-art mitigations against speculative attacks are reviewed in Section 7, in Section 8 we show that it is feasible to convert any system that is safe against classic attackers into an equivalent system that is safe against speculative attackers, and we propose three suitable program transformations for this task. In Section 9, we estimate the overhead of this transformation on real hardware. Finally, we consider related work in Section 10, and we conclude in Section 11.

Omitted proofs from Section 5 are available in the extended version of the conference version of this paper [25], proofs from Sections 6 and 8 are in Appendices A.1 and A.2, respectively.

# 2 Motivation

Each year, dozens of vulnerabilities are found in commodity operating systems' kernels, and the majority of them are memory corruption vulnerabilities [66]. A kernel suffers a memory corruption vulnerability when an unprivileged attacker can trigger it to read or write its memory in an *unexpected* way, usually, by issuing a sequence of system calls with maliciously crafted arguments. In Figure 1, we show a pair of system calls of a hypothetical system that are subject to this kind of vulnerability. The recv and send system calls are meant to implement a simple message passing protocol. The implementation supports up to K sockets, each socket can buffer up to H messages. A user can send messages by invoking the system call send, and read them with the system call recv. These system calls employ a shared buffer buf that stores messages, together with a hook for a customizable callback

pointed by buf [K] [0]. If specified, this callback is executed after a message is sent. Such a callback may, for instance, inform the sender on whether the message was sent correctly.

These system calls are meant to interact only with the memory containing the buffer, the code of the called functions and with the resources that these function in turn access. In the following, we will refer to the set of memory resources that a system call may access rightfully as the *capabilities* of that system call. Depending on the implementation of the valid function, these system calls can suffer from memory corruption vulnerabilities. For instance, if the valid function does not perform any bound checks on the value of idx, these two system calls can be used by the attacker to perform arbitrary read and write operations. In particular, if the attacker supplies an out-of-bounds value for idx to the recv system call, the system call can be used to perform an unrestricted memory read. Similarly, the send system call can be used to overwrite any value of kernel memory and, in particular, to overwrite the function pointer to the callback that is stored within the buffer. This means that the attacker can turn this memory-vulnerability into a control-flow vulnerability, as it can deviate the control flow from its intended paths. When this happens, we talk about violations of *control flow integrity* (CFI) [3].

However, if the system that implements these system calls is protected with layout randomization—like many commodity operating systems do [27, 40, 61, 55, 26, 56]—the exploitation of these vulnerabilities is not a straightforward operation. In Linux, for instance, one of the viable ways to mount a privilege-escalation attack is to disable the Supervisor Mode Execution Prevention (SMEP) by running the native\_write\_cr4 function. When this protection is disabled, the kernel is allowed to run any payload stored in user-space. The attacker can trigger the system to execute the payload, by exploiting the vulnerability of the send system call twice: the first time to run the native\_write\_cr4 function, and the second time to run the payload. However, in order to do so, the attacker has to infer the address of native\_write\_cr4. In the absence of info-leaks, an attacker can only guess this address and, due to layout randomization, the probability of success is low.

This is what we show with our first result (Theorem 1): without side-channel leaks (and speculative execution), if a system is protected with layout randomization, the probability that an unprivileged attacker leads the system to perform an unsafe memory access is very low, provided the address space is sufficiently large. Of course, the precise probability depends on the concrete randomization scheme. We emphasize that this result is compatible with the large number of kernel attacks that break Linux's kernel layout randomization, e.g. by means of heap overflows [65]. The distribution of Linux's heap addresses lack entropy [30], in consequence, the probability of mounting a successful attack are relatively high.

Although it was already well known that layout randomization can provide some security guarantees [2, 1, 4], the novelty of Theorem 1 lies in showing that these guarantees are valid even if victims can perform pointer arithmetic and indirect jumps.

Despite this positive result, the threat model considered in Theorem 1 is unrealistic nowadays. In particular, it does not take in account the ability of the attackers to access side-channel info-leaks and to steer speculative execution. There is evidence that, by leveraging similar features, the attackers can leak information on the kernel's layout [35, 39, 45, 51, 14] and compromise the security guarantees offered by layout randomization [34, 50].

In particular, if the system under consideration suffers from side-channel info-leaks that involve the layout, an attacker may break the protection offered by randomization. As an illustrative example, suppose the system contains the following system call:

```
int sc_leak(x){
  if ((void*) x == (void*) native_write_cr4)
    for(int i = 0; i < K; i++);
  return 0;
}</pre>
```

By measuring the execution time of the system call, an attacker may deduce information on the location of native\_write\_cr4. If a call sc\_leak(a) takes sufficiently long to execute, the attacker

can deduce that the address a corresponds to that of native\_write\_cr4. Once deduced, the attacker will be effectively able to disable SMEP protection via the vulnerable system call send.

Similar attacks can be mounted by taking advantage of speculative execution: in our example from Figure 1, an attacker can make use of the recy primitive to probe for readable data without crashing the system. This can be done by supplying to the system call arguments s and idx such that valid(s, idx) returns false—ideally, causing an out of bound access when the return value is fetched from memory. If the attacker manages in mis-training the branch predictor, the access to buf [\*s] [idx] is performed in transient execution. Depending on the allocation state of the address referenced by buf[\*s][idx], two cases arise. If that address does not store any readable data, the memory violation is not raised to the architectural state, because it occurred during transient execution. Most importantly, if that address stores writable data, this operation loads a new line in the system's cache and, as soon as the system detects the mis-prediction, the execution backtracks to the latest architecturally valid state. Although this operation does not affect the architectural state, the insertion of a new line in the cache can be detected from user-space. Thus, the attacker can infer that the address referenced by buf [\*s] [idx] contains readable data, and it can make use of the vulnerabilities of the send and the recv system calls to read or write the content of that memory address. This form of speculative probing is very similar to what happens, for instance, in the BlindSide attack [34] that effectively defeats Linux's KASLR.

The reader may observe that such attacks rely on the attacker's ability to reconstruct the kernel's memory layout by collecting side-channel info-leaks. For this reason, a natural question is whether these attacks can be prevented by imposing that no information of the layouts leaks to the architectural and the micro-architectural state during the execution of system calls. It turns out that this is the case, as we show in Theorem 2. In practice, this mitigation is of little help though, as it would effectively rule out all system calls that access memory at runtime.

However, we are able to show that any operating system can be pragmatically turned into another system that is architecturally equivalent to it, but that is not subject to vulnerabilities that are due to transient execution. This can be achieved by program transformation. With this approach, showing that a kernel is safe in the speculative execution model reduces to showing that the kernel under consideration is safe in the classic execution model. Notably, this holds independently of the technique that is used to enforce safety in the classic model. Concretely, with this approach, the speculative attack to the recv system call that we described above would be prevented by disallowing the transient execution of the unsafe load operation. In turn, this can be achieved by placing an instruction that stops transient execution before that operation. The efficacy of this technique is formally shown in Section 8.

Although partially blocking kernel-space speculative execution can have deleterious ramifications for performance, computer systems' execution takes place mostly in user space, amortizing the overhead that the system encounters in kernel-space. An experimental evaluation given in Section 9 confirms this claim.

# 3 The Language

In this section, we introduce the language that we employ throughout the following to study the effectiveness of kernel address space layout randomization.

Syntax and informal semantics. We are considering a simple imperative while language. The address space is explicit, and segregated into user and kernel space. The set Cmd of commands is given in Figure 2. Memories may store procedures and arrays, i.e., sequences of values  $v \in Val$  organized as contiguous regions. The set of values is left abstract, but we assume that it encompasses at least  $Boolean\ values\ Bool \triangleq \{true, false\},\ (memory)\ addresses\ Addr,\ and\ an\ undefined\ value\ null.$  Within expressions,  $x \in Reg\ ranges\ over\ registers$ ,  $a \in Arrld\ and\ f \in Funld\ over\ array\ and\ procedure\ identifiers$ , and  $op \in Ops\ over\ operations$ .  $Identifiers\ ld\ \triangleq Arrld\ \uplus\ Funld\ are\ mapped\ to\ addresses\ at$ 

```
\begin{aligned} \mathsf{Expr} \ni \mathsf{E}, \mathsf{F} &::= v \;\middle|\; x \;\middle|\; \mathsf{a} \;\middle|\; \mathsf{f} \;\middle|\; \mathsf{op}(\mathsf{E}_1, \dots, \mathsf{E}_n) \\ \mathsf{Instr} \ni \mathsf{I}, \mathsf{J} &::= \mathsf{skip} \;\middle|\; x := \mathsf{E} \;\middle|\; x \; := \mathsf{*E} \;\middle|\; \mathsf{*E} := \mathsf{F} \;\middle|\; \mathsf{call} \; \mathsf{F}(\mathsf{E}_1, \dots, \mathsf{E}_n) \;\middle|\; \mathsf{syscall} \; \mathsf{s}(\mathsf{E}_1, \dots, \mathsf{E}_n) \;\middle|\; \\ & \quad \mathsf{if} \; \mathsf{E} \; \mathsf{then} \; \mathsf{C} \; \mathsf{else} \; \mathsf{D} \; \mathsf{fi} \;\middle|\; \mathsf{while} \; \mathsf{E} \; \mathsf{do} \; \mathsf{C} \; \mathsf{od} \\ \mathsf{Cmd} \ni \mathsf{C}, \mathsf{D} &::= \epsilon \;\middle|\; \mathsf{I}; \mathsf{C} \end{aligned}
```

Figure 2: Syntax of the language. In expressions v is a value, x a register, a an array identifier, f a function identifier, and op is an operator.

runtime, as governed by a layout randomization scheme. The *size* (length) of an array a is denoted by **size**(a) and is fixed for simplicity, i.e., we do not model dynamic allocation and deallocation.

A command  $C \in Cmd$  is a sequence of instructions, evaluated in-order. The instruction x := E stores the result of evaluating E within register  $x \in Reg$ . To keep the semantics brief, expressions neither read nor write to memory. Specifically, addresses are dereferenced explicitly. To this end, the instruction x := \*E performs a memory read from the address given by E, and stores the corresponding value in register x. Dually, the instruction \*E := F stores the value of F at the address given by E. The instruction call  $F(E_1, \ldots, E_n)$  invokes the procedure residing at address F in memory, supplying arguments  $E_1, \ldots, E_n$ . Likewise, syscall  $s(E_1, \ldots, E_n)$  invokes a system call  $s \in Sys$  with arguments  $E_1, \ldots, E_n$ . The execution of a system call engages the privileged execution mode and thereby the accessible address space changes. To this end, the address space Addr is partitioned into  $\kappa_u$  user-space addresses Addr  $ext{u} = \{0, \ldots, \kappa_u\}$ , visible in unprivileged mode, and  $\kappa_k$  kernel-space addresses Addr  $ext{u} = \{\kappa_u, \ldots, \kappa_u + \kappa_k - 1\}$ , visible in unprivileged mode. The remaining constructs are standard.

Stores Regarding the address space, we categorize identifiers Id into two disjoint sets: kernel-space identifiers  $Id_k$  and user-space identifiers  $Id_u$ . This distinction signifies the intended location of the corresponding objects within the memory address space. We write  $Funld_k \subseteq Id_k$  and  $Funld_u \subseteq Id_u$  for the kernel-space and user-space procedure identifiers; similar for array identifiers we use  $ArrId_k \subseteq Id_k$  and  $ArrId_u \subseteq Id_u$  to denote kernel-space and user-space array identifiers respectively. Let  $Arr^{(n)}$  denote the set of arrays of size n, i.e., finite sequence  $\vec{v}$  of values of fixed length  $|\vec{v}| = n$ , and Arr the set of arrays of arbitrary size.

A store is a (well-sorted) mapping  $\tau: \operatorname{Id} \to \operatorname{Arr} \cup \operatorname{Cmd}$ , mapping procedure identifiers f to their implementation  $\tau(f) \in \operatorname{Cmd}$  and array identifiers a to arrays  $\tau(a) \in \operatorname{Arr}$  of size  $|\tau(a)| = \operatorname{size}(a)$ . Given a store  $\tau$ , we will always assume that the address space is sufficiently large to hold  $\tau$ ; that is,  $\kappa_b \geq \sum_{id \in \operatorname{Id}_b} \operatorname{size}(id)$  ( $b \in \{u, k\}$ ). Here, by convention,  $\operatorname{size}(f) \triangleq 1$ . In the following, we often work with pairs of stores that associate a set identifiers to the same values—e.g. that store the same procedures, but possibly different arrays. To this aim, we write  $\tau =_{Id} \tau'$  if  $\tau$  and  $\tau'$  coincide on  $Id \subseteq \operatorname{Id}$ .

Capabilities To model our notion of safety, each system call s is associated at runtime with a fixed set of identifiers that it is meant to access. In the following, we call that set the *capabilities* of s. This set identifies those memory areas that are safe to access, when a certain system call is running.

**Systems** Let Sys denote a (finite) set of system call identifiers, and let  $ids(C) \subseteq Id$  refer to the set of identifiers literally occurring in C. A system for Sys is a tuple  $\sigma = (\tau, \gamma, \xi)$ , consisting of:

- a store  $\tau: \mathsf{Id} \to \mathsf{Arr} \cup \mathsf{Cmd}$ , relating identifiers to their initial value;
- a system call map  $\gamma: \mathsf{Sys} \to \mathsf{Cmd}$  associating system calls to their implementation; and

• a capability map  $\xi: \mathsf{Sys} \to \mathcal{P}(\mathsf{Id}_k)$  associating system calls with their capabilities.

Systems define the interpretation of procedure system calls by specifying their bodies, therefore—once a system  $\sigma = (\tau, \gamma, \xi)$  is fixed—the set of identifiers that are referenced by a program  $\mathbb{C}$  can be explicitly given. Specifically,  $\mathsf{refs}_{\sigma}(\mathbb{C})$  is the least set I containing identifiers occurring in the body of  $\mathsf{s}$  ( $\mathsf{ids}(\gamma(\mathsf{s})) \subseteq I$ ), and that is closed under procedure calls (i.e., if  $\mathsf{f} \in I$  then  $\mathsf{ids}(\tau(\mathsf{f})) \subseteq I$ ).

**Safety** Our notion of safety is defined in terms of system calls' capabilities, by asking that every system call only accesses those resources that are on its capabilities. To prevent trivial memory safety violations, we impose the following two restrictions on all systems  $\sigma = (\tau, \gamma, \xi)$ :

- (i) the code  $\tau(f)$  associated to user space identifiers  $f \in \mathsf{Funld}_u$  is  $\mathit{unprivileged}, i.e. \mathsf{ids}(C) \subseteq \mathsf{Id}_u$ ;
- (ii) the capabilities  $\xi(s)$  of a system call contains all the identifiers that s refers in its body, i.e.  $\operatorname{refs}_{\sigma}(\gamma(s)) \subseteq \xi(s)$ .

Observe that by dropping (i) a user-space function may return information on the kernel layout, such as the address of a kernel function, to the attacker, undermining the protection offered by layout randomization. Restriction (ii) extends the capabilities of a system call to cover all the identifiers that appear in its code, or in the code of the procedures that it may rightfully execute at runtime.

For the sake of simplicity, we also assume that neither kernel-space procedures nor system calls perform system calls themselves. These assumptions, although not substantial, are usually verified by commodity operating systems.

Memories A store  $\tau$  determines the contents of a memory, but not its layout, in contrast, memories—modeled as functions  $m: \mathsf{Addr} \to \mathsf{Val} \cup \mathsf{Cmd} \cup \{*\}$ —associate addresses with their content (if any), or to the special symbol  $* \not\in \mathsf{Val}$  if the location is not occupied. We denote by Mem the set of all memories. As memory locations can only store values or programs, arrays will be represented by sequences of values in contiguous memory locations. Given a memory m, a value  $v \in \mathsf{Val}$ , and an address  $p \in \mathsf{Addr}$ , the notation  $m[p \leftarrow v]$  denotes the memory that is pointwise identical to m, except for the address p that is mapped to p. This operation is not defined when p is not a value, and this distinction will help in preserving the distinction of procedures from values in the semantics, thereby modeling a W^X memory protection policy, separating writable from executable memory space.

Layouts A (memory) layout is a function  $w: \mathsf{Id} \to \mathsf{Addr}$  that describes where objects are placed in memory. As we mentioned, an array  $\mathsf{a}$  is stored as continuous block at addresses  $\underline{w}(\mathsf{a}) \triangleq \{w(\mathsf{a}), \dots, w(\mathsf{a}) + \mathsf{size}(\mathsf{a}) - 1\}$  within memory. For procedure identifiers  $\mathsf{f}$ , we set  $\underline{w}(\mathsf{f}) \triangleq \{w(\mathsf{f})\}$ . We overload this notation to arbitrary sets of identifiers in the obvious way. In particular,  $\underline{w}(\mathsf{ArrId})$  and  $\underline{w}(\mathsf{FunId})$  refer to the address-spaces of arrays and procedures, under the given layout. We regard only layouts that associate identifiers with non-overlapping blocks  $(\underline{w}(\mathsf{id}_1) \cap \underline{w}(\mathsf{id}_2) = \emptyset$  for all  $\mathsf{id}_1 \neq \mathsf{id}_2$ ) and that respect address space separation  $(\underline{w}(\mathsf{id}) \subseteq \mathsf{Addr}_b$  for  $\mathsf{id} \in \mathsf{Id}_b$ ,  $b \in \{\mathsf{u}, \mathsf{k}\}$ ). The set of all such layouts is denoted by Layout. Note that, by the assumptions on  $\kappa_\mathsf{u}$  and  $\kappa_\mathsf{k}$ , Layout is non-empty.

The application of a *layout* to a *store* results in a memory and is defined in the following way:

$$(w \diamond \tau)(p) \triangleq \begin{cases} \mathtt{C} & \text{if } \tau(w^{-1}(p)) = \mathtt{C} \in \mathtt{Cmd}, \\ t & \text{if } w(\mathtt{a}) + k = p, \ \tau(\mathtt{a}) = \vec{v} \in \mathsf{Arr} \ \text{and} \ \vec{v}[k] = t \ \text{for some a} \in \mathsf{ArrId} \ \text{and} \ 0 \leq k < \mathsf{size}(\mathtt{a}), \\ * & \text{otherwise}, \end{cases}$$

where  $\vec{v}[k]$  denotes the k-th element of the tuple  $\vec{v}$ , indexed starting from 0.

Randomization scheme Abstracting from details, we model an address space randomization scheme through a probability distribution over layouts. A specific layout w is selected at random prior to system execution. For a given system  $\sigma = (\tau, \gamma, \xi)$ , this choice then dictates the initial memory configuration  $w \diamond \tau$ . Although the semantics is itself deterministic, computation can be viewed as a probabilistic process. Notably, since instructions are layout-sensitive—for example, the outcome of a memory load operation at a specific address depends on whether w places an object at that address—in this realm, kernel's safety should be construed as a property that holds in a probabilistic sense.

Register maps Together with memory locations, our program can also manipulate the content of register maps, i.e., functions  $\rho: \mathsf{Reg} \to \mathsf{Val}$ , associating a register identifiers to their value. As for memories, given a register map  $\rho$ , a value  $v \in \mathsf{Val}$  and a register  $x \in \mathsf{Reg}$ , the notation  $\rho[x \leftarrow v]$  denotes the register map that is pointwise identical to  $\rho$ , except for x, which is mapped to the value v. Accordingly to what happens with memory, this operation is only defined when v is a value, i.e., registers cannot store arrays or functions.

**Operational semantics** In the following, we endow our while-language with a small-step operational semantics. Due to the presence of (possible recursive) procedures, configurations make use of a stack of frames. Each such frame records the command under evaluation, the register contents and the execution mode. Formally, configurations are drawn from the following BNF:

$$\begin{array}{ccc} b ::= \mathtt{u} & \mathtt{k}_\mathtt{s} & execution \ mode \\ F ::= \varepsilon & \mathsf{c} & \mathsf{C}, \rho, b \rangle : F & frame \ stack \\ C, D ::= (F, m) & \mathsf{err} & \mathsf{unsafe} & configuration \end{array}$$

A configuration is either of the form (F,m), where F is a (non-empty) frame stack and m the current memory. A top-frame  $\langle \mathtt{C}, \rho, b \rangle$  indicates that  $\mathtt{C}$  is executed with allocated registers  $\rho$ , in mode b. In particular,  $b = \mathtt{k_s}$  indicates that execution proceeds in privileged kernel-mode, triggered by system call  $\mathtt{s}$ . The annotation of the *kernel-mode* flag by a system call name facilitates the instrumentation of the semantics. Indeed, every time an access to the memory is made, the semantics enforces that address is in the capabilities of the system call that is running (if any). If the address can be accessed, the execution proceeds regularly, otherwise it halts in the unsafe state. Finally, an error err signals abnormal termination (for instance, when dereferencing a kernel-space reference in user-space mode or vice versa).

The small step operational semantics takes the form

$$w \vdash_{\sigma} C \to D$$
,

indicating that, w.r.t. system  $\sigma$ , configuration C reduces to D in one step, under layout w. The reduction rules are defined in Figures 3 and 4. Rules [Load] implements a successful memory load x := \*E. Expression E is evaluated to an address  $p = \mathbb{E}[A^{\text{Addr}}]$ , and the content of the register x is updated with the value m(p) residing at address p in the memory m. Notice that the semantics of an expression depends, besides register contents, on the layout w that resolves identifiers to memory addresses. The side-condition  $p \in \underline{w}(\mathsf{ArrId}_b)$  enforces that p refers to a value accessible in the current execution mode p0 (by slight abuse of notation, we disregard the system call label in kernel-mode), otherwise the instruction leads to p1 (see rule [Load-Error]). As such, we are modeling unprivileged execution and SMAP protection, preventing respectively the access of kernel-space addresses when in user-mode, and vice versa. The final, boxed, side-condition refers to the safety instrumentation. In kernel-mode, triggered by system call p2 (see rule ensures that p3 refers to an object within the capabilities of p3 (see rule [Load-Unsafe]). When this condition is violated, unsafe execution is signaled (see rule [Load-Unsafe]). In a similar fashion, the rules for memory writes and procedure calls are defined.

$$\overline{w \vdash_{\sigma} (\langle \epsilon, \rho, b \rangle : \langle \mathtt{C}, \rho', b' \rangle : F, m) \to (\langle \mathtt{C}, \rho'[ret \leftarrow \rho(ret)], b' \rangle : F, m)}^{\text{[POP]}}$$

$$\overline{w \vdash_{\sigma} (\langle \mathtt{skip}; \mathtt{C}, \rho, b \rangle : F, m) \to (\langle \mathtt{C}, \rho, b \rangle : F, m)}^{\text{[SKIP]}}$$

$$\overline{w \vdash_{\sigma} (\langle x := \mathtt{E}; \mathtt{C}, \rho, b \rangle : F, m) \to (\langle \mathtt{C}, \rho[x \leftarrow \llbracket \mathtt{E} \rrbracket_{\rho, w}], b \rangle : F, m)}^{\text{[OP]}}}$$

$$\overline{w \vdash_{\sigma} (\langle \mathtt{if} \ \mathtt{E} \ \mathtt{then} \ \mathtt{C}_{\mathtt{true}} \ \mathtt{else} \ \mathtt{C}_{\mathtt{false}} \ \mathtt{fi}; \mathtt{D}, \rho, b \rangle : F, m) \to (\langle \mathtt{C}_{\llbracket \mathtt{E} \rrbracket_{\rho, w}^{\mathtt{Bool}}}; \mathtt{D}, \rho, b \rangle : F, m)}^{\text{[IF]}}}$$

$$\underline{C_{\mathtt{true}} = (\langle \mathtt{C}; \mathtt{while} \ \mathtt{E} \ \mathtt{do} \ \mathtt{C} \ \mathtt{od}; \mathtt{D}, \rho, b \rangle : F, m) \to C_{\llbracket \mathtt{E} \rrbracket_{\rho, w}^{\mathtt{Bool}}}}^{\mathtt{[WHILE]}}}$$

$$w \vdash_{\sigma} (\langle \mathtt{while} \ \mathtt{E} \ \mathtt{do} \ \mathtt{C} \ \mathtt{od}; \mathtt{D}, \rho, b \rangle : F, m) \to C_{\llbracket \mathtt{E} \rrbracket_{\rho, w}^{\mathtt{Bool}}}^{\mathtt{Bool}}}$$

Figure 3: Semantics w.r.t. system  $\sigma = (\tau, \gamma, \xi)$ , first part.

Rule [CALL] deals with procedure calls. It opens a new frame and, by convention, places the n evaluated arguments at registers  $x_1, \ldots, x_n$  in an initial register environment  $\rho_0$ . System calls, modeled by rule [SC], follow the same calling convention. Note that, in the newly created frame, the execution flag is set to kernel-mode. Once a procedure or system call finished evaluation, rule [POP] removes the introduced frame from the stack. Note how the rule permits return values through a designated register ret. The remaining rules are standard.

Let us denote by  $w \vdash_{\sigma} C \to^* D$  that configuration C reduces in zero or more steps to configuration D, and by  $w \vdash_{\sigma} C \uparrow$  that C diverges. In our semantics, under layout w, any non-diverging computation halts in a terminal configuration of the form  $(\langle \epsilon, \rho, b \rangle, w \diamond \tau')$ , or abnormally terminates through an error err or through a safety violation unsafe. This motivates the following definition of an evaluation function:

$$Eval_{\sigma,w}(\mathbf{C},\rho,b,\tau) \triangleq \begin{cases} \Omega & \text{if } w \vdash_{\sigma} (\langle \mathbf{C},\rho,b\rangle,w \diamond \tau) \uparrow, \\ (v,\tau') & \text{if } w \vdash_{\sigma} (\langle \mathbf{C},\rho,b\rangle,w \diamond \tau) \to^* (\langle \epsilon,\rho[ret \mapsto v],b\rangle,w \diamond \tau'), \\ \text{err} & \text{if } w \vdash_{\sigma} (\langle \mathbf{C},\rho,b\rangle,w \diamond \tau) \to^* \text{err}, \\ \text{unsafe} & \text{if } w \vdash_{\sigma} (\langle \mathbf{C},\rho,b\rangle,w \diamond \tau) \to^* \text{unsafe}. \end{cases}$$

Note how, in the case of normal termination, a computation produces a pair of a return value and a store.

## 4 Threat Model

In our threat model, attackers are unprivileged user-space programs that execute on a machine supporting two privilege rings: user-mode and kernel-mode. The victim is the host operating system which runs in kernel mode and has exclusive access to its private memory. In particular, the operating system exposes a set of procedures, the system calls, that can be invoked by the attacker and that have access to kernel's memory. The attacker's goal is to trigger a system call to perform an unsafe memory access.

In Section 5, attackers are ordinary programs that do not control speculative execution and do not have access to side-channel info-leaks. However, the target machine implements standard mitigations

Figure 4: Semantics w.r.t. system  $\sigma = (\tau, \gamma, \xi)$ , second part.

against this kind of attacks. In particular, it supports Data Execution Protection mechanisms (DEP), Supervisor Mode Access Prevention (SMAP) [20] preventing kernel-mode access to user-space data, and Supervisor Mode Execution Prevention (SMEP) [29] preventing the execution of user-space functions when running in kernel-mode. More precisely, the above-mentioned protection mechanisms are modeled in our semantics by the preconditions of the rules [Call, [Load], [Store] that prevent the system from: (i) loading and overwriting functions, (ii) execute values, (iii) accessing user-space data and functions when the system is in kernel-mode. We also assume that the victim hardware supports Intel® Indirect Branch Tracking (IBT) [42], to restrict control flow transfer only to specific program points, typically the beginning of a procedure. This is modeled by restricting the target of indirect call instruction to the beginning of procedures only. Finally, the system adopts kernel address space layout randomization, that is modeled by executing programs with a randomly chosen memory layout.

In Section 6, we consider a stronger threat model where attackers have access to side-channel observations and control *Pattern History Table* (PHT), *Branch Target Buffer* (BTB) predictions and *Store To Load* (STL) forwarding, related to Spectre v1, v2 and v4 vulnerabilities respectively [44]. Moreover, we assume that the system supports *Page Table Isolation* (PTI) [43] to prevent the specu-

lative access of kernel-space memory from user-space; this is modeled by using the same preconditions of rules [Call], [Load], [Store] for their speculative counterparts in Section 6.1.1. Against *Branch Target Injection* (BTI), we assume that the victim's machine supports protection systems akin to retpoline [70] or Intel<sup>®</sup> enhanced Indirect Branch Restricted Speculation (eIBRS) [21] to prevent attackers to speculatively steer the control flow in presence of indirect branches. This is modeled by adding an instruction for speculatively safe jumps to the victim's language.

# 5 Classic Threat Model

In this section we show how the result of Abadi et. al. [2, 1, 4] scales to the model introduced in Section 3. The safety property that we aim at is defined in terms of our instrumented semantics, as follows:

**Definition 1** (Kernel safety). We say that a system  $\sigma = (\tau, \gamma, \xi)$  is *kernel safe*, if for every layout w, unprivileged attacker  $C \in Cmd$ , and registers  $\rho$ , we have:

$$\neg (w \vdash_{\sigma} (\langle C, \rho, u \rangle, w \diamond \tau) \rightarrow^* \text{unsafe}).$$

Thus, safety is broken if an attacker C, executing in unprivileged user mode, is able to trigger a system call in such a way that it accesses, or invokes, a kernel-space object outside its capabilities. The source of such a safety violation can be twofold:

1. Scope extrusion: An obvious reason why kernel-safety may fail is due to apparent communication channels, specifically through the memory and procedure returns. As an illustrative example, consider a system  $\sigma = (\tau, \gamma, \xi)$ , where:

$$\gamma(\mathbf{s}_1) \triangleq *\mathbf{a} := \mathbf{f}$$
  $\gamma(\mathbf{s}_2) \triangleq x := *\mathbf{a}; \text{call } x()$   $\xi(\mathbf{s}_1) = \xi(\mathbf{s}_2) = \{\mathbf{a}\}$ 

A malicious program can use  $s_1$  to store the address of f at a[0], which is a shared capability. A consecutive call to  $s_2$  then breaks safety if f is not within the capabilities of  $s_2$ .

2. **Probing:** Another counterexample is given by a system call accessing memory based on its input, such as the system call s that is defined by  $\gamma$  as follows:

$$\gamma(s) = call x_1().$$

which directly invokes the procedure stored at the kernel-address corresponding to the value of its first argument  $x_1$ . This system call can potentially be used as a gadget to invoke an arbitrary kernel-space function from user-space. Since an attacker lacks knowledge of the kernel-space layout, such an invocation needs to happen effectively through probing. As any probe of an unused memory address leads to an unrecoverable error,<sup>1</sup> the likelihood of an unsafe memory access is, albeit not zero, diminishingly small when the address-space is reasonably large.

To overcome Issue 1, we impose a form of (layout) non-interference on system calls.

**Definition 2** (Layout non-interference). Given  $\sigma = (\tau, \gamma, \xi)$ , a system call **s** is *layout non-interfering*, if,

$$Eval_{\sigma,w_1}(\gamma(s), \rho, k_s, \tau') \cong Eval_{\sigma,w_2}(\gamma(s), \rho, k_s, \tau')$$

for all layouts  $w_1, w_2$ , registers  $\rho$  and stores  $\tau' =_{\mathsf{Funld}} \tau$ . Here, the equivalence  $\cong$  extends equality by identifying the abnormal termination states err and unsafe. The system  $\sigma$  is non-interfering if all its system calls are.

<sup>&</sup>lt;sup>1</sup>This is not always the case for *user-space* software protected with layout randomization, as some programs (e.g. web servers) may automatically restart after a crash to ensure availability. This behavior can be exploited by attackers to probe the entire memory space of the victim program, thus compromising the protection offered by layout randomization [67].

In effect, layout non-interfering systems do not expose layout information, neither through the memory nor through return values. In particular, observe how non-interference rules out Issue 1, as witnessed by two layouts placing f at different addresses in kernel-memory.

Concerning Issue 2, it is well known that layout randomization provides in general safety not in an absolute sense, but probabilistically [2, 12, 69]. Indeed, the chance for a probe to be successful is proportional to the ratio between occupied and free (kernel) memory space. Following Abadi and Plotkin [2], let  $\mu$  be a probability distribution of layouts, i.e., a function  $\mu$ : Layout  $\to$  [0, 1] assigning to each layout  $w \in \text{Layout}$  a probability  $\mu(w)$  (where  $\sum_{w \in \text{Layout}} \mu(w) = 1$ ). Without loss of generality, we assume that the layout of public, i.e. user-space, addresses is fixed. That is, we require for each  $w_1, w_2$  with non-zero probability in  $\mu$ , that  $w_1(\text{id}) = w_2(\text{id})$  for all  $\text{id} \in \text{Id}_u$ . For a system call  $s \in \text{Sys}$ , let  $\text{id}_1^s, \ldots, \text{id}_k^s$  be an enumeration of its capabilities  $\xi(s)$ . The following probability  $\delta_\mu$  quantifies the chance that accessing an address  $p \in \text{Addr}_k$  causes an error (rather than an unsafe access), given that capabilities  $\text{id}_1^s, \ldots, \text{id}_k^s$  are stored at addresses  $p_1, \ldots, p_h$  respectively, and that p does not refer to any of the objects within the capabilities of s.

$$\begin{split} \delta_{\mu} &\triangleq \min \big\{ \Pr_{w \leftarrow \mu} [p \notin \underline{w}(\mathsf{Id}) \mid w(\mathtt{id}_i^{\mathtt{s}}) = p_i, \text{ for } 1 \leq i \leq h] \mid \mathtt{s} \in \mathsf{Sys}, p, p_1, \dots, p_h \in \mathsf{Addr}_{\mathtt{k}} \land \\ & p \notin \{p_i, \dots, p_i + \mathsf{size}(\mathtt{id}_i^{\mathtt{s}}) - 1\}, \text{ for } 1 \leq i \leq h \big\}. \end{split}$$

More concretely,  $\delta_{\mu}$  is the probability that, during the execution of a system call s, a fixed kernel address p is not allocated, given that it does not store any object that is the capabilities of that system call. Notably, if an attacker controls the value of p, this is a lower bound to the probability that its guess fails. We arrive now at the main result of this section:

**Theorem 1** ([25], Theorem 1). Let  $\sigma = (\tau, \gamma, \xi)$  be layout non-interfering. Then

$$\mathbb{P}_{w \leftarrow \mu} \ [w \vdash_{\sigma} ((\mathtt{C}, \rho, \mathtt{u}), w \diamond \tau) \to^* \mathsf{unsafe}] \leq 1 - \delta_{\mu},$$

for any unprivileged attacker  $C \in Cmd$ , and registers  $\rho$ .

Theorem 1 extends the results of [2, 1, 4] by showing that layout randomization guarantees kernel safety probabilistically to operating systems; in contrast with [2, 1, 4], this holds even when victim's code contains unsafe programming constructs such as arbitrary pointer arithmetic and indirect jumps. This is achieved by replacing Abadi and Plotkin [2]'s restrictions on the syntax of the victims with a weaker dynamic property: layout non-interference. Notice that the strength of the security guarantee provided by Theorem 1 depends on the distribution of the layouts  $\mu$ . Therefore, in practice, it is important to determine a randomization scheme that provides a good bound. This can be done quite easily: for instance, if we assume that (i)  $\kappa_k \gg \sum_{i,d \in Id_k} \operatorname{size}(id)$  and that (ii)  $\theta \triangleq \max_{i,d \in Id_k} (\operatorname{size}(id))$  divides  $\kappa_k$ , we can think of the kernel space address range as divided in  $\frac{\kappa_k}{\theta}$  slots, each storing a single object. In this setting, we can define the distribution  $\nu$  as the uniform distribution of all the layouts that store each memory object within a slot starting from the beginning of that slot. For this simple scheme, we can approximate the bound  $\delta_{\nu}$  as the ratio between unallocated slots and all the slots that do not store any object that is referenced by the capabilities of a system call:

$$\delta_{\nu} \geq \min_{\mathbf{s} \in \mathsf{Sys}} \frac{\kappa_{\mathbf{k}}/\theta - \mathsf{size}(\mathsf{Id}_{\mathbf{k}})}{\kappa_{\mathbf{k}}/\theta - \mathsf{size}(\xi(\mathbf{s}))}.$$

In particular, the fraction in the right-hand side is the probability that by choosing a slot that is not storing any object referenced by s, we end up with a fully unoccupied slot. Observe that this lower-bound approaches 1 when  $\kappa_k$  goes to infinity. The adequacy of this bound is shown in [25, Remark 1].

Now that we have established that layout randomization provides *kernel safety* in a probabilistic sense, it is worth mentioning how this property relates with other desirable safety properties. In particular, *kernel safety* encompasses some form of *spatial memory safety* and of *control flow integrity*,

that are maybe the most sought-after security properties for operating systems, as witnessed by the large number of measures that, together with layout randomization, have been developed for their enforcement [58, 62, 64, 72, 31, 48].

**Spatial Memory Safety** Although it is difficult to find a common definition of *spatial memory safety*, many of these definitions associate a software component (a program, an instruction, or even a variable) with a fixed memory area, that this component can access rightfully [10, 59, 57, 7]. In this realm, any load or store operation that does not fall within this area is considered a violation of *spatial memory safety*. Our notion of *kernel safety* encompasses a form of spatial memory safety: if a system enjoys *kernel safety*, then no system call can access a memory region that does not appear within its capabilities.

Control Flow Integrity This property requires that the control transfer operations performed by a program can reach only specific targets that are determined statically [3]. If a system enjoys kernel safety then it also enjoys a weak form of control flow integrity. Indeed, our semantics prevents the execution of a function if its address does not belong to the set of capabilities of the current system call. This means that if a system  $\sigma = (\tau, \gamma, \xi)$  is kernel safe, we are certain that by executing that system call, the control flow will flow across the procedures that belong to  $\xi(s)$ .

# 6 Speculative Threat Model

In this section we establish to which extent a system enjoys *kernel safety* in presence of speculative attackers. To this aim, in Section 6.1, we extend the model of Section 3 for this new scenario. More precisely, we endow the semantics of Section 3 with speculative execution and side-channel observations that reveal the accessed addresses and the value of conditional branches [10, 16, 37, 38]. In Section 6.2, we refine the notion of *kernel safety* for this model, by defining *speculative kernel safety*.

## 6.1 The Speculative Execution Model

A popular way to model speculative attacks is by annotating transitions with directives that describe the choices made by microarchitectural prediction units [10, 16, 37, 38]. For instance, PHT speculation can be modeled by the directives br true and br false that, in the presence of a branching command, instruct the processor to take the true and the false branch respectively. In this realm, reductions are driven by sequences of directive governing each single transition. Therefore, stating the absence of a speculative attack boils down to stating the absence of a sequence of directives that steer the execution to a bad configuration. This also means that attacks are never explicitly represented as computational objects, but only as sequences of possibly unrelated directives.

A substantial difference between our model and above-mentioned models lies in the possibility to explicitly model attackers. More precisely, in our model, an attacker becomes a fully-fledged program that can directly interact with the system. This permits us to naturally extend the notion of *kernel safety* to the new scenario. Besides, we believe that modeling an attack explicitly can be interesting on its own. The feasibility of an attack is witnessed explicitly through a program. In this setting, for instance, assumptions on the attacker's computational capabilities can be imposed seamlessly.

#### 6.1.1 Victim Language and Semantics

The victims' language remains identical to the classic model except that, in order to model the ability of attackers to influence the speculative execution of specific instructions, we assume that load, branch and call instructions are tagged by unique labels  $\ell \in \mathsf{Lbl}$ . For example, a tagged load operation looks like the following  $x :=_{\ell} *E$ . We show labels alongside with commands only when they are relevant—e.g. the command is being executed speculatively—and they cannot be inferred from the context.

Following Barthe et al. [10], the speculative semantics is instrumented through *directives*, modeling the choice made by prediction units of the processor. Directives take the form

$$d \ni \mathsf{Dir} ::= \mathsf{br}_{\ell} \, b \, \big| \, \mathsf{run}_{\ell} \, p \, \big| \, \mathsf{Id}_{\ell} \, i \, \big| \, \mathsf{bt} \, \big| \, \mathsf{st},$$

where  $i \in \mathbb{N}$  and  $b \in \mathsf{Bool}$ . The  $\mathsf{br}_\ell \, b$  directive causes a branch instruction to be evaluated as if the guard resolved to b, modeling PHT speculation. The  $\mathsf{run}_\ell \, p$  directive causes a call instruction to execute the function stored at address p (if any), modeling BTB speculation. The directive  $\mathsf{Id}_\ell \, i$  causes the load instruction to load the i-th most recent value that is associated to an address in a (buffered) memory, modeling STL speculation. The  $\mathsf{bt}$  directive is used to direct speculations, either backtracking the most recent mis-speculation or committing the microarchitectural state. Finally, the  $\mathsf{st}$  directive evaluates an instruction without engaging into speculation, in correspondence to the semantics we have given in Section 3.

The semantics is also instrumented with observations to model timing side-channel leakage:

$$o, q \ni \mathsf{Obs} ::= \circ \mid \mathsf{br} \, b \mid \mathsf{mem} \, p \mid \mathsf{jmp} \, p \mid \mathsf{bt} \, b,$$

where  $n \in \mathbb{N}$ ,  $b \in \mathsf{Bool}$ , and  $p \in \mathsf{Addr}$ . We use  $\circ$  to label transitions that do not leak observations. The  $\mathsf{br}\,b$  observation is caused by branching instructions, with b reflecting the taken branch. The  $\mathsf{mem}\,p$  observation is caused by memory access, through loads or stores, and contains the address of the accessed location, thus modeling data-cache leaks. Likewise, the  $\mathsf{jmp}\,p$  observation is caused by calls to procedures residing at address p in memory, modeling instruction-cache leaks. Finally, the  $\mathsf{bt}\,b$  observation signals a backtracking step during speculative execution. Notice that we leak full addresses on memory accesses, and the value of the branching instructions, i.e. we adopt the  $\mathsf{baseline}\,$  leakage  $\mathsf{model}\,$  that is widely employed in the literature to model side-channel info-leaks [6, 10, 16, 9].

A reduction step now takes the form

$$w \vdash_{\sigma} S \xrightarrow{o} S',$$

indicating that for a given system  $\sigma$ , under layout  $w \in \mathsf{Layout}$ , the system evolves from state S with directive  $d \in \mathsf{Dir}$  to S' in one step, producing the side-channel observation o. The state of a system S is now modeled as a stack of backtrackable configurations. Specifically, configurations follow the following BNF:

$$C,D ::= (F,(\mu,m),b_{ms}) \mid (\mathsf{err},b_{ms}) \mid \mathsf{unsafe}$$

In a configuration  $(F, (\mu, m), b_{ms})$ , F is a call-stack as in Section 3,  $(\mu, m)$  is a memory equipped with a write buffer  $\mu$ , and  $b_{ms}$  the mis-speculation flag. Our buffered memories are based on those from [10] and they are useful to model STL speculation. Writing a value v at address p results in a delayed write  $[p \mapsto v]$  that is appended to the buffer  $\mu$ , in notation:  $([p \mapsto v] : \mu, m)$ . Reading the k-th most recent entry associated to p in a buffered memory is written  $(\mu, m)^k(p)$  and yields a value v together with a boolean flag f that is  $\bot$  if and only if v is the most recent value associated to address p. In a configuration, the mis-speculation flag  $b_{ms}$  records whether a past step of computation led to a mis-speculation. It is employed when backtracking from a speculative state: when its value is  $\top$ , some previous speculation may be incorrect, so a backtracking step will discard the configuration. As errors are recoverable under mis-speculation, error configurations carry a mis-speculation flag. Finally, as in Section 3, unsafe indicates a safety violation.

Some illustrative rules of the semantics are given in Figure 5, the complete set of rules is relegated to Appendix A.1. The rules for load instructions are very similar to the ones we give in Section 3, but attackers can take advantage of the store-to-load dependency speculation by issuing a  $\mathsf{Id}_\ell$  i directive,

$$C = (\langle x :=_{\ell} * \mathsf{E}; \mathsf{C}, \rho, b \rangle : F, (\mu, m), b_{ns}) \quad \mathbb{E} \|_{\rho, w}^{\mathsf{Add}r} = p \quad (\mu, m)^i(p) = (v, f) \quad p \in \underline{w}(\mathsf{ArrIdb}_h) \quad \overline{b = \mathtt{k}_s \Rightarrow p \in \underline{w}(\xi(s))}$$
 [SLOAD-LOAD] 
$$w \vdash_{\sigma} C : S \xrightarrow{\mathsf{mem} \, p \atop \mathsf{Id}_{\ell} \, \mathsf{i}} \vdash (\langle \mathsf{C}, \rho[x \leftarrow v], b \rangle : F, (\mu, m), b_{ns} \vee f) : C : S$$
 
$$\mathbb{E} \|_{\rho, w}^{\mathsf{Add}r} = p \quad p \notin \underline{w}(\mathsf{ArrIdb}_h)$$
 [SLOAD-ERROR] 
$$\overline{w} \vdash_{\sigma} (\langle x :=_{\ell} *\mathsf{E}; \mathsf{C}, \rho, b \rangle : F, (\mu, m), b_{ms}) : S \xrightarrow{\circ}_{\mathsf{Id}_{\ell} \, \mathsf{i}} \vdash (\mathsf{err}, b_{ns}) : S}$$
 [SLOAD-ERROR] 
$$\overline{w} \vdash_{\sigma} (\langle x :=_{\ell} *\mathsf{E}; \mathsf{C}, \rho, b_{\vee} : F, (\mu, m), b_{ms}) : S \xrightarrow{\mathsf{mem} \, p}_{d} \vdash \mathsf{ver}, b_{ns}) : S}$$
 [SLOAD-UNSAFE] 
$$\overline{w} \vdash_{\sigma} C : S \xrightarrow{\mathsf{br} \, d} \vdash (\langle \mathsf{D}_d; \mathsf{D}, \rho, b \rangle : F, (\mu, m), b_{ms}) : S \xrightarrow{\mathsf{mem} \, p}_{d} \vdash \mathsf{ver}, b_{ms}}$$
 [IF-Branch] 
$$\overline{w} \vdash_{\sigma} C : S \xrightarrow{\mathsf{br} \, d} \vdash (\langle \mathsf{D}_d; \mathsf{D}, \rho, b \rangle : F, (\mu, m), b_{ms} \vee d \neq \mathbb{E} \|_{\rho, w}^{\mathsf{Bool}}) : C : S}$$
 
$$\overline{w} \vdash_{\sigma} C : S \xrightarrow{\mathsf{br} \, d} \vdash (\langle \mathsf{D}_d; \mathsf{D}, \rho, b \rangle : F, (\mu, m), b_{ms} \vee d \neq \mathbb{E} \|_{\rho, w}^{\mathsf{Bool}}) : C : S}$$
 [SLOAD-STEP] 
$$\overline{w} \vdash_{\sigma} C : S \xrightarrow{\mathsf{bt} \, \top} S \mapsto S \xrightarrow{\mathsf{mem} \, p}_{\mathsf{st}} \lor (\langle \mathsf{C}, \rho[x \leftarrow v], b \rangle : F, (\mu, m), b_{ms}) : S}$$
 [SLOAD-STEP] 
$$\underline{C} = (F, (\mu, m), \top) \vee C = (\mathsf{err}, \top) \times F \xrightarrow{\mathsf{b} \, \top} S \mapsto S$$

Figure 5: Speculative semantics, excerpt.

as in Rule [SLOAD-LOAD]. When this happens, the *i*-th most recent value associated to the address  $p = [\![\mathbb{E}]\!]_{\rho,w}^{\mathsf{Addr}}$  is retrieved from the buffered memory, as described by the following function:

$$(\epsilon, m)^n(p) \triangleq m(a), \perp$$

$$([p' \mapsto v] : \mu, m)^n(p) \triangleq \begin{cases} (\mu, m)^n(p) & \text{if } p \neq p' \\ v', \top & \text{if } p = p', n > 0, \text{ and } (\mu, m)^{n-1}(p) = v', f \\ v, \perp & \text{if } p = p', \text{ and } n = 0. \end{cases}$$

When the value obtained from the lookup may not correspond to that of the most recent store to the address p, the flag f is  $\top$ . Depending on the value of f, Rule [SLOAD-LOAD] may be engaging mis-speculation and, for this reason, it keeps track of the starting configuration in the stack and updates the mis-speculation flag with f. A successful load produces the observation  $\mathsf{mem}\,p$  that leaks the address to the attacker. The rules for erroneous and unsafe loads are [SLOAD-ERROR] and [SLOAD-UNSAFE] and they are analogous to their non-speculative counterparts. In our semantics, every command also supports the st directive, which evaluates the configuration without speculating. For instance, the [SLOAD-STEP] rule evaluates the x := \*E command by fetching the most recent value from the write buffer, instead of an arbitrary one.

Even branch instructions can be executed speculatively by issuing the directive  $\operatorname{br} d$  by means of the rule [IF-BRANCH]. This causes the evaluation to continue as if the guard resolved to d. This operation leaks which branch is being executed by means of the observation  $\operatorname{br} d$ . Even in this case, the rule may be mis-speculating; for this reason, the current configuration is book-kept in the stack

```
\frac{C = (\langle \mathsf{call} \ \mathsf{E}(\mathsf{F}_1, \dots, \mathsf{F}_h); \mathsf{C}, \rho, b \rangle : F, (\mu, m), b_{ms}) \quad \llbracket \mathsf{E} \rrbracket_{\rho, w}^{\mathsf{Addr}} = p' \quad p \in \underline{w}(\mathsf{Funld}_b) \quad \boxed{b = \mathsf{k}_{\mathtt{S}} \Rightarrow p \in \underline{w}(\xi(\mathtt{S}))}}{w \vdash_{\sigma} C : S \xrightarrow{\mathsf{mem} \ p}{} \vdash (\langle m(p), \rho_0[x_1, \dots, x_h \leftarrow \llbracket \mathsf{F}_1 \rrbracket_{\rho, w}, \dots, \llbracket \mathsf{F}_h \rrbracket_{\rho, w}], b \rangle : \langle \mathsf{C}, \rho, b \rangle : F, (\mu, m), b_{ms} \vee (p \neq p')) : C : S}} \\ \frac{p \in \underline{w}(\mathsf{Funld}_{\mathtt{k}}) \quad \boxed{p \notin \underline{w}(\xi(\mathtt{S}))}}{w \vdash_{\sigma} (\langle \mathsf{call} \ \mathsf{E}(\vec{\mathsf{F}}); \mathsf{C}, \rho, \mathsf{k}_{\mathtt{S}} \rangle : F, (\mu, m), b_{ms}) : S \xrightarrow{\mathsf{jmp} \ p}{\mathsf{run} \ p}} \mathsf{unsafe}} \\ \underline{C = (\langle \mathsf{call} \ \mathsf{E}(\mathsf{F}_1, \dots, \mathsf{F}_h); \mathsf{C}, \rho, b \rangle : F, (\mu, m), b_{ms}) \quad \llbracket \mathsf{E} \rrbracket_{\rho, w}^{\mathsf{Addr}} = p' \quad p \notin \underline{w}(\mathsf{Funld}_b)}{w \vdash_{\sigma} C : S \xrightarrow{\mathsf{o}}{\mathsf{run} \ p}} \mathsf{ever}, b_{ms} \vee (p \neq p')) : C : S}} \\ \underline{\mathsf{SCall-Error}}
```

Figure 6: Speculative semantics, speculative call instructions.

and the mis-speculation flag is updated.

When a call instruction is encountered, the attacker can steer the control flow to any target function by supplying the directive run p, as described by the rules in Figure 6. Speculative call instructions are evaluated with the rule [SCALL], which invokes the procedure that is stored at address p (if any). This operation leaks the observation  $\mathsf{jmp}\,p$ , which reveals that some executable function is stored at address p. The mis-speculation flag is updated by comparing the address supplied by the adversary with the one targeted by the instruction ( $\llbracket E \rrbracket_{\rho,w}^{\mathsf{Addr}}$ ). The rules for erroneous and unsafe speculative call instructions are [SCALL-ERROR] and [SCALL-UNSAFE] and they are similar to their non-speculative counterparts, with the exception that the address supplied by the attacker is used to determine whether the access is erroneous or unsafe. Notice that, since the address supplied by the adversary may not correspond to the one targeted by the instruction, the rules [SCALL] and [SCALL-ERROR] may be performing mis-speculation. For this reason, these two rules keep track of the current configuration in the speculative stack and update the mis-speculation flag. When the topmost configuration of a stack carries the mis-speculation flag  $\top$ , this means that some previous speculation was wrong, so the configuration can be is discarded with the rules  $[BT_T]$ . If the mis-peculation flag is  $\bot$ , the current state is not mis-speculating, so the whole stack of book-kept configurations can be discarded with the rule  $[BT_{\perp}]$ .

### 6.1.2 Attacker's Language and Semantics

To give a definition of kernel safety w.r.t. speculative semantics, we endow an attacker with the ability to engage speculative execution, to issue directives, and to read side-channel information. To this end, from now on, we extend the instructions from Section 3 as follows:

$$\begin{array}{l} \mathsf{SpInstr} \ni \mathtt{SI} ::= \cdots \ \middle| \ \mathtt{spec} \ \mathtt{on} \ \mathtt{C} \ \middle| \ \mathtt{poison}(d) \ \middle| \ x := \mathtt{observe}() \\ \mathtt{SpAdv} \ni \mathtt{A} ::= \epsilon \ \middle| \ \mathtt{SI}; \mathtt{A} \end{array}$$

$$\overline{w \vdash_{\sigma} (\langle \mathsf{poison}(d); \mathtt{A}, \rho, b \rangle : F, m, D, O) \rightarrow (\langle \mathtt{A}, \rho, b \rangle : F, m, d : D, O)}^{\text{[POISON]}}$$

$$\overline{w \vdash_{\sigma} (\langle x := \mathsf{observe}(); \mathtt{A}, \rho, b \rangle : F, m, D, o : O) \rightarrow (\langle \mathtt{A}, \rho[x \leftarrow o], b \rangle : F, m, D, O)}^{\text{[OBSERVE]}}$$

$$\overline{w \vdash_{\sigma} (\langle \mathsf{spec} \ \mathsf{on} \ \mathsf{C}; \mathtt{A}, \rho, b \rangle : F, m, D, O) \rightarrow (\langle \mathtt{C}, \rho, b \rangle, ([], m), \bot) \mid (\langle \mathtt{A}, \rho, b \rangle : F, D, O)}^{\text{[SPEC-INIT]}}$$

$$\frac{w \vdash_{\sigma} S \stackrel{\circ}{\downarrow} S'}{\overline{w \vdash_{\sigma} S \mid (F, d : D, O) \rightarrow S' \mid (F, D, o : O)}^{\text{[SPEC-INIT]}}}^{\text{[SPEC-INIT]}}$$

$$\frac{w \vdash_{\sigma} S \downarrow_{D} \quad w \vdash_{\sigma} S \stackrel{\circ}{\downarrow} S'}{\overline{w \vdash_{\sigma} S \mid (F, D, O) \rightarrow S' \mid (F, D, o : O)}^{\text{[SPEC-BT]}}}}$$

$$\overline{w \vdash_{\sigma} (\langle \epsilon, \rho, b \rangle, (\mu, m), \bot) \mid (\langle \mathtt{A}, \rho', b' \rangle : F, D, O) \rightarrow (\langle \mathtt{A}, \rho', b' \rangle : F, \overline{(\mu, m)}, D, O)}^{\text{[SPEC-TERM]}}}$$

$$\overline{w \vdash_{\sigma} (\mathsf{err}, \bot) \mid (F, D, O) \rightarrow \mathsf{err}}^{\text{[SPEC-ERROR]}}$$

$$\overline{w \vdash_{\sigma} \mathsf{unsafe} \mid (F, D, O) \rightarrow \mathsf{unsafe}}^{\text{[SPEC-UNSAFE]}}$$

Figure 7: Semantics for speculative attackers, excerpt.

 $Obs \subseteq Val$ . As an example, the snippet

$$\begin{aligned} & \operatorname{poison}(\mathsf{br}_{\ell} \top); \\ & \operatorname{spec} \ \mathsf{on} \ \mathsf{if}_{\ell} \ \mathsf{E} \ \mathsf{then} \ \mathsf{syscall} \ \mathsf{s}(p) \ \mathsf{fi}; \\ & x := \mathsf{observe}() \end{aligned} \tag{\dagger}$$

forces the mis-speculative execution of syscall s(p), independently of the value of E. The register x will hold the final observation leaked through executing the system call.

The attacker's semantics is defined in terms of a relation

$$w \vdash_{\sigma} C \rightarrow C'$$
.

In essence, the attacker executes under the a semantics analogous to the one given in Section 3, the speculative semantics defined above plays a role only when execution of the victim is triggered by the directive spec on C. Consequently, configurations extend the ones underlying the standard semantics with the stacks D and O of directives and observations, in order to model the new constructs. In addition, hybrid configurations  $S \mid (F, D, O)$  are used to model the system when executing the victim under speculative semantics. Here S is a stack of speculative configurations concerning the victim, and F the attacker's call stack up to the invocation of speculation. Again, D gives the directives (to be processed) and O the observations (collected from executing victim's code). In summary, configurations are drawn from the following BNF:

$$C ::= (F, m, D, O) \mid S \mid (F, D, O) \mid \text{err} \mid \text{unsafe}$$

Figure 7 shows the evaluation rules for the new constructs. Rules [POISON] and [OBSERVE] define the semantics for poisoning and side-channel observations, modeled by respectively pushing and popping elements of the corresponding stacks. Rule [SPEC-INIT] deals with the initialization spec on C of speculative execution, starting from the corresponding initial configuration of the victim C in an empty speculation context. A frame for the continuation of the attacker A is left on the call stack F. This frame is used to resume execution of the attacker, once the victim has been fully evaluated. The victim itself is evaluated via the speculative semantics through rules [SPEC-D]–[SPEC-BT]. Note how execution of the victim is directed through the directive stack D (rule [SPEC-D]). Should the current directive be inapplicable, a non-speculative rewrite step (rule [SPEC-S]) or backtracking (rule [SPEC-BT]) is performed. Here, the premise  $w \vdash_{\sigma} S \downarrow_{d}$  signifies that S is irreducible w.r.t. the directive d. Likewise,  $w \vdash_{\sigma} S \downarrow_{D}$  means that S is irreducible w.r.t. the topmost directive of D, or that D is empty. Also notice how side-channel leakage, modeled through observations, is collected in the configuration via these rules. Upon normal termination, resuming of evaluation of the attacker is governed by rule [SPEC-TERM] in the case of normal termination. As a side-effect, this rule commits all buffered writes to memory, as described by the following function:

$$\overline{(\epsilon,m)} \triangleq m \qquad \qquad \overline{([p \mapsto v]: \mu, m)} \triangleq \overline{(\mu,m)}[p \leftarrow v],$$

Finally, rules [SPEC-ERROR] and [SPEC-UNSAFE] deal with abnormal termination.

We write  $\rightarrow^*$  for the multistep reduction relation induced by  $\rightarrow$ , i.e,  $S \xrightarrow[\epsilon]{\epsilon} S$  and  $S \xrightarrow[d:D]{o:O} \rightarrow^* S'$  if  $S \xrightarrow[d]{o} \rightarrow O \rightarrow^* S'$ .

### 6.2 Speculative Kernel Safety

We are now ready to extend the definition of kernel safety (Definition 1) to the speculative semantics.

**Definition 3** (Speculative kernel safety). We say that a system  $\sigma = (\tau, \gamma, \xi)$  is speculative kernel safe if for every unprivileged attacker  $A \in SpAdv$ , every layout w, and register map  $\rho$ , we have:

$$\neg \left( w \vdash_{\sigma} (\langle \mathtt{A}, \rho, \mathtt{u} \rangle, w \diamond \tau, \epsilon, \epsilon) \mathbin{\rightarrow}^* \mathsf{unsafe} \right).$$

It is important to note that this safety notion captures violations that occur during transient execution. This is in line with what happens, for instance, for Spectre and Meltdown [44, 50], both exploiting unsafe memory access under transient execution in order to reveal confidential information.

## 6.3 The Demise of Layout Randomization in the Spectre Era

A direct consequence of Definition 3 is that every system that is *speculative kernel safe* is also *kernel safe*. The inverse, of course, does not hold in general. Most importantly, the probabilistic form of safety provided by layout randomization in Section 5 does not scale to this extended threat model. This happens because *layout non-interference* (Definition 2) does not take side-channel leakage into account. In practice, even a gadget such as

if 
$$f = p$$
 then C else D fi,  $(\ddagger)$ 

can be exploited by an attacker controlling the value of p to infer information about the address of a kernel-space procedure f, through side-channel leakage distinguishing the execution of C and D. In our model, the attacker can retrieve the address of f by supplying different values for p until when it collects the side-channel observation f by the this happens, the current value of f corresponds to the address of f.

In addition, with speculative execution, unsuccessful memory probes within transient executions do not lead to abnormal termination. This fact undermines another fundamental assumption of Theorem 1 and the majority of studies demonstrating the efficacy of layout randomization (e.g., [2, 1, 4]), where memory access violations are not recoverable.

For instance, the program in  $(\dagger)$  will never reach an unrecoverable error state even if the system call s loads an arbitrary address p form the memory. If p is not allocated, the system performs a memory access violation under transient execution, that does not terminate the execution. Moreover, if p is allocated, its content is loaded into the cache, producing the observation  $\operatorname{mem} p$  before the execution of the branch and the system call are backtracked. By reading side-channel observations, an attacker can thus distinguish allocated kernel-addresses from those that are not allocated. This last example, in particular, is not at all fictitious: the BlindSide attack [34] uses the same idea to break Linux's KASLR and locate the position of kernel's executable code and data.

### 6.4 Speculative Layout Non-Interference

As this revised model significantly enhances the attackers' strength, we will need to implement more stringent countermeasures in order to restore kernel safety. To counter side-channel info-leaks, we can impose a form of side-channel non-interference that is in line with the notion of *speculative constant-time* from [16], and that is meant to prevent side-channel info-leaks from leaking information on the kernel's memory layout.

**Definition 4** (Speculative layout non-interference). Given  $\sigma = (\tau, \gamma, \xi)$ , a system call s is *speculative* layout non-interfering, if,

$$w_1 \vdash_{\sigma} (\langle \gamma(\mathtt{s}), \rho, \mathtt{k}_\mathtt{s} \rangle, (\mu, (w_1 \diamond \tau')), b_{ms}) \xrightarrow{O} \ast S_1 \quad \text{implies} \quad w_2 \vdash_{\sigma} (\langle \gamma(\mathtt{s}), \rho, \mathtt{k}_\mathtt{s} \rangle, (\mu, (w_2 \diamond \tau')), b_{ms}) \xrightarrow{O} \ast S_2,$$

for all layouts  $w_1, w_2$ , configurations over stores  $\tau' =_{\mathsf{Funld}} \tau$  coinciding on procedures  $(\tau'(\mathtt{f}) = \tau(\mathtt{f}))$  for all  $\mathtt{f} \in \mathsf{Funld}$ , directives D, observations O and register map  $\rho$ .

Speculative layout non-interference effectively prevents side-channel-related attacks, even during transient executions. Importantly, it ensures the non-leakage of layout information throughout the side-channels by requiring the identity of the sequences of observations produced by the two reductions. This, however, implies severe restrictions on memory interactions—effectively prohibiting non-static memory accesses, that are the main ingredient of layout randomization! Unsurprisingly, this form of non-interference directly establishes kernel safety of system calls:

**Lemma 1.** Suppose  $\kappa_k \ge \sum_{\mathtt{id} \in \mathsf{Id}_k} \mathsf{size}(\mathtt{id}) + 2 \cdot \max_{\mathtt{id} \in \mathsf{Id}_k} \mathsf{size}(\mathtt{id})$ . Given  $\sigma = (\tau, \gamma, \xi)$ , if s is speculative layout non-interfering then

$$\neg \left( w \vdash_{\sigma} (\langle \gamma(\mathtt{s}), \rho, \mathtt{k}_{\mathtt{s}} \rangle, (\mu, (w \diamond \tau')), b_{ms}) \xrightarrow{O} \ ^* \mathsf{unsafe} \right)$$

for all layouts w and initial configurations over stores  $\tau'$  coinciding with  $\tau$  on Funld.

The intuition behind the proof is fairly simple: by requiring the condition on the size of the memory, we ensure that the memory is large enough to allow any resource to be moved to a different location. With this precondition, we just need to observe that if an invocation of a system call s performs an unsafe memory access when executing under a layout w, the address p of the accessed resource is leaked; but the same address cannot leak if the resource is moved to another location. Therefore, if a system call is not speculatively safe, it cannot be speculative non-interferent because different memory layouts produce different observations.

In general, it is not always the case that a non-interference property has as consequence memory safety. For instance, being *non-interferent* with respect to a set of secrets does not prevent a victim program from breaking memory safety. In our case, this property holds because the layouts are not only used as the inputs for a computation, but they also determine *where* objects are placed in memory.

The following result, stating that speculative layout non-interference entails speculative kernel safety, is a direct consequence of Lemma 1.

**Theorem 2.** Under the assumption  $\kappa_k \ge \sum_{\mathtt{id} \in \mathsf{Id}_k} \mathsf{size}(\mathtt{id}) + 2 \cdot \max_{\mathtt{id} \in \mathsf{Id}_k} \mathsf{size}(\mathtt{id})$ , if a system  $\sigma$  is speculative layout non-interfering, then it is speculative kernel safe.

Observe that, in contrast with what happens in Theorem 1, the safety guarantee provided by Theorem 2 is not probabilistic and does not rely on *layout randomization*. Therefore, although layout randomization is unlikely to be restored at the software level without imposing *speculative layout non-interference*, in presence of this assumption, layout is a redundant protection measure.

However, it is important to notice that speculative layout non-interference is not a necessary condition for speculative kernel safety. As an example, we can take the system that only uses the array a with size 1 and the system call s whose body is \*a := v with  $\xi(s) = \{a\}$ . This system does not enjoy speculative non-interference because by executing s the address of a leaks, and this address changes under different layouts. However, this system is speculatively safe because it can only write to the memory storing a, and we have  $a \in \xi(s)$ .

# 7 Discussion of other Mitigations against Speculative Attacks

In this section we review state-of-art countermeasures against speculative attacks to determine to what extent they can help to enforce *speculative kernel safety*. To this aim, we introduce a simple system with system calls that are vulnerable to *speculative kernel safety* violations, and we use this system to determine whether state-of-art mitigations against speculative attacks can prevent them.

The system  $\sigma = (\tau, \gamma, \xi)$  defines a kernel function f as  $\tau(f) = ret := 0$  and four vulnerable system calls as follows:

To demonstrate that  $\sigma$  is vulnerable to *speculative kernel safety* violations, we describe in Figure 8 the attack patterns  $A_{(\cdot)}$ ,  $B_{(\cdot)}$  that take advantage of  $\sigma$ 's system calls to break *speculative kernel safety*. An attack pattern is instantiated to an attack by replacing each occurrence of the hole  $(\cdot)$  with a system call identifier. In particular, in the following we are interested to the instances of  $A_{(\cdot)}$  on s and

```
\mathsf{B}_{(\cdot)} \triangleq y := \kappa_{\mathsf{u}};
A_{(\cdot)} \triangleq y := \kappa_{\mathbf{u}};
           while x \neq \operatorname{run}_{-} \operatorname{do}
                                                                                while x \neq \operatorname{run}_{-} \operatorname{do}
               poison(br<sub>ℓ</sub> true);
                                                                                    poison(Id_{\ell} 1);
                syscall (\cdot)(false, y);
                                                                                     syscall (\cdot)(y);
               while x \notin \{\mathsf{jmp}_{-}, \mathsf{null}\}\ \mathsf{do}
                                                                                     while x \notin \{\mathsf{jmp}_{-}, \mathsf{null}\}\ \mathsf{do}
                    x := observe()
                                                                                         x := observe();
               od;
                                                                                     od:
               y := y + 1
                                                                                     y := y + 1
           od; syscall s(true, p-1)
                                                                                od; syscall t(p-1)
```

Figure 8: Attacks patterns witnessing that  $\sigma$  does not enjoy speculative kernel safety.

Protection mechanism	Attack A <sub>s</sub>	Attack As'	Attack B <sub>t</sub>	Attack $B_{t'}$
SLH and Eclipse	✓	✓	X	X
SCT	X	X	X	X
STT	X	X	X	X
SPT	✓	✓	X	X
NDA	X	X	X	X
Prospect	X	X	X	X
Our transformations (Section 8)	✓	✓	✓	✓

Table 1: Effectiveness of the protection mechanisms examined in sections 7 and 8 against the attack patterns in Figure 8.

s' and to the instances of  $\mathsf{B}_{(\cdot)}$  on the system calls t and t'. The attack pattern  $\mathsf{A}_{(\cdot)}$  is reminiscent of BlindSide [34]. It triggers the speculative execution of the conditional instruction within the victim system call to probe the kernel memory until it reads a side-channel leak  $\mathsf{jmp}\,p$  revealing the address of f. Finally, the system call s is used to trigger the execution of f by supplying the retrieved address. The other attack pattern is  $\mathsf{B}_{(\cdot)}$ . Instead of triggering the speculative execution of a conditional instruction, it relies on STL forwarding to engage speculative execution and transiently execute call x(), probing the kernel memory for executable code. Except for this variation, the remaining part of this pattern is similar to  $\mathsf{A}_{(\cdot)}$ .

SLH [15] and Eclipse [19] Speculative Load Hardening (SLH) is a software-level countermeasure that prevents speculative leaks of private information. It keeps track of a predicate indicating whether the execution follows a wrong speculation or not. The value of this predicate is used to instrument leaking operations—typically loads—in order to obfuscate leaked values during transient execution. In addition to loaded values, loaded addresses and the targets of indirect calls or jumps can also be obfuscated using SLH: later works like Strong SLH [60] and UltimateSLH [78] improved SLH by obfuscating even more operands including store addresses and inputs of time-variable arithmetical operations, respectively. Eclipse [19] is a software-level protection measure adopting a similar approach to obfuscate the operands of instructions that are used to perform speculative probing by attacks like BlindSide [34] or PACMAN [63]. Eclipse achieves lower overhead compared to SLH by protecting only a specific subset of instructions—either indirect jumps or instructions for pointer validation, depending on the type of speculative probing the user aims to prevent. However, Eclipse and SLH are specific to PHT speculation—related to Spectre v1—and they fail to detect BTB and STL speculation—related to Spectre v2 and v4, respectively. Therefore, although these measures would be effective against

attacks relying on direct branch speculation only, such as instances of the pattern  $A_{(\cdot)}$ , they would not stop instances of  $B_{(\cdot)}$  because they would not detect STL-dependency speculation.

SCT The Constant-Time (CT) discipline imposes that program do not leak secret information via side-channels. Speculative Constant Time (SCT) extends (CT) by also taking in account speculative execution [10, 16]. Therefore, SCT programs are not subject to speculative attacks like Spectre [44]. However, SCT is of little help to enforce speculative kernel safety, as leaking secrets speculatively does not necessarily involve breaking speculative kernel safety. So, by imposing SCT, speculative kernel safety is not necessarily achieved. For instance, notice that none of the attack patterns in Figure 8 leaks secrets stored in memory, although they break speculatively kernel safe.

In Section 6.4 we proposed a policy based on SCT, to prevent (secret) information on the layout to leak during speculative execution: speculative layout non-interference. It was also possible to show that speculative kernel safety can be obtained by imposing speculative layout non-interference. However, speculative layout non-interference would be too strong for our needs, as any system call interacting with the memory would violate it.

STT [77] Speculative Taint Tracking (STT) is a hardware protection mechanism to protect from speculative leakage by preventing the propagation of transiently loaded data. However, STT works by preventing the propagation only of data loaded during speculative execution to subsequent instructions. Therefore, it would not block none of the attacks that we are considering from speculatively probing the victim's memory and breaking speculative kernel safety. Specifically, STT would not prevent the execution of call  $x_2()$ , which is the instruction actually revealing the position of f, because the value  $x_2$  is not loaded during transient execution.

**SPT** [18] Speculative Privacy Tracking (SPT) is a hardware protection mechanism based on STT. Instead of preventing the propagation of speculatively loaded data, SPT prevents the propagation of speculative secrets, i.e. of data that without this measure would leak during speculative execution, but not during normal execution. To identify speculative secrets, the hardware keeps track of the data that is leaked in normal execution. When the hardware detects that some data leaks in normal execution, that data is also allowed to leak in speculative execution.

SPT would be effective against the attacks  $A_s$  and  $B_t$  because the addresses that are probed by the syscalls s and t do not leak in normal execution. However, it would not be effective against attacks the attacks  $A_{s'}$  and  $B_{t'}$  because the probed address are leaked by the load operation in normal execution before executing the probing instructions.

NDA [74] Non-speculative Data Access (NDA) is a hardware protection measure similar to STT that restricts the propagation of data during speculation. NDA supports different policies to determine whether an instruction can broadcast its output, however NDA is not effective against attacks leaking addresses via a single transient operation [74]. Therefore, it cannot prevent the speculative leaks caused by a single instruction that tries to access a memory address, such as the call  $x_2()$  instruction in all the four syscall we are examinating.

**ProSpeCT** [22] Prospect is a RISC-V processor prototype that adopts taint-tracking to prevent speculative leak of secret information, i.e. coming from a specific address range. In particular, an instruction cannot interact with the memory (performing side channel leaks) if the register holding the target address is tainted. The authors of Prospect show that any CT program running on Prospect is SCT. We already discussed the difficulties of enforcing *speculative kernel safety* on top of SCT.

It may seem that by setting the address range of the secret data to the set of kernel addresses, PROSPECT would prevent memory accesses to kernel's memory during speculative execution. However, this is not the case: PROSPECT would only block accesses to locations whose address depends on kernel data, rather than blocking all speculative accesses to kernel's memory.

**Final Remarks** The countermeasures we just discussed are designed to prevent the side-channel leakage of sensitive data during transient execution at the hardware level. However, a violation of *speculative kernel safety* does not necessarily leak sensitive data to the attacker. For instance, the attack patterns  $A_{(\cdot)}$  and  $B_{(\cdot)}$  break *speculative kernel safety* but do not leak sensitive data, as **f** returns the constant 0 which is not a secret.

We believe that speculative kernel safety could be enforced at hardware level by delaying all the operations that interact with kernel's memory until all the ongoing speculations are resolved as correct. Notice that this is not what taint-tracking mechanisms such as STT, SPT, NDA, and PROSECT do, as they only block those operations that may leak data that is considered secret according to some criteria during transient execution. With the hardware mechanism that we suggest, both the attack patterns in Figure 8 would be prevented as the indirect call instructions within the victim system calls would not execute during the speculation window.

Although hardware-based mechanisms for delaying memory operations could in principle solve speculative probing, this solution is challenging to adopt due to the need of specialized hardware. This consideration motivates us to develop an alternative software-based technique specifically aimed at enforcing speculative kernel safety.

# 8 Enforcement of Speculative Kernel Safety

Our last research goal is to identify some software-level protection mechanism by which we can enforce speculative kernel safety on a kernel. However, enforcing kernel safety in the classic model has proven challenging, as evidenced by the significant effort invested, in the last decades, to develop safe code in the classic model, [58, 62, 64, 72, 31, 48]. Therefore, instead of defining a mechanism that enforces speculative kernel safety on a system from scratch, our goal is rather to nullify the gap between kernel safety and speculative kernel safety, by making the latter property a consequence of the former. Notice that this approach is not novel in this field: as we saw in Section 7, many mitigations against speculative attacks are ultimately aimed to prevent undesirable events to take place under transient execution. With this guarantee, any software that is secure in the classic execution model becomes safe in the speculative one as well.

In terms of our model, our goal boils down to finding a transformation  $\zeta$  that turns any kernel safe system  $\sigma$  into another system  $\zeta(\sigma)$  which is architecturally equivalent to  $\sigma$  but enjoys speculatively kernel safety. The semantic requirement on the transformation  $\zeta$  is expressed by Definition 5, by which we ask that no user-space program may show different behaviors by executing in the two systems.

**Definition 5** (Semantics preservation). A system transformation  $\zeta$  is user-space semantics preserving if, for any system  $\sigma = (\tau, \gamma, \xi)$ ,

$$Eval_{\zeta(\sigma),w}(C, \rho, u, \tau') \simeq Eval_{\sigma,w}(C, \rho, u, \tau)$$

for every layout w, unprivileged command C, and registers  $\rho$ . Here,  $\tau'$  is the store underlying  $\zeta(\sigma)$ . The equivalence is given by  $(v, \tau_1) \simeq (v, \tau_2)$  if  $\tau_1 =_{\mathsf{Id}_u} \tau_2$ , and coincides with equality otherwise.

In the previous definition we require  $\tau_1 =_{\mathsf{Id}_u} \tau_2$  instead of  $\tau_1 = \tau_2$  in order to allow the transformation  $\zeta$  to modify kernel-space procedures.

Reducing speculative kernel safety to kernel safety means reducing safety violations in speculative execution to safety violations in normal execution. Therefore, the second requirement on  $\zeta$  can be expressed by asking that the system  $\zeta(\sigma)$  can violate speculative kernel safety only if it violates kernel safety, as captured by Definition 6 below.

**Definition 6.** We say that  $\zeta$  imposes speculative kernel safety if, for every system  $\sigma$  such that  $\zeta(\sigma) = (\tau, \gamma, \xi)$ , every buffer  $\mu$  with  $dom(\mu) \subseteq \underline{w}(Arrld)$  and store  $\tau' =_{Fun} \tau$  we have:

$$\left( w \vdash_{\zeta(\sigma)} ((\gamma(\mathtt{s}), \rho, \mathtt{k}_{\mathtt{s}}), (\mu, (w \diamond \tau')), b_{ms}) \xrightarrow{O} \Longrightarrow^* \mathsf{unsafe} \right) \Rightarrow \\ \left( w \vdash_{\zeta(\sigma)} ((\gamma(\mathtt{s}), \rho, \mathtt{k}_{\mathtt{s}}), \overline{(\mu, (w \diamond \tau'))}) \to^* \mathsf{unsafe} \right).$$

Finally, by combining Definition 6 and Definition 5, we obtain the following conclusion:

**Theorem 3.** If a system  $\sigma$  is kernel-safe, and the transformation  $\zeta$  (i) imposes speculative kernel safety and (ii) is user-space semantics preserving, then (i)  $\zeta(\sigma)$  is speculative kernel safe, and (ii)  $\zeta(\sigma)$  is semantically equivalent to  $\sigma$ .

This result states that every *kernel safe* system can be transformed into another system that is equivalent to it from the user's perspective and enjoys stronger security guarantees. Notice that *kernel safety* cannot be provided solely by the adoption of layout randomization: by Theorem 1, we know that layout randomization only provides *kernel safety* modulo a small probability of failure.

With Theorem 3 in mind, our next goal is to show some candidate transformations that fulfill its requirements and which enforce speculative kernel safety on classically safe kernels. To model these transformations, we extend our language with program instructions acting as speculation barriers—modeling the lfence instruction found in modern CPUs [42]—and with call instructions that does not support BTB speculation—similar to retpoline thunks [70].

The resulting language looks as follows:

$$\mathsf{Instr} \ni \mathtt{I} ::= \dots \bigm| \mathtt{fence} \bigm| \mathtt{scall} \ \mathtt{E}(\mathtt{E}_1, \dots, \mathtt{E}_n).$$

Architecturally, the fence instruction is a no-op, on the microarchitecture level it commits all buffered writes to memory. In particular, for consistency, a potentially mis-speculative state must be resolved. This is why this rule requires the mis-speculation flag to be  $\bot$ . This means that, if this configuration is reached when the flag is  $\top$ , the semantics must backtrack with the rule [B<sub>T</sub>].

At the architectural level, scall  $E(\vec{F})$  behaves in the same way as call  $E(\vec{F})$ , but it does not allow the attackers to speculate over its target. The speculative semantics of these constructs is given in Figure 9. The rules for the scall  $E(F_1, ..., F_n)$  instruction are similar to those for ordinary call instructions, but they do not accept  $\text{run}_{\ell} p$  directives. Instead, the address of the target function is obtained by evaluating E. For the ordinary semantics, the *non-speculative* call instruction is interpreted in the same way as the ordinary call instruction, as reported in Figure 10.

## 8.1 Simple Fencing Transformation

As a warm-up example, in Figure 11, we introduce a simple transformation that satisfies the requirements of Theorem 3. This transformation places a fence instruction before *all* the potentially unsafe operations, and prevents BTB speculation by rewriting ordinary call instructions with non-speculating call instructions.

By adding fences as described by  $\eta$ , any ongoing speculation is stopped before executing potentially unsafe operations (including non-speculative call instructions), and their transient execution is prevented, yet leaving the program's semantics unaltered at the architectural level. Notice that, since also non-speculating call instructions can perform transitions to unsafe, the transformation protects them by placing a fence instruction.

The transformation  $\eta$  of Figure 11, does not stop speculation completely, instead it only prevents the transient execution of those instructions that can perform unsafe operations. For instance, conditional instructions can still execute speculatively if their branches contain no potentially unsafe

$$\begin{split} & C = (\langle \operatorname{scall} \ \operatorname{E}(\mathsf{F}_1, \dots, \mathsf{F}_h); \mathsf{C}, \rho, b \rangle : F, (\mu, m), b_{ms}) \quad \mathbb{E} \mathbb{I}_{\rho, w}^{\operatorname{Addr}} = p \quad p \in \underline{w}(\operatorname{Funld}_b) \quad \boxed{b = \mathsf{k}_s \Rightarrow p \in \underline{w}(\xi(s))} \\ & w \vdash_{\sigma} C : S \xrightarrow{\operatorname{imp} p \atop \operatorname{st}} \triangleright (\langle m(p), \rho_0[x_1, \dots, x_h \leftarrow \mathbb{F}_1]_{\rho, w}, \dots, \mathbb{F}_h]_{\rho, w}], b \rangle : \langle \mathsf{C}, \rho, b \rangle : F, (\mu, m), b_{ms}) : S \\ & & \underbrace{\mathbb{E}} \mathbb{I}_{\rho, w}^{\operatorname{Addr}} = p \quad p \in \underline{w}(\operatorname{Funld}_\mathbf{k}) \quad \boxed{p \notin \underline{w}(\xi(s))} \\ & w \vdash_{\sigma} (\langle \operatorname{scall} \ \operatorname{E}(\vec{\mathsf{F}}); \mathsf{C}, \rho, \mathsf{k}_s \rangle : F, (\mu, m), b_{ms}) : S \xrightarrow{\operatorname{imp} p \atop \operatorname{st}} \triangleright \operatorname{unsafe} \\ & & \underbrace{C = (\langle \operatorname{scall} \ \operatorname{E}(\mathsf{F}_1, \dots, \mathsf{F}_h); \mathsf{C}, \rho, b \rangle : F, (\mu, m), b_{ms}) \quad \mathbb{E}} \mathbb{I}_{\rho, w}^{\operatorname{Addr}} = p \quad p \notin \underline{w}(\operatorname{Funld}_b) \\ & & w \vdash_{\sigma} C : S \xrightarrow{\circ} (\operatorname{err}, b_{ms} \vee (p \neq p')) : S \\ & & \underbrace{w \vdash_{\sigma} (\langle \operatorname{fence}; \mathsf{C}, \rho, b \rangle : F, (\mu, m), \bot) : S \xrightarrow{\circ} (\langle \mathsf{C}, \rho, b \rangle : F, \overline{(\mu, m)}, \bot) : S}^{\left[\operatorname{Fence}\right]} \end{aligned}$$

Figure 9: Speculative semantics, additional constructs.

Figure 10: Attacker's semantics for scall  $\cdot$  ( $\cdot$ ).

operations. This is not in contrast with Definition 6 because even in transient execution, a conditional instruction cannot perform any safety violation by itself. However, as their branches can contain unsafe operations, the transformation visits them in to protect any unsafe operation therein.

By observing that  $\eta$  enjoys both the properties in Definitions 5 and 6, we can draw the following conclusion:

**Lemma 2.** If a system  $\sigma$  is kernel-safe, then  $\eta(\sigma)$  is speculative kernel safe and semantically equivalent to  $\sigma$ .

## 8.2 Optimized fencing transformation

In this section we enhance the transformation  $\eta$  of Section 8.1 by means of a simple static analysis to determine whether the current instruction can be executed in transient mode or not. This way, fence instructions can be omitted before all those instructions that cannot be executed transiently. For instance, placing a single fence instruction before sequences of loads—or of stores—is sufficient

$$\begin{split} \eta(x := \mathsf{E}) &\triangleq x := \mathsf{E} \\ \eta(\mathsf{skip}) &\triangleq \mathsf{skip} \\ \eta(\mathsf{*E} := \mathsf{F}) &\triangleq \mathsf{fence}; \mathsf{*E} := \mathsf{F} \\ \eta(\mathsf{call} \ \mathsf{E}(\mathsf{F}_1, \dots, \mathsf{F}_k)) &\triangleq \mathsf{fence}; \mathsf{scall} \ \mathsf{E}(\mathsf{F}_1, \dots, \mathsf{F}_k) \\ \eta(\mathsf{if} \ \mathsf{E} \ \mathsf{then} \ \mathsf{C} \ \mathsf{else} \ \mathsf{D} \ \mathsf{fi}) &\triangleq \mathsf{if} \ \mathsf{E} \ \mathsf{then} \ \eta(\mathsf{C}) \ \mathsf{else} \ \eta(\mathsf{D}) \ \mathsf{fi} \\ \eta(\mathsf{C}; \mathsf{D}) &\triangleq \eta(\mathsf{C}); \eta(\mathsf{D}) \\ \eta(\mathsf{e}) &\triangleq \epsilon. \end{split}$$

$$\eta(\gamma) \triangleq \mathtt{s} \mapsto \eta(\gamma(\mathtt{s})) \quad \eta(\tau) \triangleq \mathtt{id} \mapsto \begin{cases} \tau(\mathtt{id}) & \text{if } \mathtt{id} \in \mathsf{Arr} \cup \mathsf{Fun_u} \\ \eta(\tau(\mathtt{id})) & \text{otherwise} \end{cases} \quad \eta((\tau,\gamma,\xi)) \triangleq (\eta(\tau),\eta(\gamma),\xi).$$

Figure 11: Simple fencing transformation

to protect the whole sequence. As an example, consider the following program:

```
C \triangleq x := *E; y := *F; z := *G. with \eta(C) = fence; x := *E; fence; y := *F; fence; z := *G.
```

Notice that the second and the third fence instruction are superfluous, as there is no way to engage speculative execution after the first fence instruction. Similarly, if we take in exam the following program:

$$D \triangleq *E := x; y := *F; z := *G,$$
 with  $\eta(D) = fence; *E := x; fence; y := *F; fence; z := *G$ 

we can observe that although the  $\eta$  transformation would place three fence instructions, but D can be protected with just two fence instructions as follows:

```
fence; *E := x; y := *F; fence; z := *G.
```

The first fence instruction prevents the store operation from reaching the unsafe state in misspeculative execution. The second one prevents z := \*G from being executed mis-speculatively: the operation \*E := x puts an entry in the write buffer and, therefore, the first load instruction may perform store-to-load dependency speculation. By placing the second fence instruction, we prevent z := \*G from reaching the unsafe state during mis-speculation.

The transformation of Figure 12 is obtained by endowing  $\eta$  with a simple static analysis mechanisms that approximates the current mis-speculation flag depending on the instructions and the number of entries in the write buffer. Based on this static analysis, the procedure can perform the optimizations we just discussed to reduce the number of **fence** instruction inserted. As we did for  $\eta$ , we can observe that it satisfies the premises of Theorem 3, thus leading to the following conclusion:

**Lemma 3.** If a system  $\sigma$  is kernel-safe, then  $\psi(\sigma)$  is speculative kernel safe and  $\psi(\sigma)$  semantically equivalent to  $\sigma$ .

# 8.3 Speculation-blocking transformation

Notice that the transformations we proposed in Sections 8.1 and 8.2 prevent the speculative execution of unsafe commands lazily, as they do not completely stop speculative execution, but they just prevent the transient execution of some instructions that may be unsafe. For instance, if in the conditional instruction if E then C else D fi neither C nor D contain potentially unsafe operations—e.g., if they only contain register assignments— $\eta$  and  $\psi$  leave that commands unchanged and the CPU can still speculate over the value of E.

```
\psi(x := \mathtt{E}, m, e) \triangleq (x := \mathtt{E}, m, e)
\psi(\mathtt{skip}, m, e) \triangleq (\mathtt{skip}, m, e)
\psi(\mathtt{*E} := \mathtt{F}, m, e) \triangleq \begin{cases} (\mathtt{*E} := \mathtt{F}, \bot, \bot) & \text{if } m = \bot \\ (\mathtt{fence}; \mathtt{*E} := \mathtt{F}, \bot, \bot) & \text{otherwise} \end{cases}
\psi(x := \mathtt{*E}, m, e) \triangleq \begin{cases} (x := \mathtt{*E}, \neg e, e) & \text{if } m = \bot \\ (\mathtt{fence}; x := \mathtt{*E}, \bot, \top) & \text{otherwise} \end{cases}
\psi(\mathtt{call} \ \mathtt{E}(\mathtt{F}_1, \dots, \mathtt{F}_k), m, e) \triangleq (\mathtt{fence}; \mathtt{scall} \ \mathtt{E}(\mathtt{F}_1, \dots, \mathtt{F}_k), \top, \bot)
\psi(\mathtt{while} \ \mathtt{E} \ \mathtt{do} \ \mathtt{C} \ \mathtt{od}, m, e) \triangleq (\mathtt{let} \ \mathtt{C}', m', e' \leftarrow \psi(\mathtt{C}, \top, \bot) \ \mathtt{in} \ (\mathtt{while} \ \mathtt{E} \ \mathtt{do} \ \mathtt{C}' \ \mathtt{od}, \top, \bot)
\psi(\mathtt{if} \ \mathtt{E} \ \mathtt{then} \ \mathtt{C} \ \mathtt{else} \ \mathtt{D} \ \mathtt{fi}, m, e) \triangleq (\mathtt{elt} \ \mathtt{C}', m', e' \leftarrow \psi(\mathtt{C}, \top, \bot) \ \mathtt{in} \ (\mathtt{if} \ \mathtt{E} \ \mathtt{then} \ \mathtt{C}' \ \mathtt{else} \ \mathtt{D}' \ \mathtt{fi}, \top, \bot)
\psi(\mathtt{C}; \mathtt{D}, m, e) \triangleq (\mathtt{elt} \ \mathtt{C}', m', e' \leftarrow \psi(\mathtt{C}, m, e) \ \mathtt{in} \ (\mathtt{if} \ \mathtt{E} \ \mathtt{then} \ \mathtt{C}' \ \mathtt{else} \ \mathtt{D}' \ \mathtt{fi}, \top, \bot)
\psi(\mathtt{C}; \mathtt{D}, m, e) \triangleq (\mathtt{elt} \ \mathtt{C}', m', e' \leftarrow \psi(\mathtt{C}, m, e) \ \mathtt{in} \ \mathtt{let} \ \mathtt{D}', m'', e'' \leftarrow \psi(\mathtt{D}, m', e') \ \mathtt{in} \ (\mathtt{C}'; \mathtt{D}', m'', e'')
\psi(\mathtt{C}; \mathtt{D}, m, e) \triangleq \mathtt{e}.
\psi(\mathtt{C}; \mathtt{D}, m, e) \triangleq \mathtt{e
```

Figure 12: Optimized fencing transformation

A different approach is to systematically prevent any form of speculation. In our model we only take in account PHT, BTB and STL speculation, therefore we block speculations by:

- placing a fence instruction after every direct branch to stop PHT speculation,
- replacing every ordinary call instruction with a non-speculative one to stop BTB speculation,
   and
- placing a fence instruction after every store operation to stop STL speculation.

The transformation we outlined above is formally defined in Figure 13. It is quite easy to observe that  $\theta$  preserves the semantics of the system it is applied to and that no potentially unsafe instruction can be reached during transient execution. As a consequence, we conclude that  $\theta$  enforces speculative kernel safety on kernel safe systems:

**Lemma 4.** If a system  $\sigma$  is kernel-safe, then  $\theta(\sigma)$  is speculative kernel safe and  $\theta(\sigma)$  semantically equivalent to  $\sigma$ .

# 9 Experimental Evaluation

In the previous section, we showed it is possible to systematically enforce speculative kernel safety on each kernel that is safe in the classical execution model, and we showed three code transformations that can be employed to this aim. In this section, we evaluate the performance overhead of the optimized fencing transformation and the speculation-blocking transformation—shown in Sections 8.2 and 8.3, respectively. We do not include the simple fencing transformation of Section 8.1 because preliminary tests revealed a sensible slowdown in terms of performance with respect to the optimized fencing transformation of Section 8.2.

$$\begin{split} \theta(x := \mathbf{E}) &\triangleq x := \mathbf{E} \\ \theta(\mathtt{skip}) &\triangleq \mathtt{skip} \\ \theta(*\mathbf{E} := \mathbf{F}) &\triangleq *\mathbf{E} := \mathbf{F}; \mathtt{fence} \\ \theta(x := *\mathbf{E}) &\triangleq x := *\mathbf{E} \\ \theta(\mathtt{if} \ \mathsf{E} \ \mathtt{then} \ \mathsf{C} \ \mathtt{else} \ \mathsf{D} \ \mathtt{fi}, m, e) &\triangleq \mathtt{if} \ \mathsf{E} \ \mathtt{then} \ \mathtt{fence}; \mathsf{C} \ \mathtt{else} \ \mathtt{fence}; \mathsf{D} \ \mathtt{fi} \\ \theta(\mathtt{call} \ \mathsf{E}(\mathsf{F}_1, \dots, \mathsf{F}_k)) &\triangleq \mathtt{scall} \ \mathsf{E}(\mathsf{F}_1, \dots, \mathsf{F}_k) \\ \theta(\mathtt{while} \ \mathsf{E} \ \mathtt{do} \ \mathsf{C} \ \mathtt{od}) &\triangleq \mathtt{while} \ \mathsf{E} \ \mathtt{do} \ \mathtt{fence}; \mathsf{C} \ \mathtt{od}; \mathtt{fence} \\ \theta(\mathsf{C}; \mathsf{D}) &\triangleq \theta(\mathsf{C}); \theta(\mathsf{D}) \\ \theta(\epsilon) &\triangleq \epsilon. \\ \\ \theta(\gamma) &\triangleq \mathtt{s} \mapsto \mathtt{fence}; \theta(\gamma(\mathtt{s})) \quad \theta(\tau) &\triangleq \mathtt{id} \mapsto \begin{cases} \tau(\mathtt{id}) & \mathtt{if} \ \mathtt{id} \in \mathsf{Arr} \cup \mathsf{Fun}_{\mathtt{u}} \\ \theta(\tau(\mathtt{id})) & \mathtt{otherwise} \end{cases} \quad \theta((\tau, \gamma, \xi)) &\triangleq (\theta(\tau), \theta(\gamma), \xi). \end{split}$$

Figure 13: Speculation-blocking transformation.

The goal of this experimental evaluation is to measure the impact of the above-mentioned transformations to *kernel-space* execution, and most importantly to user-space programs that combine *kernel-* and *user-* space execution.

# 9.1 Overview of the Implementation

We implemented the transformation of Sections 8.2 and 8.3 as parts of the LLVM/Clang compiler infrastructure [47]. We were able to generalize the transformations from our language to the more expressive x86 instruction code by relying on the LLVM API, which allows us to determine if a certain instruction may load, store, or jump. Instruction showing these behaviors can be instrumented by our implementation in the same way the corresponding transformation protects instructions x := \*E, \*E := F, and call  $E(F_1, ..., F_n)$  respectively.

As we mentioned in Section 8, we implemented fence instructions with the x86 instruction lfence. Our transformations support two ways of achieving the semantics of the non-speculative call instruction scall  $E(F_1, ..., F_n)$ :

- One way is to exchange indirect jump and call instructions with *retpoline* thunks, aimed at preventing branch target speculation. Specifically, we replace indirect jumps and calls with Linux's retpoline thunks at boot time by passing the specific boot command-line parameter spectre\_v2=retpoline.
- A faster alternative is to rely on Intel<sup>®</sup> eIBRS [21], leaving branch instructions unchanged. This measure prevents the target of any indirect branch that is executed in *kernel-space* to be predicted depending on *user-space* execution.

We chose to test both the mechanisms as retpoline offers stronger security guarantees than eIBRS: contrarily to eIBRS retpoline is not subject to confused-deputy-attacks [8, 21], but it has larger performance overheads [11].

	Off-the-shelf	Instrumented	Non-speculating	Off-the-shelf*	Instrumented*	Non-speculating $^*$
Execl Throughput	4354 lps	721 lps	426.6 lps	4455 lps	729.1 lps	425.7 lps
File Copy	4.483 GBps	1.136 GBps	0.7372 GBps	4.658 GBps	1.16 GBps	0.7383 GBps
Pipe Communication Process Creation	232.5 Klps	53.95 Klps	32.89 Klps	239.6 Klps	55.4 Klps	33.79 Klps
	9344 lps	991.4 lps	626.5 lps	9622 lps	985.8 lps	620 lps
Shell Scripts getpid	$5786 \ lpm$ $3234 \ Klps$	2087 lpm 718.8 Klps	1425 lpm 470.9 Klps	5826 lpm 3234 Klps	2091 lpm 718.1 Klps	$1534 \ lpm$ $470.6 \ Klps$

Table 2: Throughput of the UnixBench Benchmark. Measurements from columns marked with \* are collected on kernels protected with retpoline, those from columns that are not marked were collected on kernels using eIBRS. "Off-the-shelf", "Instrumented" and "Non-speculating" refer to the kernels (1), (2), and (3) respectively. The unit *lps* stands for "loops per second", indicating how many times a specific task has completed within one second. Similarly *lps* stands for "loops per minute".

## 9.2 Experimental Evaluation

**Performance Evaluation Goals** Our experimental evaluation aims to determine the performance overhead induced by the transformation of Figure 12 on both *kernel-space* execution, and user-space programs that combine *kernel-* and *user-* space execution.

**Performance Evaluation Methodology** To evaluate the performance overhead of our instrumentations, we ran our benchmarks on a machine running the Debian 12 distribution using three different Linux 6.10.0 kernels:

- 1. **Off-the-shelf kernel**. This kernel was compiled from the Linux source using Clang with Debian 12's default settings.
- 2. Instrumented kernel. This kernel is instrumented with the implementation of the  $\psi$  transformation (Figure 12), selectively placing lfence instructions before instructions that interact with the memory.
- 3. Non-speculating kernel. This kernel is obtained by instrumenting the kernel with an implementation of the  $\theta$  transformation (Figure 13) that blocks kernel-space PHT, BTB and STL speculation.

Specifically, by comparing these kernels we want to assess the overhead caused the  $\psi$  and the  $\theta$  transformations on computationally heavy kernel- and user-space tasks, measured by the UnixBench and the SPEC® CPU 2017 benchmark suites respectively. In particular, the former is meant to evaluate the performance operations that with high pressure on system calls like exec1, getpid, fork, or by executing patterns like file copy and inter-process communication. In contrast, the SPEC® benchmark suite is aimed at evaluating the performance of CPUs on tasks including compilation, interpretation, compression, simulation, and other real-world computationally heavy user-space applications that engage kernel-space execution only sporadically, by issuing system calls. All kernels were compiled with the default settings against speculative attacks. In particular, these settings include support for eIBRS [21], retpoline [70], and BHLDIS\_S [41], which is an Intel® mitigation for Branch History Injection [8]. The settings also include measures against  $Return\ Stack\ Buffer\ (RSB)$  speculation [76]—out of scope for this work—including support for call depth tracking [32] and untrained return thunks, implementing AMD's JMP2RET [5]. All the experiments were run on a machine equipped with an Intel® Core i5-13345 processor with 12 logical cores and 16GB of DDR4 memory.

Data Collection and Results The results of our experimental evaluation are shown in Tables 2 and 3 and summarized in Figure 14. The plots in Figures 14a and 14b show the runtime overhead recorded with the UnixBench benchmark suite [53]. These benchmarks are aimed at measuring the throughput of kernel-space computationally heavy Unix -like operating systems. The UnixBench suite does not report individual times for each run—preventing us from calculating the standard deviation

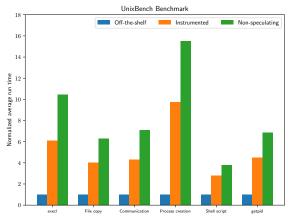
	Off-the-shelf	Instrumented	Non-speculating	Off-the-shelf*	Instrumented*	Non-speculating*
perlbench	$t = 188.039 s$ $\sigma = 0.817 s$	$t = 190.422 s$ $\sigma = 0.922 s$	$t = 190.065 s$ $\sigma = 1.18 s$	$t = 188.653 s$ $\sigma = 0.844 s$	$t = 189.02 s$ $\sigma = 0.826 s$	$t = 190.34 s$ $\sigma = 2.54 s$
gcc	$t = 319.082 s$ $\sigma = 0.638 s$	$t = 333.461 s$ $\sigma = 0.282 s$	$t = 343.766 s$ $\sigma = 0.551 s$	$t = 319.616 s$ $\sigma = 0.686 s$	$t = 333.134 s$ $\sigma = 0.473 s$	$t = 344.004 s$ $\sigma = 1.02 s$
mcf	$t = 436.329 s$ $\sigma = 5.85 s$	$t = 437.874 s$ $\sigma = 0.626 s$	$t = 437.798 s$ $\sigma = 0.679 s$	$t = 436.553 s$ $\sigma = 2.54 s$	$t = 437.095 s$ $\sigma = 0.612 s$	$t = 437.814 s$ $\sigma = 1.05 s$
omnetpp	$t = 287.511 s$ $\sigma = 6.54 s$	t = 288.243  s $\sigma = 4.82  s$	$t = 288.793 s$ $\sigma = 6.53 s$	$t = 286.753 s$ $\sigma = 3.35 s$	$t = 290.807 s$ $\sigma = 7.47 s$	$t = 288.404 s$ $\sigma = 3.53 s$
xalancbmk	t = 154.04  s $\sigma = 0.417  s$	t = 156.932  s $\sigma = 0.285  s$	t = 156.455 s $\sigma = 0.387 s$	t = 155.463 s $\sigma = 0.254 s$	t = 156.038 s $\sigma = 0.424 s$	t = 156.588 s $\sigma = 0.497 s$
x264	t = 128.383 s $\sigma = 0.486 s$	t = 129.983 s $\sigma = 0.48 s$	t = 131.243 s $\sigma = 0.634 s$	t = 128.661 s $\sigma = 0.457 s$	t = 129.994 s $\sigma = 0.588 s$	t = 131.114 s $\sigma = 0.656 s$
deepsjeng	t = 233.841 s $\sigma = 0.206 s$	$t = 234.722 s$ $\sigma = 0.264 s$	t = 234.698 s $\sigma = 0.205 s$	t = 233.996 s $\sigma = 0.241 s$	$t = 235.608 s$ $\sigma = 0.164 s$	$t = 235.884 s$ $\sigma = 0.264 s$
leela	$t = 295.555 s$ $\sigma = 0.961 s$	$t = 296.98 s$ $\sigma = 0.922 s$	$t = 297.286 s$ $\sigma = 0.856 s$	$t = 296.351 s$ $\sigma = 0.859 s$	$t = 296.869 s$ $\sigma = 0.796 s$	$t = 297.33 s$ $\sigma = 0.786 s$
exchange2	$t = 119.528 s$ $\sigma = 0.271 s$	$t = 120.174 s$ $\sigma = 0.194 s$	$t = 120.254 s$ $\sigma = 0.209 s$	$t = 119.996 s$ $\sigma = 0.264 s$	$t = 120.152 s$ $\sigma = 0.223 s$	t = 120.288 s $\sigma = 0.283 s$
xz	$t = 909.593 s$ $\sigma = 3.62 s$	$t = 923.79 s$ $\sigma = 2.94 s$	$t = 933.265 s$ $\sigma = 4.55 s$	$t = 910.888 s$ $\sigma = 2.19 s$	$t = 930.33 s$ $\sigma = 3.04 s$	$t = 937.97 s$ $\sigma = 3.19 s$

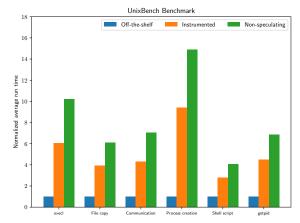
Table 3: Measurements of the average run time t and standard deviation  $\sigma$  over 40 runs of the SPEC® CPU 2017 Benchmark. Measurements from columns marked with \* are collected on kernels protected with retpoline, those from columns that are not marked were collected on kernels using eIBRS. "Off-the-shelf", "Instrumented" and "Non-speculating" refer to the kernels (1), (2), and (3) respectively.

of our sample—instead, it measures how many times a task can be completed in a fixed unit of time. By inverting this measurement, we obtained an indicator of the average time required for each task, we estimated the overhead for each task by dividing its indicator by the estimation we obtained for the off-the-shelf kernel (1). Therefore, the height of each bar represents the run time overhead of the transformations w.r.t. the off-the-shelf kernel on a specific benchmark. The plots in Figures 14c and 14d summarize the result we obtained with the SPEC® CPU 2017 benchmark suite. The height of the bar represents the average execution times recorded for each of the kernels, divided by the average required by the off-the shelf kernel (1). The black interval displays the standard deviation of our sample.

**Discussion** These measurements on the UnixBench benchmark show that the optimized  $\psi$  transformation outperforms the baseline  $\theta$  transformation, although it has an important overhead with respect to the non-instrumented kernel, ranging from 3x to 10x. In contrast, the results of the SPEC® benchmark suite show that the overhead of the  $\psi$  transformation for user-space applications is often below 1%, and always smaller than 5%. Globally, the  $\psi$  transformation performs better than the  $\theta$  transformation, and inversions are often associated with smaller overheads and high variance of the results. Therefore, we attribute this phenomenon to measurement noise, or contingent features of the specific test showing this behavior, and we conclude that it does not constitute evidence that the  $\theta$  transformation is globally preferable to the  $\psi$  transformation. Finally, we noticed that the highest performance overheads of both the transformation  $\psi$  and  $\theta$  occurred when running gcc, x264, and xz: those tests with higher pressure on the system, and in particular on file-system operations. In all these cases, the evaluation showed that there is a considerable advantage in using the transformation  $\psi$  instead of the transformation  $\theta$ .

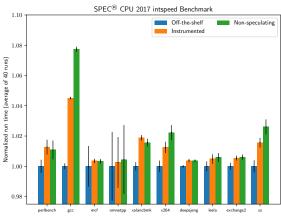
From this experimental evaluation, we conclude that although the sensible performance overhead that we recorded for computationally heavy *kernel-space* tasks, the overhead is minimal—and often negligible—for computationally heavy *user-space* tasks. Therefore, we encourage the adoption of similar measures to protect safe kernels against speculative attacks on system running almost entirely

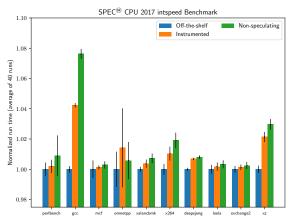




(a) Overhead on the UnixBench Benchmark with eIBRS.

(b) Overhead on the UnixBench Benchmark with retpoline.





(c) Overhead on the SPEC  $^{\circledR}$  CPU 2017 Benchmark with eIBRS.

(d) Overhead on the SPEC  $^{\circledR}$  CPU 2017 Benchmark with retpoline.

Figure 14: Overhead on the system on the UnixBench and SPEC® CPU 2017 Benchmarks.

in user space, like laptops, mobile devices or embedded systems.

# 10 Related Work

On Layout Randomization. The first work that provided a formal account of layout randomization was by Abadi and Plotkin [2], later extended in [1, 4]. In these works, the authors show that layout randomization prevents, with high probability, malicious programs from accessing the memory of a victim in an execution context with shared address space. We have already discussed this in the body of the paper how these results do not model speculative execution or side-channel observations.

**Spatial Memory Safety and Non-Interference** Spatial memory safety is typically defined by associating a software component with a memory area and requiring that, at runtime, it only accesses that area [10, 59, 57]. Azevedo de Amorim et al. [7] demonstrated that memory safety can be expressed in terms of non-interference; this property, in turn, stipulates that the final output of a computation is not influenced by secret data that a program must keep confidential [33]. Both of these properties have been extended to the speculative model. The definition of *speculative memory safety* from [10]

closely aligns with ours, while speculative non-interference was initially introduced in the context of the Spectector symbolic analyzer [37]. Spectector's property captures information flows to side-channels that occur with speculative execution but not in sequential execution. In contrast to Spectector's approach, our definition aligns with speculative constant-time [16], as it specifically targets information leaks that occur with speculative semantics.

Formal Analysis of Security Properties of Privileged Execution Environments. Barthe et al. [9] deploy a model with side-channel leaks and privileged execution mode, without specualtive execution. In particular, they are interested in studying the preservation of constant-time in virtualization platforms. They also model privilege-raising procedures *hypercalls*, similar to our system calls. They show that if one of the hosts is constant-time then the system enjoys a form of non-interference with respect to that host's secret memory. For this reason, although the two models are similar, the purposes of Barthe et al. [9] and our work are different: in [9] the victim and the attacker have the same levels of privilege and the role of the hypervisor is to ensure their separation whilst, in our work, the privileged code base is itself the victim.

Attacks to Kernel Layout Randomization Attacks that aim at leaking information on the kernel's layout are very popular and can rely on implementation bugs that reveal information the kernel's layout [46, 52, 17] or on side-channel info-leaks [35, 45, 51, 14]. In particular attacks such as EchoLoad, TagBleed and EntryBleed [45, 51, 14] are successful even in presence of state-of-art mitigations such as Intel's Page Table Isolation (PTI) [43]. These attacks motivate our decision to take into account side-channel info-leaks. Due to address-space separation between kernel and user space programs, an attacker cannot easily use a pointer to a kernel address to access the victim's memory. So, in general, if the attacker does not control the value of a pointer that is used by the victim, this kind of leak is not harmful.

The Meltdown attack [50] uses speculative execution to overcome this limitation on operating systems running on Intel processors that do not adopt KAISER [36] or PTI [43]. In particular, the hardware can speculatively access an address before checking its permissions. The attack uses this small time window to access kernel memory content and leak it by using a side-channel info-leak gadget. These attacks can also be used to leak information on the layout: by dereferencing pointers under transient execution, the whole kernel's address space can be brute-forced without crashing the system. Due to the adoption of PTI [43], this kind of attack is mitigated by removing most of the kernel-space addresses from the page tables of user-space programs. The BlindSide attack [34] overcomes this issue by probing directly from kernel-space. Similar attacks can be mounted by triggering different forms of mispredictions [54].

Branch target buffer (BTB) speculation—related to Spectre v2—can be used by attackers to defeat kernel's layout randomization. Evtyushkin et al. [28] were able to show that BTB mis-speculations reveal information on victim's layout. Barberis et al. [8] showed how, in the presence of BTB speculation, an attacker can steer the control flow of some kernel's indirect branches, even in the presence of KASLR. As shown by Wiebing et al. [75], currently the Linux kernel contains hundreds of such exploitable indirect jumps. Akin to BTB speculation, Return Stack Buffer (RSB) speculation can be used to transiently divert the victim's control flow to arbitrary locations when the victims executes a return instruction. Wikner and Razavi [76] used this vulnerability to break Layout Randomization and to leak kernel's memory. We did not consider RSB speculation as a threat in our model as, to the best of our knowledge, processors adopting eIBRS are not subject to this vulnerability [76].

Comparison with Eclipse [19] Eclipse [19] is a software-level protection measure that obfuscates the operands of instructions used to perform speculative probing by attacks like BlindSide [34] or PACMAN [63] by inserting artificial data dependencies. The performance evaluation of Eclipse shows outstanding results on *kernel-space* execution with maximum overheads smaller than 8%, i.e., more than 100 times smaller than our best performing transformation  $\psi$ . This phenomenon is not surprising,

and has to do with the high specificity of Eclipse. As Eclipse only considers direct branch speculation, it can prevent speculative safety violations by inserting artificial data dependencies with the values of the guards, in a similar manner as SLH, obtaining good performance. However, it is well known that SLH is not effective when attackers can control other forms of speculations like BTB speculation [15]. Also Eclipse only considers PHT speculation. In contrast, the threat model that we are considering is stronger than the one adopted by Eclipse [19], as in our case attackers can also control STL and BTB speculation. As a consequence, our transformation cannot rely on inserting artificial data dependencies to prevent the speculative execution of vulnerable instructions. Another consequence is that the set of instructions that can be executed speculatively for Eclipse—and therefore need protection—is smaller than for our transformations. In addition, among the speculatively executable instructions, Eclipse is only protecting a narrow class of instructions, e.g. indirect branches, whilst our transformations  $\psi$  protects indirect branches, loads and stores— i.e. all those instructions that can be used by an attacker to violate speculative kernel safety.

Relation Between Security in the Speculative and Classic models Blade [71] is a protection mechanism which is aimed at preventing speculative data-flows by selectively stopping speculations. The authors show that, with this mechanism, all those program that are constant-time in the sequential model, are constant-time in the speculative model too. This is similar to what we do in Section 8, by imposing *speculative kernel safety* on a system that enjoys *kernel safety*. Other mechanisms relating security in the speculative and classic models have been thoroughly discussed in Section 7.

# 11 Conclusion

We have formally demonstrated that the kernel's layout randomization probabilistically ensures kernel safety for a classic model, where an attacker cannot compromise the system via speculative execution or side channels. In this model, users of an operating system execute without privileges, but victims can feature pointer arithmetic, introspection, and indirect jumps. We have also shown that the protection offered by layout randomization does not naturally scale against attackers that can control side-channels and speculative execution related to Spectre v1, v2 and v4. We stipulate a sufficient condition to enforce kernel safety in the Spectre era and we proposed mechanisms based on program transformations that provably enforce speculative kernel safety on a system, provided that this system already enjoys kernel safety in the classic model. We proved the soundness of three such transformation, we implemented them as part of the LLVM/Clang compiler suite, and we established their performance overhead. To the best of our knowledge, our work is the first to formally investigate and provide ways to achieve kernel safety in the presence of speculative and side-channel vulnerabilities.

This work prepares the ground for future developments such as the evaluation of other probabilistic techniques for the enforcement of safety properties such as Arm's PA [49], and hardware backed capability machines like CHERI [73].

Finally, we also consider the possibility to enhance our model with other speculative vulnerabilities including RSB speculation, in order to develop software-level protection measures enforcing speculative kernel safety in this stronger attacker model.

# Acknowledgments

We are grateful to Gilles Barthe, Márton Bognár, Ugo Dal Lago, Lesly-Ann Daniel, Benjamin Grégoire, Jean-Pierre Lozi, Frank Piessens, Aurore Poirier, and Manuel Serrano for their comments on an early draft of the paper. This work was partially supported through the projects PPS ANR-19-C48-0014 and UCA DS4H ANR-17-EURE-0004, and by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

# References

- [1] Martín Abadi and Jérémy Planul. On layout randomization for arrays and functions. In *Principles of Security and Trust*, pages 167–185, Berlin, Heidelberg, 2013. Springer. ISBN 978-3-642-36830-1.
- [2] Martín Abadi and Gordon D. Plotkin. On protection by layout randomization. ACM Trans. Inf. Syst. Secur., 15(2), jul 2012. ISSN 1094-9224.
- [3] Martín Abadi, Mihai Budiu, Ulfar Erlingsson, and Jay Ligatti. Control-flow integrity. In Proceedings of the 12th ACM Conference on Computer and Communications Security, page 340–353, New York, NY, USA, 2005. ACM. ISBN 1595932267.
- [4] Martín Abadi, Jérémy Planul, and Gordon D. Plotkin. Layout Randomization and Nondeterminism, pages 1–39. Springer International Publishing, Berlin, Heidelberg, 2014.
- [5] AMD. Technical guidance for mitigating branch type confusion. Technical report, AMD, November 2022. URL https://www.amd.com/content/dam/amd/en/documents/resources/technical-guidance-for-mitigating-branch-type-confusion.pdf. White Paper.
- [6] Basavesh Ammanaghatta Shivakumar, Gilles Barthe, Benjamin Grégoire, Vincent Laporte, and Swarn Priya. Enforcing fine-grained constant-time policies. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, page 83–96, New York, NY, USA, 2022. ACM. ISBN 9781450394505.
- [7] Arthur Azevedo de Amorim, Cătălin Hriţcu, and Benjamin C Pierce. The meaning of memory safety. In *Proceedings of Principles of Security and Trust: 7th International Conference*, pages 79–105, Springer, 2018. Springer Berlin, Heidelberg.
- [8] Enrico Barberis, Pietro Frigo, Marius Muench, Herbert Bos, and Cristiano Giuffrida. Branch history injection: On the effectiveness of hardware mitigations against Cross-Privilege spectre-v2 attacks. In 31st USENIX Security Symposium (USENIX Security'22), pages 971–988, Boston, MA, August 2022. USENIX Association.
- [9] Gilles Barthe, Gustavo Betarte, Juan Campo, Carlos Luna, and David Pichardie. System-level non-interference for constant-time cryptography. In *Proceedings of the 2014 ACM SIGSAC Con*ference on Computer and Communications Security, page 1267–1279, New York, NY, USA, 2014. ACM.
- [10] Gilles Barthe, Sunjay Cauligi, Benjamin Grégoire, Adrien Koutsos, Kevin Liao, Tiago Oliveira, Swarn Priya, Tamara Rezk, and Peter Schwabe. High-assurance cryptography in the spectre era. In 2021 IEEE Symposium on Security and Privacy (SP), pages 1884–1901, New York, NY, USA, 2021. IEEE.
- [11] Jonathan Behrens, Adam Belay, and M. Frans Kaashoek. Performance evolution of mitigating transient execution attacks. In *Proceedings of the Seventeenth European Conference on Computer* Systems, page 251–265, New York, NY, USA, 2022. ACM. ISBN 9781450391627.
- [12] Emery D. Berger and Benjamin G. Zorn. Diehard: Probabilistic memory safety for unsafe languages. In Proceedings of the 27th ACM SIGPLAN Conference on Programming Language Design and Implementation, page 158–168, New York, NY, USA, 2006. ACM. ISBN 1595933204.
- [13] James Bucek, Klaus-Dieter Lange, and Jóakim v. Kistowski. Spec cpu2017: Next-generation compute benchmark. In *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering*, page 41–42, New York, NY, USA, 2018. ACM.

- [14] Claudio Canella, Michael Schwarz, Martin Haubenwallner, Martin Schwarzl, and Daniel Gruss. Kaslr: Break it, fix it, repeat. In Proceedings of the 15th ACM Asia Conference on Computer and Communications Security, page 481–493, New York, NY, USA, 2020. ACM. ISBN 9781450367509.
- [15] Chandler Carruth. Speculative load hardening, Sep 2018. URL https://llvm.org/docs/SpeculativeLoadHardening.html.
- [16] Sunjay Cauligi, Craig Disselkoen, Klaus v. Gleissenthall, Dean Tullsen, Deian Stefan, Tamara Rezk, and Gilles Barthe. Constant-time foundations for the new spectre era. In *Proceedings of the* 41st ACM SIGPLAN Conference on Programming Language Design and Implementation, page 913–926, New York, NY, USA, 2020. ACM.
- [17] Yueqi Chen, Zhenpeng Lin, and Xinyu Xing. A systematic study of elastic objects in kernel exploitation. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, page 1165–1184, New York, NY, USA, 2020. ACM. ISBN 9781450370899.
- [18] Rutvik Choudhary, Jiyong Yu, Christopher Fletcher, and Adam Morrison. Speculative privacy tracking (spt): Leaking information from speculative execution without compromising privacy. In MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture, page 607–622, New York, NY, USA, 2021. ACM. ISBN 9781450385572.
- [19] Neophytos Christou, Alexander J. Gaidis, Vaggelis Atlidakis, and Vasileios P. Kemerlis. Eclipse: Preventing speculative memory-error abuse with artificial data dependencies. In *Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security*, CCS '24, New York, NY, USA, 2024. ACM.
- [20] Jonathan Corbet. Supervisor mode access prevention, 2012. URL https://lwn.net/Articles/517475/.
- [21] Intel Corporation. Indirect branch restricted speculation, 2018. URL https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/technical-documentation/indirect-branch-restricted-speculation.html.
- [22] Lesly-Ann Daniel, Marton Bognar, Job Noorman, Sébastien Bardin, Tamara Rezk, and Frank Piessens. ProSpeCT: Provably secure speculation for the Constant-Time policy. In 32nd USENIX Security Symposium (USENIX Security 23), pages 7161–7178, Anaheim, CA, August 2023. USENIX Association. ISBN 978-1-939133-37-3.
- [23] Davide Davoli. Comprehensive kernel safety in the spectre era, 2024. URL https://gitlab.inria.fr/ddavoli/comprehensive-kernel-safety-in-the-spectre-era.
- [24] Davide Davoli, Martin Avanzini, and Tamara Rezk. On kernel's safety in the spectre era (and kaslr is formally dead). In Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security, CCS 2024, USA, October 14-18, 2023, New York, NY, USA, 2024. ACM.
- [25] Davide Davoli, Martin Avanzini, and Tamara Rezk. On kernel's safety in the spectre era (extended version), 2024.
- [26] Theo de Raadt. Openbsd 6.3, Oct 2017. URL https://www.openbsd.org/33.html.
- [27] Jake Edge. Kernel address space layout randomization, 2013. URL https://lwn.net/Articles/ 569635/.
- [28] Dmitry Evtyushkin, Dmitry Ponomarev, and Nael Abu-Ghazaleh. Jump over aslr: attacking branch predictors to bypass aslr. In *The 49th Annual IEEE/ACM International Symposium on Microarchitecture*, New York, NY, USA, 2016. IEEE.

- [29] Stephen Fischer. Supervisor mode execution protection, 2011. URL https://www.ncsi.com/nsatc11/presentations/wednesday/emerging\_technologies/fischer.pdf.
- [30] Thomas Garnier. Randomizing the linux kernel heap freelists, Sep 2016. URL https://mxatone.medium.com/randomizing-the-linux-kernel-heap-freelists-b899bb99c767.
- [31] Xinyang Ge, Nirupama Talele, Mathias Payer, and Trent Jaeger. Fine-grained control-flow integrity for kernel software. In 2016 IEEE European Symposium on Security and Privacy (EuroS&P), pages 179–194, New York, NY, USA, 2016. IEEE. doi: 10.1109/EuroSP.2016.24.
- [32] Thomas Gleixner. Fix rsb fill on context switch for serialize. https://lore.kernel.org/all/20220716230344.239749011@linutronix.de/, July 2022. Linux Kernel Mailing List.
- [33] J. A. Goguen and J. Meseguer. Security policies and security models. In 1982 IEEE Symposium on Security and Privacy, New York, NY, USA, 1982. IEEE.
- [34] Enes Göktas, Kaveh Razavi, Georgios Portokalidis, Herbert Bos, and Cristiano Giuffrida. Speculative probing: Hacking blind in the spectre era. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, page 1871–1885, New York, NY, USA, 2020. ACM.
- [35] Daniel Gruss, Clémentine Maurice, Anders Fogh, Moritz Lipp, and Stefan Mangard. Prefetch side-channel attacks: Bypassing smap and kernel aslr. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, page 368–379, New York, NY, USA, 2016. ACM. ISBN 9781450341394.
- [36] Daniel Gruss, Moritz Lipp, Michael Schwarz, Richard Fellner, Clémentine Maurice, and Stefan Mangard. Kaslr is dead: Long live kaslr. In *Engineering Secure Software and Systems*, pages 161–176, Berlin, Heidelberg, 2017. Springer International Publishing. ISBN 978-3-319-62105-0.
- [37] Marco Guarnieri, Boris Köpf, José F. Morales, Jan Reineke, and Andrés Sánchez. Spectector: Principled detection of speculative information flows. In 2020 IEEE Symposium on Security and Privacy (SP), pages 1–19, New York, NY, USA, 2020. IEEE.
- [38] Marco Guarnieri, Boris Köpf, Jan Reineke, and Pepe Vila. Hardware-software contracts for secure speculation. In 2021 IEEE Symposium on Security and Privacy (SP), pages 1868–1883, New York, NY, USA, 2021. IEEE.
- [39] Ralf Hund, Carsten Willems, and Thorsten Holz. Practical timing side channel attacks against kernel space aslr. In 2013 IEEE Symposium on Security and Privacy, pages 191–205, New York, NY, USA, 2013. IEEE.
- [40] Apple Inc. Mac os x has you covered, May 2011. URL http://www.apple.com/macosx/security/.
- [41] Intel. Branch history injection and intra-mode branch target injection / cve-2022-0001, cve-2022-0002 / intel-sa-00598. Technical report, Intel, May 2022. URL https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/technical-documentation/branch-history-injection.html. Technical Documentation.
- [42] Intel ®64 and IA-32 Architectures Software Developer's Manualx. Intel Corporation, September 2023.
- [43] The kernel development community. Page table isolation (pti), 2023. URL https://www.kernel.org/doc/html/next/x86/pti.html.

- [44] Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. Spectre attacks: Exploiting speculative execution. In 2019 IEEE Symposium on Security and Privacy (SP), pages 1–19, New York, NY, USA, 2019. IEEE.
- [45] Jakob Koschel, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. Tagbleed: Breaking kaslr on the isolated kernel address space using tagged tlbs. In 2020 IEEE European Symposium on Security and Privacy (EuroS&P), pages 309–321, New York, NY, USA, 2020. IEEE.
- [46] Jakob Koschel, Pietro Borrello, Daniele Cono D'Elia, Herbert Bos, and Cristiano Giuffrida. Uncontained: Uncovering container confusion in the linux kernel. In 32nd USENIX Security Symposium (USENIX Security 23), pages 5055-5072, Anaheim, CA, August 2023. USENIX Association. ISBN 978-1-939133-37-3.
- [47] C. Lattner and V. Adve. Llvm: a compilation framework for lifelong program analysis & transformation. In *International Symposium on Code Generation and Optimization*, 2004. CGO 2004., pages 75–86, New York, NY, USA, 2004. IEEE.
- [48] Jinku Li, Zhi Wang, Tyler Bletsch, Deepa Srinivasan, Michael Grace, and Xuxian Jiang. Comprehensive and efficient protection of kernel control data. *IEEE Transactions on Information Forensics and Security*, 6(4):1404–1417, 2011.
- [49] Arm Limited. Learn the architecture providing protection for complex software, 2022. URL https://developer.arm.com/documentation/102433/0100.
- [50] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. Meltdown: Reading kernel memory from user space. In 27th USENIX Security Symposium (USENIX Security 18), pages 973–990, Baltimore, MD, August 2018. USENIX Association. ISBN 978-1-939133-04-5.
- [51] William Liu, Joseph Ravichandran, and Mengjia Yan. Entrybleed: A universal kaslr bypass against kpti on linux. In *Proceedings of the 12th International Workshop on Hardware and Architectural Support for Security and Privacy*, HASP '23, page 10–18, New York, NY, USA, 2023. ACM. ISBN 9798400716232.
- [52] Ziqin Liu, Zhenpeng Lin, Yueqi Chen, Yuhang Wu, Yalong Zou, Dongliang Mu, and Xinyu Xing. Towards unveiling exploitation potential with multiple error behaviors for kernel bugs. *IEEE Transactions on Dependable and Secure Computing*, 21(1):1–18, 2023.
- [53] Kelly Lucas. byte-unixbench, 2023. URL https://github.com/kdlucas/byte-unixbench.
- [54] A. Mambretti, A. Sandulescu, A. Sorniotti, W. Robertson, E. Kirda, and A. Kurmus. Bypassing memory safety mechanisms through speculative control flow hijacks. In 2021 IEEE European Symposium on Security and Privacy (EuroS&P), pages 633–649, Los Alamitos, CA, USA, sep 2021. IEEE Computer Society.
- [55] Tarjei Mandt. Attacking the ios kernel: A look at 'evasi0n', March 2013. URL https://papers.put.as/papers/ios/2013/NISlecture201303.pdf.
- [56] Ed Maste. Address space layout randomization (aslr), July 2023. URL https://wiki.freebsd. org/AddressSpaceLayoutRandomization.
- [57] Alexandra E. Michael, Anitha Gollamudi, Jay Bosamiya, Evan Johnson, Aidan Denlinger, Craig Disselkoen, Conrad Watt, Bryan Parno, Marco Patrignani, Marco Vassena, and Deian Stefan. Mswasm: Soundly enforcing memory-safe execution of unsafe code. In Proceedings of the 50th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages, volume 7 of POPL '23, New York, NY, USA, jan 2023. ACM.

- [58] João Moreira, Sandro Rigo, Michalis Polychronakis, and Vasileios P Kemerlis. Drop the rop fine-grained control-flow integrity for the linux kernel, 2017.
- [59] Santosh Nagarakatte, Jianzhou Zhao, Milo M.K. Martin, and Steve Zdancewic. Softbound: Highly compatible and complete spatial memory safety for c. SIGPLAN Not., 44(6):245–258, 2009.
- [60] Marco Patrignani and Marco Guarnieri. Exorcising spectres with secure compilers. In Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security, page 445–461, New York, NY, USA, 2021. ACM. ISBN 9781450384544.
- [61] Android Open Source Project. Kernel hardening, August 2022. URL https://source.android.com/docs/core/architecture/kernel/hardening.
- [62] Liam Proven. Linux 6.1: Rust to hit mainline kernel, October 2022. URL https://www.theregister.com/2022/10/05/rust\_kernel\_pull\_request\_pulled/.
- [63] Joseph Ravichandran, Weon Taek Na, Jay Lang, and Mengjia Yan. Pacman: Attacking arm pointer authentication with speculative execution. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*, pages 685–698, New York, NY, USA, 2022. ACM.
- [64] Elena Reshetova, Hans Liljestrand, Andrew Paverd, and N Asokan. Toward linux kernel memory safety. Software: Practice and Experience, 48(12):2237–2256, 2018.
- [65] Michael S and Vitaly Nikolenko. Linux kernel heap feng shui in 2022, May 2022. URL https://duasynt.com/blog/linux-kernel-heap-feng-shui-2022.
- [66] SecurityScorecard. Threat overview for linux kernel, November 2022. URL https://www.cvedetails.com/product/47/Linux-Linux-Kernel.html.
- [67] Hovav Shacham, Matthew Page, Ben Pfaff, Eu-Jin Goh, Nagendra Modadugu, and Dan Boneh. On the effectiveness of address-space randomization. In *Proceedings of the 11th ACM Conference on Computer and Communications Security*, page 298–307, New York, NY, USA, 2004. ACM. ISBN 1581139616.
- [68] Andrew S. Tanenbaum and Herbert Bos. *Modern Operating Systems*. Prentice Hall Press, USA, 4th edition, 2014. ISBN 013359162X.
- [69] PaX Team. Documentation for the pax project, 2003. URL https://pax.grsecurity.net/docs/.
- [70] Paul Turner. Retpoline: a software construct for preventing branch-target-injection, 2018. URL https://support.google.com/faqs/answer/7625886.
- [71] Marco Vassena, Craig Disselkoen, Klaus von Gleissenthall, Sunjay Cauligi, Rami Gökhan Kıcı, Ranjit Jhala, Dean Tullsen, and Deian Stefan. Automatically eliminating speculative leaks from cryptographic code with blade. *Proc. ACM Program. Lang.*, 5(POPL), jan 2021.
- [72] Zhi Wang and Xuxian Jiang. Hypersafe: A lightweight approach to provide lifetime hypervisor control-flow integrity. In 2010 IEEE Symposium on Security and Privacy, pages 380–395, New York, NY, USA, 2010. IEEE. doi: 10.1109/SP.2010.30.
- [73] Robert N.M. Watson, Jonathan Woodruff, Peter G. Neumann, Simon W. Moore, Jonathan Anderson, David Chisnall, Nirav Dave, Brooks Davis, Khilan Gudka, Ben Laurie, Steven J. Murdoch, Robert Norton, Michael Roe, Stacey Son, and Munraj Vadera. Cheri: A hybrid capability-system architecture for scalable software compartmentalization. In 2015 IEEE Symposium on Security and Privacy, pages 20–37, New York, NY, USA, 2015. IEEE.

- [74] Ofir Weisse, Ian Neal, Kevin Loughlin, Thomas F. Wenisch, and Baris Kasikci. Nda: Preventing speculative execution attacks at their source. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, page 572–586, New York, NY, USA, 2019. ACM. ISBN 9781450369381.
- [75] Sander Wiebing, Alvise de Faveri Tron, Herbert Bos, and Cristiano Giuffrida. InSpectre gadget: Inspecting the residual attack surface of cross-privilege spectre v2. In 33rd USENIX Security Symposium (USENIX Security 24), pages 577–594, Philadelphia, PA, August 2024. USENIX Association. ISBN 978-1-939133-44-1.
- [76] Johannes Wikner and Kaveh Razavi. RETBLEED: Arbitrary speculative code execution with return instructions. In 31st USENIX Security Symposium (USENIX Security 22), pages 3825— 3842, Boston, MA, August 2022. USENIX Association. ISBN 978-1-939133-31-1.
- [77] Jiyong Yu, Mengjia Yan, Artem Khyzha, Adam Morrison, Josep Torrellas, and Christopher W. Fletcher. Speculative taint tracking (stt): A comprehensive protection for speculatively accessed data. In Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture, page 954–968, New York, NY, USA, 2019. ACM. ISBN 9781450369381.
- [78] Zhiyuan Zhang, Gilles Barthe, Chitchanok Chuengsatiansup, Peter Schwabe, and Yuval Yarom. Ultimate slh: taking speculative load hardening to the next level. In *Proceedings of the 32nd USENIX Conference on Security Symposium*, USA, 2023. USENIX Association. ISBN 978-1-939133-37-3.

$$\begin{split} & \mathbb{E} \mathbb{E} \mathbb{A}_{\rho,w}^{\text{Addr}} = p \quad (\mu, m)^0(p) = v, \bot \quad p \in \underline{w}(\mathsf{ArrId}_b) \quad \overline{b = \mathtt{k}_s \Rightarrow p \in \underline{w}(\xi(s))} \\ \hline w \vdash_{\sigma} (\langle x :=_{\ell} *\mathtt{E}; \mathsf{C}, \rho, b \rangle : F, (\mu, m), b_{ms}) : S \xrightarrow{\text{mem } p \\ \text{st}} \rightarrow (\langle \mathsf{C}, \rho[x \leftarrow v], b \rangle : F, (\mu, m), b_{ms}) : S \\ \hline & \mathbb{E} \mathbb{E} \mathbb{E} \mathbb{E} \mathbb{E} (\mathsf{C}, \rho, b \rangle : F, (\mu, m), b_{ms}) : S \xrightarrow{\text{mem } p \\ \text{id}_{\ell} i} } (\langle \mathsf{C}, \rho[x \leftarrow v], b \rangle : F, (\mu, m), b_{ms} \vee b') : (\langle x :=_{\ell} *\mathtt{E}; \mathsf{C}, \rho, b \rangle : F, (\mu, m), b_{ms}) : S \\ \hline & \mathbb{E} \mathbb{E} \mathbb{E} (\mathsf{C}, \rho, b \rangle : F, (\mu, m), b_{ms}) : S \xrightarrow{\text{mem } p \\ \text{id}_{\ell} i} } (\langle \mathsf{C}, \rho[x \leftarrow v], b \rangle : F, (\mu, m), b_{ms} \vee b') : (\langle x :=_{\ell} *\mathtt{E}; \mathsf{C}, \rho, b \rangle : F, (\mu, m), b_{ms}) : S \\ \hline & \mathbb{E} \mathbb{E} \mathbb{E} (\mathsf{C}, \rho, b \rangle : F, (\mu, m), b_{ms}) : S \xrightarrow{\text{mem } p \\ \text{d}} (\langle x :=_{\ell} *\mathtt{E}; \mathsf{C}, \rho, b \rangle : F, (\mu, m), b_{ms}) : S \xrightarrow{\text{mem } p \\ \text{d}} ) \text{ unsafe} \\ \hline & \mathbb{E} \mathbb{E} \mathbb{E} (\mathsf{C}, \rho, b \rangle : F, (\mu, m), b_{ms}) : S \xrightarrow{\text{mem } p \\ \text{d}} ) \text{ unsafe} \\ \hline & \mathbb{E} \mathbb{E} \mathbb{E} (\mathsf{C}, \rho, b \rangle : F, (\mu, m), b_{ms}) : S \xrightarrow{\text{mem } p \\ \text{d}} ) \text{ unsafe} \\ \hline & \mathbb{E} \mathbb{E} (\mathsf{C}, \rho, b \rangle : F, (\mu, m), b_{ms}) : S \xrightarrow{\text{mem } p \\ \text{d}} ) \text{ unsafe} \\ \hline & \mathbb{E} \mathbb{E} (\mathsf{C}, \rho, b \rangle : F, (\mu, m), b_{ms}) : S \xrightarrow{\text{mem } p \\ \text{st}} ) (\langle \mathsf{C}, \rho, b \rangle : F, ([p \mapsto \mathbb{E} \mathbb{F}]_{\rho, w}] \mu, m), b_{ms}) : S \\ \hline & \mathbb{E} \mathbb{E} (\mathsf{C}, \mathsf{C}, \mathsf{C}, \mathsf{C}) : F, (\mu, m), b_{ms}) : S \xrightarrow{\text{mem } p \\ \text{st}} ) (\langle \mathsf{C}, \mathsf{C},$$

Figure 15: Speculative rules for Cmd and a system  $\sigma = (\tau, \gamma, \xi)$ , Part I.

# A Appendix

### A.1 Appendix for Section 6

User- and Kernel-mode stack We write u(C) as a shorthand for  $ids(C) \subseteq Id_u$ , the predicate u(C) is defined inductively as follows:

$$\mathbf{u}(\varepsilon) \triangleq \top \quad \mathbf{u}(f:F) \triangleq \mathbf{u}(f) \wedge \mathbf{u}(F) \quad \mathbf{u}(\langle \mathbf{C}, \rho, b \rangle) \triangleq \mathbf{u}(\mathbf{C}) \wedge b = \mathbf{u}.$$

The predicate  $k_s$  is defined analogously:  $k_s(C)$  holds whenever  $C \in Cmd$ , and it does not contain any syscall  $\cdot (\cdot)$  command and:

$$\mathtt{k}_{\mathtt{s}}(\varepsilon) \triangleq \top \quad \mathtt{k}_{\mathtt{s}}(f:F) \triangleq \mathtt{k}_{\mathtt{s}}(f) \wedge \mathtt{k}_{\mathtt{s}}(F) \quad \mathtt{k}_{\mathtt{s}}(\langle \mathtt{C}, \rho, b \rangle) \triangleq \mathtt{k}_{\mathtt{s}}(\mathtt{C}) \wedge b = \mathtt{k}_{\mathtt{s}}.$$

We write  $k(\cdot)$  as a shorthand for  $\exists s \in Sys.k_s(\cdot)$ .

#### A.1.1 Speculative Semantics of Cmd

The speculative semantics of Cmd is in Figures 15 and 16.

## A.1.2 Semantics of SpAdv

The semantics of SpAdv is in Figures 18 and 19.

```
\begin{split} & \quad \text{I} \in \{\text{call E}(\mathsf{F}_1, \dots, \mathsf{F}_h), \text{scall E}(\mathsf{F}_1, \dots, \mathsf{F}_h)\} \quad \mathbb{E}[\frac{\mathsf{A}^{\mathsf{Addf}}}{\mathsf{A}^{\mathsf{Addf}}} = p \quad p \in \underline{w}(\mathsf{Funld}_b) \quad b = \mathsf{k}_a \Rightarrow p \in \underline{w}(\xi(s))] \\ \hline & \quad w \vdash_{\sigma} (\langle \mathsf{I}; \mathsf{C}, \rho, b \rangle : F, (\mu, m), b_{ms}) : S \xrightarrow{\mathsf{mem}\, p} (\langle m(p), \rho_0[x_1, \dots, x_h \leftarrow [\mathbb{F}_1]_{\rho,w}, \dots, [\mathbb{F}_h]_{\rho,w}], b \rangle : \langle \mathsf{C}, \rho, b \rangle : F, (\mu, m), b_{ms}) : S \\ \hline & \quad I \in \{\mathsf{call E}(\mathsf{F}_1, \dots, \mathsf{F}_h), \mathsf{scall E}(\mathsf{F}_1, \dots, \mathsf{F}_h)\} \quad \mathbb{E}[\mathbb{B}]^{\mathsf{Addf}}_{\mathsf{A}^{\mathsf{Cdf}}} = p \quad p \in \underline{w}(\mathsf{Funld}_k) \quad p \notin \underline{w}(\xi(s))] \\ \hline & \quad w \vdash_{\sigma} (\langle \mathsf{I}; \mathsf{C}, \rho, \mathsf{k}_s \rangle : F, (\mu, m), b_{ms}) : S \xrightarrow{\mathsf{imp}\, p} \mathsf{st} \mathsf{unsafe} \\ \hline & \quad I \in \{\mathsf{call E}(\mathsf{F}_1, \dots, \mathsf{F}_h), \mathsf{scall E}(\mathsf{F}_1, \dots, \mathsf{F}_h)\} \quad \mathbb{E}[\mathbb{B}]^{\mathsf{Addf}}_{\mathsf{p},w} = p \quad p \notin \underline{w}(\mathsf{Funld}_b) \\ \hline & \quad w \vdash_{\sigma} (\langle \mathsf{I}; \mathsf{C}, \rho, b \rangle : F, (\mu, m), b_{ms}) : S \xrightarrow{\mathsf{st}} \mathsf{err}, \mathsf{err}, \mathsf{h}_{ms}) : S \\ \hline & \quad W \vdash_{\sigma} (\langle \mathsf{I}; \mathsf{C}, \rho, b \rangle : F, (\mu, m), b_{ms}) : \mathbb{E}[\mathbb{B}]^{\mathsf{Addf}}_{\mathsf{p},w} = p' \quad p \notin \underline{w}(\mathsf{Funld}_b) \\ \hline & \quad w \vdash_{\sigma} (\mathsf{c}(\mathsf{call E}(\mathsf{F}_1, \dots, \mathsf{F}_h); \mathsf{C}, \rho, b) : F, (\mu, m), b_{ms}) : \mathbb{E}[\mathbb{B}]^{\mathsf{Addf}}_{\mathsf{p},w} = p' \quad p \in \underline{w}(\mathsf{Funld}_b) \\ \hline & \quad w \vdash_{\sigma} (\mathsf{c}(\mathsf{call E}(\mathsf{F}_1, \dots, \mathsf{F}_h); \mathsf{C}, \rho, b) : F, (\mu, m), b_{ms}) : S \xrightarrow{\mathsf{inp}\, p} \mathsf{unsafe} \\ \hline & \quad p \in \underline{w}(\mathsf{Funld}_k) \quad p \notin \underline{w}(\xi(\mathsf{s})) \\ \hline & \quad w \vdash_{\sigma} (\langle \mathsf{call E}(\mathsf{F}_1, \dots, \mathsf{F}_h); \mathsf{C}, \rho, b) : F, (\mu, m), b_{ms}) : S \xrightarrow{\mathsf{inp}\, p} \mathsf{unsafe} \\ \hline & \quad v \vdash_{\sigma} (\langle \mathsf{call E}(\mathsf{F}_1, \dots, \mathsf{F}_h); \mathsf{C}, \rho, b) : F, (\mu, m), b_{ms}) : S \xrightarrow{\mathsf{inp}\, p} \mathsf{unsafe} \\ \hline & \quad v \vdash_{\sigma} (\langle \mathsf{call E}(\mathsf{F}_1, \dots, \mathsf{F}_h); \mathsf{C}, \rho, b) : F, (\mu, m), b_{ms}) : S \xrightarrow{\mathsf{inp}\, p} \mathsf{unsafe} \\ \hline & \quad w \vdash_{\sigma} (\langle \mathsf{call E}(\mathsf{F}_1, \dots, \mathsf{F}_h); \mathsf{C}, \rho, b) : F, (\mu, m), b_{ms}) : S \xrightarrow{\mathsf{inp}\, p} \mathsf{unsafe} \\ \hline & \quad v \vdash_{\sigma} (\langle \mathsf{call E}(\mathsf{F}_1, \dots, \mathsf{F}_h); \mathsf{C}, \rho, b) : F, (\mu, m), b_{ms}) : S \xrightarrow{\mathsf{inp}\, p} \mathsf{unsafe} \\ \hline & \quad v \vdash_{\sigma} (\langle \mathsf{call E}(\mathsf{F}_1, \dots, \mathsf{F}_h); \mathsf{C}, \rho, b) : F, (\mu, m), b_{ms}) : S \xrightarrow{\mathsf{inp}\, p} \mathsf{unsafe} \\ \hline & \quad v \vdash_{\sigma} (\langle \mathsf{call E}(\mathsf{F}_1, \dots, \mathsf{F}_h); \mathsf{c}, \rho, b) : F, (\mu, m), b_{ms}) : S \xrightarrow{\mathsf{inp}\, p} \mathsf{unsafe} \\ \hline &
```

Figure 16: Speculative rules for Cmd and a system  $\sigma=(\tau,\gamma,\xi)$ , Part II.

$$\overline{w \vdash_{\sigma} (\langle x := \mathsf{E} ; \mathsf{C}, \rho, b \rangle : F, (\mu, m), b_{ms}) : S \xrightarrow[\mathsf{st}]{\circ} (\langle \mathsf{C}, \rho[x \leftarrow \llbracket \mathsf{E} \rrbracket_{\rho, w}], b \rangle : F, (\mu, m), b_{ms}) : S}} [SSKP]$$

$$\overline{w \vdash_{\sigma} (\langle \mathsf{skip} ; \mathsf{C}, \rho, b \rangle : F, (\mu, m), b_{ms}) : S \xrightarrow[\mathsf{st}]{\circ} (\langle \mathsf{C}, \rho, b \rangle : F, (\mu, m), b_{ms}) : S}} [SSKP]$$

$$\overline{\mathbb{E}} [\mathbb{B}_{\rho, w}^{\mathsf{Bool}} = d \quad C_{\mathsf{true}} = (\langle \mathsf{D} ; \mathsf{while}_{\ell} \; \mathsf{E} \; \mathsf{do} \; \mathsf{D} \; \mathsf{od} ; \mathsf{C}, \rho, b \rangle : F, (\mu, m), b_{ms}) \quad C_{\mathsf{false}} = (\langle \mathsf{C}, \rho, b \rangle : F, (\mu, m), b_{ms}) : S}} [SLoop]$$

$$w \vdash_{\sigma} (\langle \mathsf{while}_{\ell} \; \mathsf{E} \; \mathsf{do} \; \mathsf{D} \; \mathsf{od} ; \mathsf{C}, \rho, b \rangle : F, (\mu, m), b_{ms} \vee (d \neq \llbracket \mathsf{E} \rrbracket_{\rho, w}^{\mathsf{Bool}})) \quad C_{\mathsf{false}} = (\langle \mathsf{C}, \rho, b \rangle : F, (\mu, m), b_{ms} \vee (d \neq \llbracket \mathsf{E} \rrbracket_{\rho, w}^{\mathsf{Bool}})) : S$$

$$\mathbb{E}} [\mathbb{E} [\mathbb{B}_{\rho, w}^{\mathsf{Bool}}] = d$$

$$\mathbb{E}} [\mathbb{E}} [\mathbb{E} \; \mathsf{then} \; \mathsf{C}_{\mathsf{true}} \; \mathsf{else} \; \mathsf{C}_{\mathsf{false}} \; \mathsf{fi} ; \mathsf{C}, \rho, b \rangle : F, (\mu, m), b_{ms}) : S$$

$$\mathbb{E}} [SIF] [$$

Figure 17: Speculative rules for Cmd and a system  $\sigma = (\tau, \gamma, \xi)$ , Part III.

$$\overline{w \vdash_{\sigma} (\langle \mathsf{poison}(d); \mathtt{A}, \rho, b \rangle : F, m, D, O) \rightarrow (\langle \mathtt{A}, \rho, b \rangle : F, m, d : D, O)}^{[\mathsf{POISON}]}}$$

$$\overline{w \vdash_{\sigma} (\langle x := \mathsf{observe}(); \mathtt{A}, \rho, b \rangle : F, m, D, o : O) \rightarrow (\langle \mathtt{A}, \rho[x \leftarrow o], b \rangle : F, m, D, O)}^{[\mathsf{OBSERVE}]}}$$

$$\overline{w \vdash_{\sigma} (\langle x := \mathsf{observe}(); \mathtt{A}, \rho, b \rangle : F, m, D, e) \rightarrow (\langle \mathtt{A}, \rho[x \leftarrow \mathsf{null}], b \rangle : F, m, D, e)}^{[\mathsf{OBSERVE-END}]}}$$

$$\overline{w \vdash_{\sigma} (\langle \mathsf{spec on C}; \mathtt{A}, \rho, b \rangle : F, m, D, O) \rightarrow (\langle \mathtt{C}, \rho, b \rangle, m, \bot) | (\langle \mathtt{A}, \rho, b \rangle : F, D, O)}^{[\mathsf{Spec-Init}]}}$$

$$\overline{w \vdash_{\sigma} S \mid_{\sigma} S \mid_{$$

Figure 18: Semantics of the non-standard constructs of SpAdv for the system  $\sigma = (\tau, \gamma, \xi)$ .

 $\frac{}{w \vdash_{\sigma} (\mathsf{err}, \bot) \, | \, (F, D, O) \to \mathsf{err}} [ \underbrace{\mathsf{Spec-Error}} ] \quad \frac{}{w \vdash_{\sigma} \mathsf{unsafe} \, | \, (F, D, O) \to \mathsf{unsafe}} [ \underbrace{\mathsf{Spec-Unsafe}} ]$ 

# A.1.3 Buffered Memories

**Remark 1.** if  $(\mu, m)^{i}(p) = v, \bot$ , then  $(\mu, m)^{0}(p) = v, \bot$ .

*Proof.* the claim is:

$$\forall (\mu, m). \forall i. \forall v. (\mu, m)^i(p) = v, \bot \rightarrow (\mu, m)^0(p) = v, \bot$$

By induction on the length of the buffer.

- Case  $\epsilon$ . The claim comes from the definition of lookup.
- Case  $[p' \mapsto \overline{v}] : \mu$ . The IH says:

$$\forall i, v.(\mu, m)^{i}(p) = v, \perp \to (\mu, m)^{i}(p) = (\mu, m)^{0}(p)$$

and the claim is:

$$\forall i, v. ([p' \mapsto \overline{v}] : \mu, m)^i(p) = v, \bot \Rightarrow ([p' \mapsto \overline{v}] : \mu, m)^i(p) = ([p' \mapsto \overline{v}] : \mu, m)^0(p)$$

$$\begin{split} & \mathbb{E}[\mathbb{A}_{\rho,w}^{\mathrm{Addr}} = p \quad p \in \underline{w}(\mathrm{Arrld}_b) \quad \boxed{b = \mathbb{A}_s \Rightarrow p \in \underline{w}(\xi(s))} \\ \hline w \vdash_{\sigma} (\langle x := *\mathsf{E}; \mathsf{C}, \rho, b \rangle : F, m, D, O) \rightarrow (\langle \mathsf{C}, \rho [x \leftarrow m(p)], b \rangle : F, m, D, O)} \\ & \mathbb{E}[\mathbb{B}_{\rho,w}^{\mathrm{Addr}} = p \quad p \in \underline{w}(\mathrm{Arrld}_b) \quad \boxed{b = \mathbb{A}_s \Rightarrow p \in \underline{w}(\xi(s))} \\ \hline w \vdash_{\sigma} (\langle *\mathsf{E} := \mathsf{F}; \mathsf{C}, \rho, b \rangle : F, m, D, O) \rightarrow (\langle \mathsf{C}, \rho, b \rangle : F, m[p \leftarrow [\mathbb{F}]_{\rho,w}], D, O)} \\ & \mathbb{E}[\mathbb{B}_{\rho,w}^{\mathrm{Addr}} = p \quad p \in \underline{w}(\mathrm{Arrld}_k) \quad \boxed{p \notin \underline{w}(\xi(s))} \\ \hline w \vdash_{\sigma} (\langle x := *\mathsf{E}; \mathsf{C}, \rho, k_s \rangle : F, m, D, O) \rightarrow \mathrm{unsafe}} \\ & \mathbb{E}[\mathbb{B}_{\rho,w}^{\mathrm{Addr}} = p \quad p \in \underline{w}(\mathrm{Arrld}_k) \quad \boxed{p \notin \underline{w}(\xi(s))} \\ \hline w \vdash_{\sigma} (\langle *\mathsf{E} := \mathsf{F}; \mathsf{C}, \rho, k_s \rangle : F, m, D, O) \rightarrow \mathrm{unsafe}} \\ & \mathbb{E}[\mathbb{B}_{\rho,w}^{\mathrm{Addr}} = p \quad p \notin \underline{w}(\mathrm{Arrld}_b) \\ \hline w \vdash_{\sigma} (\langle *\mathsf{E} := \mathsf{F}; \mathsf{C}, \rho, k_s \rangle : F, m, D, O) \rightarrow \mathrm{err}} \\ & \mathbb{A} \\ \hline & \mathbb{E}[\mathbb{B}_{\rho,w}^{\mathrm{Addr}} = p \quad p \notin \underline{w}(\mathrm{Arrld}_b) \\ \hline w \vdash_{\sigma} (\langle *\mathsf{E} := \mathsf{F}; \mathsf{C}, \rho, b \rangle : F, m, D, O) \rightarrow \mathrm{err}} \\ \hline & \mathbb{A} \\ \hline & \mathbb{E}[\mathbb{B}_{\rho,w}^{\mathrm{Addr}} = p \quad p \notin \underline{w}(\mathrm{Arrld}_b) \\ \hline w \vdash_{\sigma} (\langle *\mathsf{E} := \mathsf{F}; \mathsf{C}, \rho, b \rangle : F, m, D, O) \rightarrow \mathrm{err}} \\ \hline & \mathbb{A} \\ \hline & \mathbb{A} \\ \hline & w \vdash_{\sigma} (\langle *\mathsf{E} := \mathsf{F}; \mathsf{C}, \rho, b \rangle : F, m, D, O) \rightarrow \mathrm{err}} \\ \hline & \mathbb{A} \\ \hline &$$

Figure 19: Semantics of standard construct of SpAdv for the system  $\sigma = (\tau, \gamma, \xi)$ , part I.

$$\frac{\mathbf{I} \in \{\mathsf{call} \ \mathsf{E}(\mathsf{F}_1, \dots, \mathsf{F}_h), \mathsf{scall} \ \mathsf{E}(\mathsf{F}_1, \dots, \mathsf{F}_h)\} \quad [\![\mathsf{E}]\!]_{\rho, w}^{\mathsf{Addr}} = p \quad p \in \underline{w}(\mathsf{Funld}_b) \quad \boxed{b = \mathtt{k}_{\mathtt{s}} \Rightarrow p \in \underline{w}(\xi(\mathtt{s}))}{w \vdash_{\sigma} (\langle \mathtt{I}; \mathtt{C}, \rho, b \rangle : F, m, D, O) \Rightarrow (\langle m(p), \rho_0[x_1 \leftarrow [\![\mathsf{F}_1]\!]_{\rho, w}, \dots, x_n \leftarrow [\![\mathsf{F}_n]\!]_{\rho, w}], b \rangle : \langle \mathtt{C}, \rho, b \rangle : F, m)} [\mathsf{ACALL}]$$

$$\frac{\mathtt{I} \in \{\mathtt{call}\ \mathtt{E}(\mathtt{F}_1, \dots, \mathtt{F}_h), \mathtt{scall}\ \mathtt{E}(\mathtt{F}_1, \dots, \mathtt{F}_h)\} \quad \llbracket \mathtt{E} \rrbracket_{\rho, w}^{\mathsf{Addr}} = p \quad p \in \underline{w}(\mathsf{Funld_k}) \quad \boxed{p \not\in \underline{w}(\xi(\mathtt{s}))}}{w \vdash_{\sigma} (\langle \mathtt{I}; \mathtt{C}, \rho, \mathtt{k_s} \rangle : F, m, D, O) \Rightarrow \mathsf{unsafe}} \underbrace{\mathsf{ACall-Unsafe}}_{} \\ \underbrace{\mathsf{ACall-Unsafe}_{} \\ \underbrace{\mathsf{ACall-Unsafe}}_{} \\ \underbrace{\mathsf{ACall-Unsafe}_{} \\ \underbrace{\mathsf{ACall-Unsafe}}_{} \\ \underbrace{\mathsf{ACall-Unsafe}_{} \\ \underbrace{\mathsf{ACall-Unsafe}_{} \\ } \\ \underbrace{\mathsf{ACall-Unsafe}_{} \\ \underbrace{\mathsf{ACall-Unsafe}_{} \\ } \\ \underbrace{\mathsf{ACall-Unsafe}_{} \\ } \\ \underbrace{\mathsf{ACall-Unsafe}_{} \\ } \\ \underbrace{\mathsf{ACall-Unsafe}_{} \\ } \\ \underbrace{\mathsf{ACall-Un$$

$$\frac{\mathtt{I} \in \{\mathtt{call}\ \mathtt{E}(\mathtt{F}_1, \dots, \mathtt{F}_h), \mathtt{scall}\ \mathtt{E}(\mathtt{F}_1, \dots, \mathtt{F}_h)\} \quad \llbracket \mathtt{E} \rrbracket_{\rho, w}^{\mathsf{Addr}} = p \quad p \notin \underline{w}(\mathsf{Funld}_b)}{w \vdash_{\sigma} (\langle \mathtt{I}; \mathtt{C}, \rho, b \rangle : F, m, D, O) \Rightarrow \mathsf{err}} [\mathtt{ACall-Error}]$$

$$\overline{w \vdash_{\sigma} (\langle \mathsf{syscall} \ \mathsf{s}(\mathtt{F}_1, \dots, \mathtt{F}_n); \mathtt{C}, \rho, b \rangle : F, m, D, O)} \rightarrow (\langle \gamma(\mathtt{s}), \rho_0[x_1 \leftarrow \llbracket \mathtt{F}_1 \rrbracket_{\rho, w}, \dots, x_n \leftarrow \llbracket \mathtt{F}_n \rrbracket_{\rho, w}], \mathtt{k}_{\mathtt{s}} \rangle : \langle \mathtt{C}, \rho, b \rangle : F, m, D, O)^{[\mathsf{ASC}]} \rightarrow (\langle \mathsf{S}, \mathsf{$$

$$\frac{1}{w \vdash_{\sigma} (\langle \epsilon, \rho, b \rangle : \langle \mathsf{C}, \rho', b' \rangle : F, m, D, O) \Rightarrow (\langle \mathsf{C}, \rho'[ret \leftarrow \rho(ret)], b' \rangle : F, m, D, O)} [\mathsf{APop}]$$

Figure 20: Semantics of standard construct of SpAdv for the system  $\sigma = (\tau, \gamma, \xi)$ , part II.

By cases on i.

- Case 0. The claim is:

$$\forall v. ([p' \mapsto \overline{v}] : \mu, m)^{0}(p) = v, \bot \Rightarrow ([p' \mapsto \overline{v}] : \mu, m)^{0}(p) = ([p' \mapsto \overline{v}] : \mu, m)^{0}(p)$$

Observe that the conclusion is trivial.

- Case i + 1. The claim is:

$$\forall i. \forall v. ([p' \mapsto \overline{v}]: \mu, m)^{i+1}(p) = v, \bot \Rightarrow ([p' \mapsto \overline{v}]: \mu, m)^{i+1}(p) = ([p' \mapsto \overline{v}]: \mu, m)^0(p)$$

Fix i, v, assume  $([p' \mapsto \overline{v}] : \mu, m)^{i+1}(p) = v, \bot$ , call this assumption (H). The claim becomes:

$$([p' \mapsto \overline{v}] : \mu, m)^{0}(p) = ([p' \mapsto \overline{v}] : \mu, m)^{0}(p)$$

Observe that it must be the case where  $p' \neq p$ , otherwise from (H) and the definition of lookup, we obtain  $\bot = \top$ . With this assumption, from (H) we deduce  $(\mu, m)^{i+1}(p) = v, \bot$ , and we can rewrite the claim as follows:

$$(\mu, m)^{i+1}(p) = (\mu, m)^{0}(p)$$

The claim is a consequence of the IH.

**Remark 2.** For every buffered memory  $(\mu, m)$ , and every address p we have that  $(\mu, m)^0(p) = \overline{(\mu, m)}(p)$ .

*Proof.* The proof goes by induction on  $\mu$ . If it is empty, then the claim is a trivial consequence of the definition of lookup. Otherwise, the claim is:

$$([p' \mapsto v] : \mu, m)^{0}(p) = \overline{([p' \mapsto v] : \mu, m)}(p),$$

that rewrites as follows:

$$([p' \mapsto v] : \mu, m)^{0}(p) = \overline{(\mu, m)}[p' \leftarrow v](p).$$

If p = p', the claim is a consequence of the definition of lookup and memory update. Otherwise, it is a consequence of the IH.

**Remark 3.** For every address  $p \in \mathsf{Addr}$ , array  $a \in \mathsf{Arr}$ , store  $\tau$  and  $0 \le i < \mathsf{size}(a)$ , if p = w(a) + i, then

$$(w \diamond \tau)[p \leftarrow v] = w \diamond (\tau[(p, i) \leftarrow v]).$$

*Proof.* By definition.

**Remark 4.** For every buffered memory  $(\mu, (w \diamond \tau))$  if  $dom(\mu) \subseteq w(Arrld)$ , then we have that:

$$\overline{(\mu, (w \diamond \tau))} = w \diamond \tau'$$

for some  $\tau' =_{\mathsf{Funld}} \tau$ 

*Proof.* The proof is by induction on the length of the buffer. If it is 0, then the claim is trivial. Otherwise, it is a consequence of Remark 3.  $\Box$ 

#### A.1.4 Omitted Proofs and Results

In the following we will assume, without lack of generality that all the memories m within a configuration that is reached during the evaluation of a configuration whose memory is  $w \diamond \tau$  is such that  $m = w \diamond \tau'$  for some  $\tau' =_{\mathsf{FunId}} \tau$ . This is justified by Remarks 6 and 7.

Proof of Lemma 1. Assume that a system call s of a system  $\sigma = (\gamma, \tau, \xi)$  is not speculative kernel safe; this means that for some n layout w, register map  $\rho$ , buffered memory  $(\mu, w \diamond \tau')$  with  $\tau' =_{\mathsf{Funld}} \tau$ , sequence of directives D, sequence of observations O, and mis-speculation flag  $b_{ms}$  we have that:

$$w \vdash_{\sigma} (\langle \gamma(s), \rho, \mathtt{k}_s \rangle, (,w) \diamond \tau', b_{ms}) \xrightarrow{O} {}^{n}$$
 unsafe.

By introspection on the rules, we deduce that the rule applied must be one among [SLOAD-UNSAFE], [SSTORE-UNSAFE], [SCALL-UNSAFE], and [SCALL-UNSAFE]. In all these cases, the rightmost observation within O must be  $\mathsf{mem}\,p$  or  $\mathsf{jmp}\,p$  for some address  $p \in \underline{w}(\mathsf{Id}_{\mathtt{k}})$ . More precisely, p belongs to  $\underline{w}(\mathsf{Funld}_{\mathtt{k}})$  if the rule was [SCALL-UNSAFE] and to  $\underline{w}(\mathsf{ArrId}_{\mathtt{k}})$  otherwise. In the following we just show this last case. From the definition of  $\underline{w}(\mathsf{ArrId}_{\mathtt{k}})$ , we deduce that there are  $\mathtt{a} \in \mathsf{ArrId}_{\mathtt{k}}$  and  $0 \le i < \mathsf{size}(\mathtt{a})$  such that  $w(\mathtt{a}) + i = p$ . Our goal now, is to show that there is a layout w' such that  $p \notin \underline{w}'(\mathsf{Id}_{\mathtt{k}})$ , which means that p is not allocated in w'. To build w', we go by cases on  $p' = w(\mathtt{a})$ .

- CASE  $\kappa_{\rm u}$ . In this case, the array is stored at the beginning of the kernel-space address space. From the assumption on the size of this address space, there are at least  $2 \cdot \max_{id \in Id_k} {\sf size(id)} > {\sf size(a)}$  free addresses in the set  $\{\kappa_{\rm u} + {\sf size(a)}, \ldots, \kappa_{\rm u} + \kappa_{\rm k} 1\}$ , so the array can be moved in this space, leaving the address p not allocated. We call w' one such layout.
- Case  $\kappa_k 1 \text{size}(a)$ . Analogous to the case above.
- Case  $\kappa_{\mathbf{u}} < p' < \kappa_{\mathbf{k}} 1 \mathsf{size}(\mathtt{a})$ . Due to the pigeonhole principle, in at least one of the address spaces  $\{\kappa_{\mathbf{u}}, \ldots, p' 1\}$  and  $\{p' + \mathsf{size}(\mathtt{a}), \ldots, \kappa_{\mathbf{k}} 1\mathsf{size}(\mathtt{a})\}$  there are at least  $\max_{\mathtt{id} \in \mathsf{Id}_{\mathbf{k}}} \mathsf{size}(\mathtt{id}) > \mathsf{size}(\mathtt{a})$  not allocated address, this means that a can be moved to one of these sub-spaces, leaving free the gap  $w(\mathtt{a}), \ldots, w(\mathtt{a}) + i$  and in particular p. We call w' one such layout.

From the speculative side-channel layout-non-interference assumption, we deduce that there is S' such that

$$w' \vdash_{\sigma} (\langle \gamma(s), \rho, k_s \rangle, (\mu, w' \diamond \tau'), b_{ms}) \xrightarrow{O} {}^{n} S',$$

Observe that, in particular, this transition produces the sequence of observations O. By applying Remark 11, we deduce that  $\operatorname{\mathsf{mem}} p$  does not appear in O, but this is absurd, because we showed that the rightmost observation of O was exactly  $\operatorname{\mathsf{mem}} p$ .

Proof of Theorem 2. We fix a system  $\sigma = (\tau, \gamma, \xi)$  and go by contraposition. We assume that there are an unprivileged command  $\mathbf{A} \in \mathbf{SpAdv}$ , an initial register map  $\rho$ , a number of steps n and a layout such that:

$$w \vdash_{\sigma} (\langle A, \rho, u \rangle, w \diamond \tau, \epsilon, \epsilon) \rightarrow^{n} unsafe.$$

We first observe that  $n \neq 0$ , and by introspection of the rules of the semantics, we observe that the last rule must be one among [ALOAD-UNSAFE], [ASTORE-UNSAFE], [ACALL-UNSAFE] and [SPEC-UNSAFE] We go by cases on these rules; in particular, the proof in the case of the first three rules is analogous, so we take the case of the rule [ALOAD-UNSAFE] as an example.

- Case [Aload-Unsafe]. By introspection of the rule, we deduce that there is a configuration

$$(\langle x := *E, \rho', k_s \rangle : F, w \diamond \tau', D, O)$$

such that

$$w \vdash_{\sigma} (\langle A, \rho, u \rangle, w \diamond \tau, \epsilon, \epsilon) \rightarrow^{n} (\langle x := *E, \rho', k_s \rangle : F, w \diamond \tau', D, O) \rightarrow \text{unsafe}.$$

with  $\tau' =_{\mathsf{Funld}} \tau$ . With Remark 8, we observe that F is the concatenation of a kernel mode stack F' and a user mode stack F'', so we can apply Lemma 5, and deduce that there is a configuration

$$(\langle \gamma(s), \rho_0[x_1, \dots, x_k \leftarrow v_1, \dots, v_k], k_s \rangle, w \diamond \tau'', D', O'),$$

with  $\tau'' =_{\mathsf{FunId}} \tau$ , a prefix F' of F and  $n' \in \mathbb{N}$  such that:

$$w \vdash_{\sigma} (\langle \gamma(s), \rho_0[x_1, \dots, x_k \leftarrow v_1, \dots, v_k], k_s \rangle, w \diamond \tau'', D', O') \rightarrow^{n'} (\langle x := *E, \rho', k_s \rangle : F', w \diamond \tau', D, O)$$

By introspection of the rule [ALOAD-UNSAFE] we deduce that:

- $\ \llbracket \mathtt{E} \rrbracket_{\rho',w} \in \underline{w}(\mathsf{ArrId}_{\mathtt{k}}).$
- $[\![\mathtt{E}]\!]_{\rho',w} \notin \underline{w}(\xi(\mathtt{s})).$

and this allows us to conclude that the same rule applies to

$$(\langle x := *E, \rho', k_s \rangle : F', w \diamond \tau', D, O),$$

thus showing that

$$(\langle \gamma(s), \rho_0[x_1, \dots, x_k \leftarrow v_1, \dots, v_k], k_s \rangle, w \diamond \tau'', D', O')$$

Reduces in n' + 1 steps to unsafe. With an application of Lemma 6, we deduce that also the speculative configuration

$$(\langle \gamma(s), \rho_0[x_1, \dots, x_k \leftarrow v_1, \dots, v_k], k_s \rangle, w \diamond \tau', \bot)$$

reduces in n'+1 steps to unsafe using the sequence of directives  $\mathsf{st}^{n'+1}$  and this contradicts Lemma 1 applied to the system call  $\mathsf{s}$ .

- Case [Spec-Unsafe]. By introspection of the rule, we deduce that there is a hybrid configuration

$$\mathsf{unsafe} \, | \, (\langle \mathtt{A}, \rho, b \rangle : F, D, O)$$

such that

$$w \vdash_{\sigma} \mathsf{unsafe} \mid (\langle A, \rho, b \rangle : F, D, O) \rightarrow \mathsf{unsafe}$$

and

$$w \vdash_{\sigma} (\langle A, \rho, u \rangle, w \diamond \tau, \epsilon, \epsilon) \rightarrow^{n} \text{unsafe} | (\langle A, \rho, b \rangle : F, D, O)$$

With an application of Remark 14, we deduce that there is a configuration

$$(\langle C, \rho', \mathbf{u} \rangle, (\mu, w \diamond \tau'), \bot),$$

with  $\tau' =_{\mathsf{Funld}} \tau$ , a sequence of directives D', a sequence of observations O' and a natural number  $n' \leq n$  such that:

$$w \vdash_{\sigma} (\langle \mathtt{C}, \rho', \mathtt{u} \rangle, (\mu, w \diamond \tau'), \bot) \xrightarrow{O'} {}^{n'} \mathsf{unsafe}.$$

Because of Lemma 10 we can assume without lack of generality that D' does not contain any bt directive. From Lemma 7, and by introspection of the semantics, we deduce that there are configurations

$$(\langle \gamma(s), \rho'', k_s \rangle, (\mu', w \diamond \tau''), b)$$
 and  $(\langle D, \rho''', k_s \rangle : F'', (\mu'', w \diamond \tau'''), b_{ms})$ 

a sequence of directives D'', a sequence of observations O'' and a store  $\tau''$  such that:

$$w \vdash_{\sigma} (\langle \gamma(\mathtt{s}), \rho'', \mathtt{k}_{\mathtt{s}} \rangle, (\mu', w \diamond \tau''), b) \xrightarrow[D'']{O''} {}^{n''} (\langle \mathtt{D}, \rho''', \mathtt{k}_{\mathtt{s}} \rangle : F'', (\mu'', w \diamond \tau'''), b_{ms}) \xrightarrow[d]{o} \mathsf{unsafe}.$$

but this is in contradiction with Lemma 1 applied to s.

**Lemma 5.** For every system  $\sigma = (\tau, \gamma, \xi)$ , natural number n, configurations

$$C = (\langle \mathtt{A}, \overline{\rho}, \mathtt{u} \rangle, w \diamond \tau', \overline{D}, \overline{O})$$

and

$$(\langle \mathtt{C}, \rho, \mathtt{k_s} \rangle : F_\mathtt{k} : F_\mathtt{u}, w \diamond \tau'', D, O)$$

where A is unprivileged,  $k(F_k)$  and  $u(F_u)$ , such that

$$w \vdash_{\sigma} C \Rightarrow^{n} (\langle \mathsf{C}, \rho, \mathsf{k}_{\mathsf{s}} \rangle : F_{\mathsf{k}} : F_{\mathsf{u}}, w \diamond \tau'', D, O),$$

there is a third configuration

$$(\langle \gamma(\mathbf{s}), \rho', \mathbf{k}_{\mathbf{s}} \rangle, w \diamond \tau''', D, O),$$

a natural number n' such that

$$w \vdash_{\sigma} (\langle \gamma(s), \rho', k_s \rangle, w \diamond \tau''', D, O) \rightarrow^{n'} (\langle C, \rho, k_s \rangle : F_k, w \diamond \tau'', D, O).$$

*Proof.* The proof goes by induction on n. The base case holds for vacuity of the premise The inductive case goes by cases on the rule that has been used to show the last transition. All the rules that do have as target a non-terminal classic configuration except for [APOP], [SPEC-TERM] and [ASC] share a similar proof, so we just show the case of loads:

- Case [Aload]. In this case, the assumption is that

$$w \vdash_{\sigma} C \to^{n} (\langle x := *\mathtt{E}; \mathtt{C}, \rho'', b \rangle : F, w \diamond \tau', D, O) \to (\langle \mathtt{C}, \rho''[x \leftarrow [\![\mathtt{E}]\!]_{\rho'', w}], \mathtt{k_s} \rangle : F_\mathtt{k} : F_\mathtt{u}, w \diamond \tau'', D, O)$$

by introspection of the rule, we deduce  $\rho = \rho''[x \leftarrow [\![\mathtt{E}]\!]_{\rho'',w}]$  and that flag of the source configuration is equal to that of the target one it must be  $\mathtt{k}_{\mathtt{s}}$  as well, so we can apply the IH. This shows that there is a configuration

$$(\langle \gamma(s), \rho', k_s \rangle, w \diamond \tau''', D, O)$$

a natural number n' such that

$$w \vdash_{\sigma} (\langle \gamma(\mathtt{s}), \rho', \mathtt{k}_{\mathtt{s}} \rangle, w \diamond \tau''', D, O) \Rightarrow^{n'} (\langle x := \mathtt{*E}; \mathtt{C}, \rho'', \mathtt{k}_{\mathtt{s}} \rangle : F_{\mathtt{k}}, w \diamond \tau'', D, O).$$

Then we observe that the rule [ALOAD] can be applied to the configuration

$$(\langle x := *\mathbf{E}; \mathbf{C}, \rho'', \mathbf{k_s} \rangle : F_{\mathbf{k}}, w \diamond \tau'', D, O)$$

to show the transition to

$$(\langle \mathtt{C}, \rho''[x \leftarrow [\mathtt{E}]_{\rho'',w}], \mathtt{k}_{\mathtt{S}} \rangle : F_{\mathtt{k}}, w \diamond \tau'', D, O)$$

and this shows the claim.

- Case [ASC]. In this case, the assumption is that

$$w \vdash_{\sigma} C \Rightarrow^{n} (\langle \operatorname{syscall} \operatorname{s}(\mathsf{E}_{1}, \dots, \mathsf{E}_{k}); \mathsf{C}, \rho'', b \rangle : F, w \diamond \tau', D, O) \Rightarrow (\langle \gamma(\mathsf{s}), \rho, \mathsf{k}_{\mathsf{s}} \rangle : \langle \mathsf{C}, \rho'', b \rangle : F_{\mathsf{k}} : F_{\mathsf{u}}, w \diamond \tau', D, O)$$

From Remark 8, we deduce that u(F) and b = u. This shows that the claim holds, in particular  $F_k = \epsilon$ ,  $F_u = F$ , and it is easy to verify that

$$w \vdash_{\sigma} (\langle \mathsf{syscall} \ \mathsf{s}(\mathsf{E}_1, \dots, \mathsf{E}_k); \mathsf{C}, \rho'', b \rangle, w \diamond \tau', D, O) \rightarrow (\langle \gamma(\mathsf{s}), \rho, \mathsf{k}_\mathsf{s} \rangle : \langle \mathsf{C}, \rho'', b \rangle, w \diamond \tau', D, O)$$

holds.

- Case [APop]. In this case, the assumption is that

$$w \vdash_{\sigma} C \Rightarrow^{n} (\langle \mathtt{D}, \rho'', b \rangle : \langle \mathtt{C}, \rho, \mathtt{k}_{\mathtt{s}} \rangle : F, w \diamond \tau', D, O) \Rightarrow (\langle \mathtt{C}, \rho'[ret \leftarrow \rho''(ret)], \mathtt{k}_{\mathtt{s}} \rangle : F, w \diamond \tau', D, O),$$

With an application of Remark 8, we rewrite F as  $F_k : F_u$ , and we deduce that  $b = k_s$ , so we can apply the IH. It shows that there is a configuration

$$(\langle \gamma(\mathbf{s}), \rho', \mathbf{k}_{\mathbf{s}} \rangle, w \diamond \tau''', D, O)$$

a natural number n' such that

$$w \vdash_{\sigma} (\langle \gamma(s), \rho', k_s \rangle, w \diamond \tau''', D, O) \Rightarrow^{n'} (\langle D, \rho'', b \rangle : \langle C, \rho, k_s \rangle : F_k, w \diamond \tau', D, O).$$

To conclude the proof it suffices to verify that

$$w \vdash_{\sigma} (\langle \mathtt{D}, \rho'', b \rangle : \langle \mathtt{C}, \rho, \mathtt{k}_{\mathtt{s}} \rangle : F_{\mathtt{k}}, w \diamond \tau', D, O) \Longrightarrow (\langle \mathtt{C}, \rho'[ret \leftarrow \rho''(ret)], \mathtt{k}_{\mathtt{s}} \rangle : \langle \mathtt{C}, \rho, \mathtt{k}_{\mathtt{s}} \rangle : F_{\mathtt{k}}, w \diamond \tau', D, O).$$

- Case [Spec-Term]. From Remark 10 and introspection on the rules, we conclude that this case is absurd.

**Lemma 6.** For every system  $\sigma = (\overline{\tau}, \gamma, \xi)$ , store  $\tau =_{\mathsf{Funld}} \overline{\tau}$ , configuration  $(\langle \mathtt{C}, \rho, \mathtt{u} \rangle, w \diamond \tau, D, O)$ ,  $n \in \mathbb{N}$ , and every configuration  $(F, w \diamond \tau', D, O)$  such that

$$w \vdash_{\sigma} (\langle \mathtt{C}, \rho, \mathtt{k}_{\mathtt{S}} \rangle, w \diamond \tau, D, O) \rightarrow^{n} (F, w \diamond \tau', D, O),$$

there is a buffered memory  $(\mu, (w \diamond \tau''))$  and a sequence of observations O such that

$$w \vdash_{\sigma} (\langle \mathtt{C}, \rho, \mathtt{k}_{\mathtt{s}} \rangle, w \diamond \tau, \bot) \xrightarrow[\mathtt{st}^{n}]{O} {}^{n} (F, (\mu, (w \diamond \tau'')), \bot)$$

and 
$$\overline{(\mu,(w\diamond\tau''))}=w\diamond\tau'.$$

*Proof.* Notice that for Remarks 8 and 10, the last transition cannot be shown with [SPEC-UNSAFE]. By introspection of the rules, we also deduce that it cannot be shown with [SPEC-TERM] and [SPEC-ERROR]. With this additional observation we go by induction on n, and in the inductive case, we can assume that the last transition was not showed with one of the above-mentioned rules.

- Case 0. Trivial.
- CASE n + 1. In this case, the last rule that has been used cannot be one of [SPEC-TERM], [SPEC-UNSAFE] and [SPEC-ERROR], therefore the n-th configuration cannot be a hybrid one. For this reason, we can restate the premise as:

$$w \vdash_{\sigma} (\langle \mathsf{C}, \rho, \mathsf{k}_{\mathsf{s}} \rangle, w \diamond \tau, D, O) \Rightarrow^{n} (\langle \mathsf{C}', \rho', b' \rangle : F', w \diamond \tau'', D, O) \Rightarrow (F, w \diamond \tau', D, O)$$

We can apply The IH to the first n steps. This shows that there are  $\mu$  and O

such that  $\overline{(\mu,(w\diamond\omega))}=w\diamond\tau''$  and  $k_s(\langle C',\rho',k_s\rangle :F')$ . We are required to show that

$$w \vdash_{\sigma} (\langle \mathtt{C}', \rho', \mathtt{k}_{\mathtt{s}} \rangle : F', (\mu, (w \diamond \omega)), \bot) \xrightarrow[\mathtt{st}]{o} (F, (\mu', (w \diamond \omega')), \bot),$$

that  $\overline{(\mu',(w\diamond\omega'))}=w\diamond\tau'$ . The proof cases on the  $\rightarrow$  relation; many of the cases are similar with the others, so we just show the most important ones. Here, we also observe that the last step cannot be shown with [POISON] or [OBSERVE] because it would again Remark 8 which shows that  $\mathbf{k}(\langle C', \rho', b' \rangle : F')$  holds, and in particular the program C' cannot be an attacker.

- Case [AFENCE]. In this case, the assumption rewrites as follows:

$$w \vdash_{\sigma} (\langle \mathtt{fence}; \mathtt{D}, \rho, \mathtt{k_s} \rangle : F', w \diamond \tau'', D, O) \Rightarrow (\langle \mathtt{D}, \rho, \mathtt{k_s} \rangle : F', w \diamond \tau'', D, O)$$

and the goal is to show that there is an observation o, a buffer  $\mu'$  and a store  $\omega'$  such that:

$$w \vdash_{\sigma} (\langle \mathtt{fence}; \mathtt{D}, \rho, \mathtt{k_s} \rangle : F', (\mu, (w \diamond \omega)), \bot) \xrightarrow{\bullet} (\langle \mathtt{D}, \rho, \mathtt{k_s} \rangle : F', (\mu', (w \diamond \omega')), \bot)$$

and  $\overline{(\mu',(w\diamond\omega'))}=w\diamond\tau''$ . By applying the rule [Fence], suitable buffers and stores for the target configuration are  $\mu'=\epsilon$  and  $\omega'=\tau''$ , and the transition produces the observation  $\circ$ ; we need to observe that  $w\diamond\tau''=\overline{(\epsilon,(w\diamond\tau''))}$ , that is a consequence of the IH and of the definition of  $\bar{\cdot}$  on memories. Finally, we must observe that  $k_s(\langle D,\rho,k_s\rangle:F')$ , but this is a direct consequence of the IH, and of the fact that D is a sub-term of fence; D.

- Case [Aload]. In this case, the assumption rewrites as follows:

$$w \vdash_{\sigma} (\langle x := *\mathtt{E}; \mathtt{D}, \rho, \mathtt{k_s} \rangle : F', w \diamond \tau'', D, O) \Rightarrow (\langle \mathtt{D}, \rho[x \leftarrow w \diamond \tau''([\![\mathtt{E}]\!]^{\mathsf{Addr}}_{\rho, w})], \mathtt{k_s} \rangle : F', w \diamond \tau'', D, O)$$

and the goal is to show that there is an observation o and a buffer  $\mu'$  such that:

$$\begin{split} w \vdash_{\sigma} (\langle x := \star \mathtt{E}; \mathtt{D}, \rho, \mathtt{k}_{\mathtt{s}} \rangle : F', (\mu, (w \diamond \omega)), \bot) &\xrightarrow{\sigma} \\ & (\langle \mathtt{D}, \rho[x \leftarrow w \diamond \tau''(\llbracket \mathtt{E} \rrbracket^{\mathsf{Addr}}_{\rho, w})], \mathtt{k}_{\mathtt{s}} \rangle : F', (\mu', (w \diamond \omega')), \bot) \end{split}$$

and  $\overline{(\mu',(w\diamond\omega))}=w\diamond\tau''$ . We choose  $\mu'=\mu$  and  $\omega=\omega'$  and the equality is a consequence of the IH. The only rule that can apply is [SLOAD-STEP]; with this choice the observation produced is  $\text{mem } [\![\!E]\!]_{\rho,w}^{\mathsf{Addr}}$ ; we need to observe that  $w\diamond\tau''([\![\!E]\!]_{\rho,w}^{\mathsf{Addr}})=(\mu',(w\diamond\omega))^0([\![\!E]\!]_{\rho,w}^{\mathsf{Addr}})$ , that is a consequence of Remark 2. Finally, we must observe that

$$\mathtt{k}_{\mathtt{s}}(\langle \mathtt{D}, \rho[x \leftarrow w \diamond \tau''([\![\mathtt{E}]\!]_{\rho,w}^{\mathsf{Addr}})], \mathtt{k}_{\mathtt{s}} \rangle : F'),$$

but this is a direct consequence of the IH, and of the fact that D is a sub-term of x := \*E; D. Case [ASTORE]. In this case, the assumption rewrites as follows:

$$w \vdash_{\sigma} (\langle *\mathtt{E} := \mathtt{F}; \mathtt{D}, \rho, \mathtt{k_s} \rangle : F', w \diamond \tau'', D, O) \Rightarrow (\langle \mathtt{D}, \rho, \mathtt{k_s} \rangle : F', w \diamond \tau'' [\llbracket \mathtt{E} \rrbracket^{\mathsf{Addr}}_{a,w} \leftarrow \llbracket \mathtt{F} \rrbracket_{\rho,w}], D, O)$$

and the goal is to show that there is an observation o and a buffer  $\mu'$  such that:

$$w \vdash_{\sigma} (\langle *x := \mathtt{E}; \mathtt{D}, \rho, \mathtt{k_s} \rangle : F', (\mu, (w \diamond \omega)), \bot) \xrightarrow[\mathtt{st}]{o} (\langle \mathtt{D}, \rho, \mathtt{k_s} \rangle : F', (\mu', (w \diamond \omega)), \bot)$$

and  $\overline{(\mu',(w\diamond\omega))}=w\diamond\tau''[\llbracket\mathbb{E}\rrbracket_{\rho,w}^{\mathsf{Addr}}\leftarrow\llbracket\mathbb{F}\rrbracket_{\rho,w}]$ . We observe that the rule [SSTORE] applies and produces a target configuration that matches the one we are looking for, in particular, the buffer it produces is

$$[\llbracket \mathtt{E} \rrbracket_{\rho,w}^{\mathsf{Addr}} \mapsto \llbracket \mathtt{F} \rrbracket_{\rho,w}] : \mu.$$

We observe that the conclusion

$$\overline{([\llbracket \mathbb{E} \rrbracket_{\rho,w}^{\mathsf{Addr}} \mapsto \llbracket \mathbb{F} \rrbracket_{\rho,w}] : \mu, (w \diamond \omega))} = w \diamond \tau''[\llbracket \mathbb{E} \rrbracket_{\rho,w}^{\mathsf{Addr}} \leftarrow \llbracket \mathbb{F} \rrbracket_{\rho,w}]$$

comes from the rewriting of the function  $\overline{\cdot}$  and form the assumption  $\overline{(\mu,(w\diamond\omega))}=w\diamond\tau''$ .

- CASE [ACALL]. In this case, we observe that this rule and [SCALL] share the same premises, so also the second one can be applied. By introspection of these rules, we deduce that said  $C' = call \ F(E_1, \ldots, E_k); D$ , the target configurations are respectively

$$(\langle w \diamond \tau''(\llbracket \mathtt{F} \rrbracket_{\rho',w}), \rho'_0, \mathtt{k_s} \rangle : \langle \mathtt{D}, \rho', \mathtt{k_s} \rangle : F', w \diamond \tau'', D, O)$$

and

$$(\langle w \diamond \tau''(\llbracket F \rrbracket_{\rho',w}), \rho'_0, k_s \rangle : \langle D, \rho', k_s \rangle : F', \mu(w \diamond \omega), \bot),$$

where  $\rho'_0 = \rho_0[x_1, \dots, x_k \leftarrow [\![\mathbf{E}_1]\!] \rho', w, \dots, \mathbf{E}_k \rho', w]$ . For this reason, the conclusion on the buffered memory is a consequence of the IH, and we can deduce

$$\mathtt{k}_{\mathtt{s}}(\langle w \diamond \tau''(\llbracket \mathtt{F} \rrbracket_{\rho',w}), \rho'_0, \mathtt{k}_{\mathtt{s}} \rangle : \langle \mathtt{D}, \rho', \mathtt{k}_{\mathtt{s}} \rangle : F')$$

from the IH and the premises of the rule [ACall], which guarantee  $[\![\mathbf{F}]\!]_{\rho',w} \in \underline{w}(\mathsf{Funld_k});$  this, in turn, has as consequence that there is  $\mathbf{f} \in \mathsf{Funld_k}$  such that  $w(\mathbf{f}) = [\![\mathbf{F}]\!]_{\rho',w}$ . Thus, by definition of  $\cdot \diamond \cdot$ , we conclude that

$$w \diamond \tau''(\llbracket \mathbf{f} \rrbracket_{\rho',w}) = \omega(\mathbf{f}) = \overline{\tau}(\mathbf{f}).$$

For this reason, it suffices to observe that  $k_s(\bar{\tau}(f))$  holds by definition of system.

**Lemma 7.** For every system  $\sigma = (\tau, \gamma, \xi)$ , configuration

$$(\langle C, \rho, u \rangle, (\mu, m), \bot)$$

such that u(C) speculative stack  $S = (\langle C, \rho', k_s \rangle : F', (\mu, m)', b_{ms}) : S'$ , sequence of directives D without bt directives, sequence of observations O and layout w such that

$$w \vdash_{\sigma} (\langle C, \rho, \mathbf{u} \rangle, (\mu, m), \bot) \xrightarrow{O} {}^{n} S,$$

there is a configuration  $(\langle \gamma(s), \rho_0[x_0, \ldots, x_h \leftarrow v_1, \ldots, v_h], k_s \rangle : F'', (\mu, m)'', b_{ms})$  a sequence of directives D', a sequence of observations O', and a natural number  $n' \leq n$  such that:

$$w \vdash_{\sigma} (\langle \gamma(\mathtt{s}), \rho'', \mathtt{k}_{\mathtt{s}} \rangle, (\mu, m)'', b''_{ms}) \xrightarrow{O'} {}^{n'} (\langle \mathtt{C}, \rho', \mathtt{k}_{\mathtt{s}} \rangle : \overline{F}, (\mu, m)', b'_{ms}) : \overline{S},$$

for some  $\overline{S}$ , where in particular  $\overline{F}$  is a prefix of F' such that  $k_s(\langle \mathtt{C}, \rho', k_s \rangle : \overline{F})$  and there is a stack F'' such that  $\mathfrak{u}(F'')$  and  $F' = \overline{F} : F''$ .

*Proof.* By induction on n.

- Case 0. Holds by vacuity of the premise.
- Case n+1. The premise rewrites as

$$w \vdash_{\sigma} (\langle \mathtt{C}, \rho, \mathtt{u} \rangle, (\mu, m), \bot) \xrightarrow{O} {}^{n} T \xrightarrow{o} S.$$

We go by cases on the rule that has been applied to show the last transition. Most of these cases are similar to the others; for this reason, we just show some of the most interesting ones. In particular, since D does not contain any bt directive, we can assume without lack of generality that the rules for backtracking are not employed.

- Case [SOP]. In this case we can assume that

$$T = (\langle x := E; D, \rho''', b \rangle : F''', (\mu, m)''', b'''_{ms}) : T'.$$
(†)

By introspection of the rule, we deduce that the flag of the target configuration is equal to that of the source configuration, so we deduce  $b = k_s$ , this also means that we can apply the IH on the first n steps and deduce that

$$w \vdash_{\sigma} (\langle \gamma(\mathtt{s}), \rho'', \mathtt{k}_{\mathtt{s}} \rangle, (\mu, m)'', b''_{ms}) \xrightarrow{O'} {}^{n'} (\langle x := \mathtt{E}; \mathtt{D}, \rho''', b \rangle : \overline{F}''', (\mu, m)''', b'''_{ms}) : \overline{T}',$$

 $\overline{F}'''$  is a prefix of F''' such that  $k_s(\overline{F}''')$  and there is a stack G such that u(G) and  $F''' = \overline{F}''' : G$ . Then, we observe that

$$\begin{split} w \vdash_{\sigma} (\langle x := \mathtt{E}; \mathtt{D}, \rho''', b \rangle : \overline{F}''', (\mu, m)''', b'''_{ms}) : \overline{T}' \xrightarrow[\mathtt{st}]{\circ} \\ (\langle \mathtt{D}, \rho'''[x \leftarrow [\![\mathtt{F}]\!]_{\rho''', w}], b \rangle : \overline{F}''', (\mu, m)''', b'''_{ms}) : \overline{T}', \end{split}$$

and that:

$$\begin{split} w \vdash_{\sigma} (\langle x := \mathtt{E}; \mathtt{D}, \rho''', b \rangle : F''', (\mu, m)''', b'''_{ms}) : T' \xrightarrow{\circ} \\ (\langle \mathtt{D}, \rho'''[x \leftarrow [\![\mathtt{F}]\!]_{\rho''', w}], b \rangle : F''', (\mu, m)''', b'''_{ms}) : T', \end{split}$$

The conclusions that we are required to show are a direct consequence of the IH, and of the observation that  $k_s(x := E; D)$  has as consequence  $k_s(D)$ .

- Case [Sload]. In this case we can assume that

$$T = (\langle x :=_{\ell} *E; D, \rho''', b \rangle : F''', (\mu, m)''', b_{ms}''') : T'.$$
 (†)

By introspection of the rule, we deduce that the execution mode flag of the target configuration is equal to that of the source configuration, so we deduce  $b = k_s$ , this also means that we can apply the IH on the first n steps and deduce that

 $\overline{F}'''$  is a prefix of F''' such that  $k_s(\overline{F}''')$  and there is a stack G such that u(G) and  $F''' = \overline{F}''' : G$ . Then, we call p the value of  $[\![E]\!]_{\rho''',w}^{\mathsf{Addr}}$  and (v,b') the pair that is returned by  $(\mu,m)^{\prime\prime\prime}(p)$ . Then, we observe that

$$w \vdash_{\sigma} (\langle x :=_{\ell} *\mathtt{E}; \mathtt{D}, \rho''', b \rangle : F''', (\mu, m)''', b'''_{ms}) : T' \xrightarrow{\mathsf{mem } p} \\ (\langle \mathtt{D}, \rho'''[x \leftarrow v], b \rangle : F''', (\mu, m)''', b'''_{ms} \vee b') : \\ (\langle x :=_{\ell} *\mathtt{E}; \mathtt{D}, \rho''', b \rangle : F''', (\mu, m)''', b'''_{ms}) : T', (\mu, m)''', b'''', (\mu, m)''', b'''', (\mu, m)'''', b'''', (\mu, m)'''', b'''', (\mu, m)'''', b'''', (\mu, m)'''', b''''', (\mu, m)'''', b''''', (\mu, m)'''', b'''', (\mu, m)'''', (\mu, m)'''', (\mu, m)'''', (\mu,$$

and that:

$$w \vdash_{\sigma} (\langle x :=_{\ell} *\mathtt{E}; \mathtt{D}, \rho''', b \rangle : \overline{F}''', (\mu, m)''', b'''_{ms}) : \overline{T}' \xrightarrow{\mathsf{mem} \, p} \\ (\langle \mathtt{D}, \rho'''[x \leftarrow v], b \rangle : \overline{F}''', (\mu, m)''', b'''_{ms} \lor b') : \\ (\langle x :=_{\ell} *\mathtt{E}; \mathtt{D}, \rho''', b \rangle : \overline{F}''', (\mu, m)''', b'''_{ms}) : \overline{T}', (\mu, m)'''', b'''_{ms}) : \overline{T}', (\mu, m)''', b'''_{ms}) : \overline{T}',$$

The conclusions that we are required to show are a direct consequence of the IH and of the observation that  $k_s(x := *E; D)$  has as consequence  $k_s(D)$ .

- Case [SCall]. In this case we can assume that T is

$$(\langle \mathtt{call} \ \mathtt{E}(\mathtt{F}_1, \dots, \mathtt{F}_k); \mathtt{D}, \rho''', b \rangle : F''', (\mu, m)''', b'''_{ms}) : T'.$$

By introspection of the rule, and by knowing that the execution flag of the target configuration is  $k_s$ , we deduce that this must be the case also for the source configuration, so  $b = k_s$ , this also means that we can apply the IH on the first n steps and deduce that

$$w \vdash_{\sigma} (\langle \gamma(\mathtt{s}), \rho'', \mathtt{k}_{\mathtt{s}} \rangle, (\mu, m)'', b''_{ms}) \xrightarrow{O'} {}^{n'} (\langle \mathtt{call} \ \mathtt{E}(\mathtt{F}_1, \dots, \mathtt{F}_k); \mathtt{D}, \rho''', b \rangle : \overline{F}''', (\mu, m)''', b'''_{ms}) : \overline{T}', (\mu, m)''', b'''', b'''', b'''', b'''', b'''', b'''', b'''', b''''', b'''', b'''', b'''', b'''', b'''', b'''', b'''', b''''', b'''', b'''', b'''', b'''', b''''', b'''', b'''', b'''', b'''', b''''', b''''', b'''', b'''', b'''', b'''', b'''', b'''', b''''', b'''', b'''', b'''', b'''', b''''', b''''', b''''', b''''', b''''', b'''', b''''', b''''', b''''', b''''', b''''', b''''', b''''',$$

 $\overline{F}'''$  is a prefix of F''' such that  $k_s(\overline{F}''')$  and there is a stack G such that u(G) and  $F'''=\overline{F}''':G$ . Then, we call p the value of  $[\![E]\!]_{\rho''',w}^{\operatorname{Addr}}$  and we observe that form the premise of the rule and the definition of  $\underline{w}$ , we can deduce that there is  $\mathbf{f} \in \operatorname{Funld}_b$  such that  $w(\mathbf{f}) = p$ . From this observation, and Remark 7, we deduce that  $(\mu,m)''' = (\mu',w\diamond\tau')$  for a  $\tau' =_{\operatorname{Funld}}\tau$  So, from the definition of  $\cdot \diamond \cdot$  and these observations we conclude that the executed procedure is exactly  $\tau(\mathbf{f})$ . We also call  $\overline{\rho}$  the register map that is obtained by evaluating the semantics of the arguments in  $\rho'''$  and updating the argument registers of  $\rho_0$  with these values. Then, by introspection of the rule, we observe that

$$w \vdash_{\sigma} (\langle \mathtt{call} \ \mathtt{E}(\mathtt{F}_{1}, \dots, \mathtt{F}_{k}); \mathtt{D}, \rho''', b \rangle : F''', (\mu, m)''', b'''_{ms}) : T' \xrightarrow{\mathtt{jmp} \ p} (\langle \tau(\mathtt{f}), \overline{\rho}, b \rangle : \langle \mathtt{D}, \rho''', b \rangle : F''', (\mu, m)''', b'''_{ms}) : T',$$

and that also:

$$w \vdash_{\sigma} (\langle \mathtt{call} \ \mathtt{E}(\mathtt{F}_{1}, \dots, \mathtt{F}_{k}); \mathtt{D}, \rho''', b \rangle : \overline{F}''', (\mu, m)''', b'''_{ms}) : \overline{T}' \xrightarrow{\mathtt{jmp} \, p} \\ (\langle \tau(\mathtt{f}), \overline{\rho}, b \rangle : \langle \mathtt{D}, \rho''', b \rangle : \overline{F}''', (\mu, m)''', b'''_{ms}) : \overline{T}'.$$

can be shown. The conclusions that we are required to show are a direct consequence of the IH, and of the observation that  $k_s(\tau(f))$  holds by definition of  $\tau$  because  $f \in \mathsf{Funld}_k$ .

- Case [SSystemCall]. In this case we can assume that T is

$$(\langle \text{syscall } \mathsf{t}(\mathsf{F}_1,\ldots,\mathsf{F}_k); \mathsf{D},\rho''',b\rangle : F''',(\mu,m)''',b'''_{ms}) : T'.$$

By introspection of the rule and the target configuration, we deduce that t = s; and that S has the following shape:

$$(\langle \gamma(\mathbf{s}), \rho'_0, \mathbf{k}_{\mathbf{s}} \rangle : \langle \mathbf{D}, \rho''', b \rangle : F''', (\mu, m)''', b'''_{ms}) : T'.$$

where  $\rho'_0$  is obtained by updating the argument registers of  $\rho_0$  with the evaluation of  $E_1, \ldots, E_k$ . To show the claim, it suffices to set n'=0,  $D'=\epsilon$ ,  $O=\epsilon$ ,  $\overline{F}=\varepsilon$ , F''=F', and the observation  $k_s(\langle \gamma(s), \rho'_0, k_s \rangle)$  holds for definition of  $\gamma$ , while  $u(\langle D, \rho''', b \rangle : F''')$  is a consequence of Remark 12 and of u(C). In particular, we can refuse the assumption that  $b=k_t$  for some t, because in such case we could not have a system call invocation as a command. Therefore, it must be the case that  $u(syscall\ s(E_1, \ldots, E_k); D)$ .

- Case [SPOP]. In this case we can assume that T is

$$(\langle \epsilon, \rho''', b \rangle : F''', (\mu, m)''', b'''_{ms}) : T'.$$

And by introspection of the rule, we deduce that

$$F''' = \langle \mathtt{C}, \rho'[ret \leftarrow v], \mathtt{k}_{\mathtt{s}} \rangle : F'$$

for some v; from this observation and Remark 12, we deduce that  $b = k_s$ , so we can apply the IH to the first n steps and obtain that

$$w \vdash_{\sigma} (\langle \gamma(\mathtt{s}), \rho'', \mathtt{k}_{\mathtt{s}} \rangle, (\mu, m)'', b'''_{ms}) \xrightarrow{O'} \ ^{n'} (\langle \epsilon, \rho''', b \rangle : \overline{F}''', (\mu, m)''', b'''_{ms}) : \overline{T}',$$

 $\overline{F}'''$  is a prefix of F''' such that  $k_s(\overline{F}''')$  and there is a stack G such that u(G) and  $F''' = \overline{F}''' : G$ . This shows that, in particular, the topmost frame of  $\overline{F}'''$  must also be  $\langle C, \rho'[ret \leftarrow v], k_s \rangle$ , so we have  $\overline{F}''' = \langle C, \rho'[ret \leftarrow v], k_s \rangle : \overline{F}'$ . Thanks to this observation, by introspection of the rule, we observe that

$$w \vdash_{\sigma} (\langle \epsilon, \rho''', \mathtt{k_s} \rangle : \langle \mathtt{C}, \rho'[ret \leftarrow v], \mathtt{k_s} \rangle : F', (\mu, m)''', b'''_{ms}) : T' \xrightarrow{\circ}_{\mathsf{st}} (\langle \mathtt{C}, \rho', \mathtt{k_s} \rangle : F', (\mu, m)''', b'''_{ms}) : T', (\mu, m)''', b'''_{ms}) : T' \xrightarrow{\circ}_{\mathsf{st}}_{\mathsf{st}} (\langle \mathtt{C}, \rho', \mathtt{k_s} \rangle : F', (\mu, m)''', b'''_{ms}) : T' \xrightarrow{\circ}_{\mathsf{st}}_{\mathsf{st}}_{\mathsf{st}} (\langle \mathtt{C}, \rho', \mathtt{k_s} \rangle : F', (\mu, m)''', b'''_{ms}) : T' \xrightarrow{\circ}_{\mathsf{st}}_{$$

and that also:

$$w \vdash_{\sigma} (\langle \epsilon, \rho''', \mathtt{k_s} \rangle : \langle \mathtt{C}, \rho'[ret \leftarrow v], \mathtt{k_s} \rangle : \overline{F}', (\mu, m)''', b'''_{ms}) : \overline{T}' \xrightarrow{\circ} (\langle \mathtt{C}, \rho', \mathtt{k_s} \rangle : \overline{F}', (\mu, m)''', b'''_{ms}) : \overline{T}'.$$

Notice that  $k_s(\overline{F}')$ , and  $F' = \overline{F}' : G$ . This concludes the proof.

### A.2 Proofs from Section 8

Before going to the proof of Theorem 3, we give a simple remark that will turn out to be useful in the following:

**Remark 5.** If  $\sigma$  is kernel safe,  $\zeta$  is user-space semantics preserving, and  $\zeta(\sigma) = \sigma'$ , then  $\sigma'$  is also kernel safe.

Proof of Theorem 3. We fix a system  $\sigma = (\tau, \gamma, \xi) = \zeta(\sigma')$  for some safe system  $\zeta$  that preserves systems' semantics and enforces speculative safety. Claim (ii) is trivial, therefore we only focus on Claim (i). The proof goes by contraposition. We assume that there are an unprivileged command  $A \in SpAdv$ , an initial register map  $\rho$ , a number of steps n and a layout such that:

$$w \vdash_{\sigma} (\langle A, \rho, u \rangle, w \diamond \tau, \epsilon, \epsilon) \rightarrow^{n} unsafe.$$

We first observe that  $n \neq 0$ , and by introspection of the rules of the semantics, we observe that the last rule must be one among [ALOAD-UNSAFE], [ASTORE-UNSAFE], [ACALL-UNSAFE] and [SPEC-UNSAFE] We go by cases on these rules; in particular, the proof in the case of the first three rules is analogous, so we take the case of the rule [ALOAD-UNSAFE] as an example.

- CASE [ALOAD-UNSAFE]. By introspection of the rule, we deduce that there is a configuration

$$(\langle x := *E, \rho', k_s \rangle : F, w \diamond \tau', D, O)$$

such that

$$w \vdash_{\sigma} (\langle A, \rho, u \rangle, w \diamond \tau, \epsilon, \epsilon) \rightarrow^{n} (\langle x := *E, \rho', k_s \rangle : F, w \diamond \tau', D, O) \rightarrow \text{unsafe}.$$

with  $\tau' =_{\mathsf{FunId}} \tau$ . With Remark 8, we observe that F is the concatenation of a kernel mode stack F' and a user mode stack F'', so we can apply Lemma 5, and deduce that there is a configuration

$$(\langle \gamma(s), \rho_0[x_1, \dots, x_k \leftarrow v_1, \dots, v_k], k_s \rangle, w \diamond \tau'', D', O'),$$

with  $\tau'' =_{\mathsf{FunId}} \tau$ , a prefix F' of F and  $n' \in \mathbb{N}$  such that:

$$w \vdash_{\sigma} (\langle \gamma(\mathbf{s}), \rho_0[x_1, \dots, x_k \leftarrow v_1, \dots, v_k], \mathbf{k}_{\mathbf{s}} \rangle, w \diamond \tau'', D', O')$$

$$\rightarrow^{n'} (\langle x := *\mathbf{E}, \rho', \mathbf{k}_{\mathbf{s}} \rangle : F', w \diamond \tau', D, O) \rightarrow \text{unsafe}$$

with an application of lemma 11, we deduce:

$$w \vdash_{\sigma} (\langle \gamma(s), \rho_0[x_1, \dots, x_k \leftarrow v_1, \dots, v_k], \mathtt{k}_s \rangle, w \diamond \tau'') \rightarrow^{n'} (\langle x := *\mathtt{E}, \rho', \mathtt{k}_s \rangle : F', w \diamond \tau') \rightarrow \mathsf{unsafe},$$

this shows that  $\sigma$  is not kernel safe, but is in contradiction with Remark 5, that shows that  $\sigma$  is safe

- CASE [Spec-Unsafe]. By introspection of the rule, we deduce that there is a hybrid configuration

$$\mathsf{unsafe} \mid (\langle \mathtt{A}, \rho, b \rangle : F, D, O)$$

such that

$$w \vdash_{\sigma} \mathsf{unsafe} \mid (\langle \mathtt{A}, \rho, b \rangle : F, D, O) \Rightarrow \mathsf{unsafe}$$

and

$$w \vdash_{\sigma} (\langle \mathtt{A}, \rho, \mathtt{u} \rangle, w \diamond \tau, \epsilon, \epsilon) \rightarrow^{n} \mathsf{unsafe} \,|\, (\langle \mathtt{A}, \rho, b \rangle : F, D, O)$$

With an application of Remark 14, we deduce that there is a configuration

$$(\langle \mathsf{C}, \rho', \mathsf{u} \rangle, (\mu, w \diamond \tau'), \bot),$$

with  $\tau' =_{\mathsf{Funld}} \tau$ , a sequence of directives D', a sequence of observations O' and a natural number  $n' \le n$  such that:

$$w \vdash_{\sigma} (\langle \mathtt{C}, \rho', \mathtt{u} \rangle, (\mu, w \diamond \tau'), \bot) \xrightarrow{O'}_{D'} {}^{n'} \mathsf{unsafe}.$$

Because of Lemma 10 we can assume without lack of generality that D' does not contain any bt directive. From Lemma 7, and by introspection of the semantics, we deduce that there are configurations

$$(\langle \gamma(\mathtt{s}), \rho'', \mathtt{k}_\mathtt{s} \rangle, (\mu', w \diamond \tau''), b) \quad \text{and} \quad (\langle \mathtt{D}, \rho''', \mathtt{k}_\mathtt{s} \rangle : F'', (\mu'', w \diamond \tau'''), b_{ms})$$

a sequence of directives D'', a sequence of observations O'' and a store  $\tau''$  such that:

$$w \vdash_{\sigma} (\langle \gamma(\mathtt{s}), \rho'', \mathtt{k}_{\mathtt{s}} \rangle, (\mu', w \diamond \tau''), b) \xrightarrow[D'']{O''} {}^{n''} (\langle \mathtt{D}, \rho''', \mathtt{k}_{\mathtt{s}} \rangle : F'', (\mu'', w \diamond \tau'''), b_{ms}) \xrightarrow[d]{o} \mathsf{unsafe}.$$

We apply our assumption on  $\zeta$ , which yields:

$$w \vdash_{\sigma} (\langle \gamma(\mathtt{s}), \rho'', \mathtt{k}_{\mathtt{s}} \rangle, \overline{(\mu', w \diamond \tau'')}, b) \to^* \mathsf{unsafe},$$

this shows that  $\sigma$  is unsafe, but it is also in contradiction with Remark 5, showing that it is actually safe.

Proof of Lemma 2. This result is a direct consequence of Corollary 2, Lemma 8 and of Theorem 3.

*Proof of Lemma 3.* This result is a direct consequence of Corollary 2, Lemma 18 and of Theorem 3.

Proof of Lemma 4. This result is a direct consequence of Corollary 2, Lemma 22 and of Theorem 3.

We extend the semantics of Figures 3 and 4 by giving the following semantics to the fence instruction:

$$\overline{w \vdash_{\sigma} (\langle \mathtt{fence}; \mathtt{C}, \rho, b \rangle : F, m) \to (\langle \mathtt{C}, \rho, b \rangle : F, m)}^{\big[\mathtt{Fence}\big]}$$

and the following rules for safe calls:

$$\frac{\forall \mathtt{f} \in \mathsf{Funld}_\mathtt{k}. \exists \mathtt{C}. \tau'(\mathtt{f}) = \eta(\mathtt{C}) \quad \tau =_{\mathsf{Funld}} \tau'}{\eta, \sigma \vdash \mathsf{twf}_{w,\mathtt{s}}(\tau')} \quad \frac{\eta, \sigma \vdash \mathsf{twf}_{w,\mathtt{s}}(F) \quad \exists \mathtt{C}'. \mathtt{C} = \eta(\mathtt{C}')}{\eta, \sigma \vdash \mathsf{twf}_{w,\mathtt{s}}(\langle \mathtt{C}, \rho, \mathtt{k}_\mathtt{s} \rangle : F)}$$
$$\overline{\eta, \sigma \vdash \mathsf{twf}_{w,\mathtt{s}}(\mathsf{err})} \quad \overline{\eta, \sigma \vdash \mathsf{twf}_{w,\mathtt{s}}(\mathsf{unsafe})}$$

$$\frac{\eta, \sigma \vdash \mathsf{twf}_{w, \mathtt{s}}(\tau') \quad \eta, \sigma \vdash \mathsf{twf}_{w, \mathtt{s}}(F) \quad \mathsf{dom}(\mu) \subseteq \underline{w}(\mathsf{ArrId})}{\eta, \sigma \vdash \mathsf{twf}_{w, \mathtt{s}}((F, (\mu, w \diamond \tau'), b_{ms}))} \quad \frac{\eta, \sigma \vdash \mathsf{twf}_{w, \mathtt{s}}(C)}{\eta, \sigma \vdash \mathsf{twf}_{w, \mathtt{s}}(C : S)} \quad \frac{\eta, \sigma \vdash \mathsf{twf}_{w, \mathtt{s}}(C)}{\eta, \sigma \vdash \mathsf{twf}_{w, \mathtt{s}}(C : S)} \quad \frac{\eta, \sigma \vdash \mathsf{twf}_{w, \mathtt{s}}(C : S)}{\eta, \sigma \vdash \mathsf{twf}_{w, \mathtt{s}}(C : S)} \quad \frac{\eta, \sigma \vdash \mathsf{twf}_{w, \mathtt{s}}(C : S)}{\eta, \sigma \vdash \mathsf{twf}_{w, \mathtt{s}}(C : S)} \quad \frac{\eta, \sigma \vdash \mathsf{twf}_{w, \mathtt{s}}(C : S)}{\eta, \sigma \vdash \mathsf{twf}_{w, \mathtt{s}}(C : S)} \quad \frac{\eta, \sigma \vdash \mathsf{twf}_{w, \mathtt{s}}(C : S)}{\eta, \sigma \vdash \mathsf{twf}_{w, \mathtt{s}}(C : S)} \quad \frac{\eta, \sigma \vdash \mathsf{twf}_{w, \mathtt{s}}(C : S)}{\eta, \sigma \vdash \mathsf{twf}_{w, \mathtt{s}}(C : S)} \quad \frac{\eta, \sigma \vdash \mathsf{twf}_{w, \mathtt{s}}(C : S)}{\eta, \sigma \vdash \mathsf{twf}_{w, \mathtt{s}}(C : S)} \quad \frac{\eta, \sigma \vdash \mathsf{twf}_{w, \mathtt{s}}(C : S)}{\eta, \sigma \vdash \mathsf{twf}_{w, \mathtt{s}}(C : S)} \quad \frac{\eta, \sigma \vdash \mathsf{twf}_{w, \mathtt{s}}(C : S)}{\eta, \sigma \vdash \mathsf{twf}_{w, \mathtt{s}}(C : S)} \quad \frac{\eta, \sigma \vdash \mathsf{twf}_{w, \mathtt{s}}(C : S)}{\eta, \sigma \vdash \mathsf{twf}_{w, \mathtt{s}}(C : S)} \quad \frac{\eta, \sigma \vdash \mathsf{twf}_{w, \mathtt{s}}(C : S)}{\eta, \sigma \vdash \mathsf{twf}_{w, \mathtt{s}}(C : S)} \quad \frac{\eta, \sigma \vdash \mathsf{twf}_{w, \mathtt{s}}(C : S)}{\eta, \sigma \vdash \mathsf{twf}_{w, \mathtt{s}}(C : S)}$$

Figure 21: Well-formedness relation for the system transformation  $\eta$  with respect to a system  $\sigma = (\tau, \gamma, \xi)$ .

$$\frac{\forall \mathbf{f} \in \mathsf{Funld}_{\mathbf{k}}. \exists \mathtt{C}.\tau'(\mathbf{f}) = \psi(\mathtt{C}, \top, \bot) \quad \tau =_{\mathsf{Funld}} \tau'}{\psi, \sigma \vdash \mathsf{twf}_{w,\mathtt{s}}(\tau')} \quad \frac{\psi, \sigma \vdash \mathsf{twf}_{w,\mathtt{s}}(\tau') \quad \mathsf{dom}(\mu) \subseteq \underline{w}(\mathsf{Arrld})}{\psi, \sigma \vdash \mathsf{twf}_{w,\mathtt{s}}((\epsilon, (\mu, w \diamond \tau'), b_{ms}))} \\ \frac{\psi, \sigma \vdash \mathsf{twf}_{w,\mathtt{s}}(\tau') \quad \Sigma(\mathtt{C}, m, e), \wedge \Sigma(\mathtt{C}_1, \top, \bot) \wedge \ldots \wedge \Sigma(\mathtt{C}_k, \top, \bot) \quad b_{ms} \Rightarrow m \quad e \Rightarrow \mu = \epsilon \quad h, k \in \mathbb{N} \quad \mathsf{dom}(\mu) \subseteq \underline{w}(\mathsf{Arrld})}{\psi, \sigma \vdash \mathsf{twf}_{w,\mathtt{s}}((\langle \mathtt{C}, \rho, \mathtt{k}_\mathtt{s} \rangle : \langle \mathtt{C}_1, \rho_1, \mathtt{k}_\mathtt{s} \rangle \ldots \langle \mathtt{C}_k, \rho_k, \mathtt{k}_\mathtt{s} \rangle, (\mu, w \diamond \tau'), b_{ms}))} \\ \frac{\psi, \sigma \vdash \mathsf{twf}_{w,\mathtt{s}}(C)}{\psi, \sigma \vdash \mathsf{twf}_{w,\mathtt{s}}(C)} \quad \overline{\psi, \sigma \vdash \mathsf{twf}_{w,\mathtt{s}}(\epsilon)} \quad \overline{\psi, \sigma \vdash \mathsf{twf}_{w,\mathtt{s}}(\mathsf{err})} \quad \overline{\psi, \sigma \vdash \mathsf{twf}_{w,\mathtt{s}}(\mathsf{unsafe})}$$

Figure 22: Well-formedness relation for the system transformation  $\psi$  with respect to a system  $\sigma = (\tau, \gamma, \xi)$ .

In addition, in order to facilitate the proof of the forthcoming results, we introduce the relations of well-formedness for the transformations  $\eta, \psi$ , and  $\theta$ . These relations are aimed to express invariant properties that are valid when the execution of a system call begins and that are preserved during kernel space evaluation. These predicates are in Figures 21 to 23.

For the well-formedness relation for  $\psi$ , we use the auxiliary predicate  $\Sigma(C, m, e)$ , which holds whenever  $C = \psi(C', m, e); \psi(D_1, \top, \bot); \ldots; \psi(D_h, \top, \bot)$ . In addition, in the rule for configurations, we overload; to denote the lexical concatenation of two programs C, D, and we take  $\epsilon$  as null element for this operation. Finally, notice that as h and k can be any natural number, we are also considering the case where  $D_0, \ldots, D_k$  is an empty sequences when they are 0.

In order to show that  $\theta$  imposes speculative kernel safety, we adopt the same approach we used for  $\psi$ : we introduce an invariant  $\theta, \sigma \vdash \mathsf{twf}_{w,s}(\cdot)$  on configurations, and we show that it is (weakly) preserved during the execution,

Notice that this relation imposes that the mis-speculation flag is  $\bot$  and that the write buffer is empty.

**Lemma 8.** The transformation  $\eta$  imposes speculative kernel safety.

$$\frac{\forall \mathbf{f} \in \mathsf{Funld}_{\mathtt{k}}. \exists \mathtt{C}.\tau'(\mathtt{f}) = \theta(\mathtt{C}) \quad \tau =_{\mathsf{Funld}} \tau'}{\theta, \sigma \vdash \mathsf{twf}_{w,\mathtt{s}}(\tau')} \quad \frac{\theta, \sigma \vdash \mathsf{twf}_{w,\mathtt{s}}(F) \quad \exists \mathtt{C}'.\mathtt{C} = \theta(\mathtt{C}')}{\theta, \sigma \vdash \mathsf{twf}_{w,\mathtt{s}}(\langle \mathtt{C}, \rho, \mathtt{k}_\mathtt{s} \rangle : F)}$$

$$\overline{\theta, \sigma \vdash \mathsf{twf}_{w,\mathtt{s}}(\mathsf{err})} \quad \overline{\theta, \sigma \vdash \mathsf{twf}_{w,\mathtt{s}}(\mathsf{unsafe})}$$

$$\frac{\theta, \sigma \vdash \mathsf{twf}_{w,\mathtt{s}}(\tau') \quad \theta, \sigma \vdash \mathsf{twf}_{w,\mathtt{s}}(F)}{\theta, \sigma \vdash \mathsf{twf}_{w,\mathtt{s}}((F, (\epsilon, w \diamond \tau'), \bot))} \quad \frac{\theta, \sigma \vdash \mathsf{twf}_{w,\mathtt{s}}(C)}{\theta, \sigma \vdash \mathsf{twf}_{w,\mathtt{s}}(C : S)} \quad \overline{\theta, \sigma \vdash \mathsf{twf}_{w,\mathtt{s}}(\epsilon)}$$

Figure 23: Well-formedness relation for the system transformation  $\theta$  with respect to a system  $\sigma = (\tau, \gamma, \xi)$ .

*Proof.* We assume that there is a system  $\sigma = (\tau, \gamma, \varphi)$ , a system call s a register map  $\rho$  buffer  $\mu$ , an array store  $\tau' =_{\mathsf{FunId}} \tau$ , and a natural number n such that

$$w \vdash_{\sigma} (\langle \gamma(\mathtt{s}), \rho, \mathtt{k}_{\mathtt{s}} \rangle, (\mu, (w \diamond \tau')), b_{ms}) \xrightarrow{O} {}^{n} \mathsf{unsafe}.$$

for a sequence of directives D (that we assume free of bt because of Lemma 10) producing a set of observations O. By case analysis on n, we refuse the case where n = 0, because that would mean

$$(\langle \gamma(\mathbf{s}), \rho, \mathbf{k}_{\mathbf{s}} \rangle, (\mu, (w \diamond \tau')), b_{ms}) = \mathsf{unsafe}.$$

For this reason, in the following, we assume that n > 0. By case analysis on the proof relation, we deduce that the rule that has been used to show the last transition must be one among [SLOAD-UNSAFE], [SSTORE-UNSAFE], [SCALL-UNSAFE] and [SCALL-STEP-UNSAFE].

We start by showing the case for [SLOAD-UNSAFE], for [SSTORE-UNSAFE] the proof is analogous. This means that

$$\begin{split} w \vdash_{\sigma} (\langle \gamma(\mathbf{s}), \rho, \mathbf{k}_{\mathbf{s}} \rangle, (\mu, (w \diamond \tau')), b_{ms}) &\xrightarrow{O'} {}^{n-1} \\ (\langle x := *\mathbf{E}; \mathbf{D}, \rho', \mathbf{k}_{\mathbf{s}} \rangle : F, (\mu', (w \diamond \tau'')), b'_{ms}) : S \xrightarrow{\mathsf{mem}\, p} \mathsf{unsafe}. \end{split}$$

Observe that n > 1: otherwise, if n = 1, we would have  $x := *E; D = \gamma(s)$ , but from  $\sigma \in \text{im}(\eta)$ , we deduce that there is a system  $\sigma' = (\tau'', \gamma', \xi')$  such that  $\eta(\sigma') = \sigma$ . This, in particular, would mean that  $\gamma(s) = \eta(\gamma'(s))$ , so there is a command  $C \in \text{Cmd}$  such that  $\gamma(s) = \eta(\gamma'(C))$  but, by induction on the syntax of the command, we observe that this is not possible. However, we can apply Lemma 13, so to deduce that there must be a stack of configurations

$$(\langle \mathtt{C}, \rho'', \mathtt{k_s} \rangle : F'', (\mu'', (w \diamond \tau''')), b'''_{ms}) : S'$$

such that:

$$\begin{split} w \vdash_{\sigma} (\langle \gamma(\mathtt{s}), \rho, \mathtt{k}_{\mathtt{s}} \rangle, (\mu, (w \diamond \tau')), b_{ms}) &\xrightarrow{O''} \\ \\ (\langle \mathtt{C}, \rho'', \mathtt{k}_{\mathtt{s}} \rangle : F'', (\mu'', (w \diamond \tau''')), b''_{ms}) : S' \xrightarrow{\sigma} \\ \\ (\langle x := *\mathtt{E}; \mathtt{D}, \rho', \mathtt{k}_{\mathtt{s}} \rangle : F, (\mu', (w \diamond \tau'')), b'_{ms}) : S \xrightarrow{\mathsf{mem}\, p} \mathsf{vunsafe}. \end{split}$$

and, in particular,  $\eta, \sigma \vdash \mathsf{twf}_{w,s}(\langle \mathtt{C}, \rho'', \mathtt{k}_s \rangle : F'')$ . From this observation, we deduce that  $\mathtt{C} = \eta(\mathtt{D})$  for some  $\mathtt{D} \in \mathtt{Cmd}$  and that  $\eta, \sigma \vdash \mathsf{twf}_{w,s}(F'')$  holds. For these reasons, going by cases on  $\mathtt{D}$  and the rule

that has been used to show the transition with the directive d (knowing that  $d \neq bt$  by assumption), we can rewrite the reduction above, that rewrites as follows:

$$\begin{split} w \vdash_{\sigma} (\langle \gamma(\mathtt{s}), \rho, \mathtt{k}_{\mathtt{s}} \rangle, (\mu, (w \diamond \tau')), b_{ms}) &\xrightarrow{D''} \flat^{n-2} \\ (\langle \mathtt{fence}; x := \mathtt{*E}; \mathtt{D}, \rho'', \mathtt{k}_{\mathtt{s}} \rangle : F'', (\mu'', (w \diamond \tau''')), b''_{ms}) : S \xrightarrow[d]{o} \flat \\ (\langle x := \mathtt{*E}; \mathtt{D}, \rho', \mathtt{k}_{\mathtt{s}} \rangle : F, \overline{(\mu'', (w \diamond \tau'''))}, \bot) : S \xrightarrow[\mathtt{st}]{\mathtt{mem} \, p} \flat \, \mathsf{unsafe}. \end{split}$$

In particular, none of the rules except for [Fence] can have been used. This also means that  $b'_{ms} = b''_{ms} = \bot$ . From Lemma 9, we deduce that there is n' such that:

$$\begin{split} w \vdash_{\sigma} (\langle \gamma(\mathbf{s}), \rho, \mathbf{k}_{\mathbf{s}} \rangle, (\mu, (w \diamond \tau')), b_{ms}) &\xrightarrow{O'''} \xrightarrow{\mathsf{st}^{n'}} \\ (\langle x := *\mathbf{E}; \mathbf{D}, \rho', \mathbf{k}_{\mathbf{s}} \rangle : F, \overline{(\mu'', (w \diamond \tau'''))}, \bot) : S \xrightarrow{\mathsf{mem} \, p} \mathsf{unsafe}. \end{split}$$

Finally, we apply Lemma 14 that shows:

$$w \vdash_{\sigma} (\langle \gamma(s), \rho, k_s \rangle, \overline{(\mu, (w \diamond \tau'))}) \rightarrow^{n'} (\langle x := *E; D, \rho', k_s \rangle : F, \overline{(\mu'', (w \diamond \tau'''))}).$$

Finally, by assumption we know that the rule [SLOAD-UNSAFE] has been applied to show the transition in the speculative semantics. By introspection of that rule, we conclude that its premises are also verified by the configuration

$$(\langle x := *\mathbf{E}; \mathbf{D}, \rho', \mathbf{k_s} \rangle : F, \overline{(\mu'', (w \diamond \tau'''))});$$

this shows that [LOAD-UNSAFE] applies, i.e.

$$w \vdash_{\sigma} (\langle x := *\mathtt{E}; \mathtt{D}, \rho', \mathtt{k_s} \rangle : F, \overline{(\mu'', (w \diamond \tau'''))}) \to^{n'} \mathsf{unsafe},$$

and this shows the claim.

Now, we show the case where the applied rule is [SCALL-UNSAFE]. In this case, by going similarly to what we did for the previous case, we observe that

$$w \vdash_{\sigma} (\langle \gamma(\mathtt{s}), \rho, \mathtt{k}_{\mathtt{s}} \rangle, (\mu, (w \diamond \tau')), b_{ms}) \xrightarrow{O'} {}^{n-1} \\ (\langle \mathtt{call} \ \mathtt{E}(\mathtt{E}_1, \dots, \mathtt{E}_n); \mathtt{C}', \rho', \mathtt{k}_{\mathtt{s}} \rangle : F, (\mu', (w \diamond \tau'')), b'_{ms}) : S \xrightarrow{\mathtt{jmp} \, p} \mathtt{unsafe}.$$

As we did before we observe that we must have that n > 1. By an application of Lemma 13, we deduce that there must be a stack of configurations

$$(\langle \mathtt{C}, \rho'', \mathtt{k_s} \rangle : F'', (\mu'', (w \diamond \tau''')), b''_{ms}) : S'$$

such that:

$$\begin{split} w \vdash_{\sigma} (\langle \gamma(\mathtt{s}), \rho, \mathtt{k}_{\mathtt{s}} \rangle, (\mu, (w \diamond \tau')), b_{ms}) &\xrightarrow{O''} {}^{\flat} {}^{n-2} \\ & (\langle \mathtt{C}, \rho'', \mathtt{k}_{\mathtt{s}} \rangle : F'', (\mu'', (w \diamond \tau''')), b''_{ms}) : S' \xrightarrow{o}_{d} {}^{\flat} \\ & (\langle \mathtt{call} \ \mathtt{E}(\mathtt{E}_{1}, \dots, \mathtt{E}_{n}); \mathtt{C}', \rho', \mathtt{k}_{\mathtt{s}} \rangle : F, (\mu', (w \diamond \tau'')), b'_{ms}) : S \xrightarrow{\mathsf{jmp} \, p}_{\mathsf{run} \, p} \mathsf{b} \ \mathsf{unsafe}. \end{split}$$

and, in particular,  $\eta, \sigma \vdash \mathsf{twf}_{w,s}(\langle \mathtt{C}, \rho'', \mathtt{k}_s \rangle : F'')$ . From this observation, we deduce that  $\mathtt{C} = \eta(\mathtt{D})$  for some  $\mathtt{D} \in \mathsf{Cmd}$  and that  $\eta, \sigma \vdash \mathsf{twf}_{w,s}(F'')$  holds. However, this conclusion is absurd because, by cases on the syntax of  $\mathtt{D}$  we can easily observe that none of the images of a command may reduce in one step to call  $\mathtt{E}(\mathtt{E}_1, \ldots, \mathtt{E}_n)$ .

Finally, we show the case where the rule is [SCALL-STEP-UNSAFE]. <sup>2</sup>In this case, the command that caused the unsafe memory access can either be a call  $\cdot$  ( $\cdot$ ) or a scall  $\cdot$  ( $\cdot$ ), reasoning similar to the previous case, we can exclude call  $\cdot$  ( $\cdot$ ), so we have:

$$w \vdash_{\sigma} (\langle \gamma(\mathtt{s}), \rho, \mathtt{k}_{\mathtt{s}} \rangle, (\mu, (w \diamond \tau')), b_{ms}) \xrightarrow{O'} {}^{n-1}$$
 
$$(\langle \mathtt{scall} \ \mathtt{E}(\mathtt{E}_1, \dots, \mathtt{E}_n); \mathtt{C}', \rho', \mathtt{k}_{\mathtt{s}} \rangle : F, (\mu', (w \diamond \tau'')), b'_{ms}) : S \xrightarrow{\mathtt{jmp} \ p} \mathtt{unsafe}.$$

Observe that n>1: otherwise, if n=1, we would have scall  $E(E_1,\ldots,E_n)=\gamma(s)$ , but from  $\sigma\in \text{im}(\eta)$ , we deduce that there is a system  $\sigma'=(\tau'',\gamma',\xi')$  such that  $\eta(\sigma')=\sigma$ . This, in particular, would mean that  $\gamma(s)=\eta(\gamma'(s))$ , so there is a command  $C\in \text{Cmd}$  such that  $\gamma(s)=\eta(\gamma'(C))$  but, by cases on the syntax of the command, we observe that this is not possible. Then, we can apply Lemma 13, so to deduce that there is a stack of configurations

$$(\langle \mathtt{C}, \rho'', \mathtt{k_s} \rangle : F'', (\mu'', (w \diamond \tau''')), b'''_{ms}) : S'$$

such that:

and, in particular,  $\eta, \sigma \vdash \mathsf{twf}_{w,s}(\langle \mathsf{C}, \rho'', \mathsf{k}_s \rangle : F'')$ . From this observation, we deduce that  $\mathsf{C} = \eta(\mathsf{D})$  for some  $\mathsf{D} \in \mathsf{Cmd}$  and that  $\eta, \sigma \vdash \mathsf{twf}_{w,s}(F'')$  holds. Reasoning in a similar way of what we did for the [SLOAD-UNSAFE] rule, we can rewrite the reduction above as follows:

$$\begin{split} w \vdash_{\sigma} (\langle \gamma(\mathtt{s}), \rho, \mathtt{k}_{\mathtt{s}} \rangle, (\mu, (w \diamond \tau')), b_{ms}) & \xrightarrow{O''}_{D''} \flat^{-n-2} \\ & (\langle \mathtt{fence}; \mathtt{scall} \ \mathtt{E}(\mathtt{E}_1, \dots, \mathtt{E}_n); \mathtt{C}', \rho'', \mathtt{k}_{\mathtt{s}} \rangle : F'', (\mu'', (w \diamond \tau''')), b''_{ms}) : S \xrightarrow[d]{o} \flat \\ & (\langle \mathtt{scall} \ \mathtt{E}(\mathtt{E}_1, \dots, \mathtt{E}_n); \mathtt{C}', \rho', \mathtt{k}_{\mathtt{s}} \rangle : F, \overline{(\mu'', (w \diamond \tau'''))}, \bot) : S \xrightarrow[\mathtt{st}]{\mathsf{mem} \ p} \flat \mathsf{unsafe}. \end{split}$$

In particular, none of the rules except for [Fence] can have been used. This also means that  $b'_{ms} = b''_{ms} = \bot$ . From Lemma 9, we deduce that there is n' such that:

$$\begin{split} w \vdash_{\sigma} (\langle \gamma(\mathbf{s}), \rho, \mathbf{k}_{\mathbf{s}} \rangle, (\mu, (w \diamond \tau')), b_{ms}) & \xrightarrow{O'''} \xrightarrow{\mathbf{st}^{n'}} \xrightarrow{n'} \\ & (\langle \mathbf{scall} \ \mathbf{E}(\mathbf{E}_{1}, \dots, \mathbf{E}_{n}); \mathbf{C}', \rho', \mathbf{k}_{\mathbf{s}} \rangle : F, \overline{(\mu'', (w \diamond \tau'''))}, \bot) : S \xrightarrow{\mathsf{mem} \, p} \mathsf{unsafe}. \end{split}$$

Finally, we apply Lemma 14 that shows:

$$w \vdash_{\sigma} (\langle \gamma(\mathtt{s}), \rho, \mathtt{k}_{\mathtt{s}} \rangle, \overline{(\mu, (w \diamond \tau'))}) \to^{n'} (\langle \mathtt{scall} \ \mathtt{E}(\mathtt{E}_1, \dots, \mathtt{E}_n); \mathtt{C}', \rho', \mathtt{k}_{\mathtt{s}} \rangle : F, \overline{(\mu'', (w \diamond \tau'''))}).$$

<sup>&</sup>lt;sup>2</sup>The proof of this case is similar to the one we showed for the [SLOAD-UNSAFE] rule. Nevertheless, we show the case explicitly to stress out that the same proof strategy used for the previous case—that is taken from [25]—scales smoothly to BTB speculation.

Finally, by assumption we know that the rule [SCall-Step-Unsafe] has been applied to show the transition in the speculative semantics. By introspection of that rule, we conclude that its premises are also verified by the configuration

$$(\langle \text{scall } E(E_1, \dots, E_n); C', \rho', k_s \rangle : F, \overline{(\mu'', (w \diamond \tau'''))});$$

this shows that [SCALL-UNSAFE] applies, i.e.

$$w \vdash_{\sigma} (\langle \text{scall } \mathsf{E}(\mathsf{E}_1, \dots, \mathsf{E}_n); \mathsf{C}', \rho', \mathsf{k}_{\mathsf{S}} \rangle : F, \overline{(\mu'', (w \diamond \tau'''))}) \to^{n'} \mathsf{unsafe},$$

and this shows the claim.

#### A.2.1 Technical Observations on the Speculative Semantics

We start by defining a predicate  $\sigma \vdash \mathsf{wf}_w(\cdot)$  that captures the well-formedness of a buffered memory with respect to a layout w. For a system  $\sigma = (\tau, \gamma, \xi)$ , and a layout w such predicate is defined as follows:

$$\overline{\sigma \vdash \mathsf{wf}_w((\mathsf{err}, \bot))} \quad \overline{\sigma \vdash \mathsf{wf}_w(\mathsf{unsafe})} \quad \overline{\sigma \vdash \mathsf{wf}_w(\epsilon)}$$

$$(C) \quad \sigma \vdash \mathsf{wf}_*(S) \quad \mathsf{dom}(u) \subseteq w(\mathsf{ArrId}) \quad m = w \diamond \tau' \quad \tau' = w \diamond \tau' \quad$$

$$\frac{\sigma \vdash \mathsf{wf}_w(C) \quad \sigma \vdash \mathsf{wf}_w(S)}{\sigma \vdash \mathsf{wf}_w(C:S)} \quad \frac{\mathsf{dom}(\mu) \subseteq w(\mathsf{ArrId}) \quad m = w \diamond \tau' \quad \tau' =_{\mathsf{FunId}} \tau}{\sigma \vdash \mathsf{wf}_w((F, (\mu, m), b_{ms}))}$$

**Remark 6.** For every system  $\sigma = (\tau, \gamma, \xi)$ , layout  $w \in \text{Layout}$ , pair of configurations S and S', sequences of directives D and of observations O, if  $\sigma \vdash \text{wf}_w(S)$  and  $w \vdash_{\sigma} S \xrightarrow{O} * S'$ , then we have  $\sigma \vdash \text{wf}_w(S')$ .

*Proof.* The proof goes by induction on the number of steps.

- Case 0. Trivial.
- CASE n+1. The claim always comes from the IH, and in most of the cases the proof is straightforward. The most interesting cases are those for stores and fences. In the first case, the premise of the rule insures that the address where we write is part of the memory of an array. In the second case, it is a consequence of Remark 4.

Remark 7. For every system  $\sigma = (\tau, \gamma, \xi)$ , layout  $w \in \text{Layout}$ , pair of configurations  $(F, w \diamond \tau, D, O)$  and (F', m, D', O') such that  $w \vdash_{\sigma} (F, w \diamond \tau, D, O) \rightarrow^* (F', m, D', O')$ , we have that  $m = w \diamond \tau'$  for some  $\tau' =_{\text{Funld }} \tau$ .

*Proof.* To show this result we need to show, in conjunction with it, that if

$$w \vdash_{\sigma} (F, w \diamond \tau, D, O) \Longrightarrow^* (S, F, D', O')$$

then  $\sigma \vdash \mathsf{wf}_w(S)$  The proof goes by induction on the number of steps.

- Case 0. Trivial.
- CASE n+1. The proof goes by cases on the rule that has been applied to show the last transition. The cases for the ordinary rules are simple. For this reason we just focus on those for the hybrid configuration.
  - CASE [Spec-Init]. In this case, we can verify that the speculative frame of the target configuration enjoys  $\sigma \vdash \mathsf{wf}_w(\cdot)$  simply by introspection of the target configuration and because of the IH.
  - Case [Spec-D], [Spec-S], [Spec-BT]. The claim is a consequence of the IH and of Remark 6.

- Case [Spec-Term]. The claim is a consequence of the IH, and of Remark 4 which ensures that the memory that is extracted form the speculative stack enjoys the desired property.

**Remark 8.** For every system  $\sigma = (\tau, \gamma, \xi)$ , every store  $\tau =_{\mathsf{Funld}} \tau'$ , program configuration  $C = (\langle \mathtt{C}, \rho, \mathtt{u} \rangle, w \diamond \tau', O, D)$ , configuration  $(F', w \diamond \tau'', O, D)$ , and hybrid configuration (S, (F', O, D)), that are reachable in n step from C, there are a pair of stacks  $F_{\mathtt{k}}$ ,  $F_{\mathtt{u}}$  such that  $F' = F_{\mathtt{k}} : F_{\mathtt{u}}$ ,  $\mathtt{k}(F_{\mathtt{k}})$  and  $\mathtt{u}(F_{\mathtt{u}})$ .

*Proof.* By induction on n. The base case is trivial, the inductive one comes by cases on the rule that is applied. Most of these rules are relatively simple. We just show the most interesting cases:

- Case [APOP]. The target configuration carries a stack which is a suffix of the one in the source configuration, so the conclusion is a trivial consequence of the IH.
- CASE [ASC]. We first apply the IH, which shows that:

$$w \vdash_{\sigma} (\langle \mathtt{C}, \rho, \mathtt{u} \rangle : F, w \diamond \tau, O, D) \Rightarrow^{n} (\langle \mathtt{syscall} \ \mathtt{s}(\mathtt{E}_{1}, \dots, \mathtt{E}_{k}); \mathtt{D}, \rho', b \rangle : F', w \diamond \tau, O, D)$$

and that  $\langle \text{syscall } \text{s}(\text{E}_1, \dots, \text{E}_k), \rho', b \rangle : F'$  is a concatenation of a pair of stacks  $F_k, F_u$  such that  $k(F_k)$  and  $u(F_u)$ . and since the current command is a system call,  $F_k$  must be empty and b must be u. Observe that

$$w \vdash_{\sigma} (\langle \text{syscall } \mathbf{s}(\mathbf{E}_1, \dots, \mathbf{E}_k); \mathbf{D}, \rho', \mathbf{u} \rangle : F', w \diamond \tau, O, D) \rightarrow (\langle \gamma(\mathbf{s}), \rho'', \mathbf{k}_{\mathbf{s}} \rangle : \langle \mathbf{D}, \rho', \mathbf{u} \rangle : F', w \diamond \tau, O, D)$$

for a suitable  $\rho''$ . The claim just requires verifying that the target configuration above satisfies the premises. In particular that  $k(\langle \gamma(s), \rho'', k_s \rangle)$  holds because  $\gamma(s)$  cannot contain system call invocations for the definition of system.

- Case [ACall]. This case is analogous to the previous one, but instead of setting the flag to k in the new frame, the rule copies it from the topmost frame of the source configuration, and this does not break the invariant. In particular, we observe that the target configuration of the rule looks like the following one:

$$\langle \langle w \diamond \tau''(p), \rho_0 [x_1 \leftarrow \llbracket F_1 \rrbracket_{\rho, w}, \dots, x_n \leftarrow \llbracket F_n \rrbracket_{\rho, w}], b \rangle : \langle C, \rho, b \rangle : F, w \diamond \tau'' \rangle.$$

From Remark 6, we deduce that  $\tau'' =_{\mathsf{Funld}} \tau$ , this means that  $w \diamond \tau''(p)$ , because from the premises of the rule, we know that  $p \in \underline{w}(\mathsf{Funld}_b)$ , which means that there is a function  $\mathbf{f} \in \mathsf{Funld}_b$  such that  $w(\mathbf{f}) = p$ . From this observation and the definition of  $\cdot \diamond \cdot$ , we deduce that  $w \diamond \tau''(p) = \tau(\mathbf{f})$ . So, we go by cases on b. If it is  $\mathbf{u}$ , the body of the function is unprivileged, and this shows  $\mathbf{u}(w \diamond \tau''(p))$ , analogously, if it is  $\mathbf{k}$ , we observe that, this program is in Cmd and does not have any syscall  $\cdot (\cdot)$  instruction inside. This shows  $\mathbf{k}_{\mathbf{s}}(w \diamond \tau''(p))$ , as required.

- Case [Spec-Init]. Observe that this rule simply copies the stack from the source configuration to the target configuration, and removes the executed command from the topmost frame; the conclusion is a consequence of the IH.

**Remark 9.** For every system  $\sigma = (\overline{\tau}, \gamma, \xi)$ , store  $\tau =_{\mathsf{Funld}} \overline{\tau}$ , configuration  $(\langle \mathtt{A}, \rho, \mathtt{u} \rangle, w \diamond \tau, D, O)$ ,  $n \in \mathbb{N}$ , and every configuration  $(F, w \diamond \tau, D, O)$  if

$$w \vdash_{\sigma} (\langle A, \rho, u \rangle, w \diamond \tau, D, O) \Rightarrow^{n} (\langle \text{spec on C}; A', \rho', b_{ms} \rangle : F, w \diamond \tau', D', O'),$$

then  $u(\langle \operatorname{spec} \operatorname{on} \operatorname{C}; \operatorname{A}', \rho', b_{ms} \rangle : F)$ .

*Proof.* Consequence of Remark 8: in order for  $k(\langle \text{spec on } C; A', \rho', b_{ms} \rangle : F)$  to hold, it must be the case that  $\text{spec on } C; A' \in Cmd$ , but this is absurd.

**Remark 10.** For every system  $\sigma = (\overline{\tau}, \gamma, \xi)$ , store  $\tau =_{\mathsf{Funld}} \overline{\tau}$ , configuration  $(\langle \mathtt{A}, \rho, \mathtt{u} \rangle, w \diamond \tau, D, O)$ ,  $n \in \mathbb{N}$ , and every configuration  $(F, w \diamond \tau, D, O)$  if

$$w \vdash_{\sigma} (\langle A, \rho, \mathbf{u} \rangle, w \diamond \tau, D, O) \rightarrow^{n} S \mid (F', D', O'),$$

then u(F').

*Proof.* We go by induction on n. The base case is trivial, and for the inductive one, the claim is always trivial, or a direct consequence of the IH except for when the rule is [SPEC-INIT]. In that case, the claim is a consequence of Remark 9.

**Remark 11.** For every pair of speculative stacks S, S', system  $\sigma = (\tau, \gamma, \xi)$ ,  $n \in \mathbb{N}$ , sequence of directives D, address p and layout w such that  $p \notin \underline{w}(\mathsf{Id})$ , if:

$$w \vdash_{\sigma} S \xrightarrow{O} {}^{n} S',$$

then O does not contain the observation mem p.

*Proof.* The proof goes by induction on n.

- Case 0. Trivial
- CASE n+1. We apply the IH and we go by cases on the rule that has been used to show the last transition. For those rules that do not produce a transition like  $\operatorname{mem} p'$ , the claim is trivial. The rules that can produce a similar transition are [SLOAD-STEP], [SLOAD], [SLOAD-UNSAFE], [SSTORE], [SSTORE-UNSAFE], [SCALL], [SCALL-UNSAFE]. All these rules require in their premises that  $p' \in \underline{w}(\operatorname{Arrld}_k), \ p' \in \underline{w}(\operatorname{Funld}_k), \ p' \in \underline{w}(\operatorname{Arrld}_u), \ or that \ p' \in \underline{w}(\operatorname{Funld}_u), \ but \ since \ p \notin \underline{w}(\operatorname{Id}), \ we deduce that \ p \ does not belong to any of these sets, so it must be that <math>p' \neq p$ .

In order to show that the stack and memory are well-formed also during speculative execution, we define the predicate  $\sigma \vdash \mathsf{swf}_w(\cdot)$  that is defined as follows:

$$\frac{\sigma \vdash \mathsf{wf}_w(C) \quad \sigma \vdash \mathsf{swf}_w(C) \quad \sigma \vdash \mathsf{swf}_w(S)}{\sigma \vdash \mathsf{swf}_w(C:S)} \quad \frac{F = F_{\mathtt{k}} : F_{\mathtt{u}} \quad \mathtt{k}_{\mathtt{s}}(F_{\mathtt{k}}) \quad \mathtt{u}(F_{\mathtt{u}})}{\sigma \vdash \mathsf{swf}_w((F,(\mu,m),b_{ms}))}$$

**Remark 12.** For every system  $\sigma = (\tau, \gamma, \xi)$ , every stack S such that  $\sigma \vdash \mathsf{swf}_w(S)$ , stack S', such that  $w \vdash_{\sigma} S \xrightarrow[]{O} {}^{n}S'$ , we have  $\sigma \vdash \mathsf{swf}_w(S')$ .

*Proof.* By induction on n. The base case is trivial, the inductive one comes by cases on the rule that is applied. Thanks to Remark 6, we can just focus in showing the additional requirements on the stack. Most of these rules are relatively simple. We just most interesting cases:

- Case [SPOP]. The target configuration carries a frame-stack which is a suffix of the one in the source configuration, so the conclusion is a trivial consequence of the IH.
- Case [ACall]. We apply the IH and we rewrite the configuration reached after n steps as follows:

$$(\langle \mathtt{call} \ \mathtt{E}(\mathtt{E}_1, \dots, \mathtt{E}_k); \mathtt{D}, \rho, b \rangle : F, (\mu', w \diamond \tau), b_{ms}) : S''$$

we know by IH that it enjoys  $\sigma \vdash \mathsf{swf}_w(\cdot)$ , and we observe that the target configuration looks like the following one:

$$(\langle \mathtt{call} \ \mathtt{E}(\mathtt{E}_1,\ldots,\mathtt{E}_k);\mathtt{D},\rho',b\rangle:\langle \mathtt{D},\rho,\mathtt{k}\rangle:F,(\mu',w\diamond\tau),b_{ms}):S''$$

for some  $\rho_0$ . From the premises of the rule, we know that  $p \in \underline{w}(\mathsf{Funld}_b)$ , which means that there is a function  $\mathbf{f} \in \mathsf{Funld}_b$  such that  $w(\mathbf{f}) = p$ . From this observation and the definition of  $\cdot \diamond \cdot$ ,

we deduce that  $w \diamond \tau''(p) = \tau(f)$ . So, we go by cases on b. If it is u, the body of the function is unprivileged, and this shows  $w \diamond \tau''(p)$ , analogously, if it is k, we observe that, this program is in Cmd and does not have any syscall  $\cdot(\cdot)$  instruction inside. This shows  $k_s(\tau(f))$ , as required

- CASE [ASC]. We first apply the IH, which shows that the configuration reached after n steps enjoys  $\sigma \vdash \mathsf{swf}_w(\cdot)$ , namely:

$$\sigma \vdash \mathsf{swf}_w((\langle \mathsf{syscall} \ \mathsf{s}(\mathsf{E}_1, \dots, \mathsf{E}_k); \mathsf{D}, \rho, \mathsf{u} \rangle : F, (\mu', w \diamond \tau), b_{ms}) : S'')$$

form this observation, we deduce that

$$\mathbf{u}(\langle \mathtt{syscall} \ \mathtt{s}(\mathtt{E}_1,\ldots,\mathtt{E}_k);\mathtt{D},\rho,\mathbf{u}\rangle:F)$$

must hold. The claim is comes from introspection of the rule, and from the definition of  $\gamma$ .

**Remark 13.** For every n, if we define  $\sqsubseteq$  as the smallest partial order such that  $\bot \sqsubseteq \top$ , we have that for every speculative configuration C and speculative stack S, if:

$$w \vdash_{\sigma} (F, (\mu, m), b_{ms}) : S \xrightarrow{O} (F', (\mu', m'), b'_{ms}) : S',$$

and D does not contain bt directives, then  $b_{ms} \sqsubseteq b'_{ms}$ .

*Proof.* The proof goes by induction on n. The base case is trivial, as  $\sqsubseteq$  is reflexive. The inductive claim follows from an application of the inductive hypothesis and by introspection of the rules.  $\Box$ 

**Remark 14.** For every system  $\sigma = (\tau, \gamma, \xi)$ , natural number n, layout w, and every configuration  $(\langle A, \rho, u \rangle, w \diamond \tau, \epsilon, \epsilon)$  where A is unprivileged, and stack of directives D and stack of observations O, if

$$w \vdash_{\sigma} (\langle \mathtt{A}, \rho, \mathtt{u} \rangle, w \diamond \tau, \epsilon, \epsilon) \mathop{{\Rightarrow}^{n}} S \, | \, (F, D, O)$$

then there is  $n' \leq n$  and a configuration  $(\langle C', \rho', u \rangle, w \diamond \tau', \bot)$  with u(C'), a sequence of directives D' and a sequence of observations O' such that

$$w \vdash_{\sigma} (\langle \mathsf{C}', \rho', \mathsf{u} \rangle, w \diamond \tau', \bot) \xrightarrow{O'} {n'} S.$$

*Proof.* The proof goes by induction on n.

- Case 0. Follows from vacuity of the premise.
- Case n+1. The premise is:

$$w \vdash_{\sigma} (\langle A, \rho, u \rangle, w \diamond \tau, \epsilon, \epsilon) \rightarrow^{n} D \rightarrow (S, F, D, O)$$

We go by cases on the rule that has been applied to show the last transition.

- Case [Spec-Init]. We observe that

$$D = (\langle \mathtt{spec} \ \mathtt{on} \ \mathtt{C}''; \mathtt{D}, \rho'', b \rangle : F, w \diamond \tau'')$$

and the claim follows by introspection of the rule. u(C'') and b = u follow from Remark 8: in particular, it cannot be that  $b = k_s$  because, otherwise, we could not have spec on C'; D.

- Case [Spec-S]. The IH provides

$$w \vdash_{\sigma} (\langle \mathsf{C}', \rho', \mathsf{u} \rangle, w \diamond \tau', \bot) \xrightarrow{O'} {n'} S.$$

from the premise of the rule, we conclude

$$w \vdash_{\sigma} S \xrightarrow[st]{o} S'$$

and this concludes the proof.

The other cases are analogous.

Remark 15. If

$$w \vdash_{\sigma} C \xrightarrow{O} {}^{n} S$$

and S = D : S' and S' not empty, then there are  $n' \leq n$ , D', O' such that:

$$w \vdash_{\sigma} C \xrightarrow{O'} {}^{n'} S'$$

*Proof.* By induction on S and then on n.

- Case  $\epsilon$ . Absurd.
- CASE D: S'. The IH tells that for every m, there are  $m' \leq m$ , D', O' such that:

$$w \vdash_{\sigma} C \xrightarrow{O'} {}^{m'} S'$$
 (IHS)

we go by induction on n.

- Case 0. From the premise we can deduce  $\epsilon = S'$ , which is absurd.
- Case n+1. We go by cases on the directive used for the last step:
  - Case st. The claim is a consequence of the IH on n.
  - Case Id i, br b. The witness we need to introduce is n step transition  $w \vdash_{\sigma} C \xrightarrow{O} {}^{n} D : S'$ .
  - CASE bt. We go by cases on the rule that has been applied. It must be one of [B<sub>T</sub>], [B<sub>T</sub>]. We will just show the result for the case of ordinary configurations. The proof with error configurations is analogous.
    - Case  $[B\tau_{\top}]$ . In this case, we go by cases on the n+1-th target stack. If it is empty or it has one element, the claim holds for vacuity of the premise, if it has more than one element, the claim is a consequence of two application of the IH.
    - Case [BT $_{\perp}$ ]. The claim holds for vacuity of the premise: the n+1-th target stack has just one element.

**Remark 16.** For every n, initial configuration  $C = (\langle C, \rho, u \rangle, m, \bot)$ , if:

$$w \vdash_{\sigma} C \xrightarrow{O} {}^{n} D,$$

and D has a mis-speculation flag (D  $\neq$  unsafe) then the mis-speculation flag of D must be  $\perp$ .

*Proof.* The proof is by induction, and the base case is trivial. In the inductive case, we go by cases on the directive of the last transition:

- Case st. By introspection of this fragment of the semantics, we deduce that the number of stacks in the n-th and in the n+1-th target configurations is the same, so in particular the n-th target stack must have one entry only. For this reason we can apply the IH and deduce that the mis-speculation flag of the n-th target configuration  $\bot$ . Then we observe that for all these rules, the flag is always copied from the source configuration to the target one, so we conclude.
- Case  $\operatorname{Id} i$ ,  $\operatorname{br} b$ . For all the rules that match these directives, the claim hold for vacuity of the premise: the n+1-th target configuration stack has height greater than 1. The only exception is [SLOAD-ERROR]. In this case, the proof is analogous to the case of the same but with  $\operatorname{st}$  directive, which has already taken in account in the previous step.
- Case bt. We go by cases on the rule that has been applied. It must be one of [BT<sub>⊤</sub>], [BT<sub>⊥</sub>]. We will just show the result for the ordinary configurations, the case for errors is analogous.
  - Case [BT $_{\perp}$ ]. In this case, the claim follows directly from the definition of the rule.

- CASE  $[BT_{\top}]$ . Call  $S_n$  the n-th target configuration stack. Observe that its height cannot be neither 0 nor 1. In the first case there would be no next transition, in the other case, from the IH we would deduce that that its mis-speculation flag is unset, which is in contradiction with the assumption on the applied rule. Observe that if its height is greater than 2, then the height of the n+1-th target stack is greater than 1, so the claim holds for vacuity of the premise. Finally, if its height is exactly 2, then it must be in the shape  $D_n: D'_n$ . From Remark 15, we deduce that there is a sequence of transitions from C to  $D'_n$  whose length is  $n' \leq n$ . By applying the IH on this sequence, we can show that  $D'_n$  has mis-speculation flag unset. We conclude observing that the n+1-th target configuration stack is exactly  $D'_n$ .

**Lemma 9.** For every n, configuration C, if:

$$w \vdash_{\sigma} C \xrightarrow{O} {}^{n} D : S,$$

and the mis-speculation flag of D is  $\perp$ , then there are  $\overline{O}$  and  $n' \leq n$  such that :

$$w \vdash_{\sigma} C \xrightarrow{\overline{O}} {}^{n'} D$$

*Proof.* The proof is by induction on n. The base case is trivial. For the inductive case, we go by cases on the directive of the n + 1-th transition. The premise tells us that:

$$w \vdash_{\sigma} C \xrightarrow{O} ^{n} D : S \xrightarrow{o} D' : S'$$

and we know that the mis-speculation flag of D' is  $\bot$ . We must show that there are  $\overline{O}'$  and  $n'' \le n+1$  such that :

$$w \vdash_{\sigma} D \xrightarrow[\operatorname{\mathsf{st}^{n''}}]{\overline{O}} n'' D'$$

- Case st. Observe that the mis-speculation flag of D must be  $\bot$ , because no rules for the directive st changes it. For this reason we can apply the IH. This shows that:

$$w \vdash_{\sigma} C \xrightarrow{\overline{O}} n' D$$

We examine all the rules matching the transition

$$w \vdash_{\sigma} D : S \xrightarrow[d]{o} D' : S'$$

and we observe that the premises these rules do not depend on S, but they depend on D only, so if one of these rules is applied to show the transition above, it can be applied only on the transition of the claim. By introspection of all these rules, we observe that the configuration they produce is exactly D''.

- CASE Id i. There are two rules that match this directive and our premises, namely [SLOAD-ERROR] and [SLOAD]. We show just the case of the second one. The first one can be reduced for the same rule with the st directive. If this rule is applied, we can rewrite

$$w \vdash_{\sigma} D : S \xrightarrow[d]{o} D' : S'$$

as follows

$$\begin{split} w \vdash_{\sigma} (\langle x :=_{\ell} *\mathtt{E}; \mathtt{C}, \rho, b \rangle : F, (\mu, m), b_{ms}) : S \xrightarrow{\mathsf{mem} \, p} \\ (\langle \mathtt{C}, \rho[x \leftarrow v], b \rangle : F, (\mu, m), b_{ms} \vee b') : (\langle x :=_{\ell} *\mathtt{E}; \mathtt{C}, \rho, b \rangle : F, (\mu, m), b_{ms}) : S \end{split}$$

From the premises of the rule, we deduce:

$$\begin{split} &- \ [\![\mathbf{E}]\!]_{\rho,w}^{\mathsf{Addr}} = p \\ &- (\mu, m)^i(p) = v, b' \\ &- p \in \underline{w}(\mathsf{Arrld}_b) \\ &- b = \mathtt{k}_{\mathtt{s}} \Rightarrow p \in \underline{w}(\xi(\mathtt{s})) \end{split}$$

In particular, from the main premise of the claim, we deduce that b' cannot be  $\top$ . This means that  $(\mu, m)^i(p) = v, \bot$ . Our goal is to show

$$w \vdash_{\sigma} (\langle x :=_{\ell} *\mathtt{E}; \mathtt{C}, \rho, b \rangle : F, (\mu, m), b_{ms}) \xrightarrow{\mathsf{mem} \, p} (\langle \mathtt{C}, \rho[x \leftarrow v], b \rangle : F, (\mu, m), b_{ms} \vee b')$$

To do so, we can observe that, if we show that  $(\mu, m)^0(p) = v, \perp$ , then premises of the rule [SLOAD-STEP] are matched and thus the proof is concluded. This is a consequence of Remark 1.

- Case br b. Analogous to the case above.
- CASE bt. If the rule which has been applied is  $[BT_{\perp}]$ , the claim is a direct consequence of the IH. Otherwise, the rule applied is  $[BT_{\perp}]$ . We go by cases on the height of D:S. If it is 0, we reached a contradiction, if its height is 1, then the target configuration of this rule is  $\epsilon$ , and this is in contradiction with the assumption that it had a topmost configuration D with unset misspeculation flag. If the height is greater than 1, we deduce that D:S=D:D':S' and that the mis-speculation flag of D' is  $\perp$  by introspection of the rule and by the main assumption of this claim. We apply Remark 15 on the n-step reduction from C to D:D':S' in order to show that there is a  $n' \leq n$ -long reduction from C to D':S'. For the IH, there is a  $n'' \leq n'$  step reduction from C to D' which employs only the directive st. Which is our claim.

**Lemma 10.** For every system  $\sigma = (\tau, \gamma, \xi)$ , speculative configuration  $C = (\langle C, \rho, b \rangle, m, \bot)$  non-empty speculative stack S, sequence of directives D and sequence of observations O, if:

$$w \vdash_{\sigma} C \xrightarrow{O} {}^{n} S$$

then there are  $n' \leq n$ , D', O' such that

$$w \vdash_{\sigma} C \xrightarrow{O'} {}^{n'} S.$$

and D' does not contain any bt directive.

*Proof.* By induction on n.

- Case 0. Trivial.
- Case n+1. The premise tells

$$w \vdash_{\sigma} C \xrightarrow{O} {}^{n} S' \xrightarrow{o} S$$

The IH shows that there are  $n' \leq n$ , D' without bt directives and O' such that:

$$w \vdash_{\sigma} C \xrightarrow{O'} \stackrel{n'}{\triangleright} n' S'.$$

We are required to show that if

$$w \vdash_{\sigma} S' \xrightarrow[d]{o} S,$$
 (†)

then there is  $n'' \le n+1$ , D'' and O' such that

$$w \vdash_{\sigma} C \xrightarrow{O''} ^{n''} S'.$$

where in particular D'' does not contain bt directives. We go by cases on the directive d that is used in  $(\dagger)$ .

- CASE  $d \neq bt$ . In these cases, the claim holds if we take D'' = D' : d, O'' = O' : o and n'' = n' + 1.
- Case d = bt. The applied rule can either be  $[BT_{\perp}]$ , or  $[BT_{\perp}]$ . We just take in exam the case of ordinary configurations:
  - CASE [B<sub>T</sub>]. In this case, we conclude that S' = D : S'' by introspection of the rule, we deduce that the mis-speculation flag of the topmost configuration of S' is  $\top$ , so we apply Remark 16 to deduce that  $S'' \neq \epsilon$  (otherwise D could not have the mis-speculation flag set to  $\top$ ). Thanks to this observation, we can use Remark 15 to show that there is a sequence of transitions from S to S'' whose length is  $n' \leq n$ . The conclusion is a consequence of the application of the IH on this intermediate result. This concludes the sub-derivation.
  - CASE [BT<sub>⊥</sub>]. In this case, we use Lemma 9 to show that there is a sequence of transitions from C to C containing only the directive st, which is stronger than the claim we need.

**Lemma 11.** For every system  $\sigma(\tau, \gamma, \xi)$ , every  $\tau' =_{\mathsf{Funld}} \tau$  and register map  $\rho$ , if:

$$w \vdash_{\sigma} (\langle \gamma(\mathtt{s}), \rho, \mathtt{k}_{\mathtt{s}} \rangle, w \diamond \tau', O, D) \rightarrow^{n} (\langle \mathtt{C}, \rho', \mathtt{k}_{\mathtt{s}} \rangle : F, w \diamond \tau'', O', D')$$

then:

$$w \vdash_{\sigma} (\langle \gamma(\mathtt{s}), \rho, \mathtt{k}_{\mathtt{s}} \rangle, w \diamond \tau') \mathop{{\longrightarrow}^{n}} (\langle \mathtt{C}, \rho', \mathtt{k}_{\mathtt{s}} \rangle : F, w \diamond \tau'')$$

*Proof.* The proof goes by induction n n. The base case is trivial. The inductive case goes by cases on the last rule and follows by introspection of the rules' premises and IH. Notice that because of Remarks 8 and 10, rules [Poison], [Observe], [Observe-End], [Spec-Init] cannot be applied. The main premise of this lemma also allows us to exclude the remaining no-standard adversarial rules, i.e. all the remaining rules of Figure 18.  $\Box$ 

## A.2.2 Technical Observations on the $\eta$ Transformation

We say that two programs C, C' are in relation  $C \lesssim C'$  if and only if C' can be obtained by substituting instructions call  $E(F_1, \ldots, F_k)$  with scall  $E(F_1, \ldots, F_k)$  and by placing fence instruction inside C. We extend this relation to configurations by stipulating

$$\frac{\tau =_{\mathsf{Id_u} \cup \mathsf{ArrId_k}} \ \tau' \quad \forall \mathtt{f} \in \mathsf{FunId_k}. \tau'(\mathtt{f}) \precsim (\tau(\mathtt{f}))}{\tau \precsim \tau'} \quad \frac{\forall \mathtt{s} \in \mathsf{Sys}. \gamma(\mathtt{s}) \precsim \gamma(\mathtt{s})'}{\gamma \precsim \gamma'} \quad \frac{\tau \precsim \tau' \quad \gamma \precsim \gamma'}{(\tau, \gamma, \xi) \precsim (\tau', \gamma', \xi)} \\ \frac{F \precsim F' \quad \mathtt{C} \precsim \mathtt{C}'}{\langle \mathtt{C}, \rho, \mathtt{k_s} \rangle : F \precsim \langle \mathtt{C}', \rho, \mathtt{k_s} \rangle : F'} \quad \frac{F \precsim F'}{\langle \mathtt{C}, \rho, \mathtt{u} \rangle : F \precsim \langle \mathtt{C}, \rho, \mathtt{u} \rangle : F'} \\ \frac{T \precsim \tau' \quad F \precsim F'}{\mathsf{err} \precsim \mathsf{err}} \quad \frac{\tau \precsim \tau' \quad F \precsim F'}{\mathsf{unsafe} \precsim \mathsf{unsafe}} \quad \frac{\tau \precsim \tau' \quad F \precsim F'}{(F, w \diamond \tau) \precsim (F', w \diamond \tau)}$$

**Lemma 12.** For every pair of systems  $\sigma \lesssim \sigma'$ , configurations  $C = (\langle C, \rho, b \rangle : F, w \diamond \tau)$ , and D such that  $C \lesssim D$ , if

$$w \vdash_{\sigma} C \to C'$$

then

$$w \vdash_{\sigma'} D \to^* D'$$

for some D' such that  $C' \preceq D'$ .

*Proof.* We go by cases on b.

- Case u. We can assume that

$$C = (\langle \mathtt{C}, \rho, \mathtt{u} \rangle : F, w \diamond \tau)$$

and

$$D = (\langle \mathtt{C}, \rho, \mathtt{u} \rangle : F', w \diamond \tau').$$

with  $F \lesssim F'$  and  $\tau \lesssim \tau'$ . The proof goes by cases on the rule that has been applied to C

- Case [Op]. The assumption is

$$w \vdash_{\sigma} (\langle x := \mathtt{E}; \mathtt{D}, \rho, \mathtt{u} \rangle : F, w \diamond \tau) \to (\langle \mathtt{D}, \rho[x \leftarrow [\![\mathtt{E}]\!]_{\rho, w}], \mathtt{u} \rangle : F, w \diamond \tau),$$

and by applying the same rule to D, we obtain

$$w \vdash_{\sigma'} (\langle x := \mathsf{E}; \mathsf{D}, \rho, \mathsf{u} \rangle : F', w \diamond \tau') \to (\langle \mathsf{D}, \rho [x \leftarrow \llbracket \mathsf{E} \rrbracket_{\rho, w}], \mathsf{u} \rangle : F', w \diamond \tau')$$

and this shows the claim.

- Case [Skip]. analogous to the previous one.
- Case [Fence]. analogous to the case of assignments.
- Case [Pop]. analogous to the case of assignments.
- Case [If]. Analogous to the case of assignments.
- Case [While]. Analogous to the case of assignments.
- Case [Store]. The assumption is

$$w \vdash_{\sigma} (\langle *\mathtt{E} := \mathtt{F}; \mathtt{D}, \rho, \mathtt{u} \rangle : F, w \diamond \tau) \to (\langle \mathtt{D}, \rho, \mathtt{u} \rangle : F, w \diamond \tau[p \leftarrow v]),$$

where  $v = \llbracket \mathbf{F} \rrbracket_{\rho,w}$  and  $p = \llbracket \mathbf{E} \rrbracket_{\rho,w}^{\mathsf{Addr}}$ . observe that  $p \in \underline{w}(\mathsf{ArrId_u})$ . Which means that there is a pair  $(\mathtt{a},i)$  such that  $w(\mathtt{a})+i=p$  and  $\mathtt{a} \in \mathsf{ArrId_u}$ . So, in particular, we have that  $w \diamond \tau[p \leftarrow v] = w \diamond \tau[(\mathtt{a},i) \leftarrow v]$  because of Remark 3. We also deduce that by applying the same rule to D, we obtain

$$w \vdash_{\sigma} (\langle *E := F; D, \rho, u \rangle : F', w \diamond \tau') \rightarrow (\langle D, \rho, u \rangle : F', w \diamond \tau'[(a, i) \leftarrow v]),$$

and to show the claim, we just need to observe  $\tau[(\mathtt{a},i)\leftarrow v] \lesssim \tau'[(\mathtt{a},i)\leftarrow v]$ , which is a consequence of the assumption  $\mathtt{a}\in\mathsf{Arrld}_\mathtt{u}$ .

- Case [Load]. Analogous to the previous case
- Case [Call]. The assumption is

$$w \vdash_{\sigma} (\langle \mathsf{call} \ \mathsf{E}(\mathsf{F}_1, \dots, \mathsf{F}_k); \mathsf{D}, \rho, \mathsf{u} \rangle : F, w \diamond \tau) \to (\langle w \diamond \tau(\llbracket \mathsf{E} \rrbracket_{\rho, w}), \rho_0', \mathsf{u} \rangle : (\langle \mathsf{D}, \rho, \mathsf{u} \rangle : F, w \diamond \tau)),$$

where  $\rho'_0$  is a shorthand for

$$\rho_0[x_1,\ldots,x_k \leftarrow \llbracket \mathsf{F}_1 \rrbracket_{\rho,w},\ldots,\llbracket \mathsf{F}_1 \rrbracket_{\rho,w}]$$

and from the premise of the rule, we obtain that there is  $f \in \mathsf{Funld}_u$  such that  $[\![E]\!]_{\rho,w} = w(f)$ . Thus from the definition of  $w \diamond \tau$ , we deduce that  $w \diamond \tau = \tau(f) = \tau'(f)$  for the definition of the  $\lesssim$  relation. Thanks to these observations, we can show that the application of the same rule to D gives rule to D, we obtain

$$w \vdash_{\sigma'} (\langle \text{call } E(F_1, \dots, F_k); D, \rho, \mathbf{u} \rangle : F', w \diamond \tau') \rightarrow (\langle w \diamond \tau(\llbracket E \rrbracket_{\rho, w}), \rho'_0, \mathbf{u} \rangle : (\langle D, \rho, \mathbf{u} \rangle : F', w \diamond \tau')),$$

and this shows the claim.

- Case [SC]. The assumption is

$$w \vdash_{\sigma} (\langle \mathtt{syscall} \ \mathtt{s}(\mathtt{F}_1, \dots, \mathtt{F}_k); \mathtt{D}, \rho, \mathtt{u} \rangle : F, w \diamond \tau) \rightarrow (\langle \gamma(\mathtt{s}), \rho'_0, \mathtt{k}_\mathtt{s} \rangle : (\langle \mathtt{D}, \rho, \mathtt{u} \rangle : F, w \diamond \tau)),$$

where  $\rho'_0$  is a shorthand for

$$\rho_0[x_1, \dots, x_k \leftarrow [\![ \mathbf{F}_1 ]\!]_{\rho, w}, \dots, [\![ \mathbf{F}_1 ]\!]_{\rho, w}]$$

By applying the same rule to the configuration D, we obtain

$$w \vdash_{\sigma'} (\langle \text{syscall } s(F_1, \dots, F_k); D, \rho, u \rangle : F', w \diamond \tau') \rightarrow (\langle \gamma'(s), \rho'_0, k_s \rangle : (\langle D, \rho, u \rangle : F', w \diamond \tau')),$$

to conclude, we just need to observe that

$$\langle \gamma(s), \rho'_0, k_s \rangle \preceq \langle \gamma'(s), \rho'_0, k_s \rangle$$
.

- Case [Store-Error]. The assumption is

$$w \vdash_{\sigma} (\langle *E := F; D, \rho, u \rangle : F, w \diamond \tau) \rightarrow err,$$

We call  $p = [E]_{\rho,w}^{\mathsf{Addr}}$ , and from the premises of the rule, we deduce that  $p \notin \underline{w}(\mathsf{Arrld_u})$ . This suffices to show

$$w \vdash_{\sigma} (\langle *E := F; D, \rho, u \rangle : F', w \diamond \tau') \rightarrow err$$

and to show the claim.

- Case  $k_s$ . Under this assumption, most of the cases are analogous to the corresponding ones for user-mode execution. Even in this case, we go by induction on the rules. We just show some among the most important cases:
  - Case [Op]. The assumption is

$$w \vdash_{\sigma} (\langle x := \mathsf{E}; \mathsf{D}, \rho, \mathsf{k}_{\mathsf{S}} \rangle : F, w \diamond \tau) \to (\langle \mathsf{D}, \rho [x \leftarrow [\![\mathsf{E}]\!]_{\varrho,w}], \mathsf{k}_{\mathsf{S}} \rangle : F, w \diamond \tau),$$

from the assumption D, we deduce that it may either be:

- $(\langle x := E; D', \rho, k_s \rangle : F', w \diamond \tau'),$
- ( $\langle \text{fence}; x := E; D', \rho, k_s \rangle : F', w \diamond \tau'$ ), or
- $(\langle x := E; fence; D', \rho, k_s \rangle : F', w \diamond \tau')$

for some  $C \lesssim D'$ . In the first case, the same rule can be applied on D to show:

$$w \vdash_{\sigma'} (\langle x := \mathtt{E}; \mathtt{D}, \rho, \mathtt{k_s} \rangle : F', w \diamond \tau') \to (\langle \mathtt{D}'', \rho [x \leftarrow [\![\mathtt{E}]\!]_{\rho,w}], \mathtt{k_s} \rangle : F', w \diamond \tau')$$

for  $D'' \in \{D', \mathtt{fence}; D'\}$  and the claim comes from the observation that  $D \lesssim D''$ . In the other cases, we observe that:

$$w \vdash_{\sigma'} (\langle \text{fence}; x := E: D', \rho, k_s \rangle : F', w \diamond \tau') \rightarrow^2 (\langle D', \rho[x \leftarrow [E]_{\sigma,w}], k_s \rangle : F', w \diamond \tau')$$

and the conclusion is trivial, because  $D \lesssim D'$ 

- Case [Call]. The assumption is

$$w \vdash_{\sigma} (\langle \mathtt{call} \ \mathtt{E}(\mathtt{F}_1, \dots, \mathtt{F}_k); \mathtt{D}, \rho, \mathtt{k}_\mathtt{s} \rangle : F, w \diamond \tau) \rightarrow (\langle w \diamond \tau(\llbracket \mathtt{E} \rrbracket_{\rho, w}), \rho'_0, \mathtt{k}_\mathtt{s} \rangle : (\langle \mathtt{D}, \rho, \mathtt{k}_\mathtt{s} \rangle : F, w \diamond \tau)),$$

where  $\rho'_0$  is a shorthand for

$$\rho_0[x_1,\ldots,x_k \leftarrow \llbracket \mathsf{F}_1 \rrbracket_{\rho,w},\ldots,\llbracket \mathsf{F}_1 \rrbracket_{\rho,w}]$$

and from the premise of the rule, we obtain that there is  $f \in \mathsf{Funld}_k$  such that  $[\![\mathtt{E}]\!]_{\rho,w} = w(f)$ . Thus from the definition of  $w \diamond \tau$ , we deduce that  $w \diamond \tau = \tau(f)$ . We observe that D is:

$$(\langle \mathtt{C}'; \mathtt{D}', \rho, \mathtt{k_s} \rangle : F', w \diamond \tau')$$

For some  $C' \in \{\text{call } E(F_1, \dots, F_k), \text{fence}; \text{call } E(F_1, \dots, F_k), \text{scall } E(F_1, \dots, F_k), \dots\}$ . In any of these cases an application of the [Fence] rule and of the [Call] rule to D shows:

$$w \vdash_{\sigma'} (\langle \mathtt{call} \ \mathtt{E}(\mathtt{F}_1, \dots, \mathtt{F}_k); \mathtt{D}', \rho, \mathtt{k}_\mathtt{s} \rangle : F', w \diamond \tau') \rightarrow^2 \\ (\langle w \diamond \tau'(\llbracket \mathtt{E} \rrbracket_{\rho, w}), \rho'_0, \mathtt{k}_\mathtt{s} \rangle : (\langle \mathtt{D}', \rho, \mathtt{k}_\mathtt{s} \rangle : F', w \diamond \tau')),$$

To conclude, we must observe that  $w \diamond \tau(\llbracket \mathtt{E} \rrbracket_{\rho,w}) \preceq w \diamond \tau'(\llbracket \mathtt{E} \rrbracket_{\rho,w})$ . This is a consequence of the assumptions  $\llbracket \mathtt{E} \rrbracket_{\rho,w} = w(\mathtt{f}), \ \mathtt{f} \in \mathsf{Funld}_{\mathtt{k}} \ \mathrm{and} \ \tau \preceq \tau'.$ 

Corollary 1. For every two systems  $\sigma = (\tau_1, \gamma_1, \xi) \lesssim (\tau_2, \gamma_2, \xi) = \sigma'$ , every layout w, unprivileged command C, and registers  $\rho$ , we have

$$Eval_{\sigma,w}(C, \rho, u, \tau_1) \simeq Eval_{\sigma',w}(C, \rho, u, \tau_2),$$

where the equivalence is that of Definition 5, i.e.  $(v, \tau_1) \simeq (v, \tau_2)$  if  $\tau_1 =_{\mathsf{Id}_u} \tau_2$ , and it coincides with equality otherwise.

*Proof.* Observe that the initial configurations are in  $\lesssim$  relation, therefore the claim is a trivial consequence of Lemma 12.

Corollary 2. Transformations  $\eta, \psi, \theta$  preserve the semantics of the systems.

*Proof.* Observe that said  $\zeta \in \{\eta, \psi, \theta\}$ , we always have  $\sigma \lesssim \zeta(\sigma)$ . So, the conclusion is a trivial consequence of Corollary 1.

**Lemma 13.** For every stack of speculative configurations S layout w, system  $\sigma \in \operatorname{im}(\eta)$  such that  $\eta, \sigma \vdash \operatorname{twf}_{w,s}(S)$ , bt-free sequence of directives d : D, sequence of observations o : O, and S' such that  $\neg(\eta, \sigma \vdash \operatorname{twf}_{w,s}(S'))$ , there is a stack S'' such that

$$(w \vdash_{\sigma} S \xrightarrow{O:o} {^{n}S'}) \Rightarrow (w \vdash_{\sigma} S \xrightarrow{O} {^{n-1}S''})$$

and  $\eta, \sigma \vdash \mathsf{twf}_{w,s}(S'')$ .

*Proof.* By cases on n.

- Case 0. Absurd.
- Case n+1. In this case, we assume that

$$(w \vdash_{\sigma} S \xrightarrow{O:o} {n+1} S')$$

and  $\neg(\eta, \sigma \vdash \mathsf{twf}_{w,s}(S'))$ . We need to show that

$$(w \vdash_{\sigma} S \xrightarrow{O} {}^{n} S'')$$

for some S'' such that  $\eta, \sigma \vdash \mathsf{twf}_{w,s}(S'')$ . Assume that the claim does not hold, i.e. that  $\neg \eta, \sigma \vdash \mathsf{twf}_{w,s}(S'')$ . If n=0, that stack is exactly S, but this is absurd, so we can assume that n>0, however, in this case we would have a contradiction of Remark 17 for the configuration that is reached after n-1 steps because none of the two configurations that follow it satisfies  $\eta, \sigma \vdash \mathsf{twf}_{w,s}(\cdot)$ .

**Remark 17.** For every stack of speculative configurations S every layout w, every system  $\sigma \in \text{im}(\eta)$  such that  $\eta, \sigma \vdash \text{twf}_{w,s}(S)$ , and every system call s, one of the three following cases holds:

- $w \vdash_{\sigma} S \downarrow$
- $\ \forall d \neq \mathsf{bt}. \forall o.w \vdash_{\sigma} S \xrightarrow[d]{o} S' \Rightarrow \eta, \sigma \vdash \mathsf{twf}_{w,\mathtt{s}}(S')$
- $\ \forall d_1, d_2 \neq \mathsf{bt}. \forall o_1, o_2.w \vdash_{\sigma} S \xrightarrow[d_1:d_2]{o_1:o_2} ^2 S' \Rightarrow \eta, \sigma \vdash \mathsf{twf}_{w,\mathtt{s}}(S')$

*Proof.* The proof goes by cases on the proof of the predicate  $\eta, \sigma \vdash \mathsf{twf}_{w,s}(\cdot)$ . Many of the cases are trivial. The most interesting ones are when the stack of configurations is not-empty, i.e. it is like  $(\langle \eta(\mathtt{C}), \rho, \mathtt{k}_{\mathtt{s}} \rangle : F, (\mu, w \diamond \tau'), b_{ms}) : S$ . The proof goes by cases on C. Observe that by Remark 12, we can avoid the case of system calls.

- Case  $\epsilon$ . Observe that if  $F = \epsilon$ , the first claim holds, otherwise:

$$w \vdash_{\sigma} (\langle \eta(\epsilon), \rho, \mathtt{k}_{\mathtt{s}} \rangle : \langle \mathtt{D}, \rho', \mathtt{k}_{\mathtt{s}} \rangle : F', (\mu, w \diamond \tau'), b_{ms}) : S \xrightarrow[\mathtt{st}]{\circ}$$

$$(\langle \mathtt{D}, \rho'[ret \leftarrow \rho(ret)], \mathtt{k}_{\mathtt{s}} \rangle : F', (\mu, w \diamond \tau'), b_{ms}) : S \xrightarrow[\mathtt{st}]{\circ}$$

and the claim is a consequence of the premise of the proof rule for  $\eta, \sigma \vdash \mathsf{twf}_{w,s}(\cdot)$ .

- CASE x := \*F; D. Observe that one rule among [SLOAD-STEP], [SLOAD], [SLOAD-UNSAFE], and [SLOAD-ERROR] must apply. If one of the last two rules applies, the claim is trivial. Otherwise the stack looks like:

$$(\langle \eta(x := *F; D), \rho, k_s \rangle : F', (\mu, w \diamond \tau'), b_{ms}) : S$$

And if  $b_{ms} = \top$ , the first claim holds, because, without backtracking, this configuration cannot reduce further. So we can assume that  $b_{ms} = \bot$  Otherwise, we just show the case for the [SLOAD] rule:

$$\begin{split} w \vdash_{\sigma} (\langle \eta(x := *\mathtt{F}; \mathtt{D}), \rho, \mathtt{k}_{\mathtt{S}} \rangle : F', (\mu, w \diamond \tau'), b_{ms}) : S \xrightarrow{\circ : \mathsf{mem} \, p} {}^{2} \\ (\langle \eta(\mathtt{D}), \rho[x \leftarrow v], \mathtt{k}_{\mathtt{S}} \rangle : F', (\mu, w \diamond \tau'), b_{ms} \vee b') : \\ (\langle x := *\mathtt{F}; \eta(\mathtt{D}), \rho, \mathtt{k}_{\mathtt{S}} \rangle : F', \overline{(\mu, w \diamond \tau')}, b_{ms}) : S \end{split}$$

where  $p = [\![ \mathbf{F} ]\!]_{\rho,w}^{\mathsf{Addr}}$ ,  $v = (\mu, w \diamond \tau')^i(p)$ . Observe that in particular, the fence instruction that precedes the assignment has flushed the memory. For this reason, in order to show the claim, we just need to verify that:

$$\eta, \sigma \vdash \mathsf{twf}_{w,s}((\langle \eta(\mathtt{D}), \rho, \mathtt{k}_{\mathtt{s}} \rangle : F', \overline{(\mu, w \diamond \tau')}, b_{ms} \lor b'))$$

which is almost entirely consequence of the premise of the proof rule for  $\eta, \sigma \vdash \mathsf{twf}_{w,s}(\cdot)$ . in particular, since the domain of  $\mu$  does not contain any function address, the result is a consequence of Remark 4, which ensures that the resulting memory is  $w \diamond \tau''$  and  $\tau'' =_{\mathsf{Funld}} \tau' =_{\mathsf{Funld}} \tau$ .

- Case \*E := F; D. Observe that one rule among [SSTORE], [SSTORE-UNSAFE], and [SSTORE-ERROR] must apply. If one of the last two rules applies, the claim is trivial. Otherwise the stack looks like:

$$(\langle \eta(*\mathbf{E} := \mathbf{F}; \mathbf{D}), \rho, \mathbf{k}_{\mathrm{s}} \rangle : F', (\mu, w \diamond \tau'), b_{ms}) : S$$

We just show the case for the [SSTORE] rule:

$$w \vdash_{\sigma} (\langle \eta(*\mathtt{E} := \mathtt{F}; \mathtt{D}), \rho, \mathtt{k}_{\mathtt{s}} \rangle : F', (\mu, w \diamond \tau'), b_{ms}) : S \xrightarrow{\circ : \mathsf{mem} \, p} {}^{2}$$

$$(\langle \eta(\mathtt{D}), \rho, \mathtt{k}_{\mathtt{s}} \rangle : F', [p \mapsto v] \overline{(\mu, w \diamond \tau')}, b_{ms} \vee b') : S$$

where  $p = [\![\mathbf{F}]\!]_{\rho,w}^{\mathsf{Addr}}$ ,  $v = [\![\mathbf{F}]\!]_{\rho,w}$ . Observe that the claim requires to verify just that

$$\eta, \sigma \vdash \mathsf{twf}_{w,\mathtt{s}}((\langle \eta(\mathtt{D}), \rho, \mathtt{k}_\mathtt{s} \rangle : F', [p \mapsto v] \overline{(\mu, w \diamond \tau')}, b_{ms} \vee b'))$$

which is almost completely a consequence of the premise of the proof rule for  $\eta, \sigma \vdash \mathsf{twf}_{w,s}(\cdot)$ . In particular, as in the case above, we observe that we just need to observe that  $\overline{(\mu, w \diamond \tau')}$  is such that  $\eta, \sigma \vdash \mathsf{twf}_{w,s}(\overline{(\mu, w \diamond \tau')})$  and from the premise of the rule [SSTORE], we deduce that  $p \in \mathsf{ArrId}_k$ , that shows that the new buffer satisfies the desired conditions.

- CASE call  $E(F_1, ..., F_k)$ ; D. Observe that one rule among [SCALL], [SCALL-UNSAFE], and rule [SCALL-ERROR] must apply. If one of the last two rules applies, the claim is trivial. Otherwise,

The proof proceeds as in the previous cases, but the important observation is that:

$$\begin{split} w \vdash_{\sigma} (\langle \eta(\star \mathtt{E} := \mathtt{F}; \mathtt{D}), \rho, \mathtt{k}_{\mathtt{s}} \rangle : F', (\mu, w \diamond \tau'), b_{ms}) : S \xrightarrow{\circ : \mathsf{mem} \, p} {}^{2} \\ (\langle \eta(\mathtt{D}), \rho, \mathtt{k}_{\mathtt{s}} \rangle : \langle \overline{(\mu, w \diamond \tau')}(p), \rho, \mathtt{k}_{\mathtt{s}} \rangle : F', \overline{(\mu, w \diamond \tau')}, b_{ms} \vee b') : S \end{split}$$

where  $p = \llbracket \mathtt{F} \rrbracket_{\rho,w}^{\mathsf{Addr}}$ ,  $v = \llbracket \mathtt{F} \rrbracket_{\rho,w}$ . And we must ensure that the loaded program (namely  $\overline{(\mu,w \diamond \tau')}(p)$ ) is equal to  $\eta(\mathtt{C}')$  for some command. From the premise of the rule [SCALL] that has been applied, we deduce that  $p \in \underline{w}(\mathsf{Funld_k})$  means that there is a function in  $\mathsf{Funld_k}$  such that  $w(\mathtt{f}) = p$ . From Remark 4, we deduce that  $\overline{(\mu,w \diamond \tau')} = w \diamond \tau''$  for some  $\tau'' =_{\mathsf{Funld}} \tau' =_{\mathsf{Funld}} \tau$ . By definition of  $\cdot \diamond \cdot$ , this also means that  $\overline{w \diamond \tau''}(p) = \tau(p)$  that satisfies the requirement by hypothesis on  $\sigma$ .

#### Lemma 14. For every configurations

$$C = (\langle C_1, \rho_1, k_s \rangle : F_1, (\mu_1, m_1), b_{ms1})$$

and

$$D = (\langle C_2, \rho_2, k_s \rangle : F_2, (\mu_2, m_2), b_{ms2}),$$

and stack if

$$w \vdash_{\sigma} (\langle \mathtt{C}_1, \rho_1, \mathtt{k}_\mathtt{s} \rangle : F_1, (\mu_1, m_1), b_{ms\,1}) \xrightarrow[\mathtt{st}]{o} (\langle \mathtt{C}_2, \rho_2, \mathtt{k}_\mathtt{s} \rangle : F_2, (\mu_2, m_2), b_{ms\,2})$$

for some observation o, then

$$w \vdash_{\sigma} (\langle \mathtt{C}_1, \rho_1, \mathtt{k}_\mathtt{s} \rangle : F_1, \overline{(\mu_1, m_1)}) \to (\langle \mathtt{C}_2, \rho_2, \mathtt{k}_\mathtt{s} \rangle : F_2, \overline{(\mu_2, m_2)})$$

*Proof.* The proof goes by cases on the transition relation. Most of the cases are trivial. The most interesting ones are those which interact with memory.

- Case [Fence]. We rewrite the assumption as follows:

$$w \vdash_{\sigma} (\langle \mathtt{fence}; \mathtt{C}, \rho_1, \mathtt{k_s} \rangle : F_1, (\mu_1, m_1), b_{ms1}) \xrightarrow[\mathtt{st}]{o} (\langle \mathtt{C}, \rho_1, \mathtt{k_s} \rangle : F_1, \overline{(\mu_1, m_1)}, b_{ms1})$$

The claim is

$$w \vdash_{\sigma} (\langle \mathtt{fence}; \mathtt{C}, \rho_1, \mathtt{k_s} \rangle : F_1), \overline{(\mu_1, m_1)} \to (\langle \mathtt{C}_1, \rho_1, \mathtt{k_s} \rangle : F_1), \overline{(\mu_1, m_1)}$$

that is trivial.

- Case [SStore]. We rewrite the assumption as follows:

$$w \vdash_{\sigma} (\langle \mathbf{*E} := \mathbf{F}; \mathbf{C}, \rho_1, \mathbf{k_s} \rangle : F_1, (\mu_1, m_1), b_{ms\,1}) \xrightarrow[\mathbf{st}]{o} (\langle \mathbf{C}, \rho_1, \mathbf{k_s} \rangle : F_1, ([p \mapsto v] : \mu_1, m_1), b_{ms\,1}).$$

Where  $p = [\![\mathbf{E}]\!]_{\rho_1,w}^{\mathsf{Addr}}$  and  $v = [\![\mathbf{F}]\!]_{\rho_1,w}$ . To show the claim is suffices to observe that

$$w \vdash_{\sigma} (\langle \mathtt{*E} := \mathtt{F}; \mathtt{C}, \rho_1, \mathtt{k_s} \rangle : F_1), \overline{(\mu_1, m_1)} \to (\langle \mathtt{C}_1, \rho_1, \mathtt{k_s} \rangle : F_1, \overline{(\mu_1, m_1)}[p \leftarrow v])$$

and that  $\overline{([p \mapsto v] : \mu_1, m_1)} = \overline{(\mu_1, m_1)}[p \leftarrow v]$  that is a consequence of the definition of  $\overline{\cdot}$ .

- Case [Sload]. We rewrite the assumption as follows:

$$w \vdash_{\sigma} (\langle x := \star \mathtt{E}; \mathtt{C}, \rho_1, \mathtt{k_s} \rangle : F_1, (\mu_1, m_1), b_{ms1}) \xrightarrow[\mathtt{s}]{o} (\langle \mathtt{C}, \rho_1[x \leftarrow v], \mathtt{k_s} \rangle : F_1, (\mu_1, m_1), b_{ms1}).$$

Where  $v = (\mu_1, m_1)^0(\llbracket \mathbf{E} \rrbracket_{\rho_1, w}^{\mathsf{Addr}})$ . To show the claim is suffices to apply Remark 2.

### A.2.3 Technical Observations on the $\psi$ Transformation

The next result is crucial to show that  $\psi$  imposes speculative kernel safety. It relies on a predicate  $\Psi$  that is defined as follows:

$$\frac{\mathtt{C} \neq \mathtt{call} \ \mathtt{E}(\mathtt{F}_1, \dots, \mathtt{F}_k) \quad \mathtt{C} \in \{x := \mathtt{*E}, \mathtt{*E} := \mathtt{F}, \mathtt{scall} \ \mathtt{E}(\mathtt{F}_1, \dots, \mathtt{F}_n)\} \Rightarrow b_{ms} = \bot}{\Psi((\langle \mathtt{C}; \mathtt{D}, \rho, b \rangle : F, (\mu, m), b_{ms}) : S)}$$

**Lemma 15.** For every system  $\sigma = (\tau'', \gamma, \xi) \in \text{im}(\psi)$ , directive  $d \neq \text{bt}$ , and speculative stack of configurations  $(\langle P, \rho, k_s \rangle : F, (\mu, w \diamond \tau), b_{ms}) : S \text{ such that:}$ 

(H1) 
$$(\langle P, \rho, k_s \rangle : F, (\mu, w \diamond \tau), b_{ms}) : S \in \Psi,$$

(H2) 
$$\psi, \sigma \vdash \mathsf{twf}_{w,s}((\langle P, \rho, k_s \rangle : F, (\mu, w \diamond \tau), b_{ms}) : S),$$

$$(H3) \neg (w \vdash_{\sigma} (\mathtt{P}, \rho, \mathtt{k_s}) : F(\mu, w \diamond \tau) b_{ms} : S \downarrow_d),$$

there is a set Z containing:

• a speculative stack of configurations  $(\langle P', \rho', k_s \rangle : F', (\mu', w \diamond \tau'), b'_{ms}) : S'$  and an observation of such that

(C1) 
$$(\langle \mathsf{P}', \rho', \mathsf{k}_{\mathsf{s}} \rangle : F', (\mu', w \diamond \tau'), b'_{ms}) : S' \in \Psi,$$
  
(C2)  $\psi, \sigma \vdash \mathsf{twf}_{w,\mathsf{s}}((\langle \mathsf{P}', \rho', \mathsf{k}_{\mathsf{s}} \rangle : F', (\mu', w \diamond \tau'), b'_{ms}) : S'),$ 

- a speculative stack (err,  $\perp$ ): S',
- the configuration unsafe,

and either:

$$w \vdash_{\sigma} (\langle P, \rho, k_s \rangle : F, (\mu, w \diamond \tau), b_{ms}) : S \xrightarrow{o} z \text{ with } z \in Z$$
 (C3A)

$$w \vdash_{\sigma} (\langle P, \rho, k_s \rangle : F, (\mu, w \diamond \tau), b_{ms}) : S \xrightarrow[d]{o} S'' \xrightarrow[st]{o} z \text{ with } z \in Z \text{ and } S'' \in \Psi.$$
 (C3B)

*Proof.* From (H2), it must be the case where  $P = \psi(C, m, e)$ ;  $\vec{Q}$ , where  $\vec{Q}$  is a (possibly empty) sequence of commands  $Q_1, \dots Q_h$  such that for every  $1 \le i \le h.Q_i = \psi(C_i, \top, \bot)$ . We go by cases on  $\vec{Q}$ :

- Case  $\vec{Q} = \epsilon$ . The proof goes by cases on C
  - Case  $C = \epsilon$ . In this case, from (H3) and by introspection of the rules, we deduce that the only directive that is compatible with this command is st and that the transition rule that is applied must be [SPOP]. By introspection of this rule, we also deduce that F is not empty, and therefore from (H2), we deduce that  $F = \langle \psi(C_1, \top, \bot), \rho_1, k_s \rangle : \ldots : \langle \psi(C_k, \top, \bot), \rho_k, k_s \rangle$ , and therefore the source configuration can only reach the target stack of configuration is  $(F, (\mu, w \diamond \tau), b_{ms}) : S$  which we put in Z, the other elements of Z are not important for this part of proof, so they can be witnessed by any suitable values. This stack satisfies (C1), (C2) and (C3A). Claim (C1) holds because, by introspection of the definition of  $\psi(\cdot, \top, \bot)$ , we deduce that  $\psi(C_1, \top, \bot)$  cannot be a possibly unsafe command (i.e. a member of  $\{x := *E, *E := F, scall\ E(F_1, \ldots, F_n)\}$ ), and claim (C2) is a consequence of (H2).
  - Case C = I; D. In this case, we go by cases on the first instruction.
    - CASE I = skip. We start by observing that  $\psi(\text{skip}; D, m, e) = \text{skip}; \psi(D, m, e)$ . As we did for the case where  $C = \epsilon$ , from (H3) and by introspection of the rules, we deduce that the only directive that is compatible with this command is st and that the transition rule that is applied must be [SSKIP]. The target stack of configuration is  $(\langle \psi(D, m, e), \rho, b \rangle : F, (\mu, w \diamond \tau), b_{ms}) : S$ , which we put in Z together with any other suitable configurations that are not important for this part of the proof. This stack satisfies (C1), (C2) and (C3A). In order to establish (C1), we go by cases on m.

- CASE  $m = \top$ . If this is the case, by introspection of the definition of  $\psi(\cdot, \top, e)$ , we deduce that  $\psi(D, \top, e)$  cannot be a possibly unsafe command.
- CASE  $m = \bot$ . If this is the case, from (H2) we deduce that  $b_{ms} = \bot$ , so (C1) is trivially verified.

Claim (C2) is a direct consequence of (H2), as this transition does not modify the state.

- Case I = x := E. This case is analogous to the previous one, with the only difference that the transition causes a modification in  $\rho$ , however this modification does not influence the proof.
- Case I = \*E := F. We start going by cases on m.
  - CASE  $m=\top$ . If this is the case, by introspection of the definition of  $\psi(\cdot,\top,e)$ , we deduce that  $\psi(*\mathtt{E}:=\mathtt{F};\mathtt{D},m,e)=\mathtt{fence};*\mathtt{E}:=\mathtt{F};\psi(\mathtt{D},\bot,\bot)$ . From (H3), and by introspection of the rules, we deduce that  $d=\mathtt{st}$  and that the rule for showing the first transition is [FENCE], after that, another transition applies with the directive st. This means that the whole reduction has this shape:

$$w \vdash_{\sigma} (\langle \mathtt{fence}; \mathtt{*E} := \mathtt{F}; \psi(\mathtt{D}, \bot, \bot), \rho, \mathtt{k_s} \rangle : F, (\mu, w \diamond \tau), b_{ms}) : S \xrightarrow[\mathtt{d}]{o} S'' \xrightarrow[\mathtt{st}]{o} z$$

Depending on the rule that is applied to show the last transition (respectively [SSTORE], [SSTORE-ERROR], or [SSTORE-UNSAFE]), the final configuration z can either be:

- $\bullet \ (\langle \psi(\mathtt{D},\bot,\bot),\rho,\mathtt{k_s}\rangle: F, ([[\![\mathtt{E}]\!]_{\rho,w} \mapsto [\![\mathtt{F}]\!]_{\rho,w}], \overline{(\mu,w\diamond\tau)}),\bot): S,$
- $(\operatorname{err}, \bot) : S$ , or
- unsafe.

We take these three possible outcomes give us the elements of the set Z. Notice that because of the existence of the first reduction step—which, in turn, is a consequence of (H3)—it must be the case that  $b_{ms} = \bot$ . Therefore, S'' satisfies  $\Psi$ . Moreover, depending on the value of  $[\![E]\!]_{\rho,w}$  the final configuration can be any of the elements of Z. Finally, we are required to show that  $(\langle \psi(D,\bot,\bot), \rho, k_s \rangle : F, ([\![E]\!]_{\rho,w} \mapsto [\![F]\!]_{\rho,w}], \overline{(\mu,w \diamond \tau)}),\bot) : S$  satisfies (C1) and (C2). (C1) holds trivially, because the mis-speculation flag of the target configuration is  $\bot$ . For (C2), we are required to establish:

- 1.  $\overline{(\mu, w \diamond \tau)} = w \diamond \tau'$  for some  $\tau'$  such that  $\psi, \sigma \vdash \mathsf{twf}_{w,s}(\tau')$ .
- 2.  $F = \langle \mathsf{C}_0, \rho_0, b_0 \rangle : \ldots : \langle \mathsf{C}_k, \rho_k, b_k \rangle$  for  $\mathsf{C}_0, \ldots, \mathsf{C}_k$  such that  $\Sigma(\mathsf{C}_0, \top, \bot), \ldots, \Sigma(\mathsf{C}_k, \top, \bot)$ .
- 3.  $C = \psi(C', m, e) \wedge b_{ms} \Rightarrow m \wedge e \Rightarrow \mu = \epsilon$
- 4.  $\operatorname{dom}(\llbracket \mathbb{E} \rrbracket_{\rho,w} \mapsto \llbracket \mathbb{F} \rrbracket_{\rho,w}]) \subseteq \underline{w}(\operatorname{ArrId})$

From (H2), we deduce that the domain of  $\mu$  is a subset of Arrld, therefore, we can apply Remark 4 to deduce that  $\tau' =_{\mathsf{Funld}} \tau =_{\mathsf{Funld}} \tau''$  (for (H2)), and therefore  $\psi, \sigma \vdash \mathsf{twf}_{w,s}(\tau')$ , because  $\sigma \in \mathsf{im}(\psi)$ . Point (2) follows from (H2), as the frame stack below the topmost configuration is unchanged, point (3) can easily be verified by introspection of the target configuration, point (4) follows from the premise of the rule [SSTORE] that has been used to prove the transition.

- Case  $m=\bot$ . If this is the case, by introspection of the definition of  $\psi(\cdot,\bot,e)$ , we deduce that  $\psi(*\mathtt{E}:=\mathtt{F};\mathtt{D},m,e)=*\mathtt{E}:=\mathtt{F};\psi(\mathtt{D},\bot,\bot)$ . From (H3), and by introspection of the rules, we deduce that  $d=\mathtt{st}$  and that the rule for showing the transition [SSTORE], [SSTORE-ERROR], or [SSTORE-UNSAFE], and the target configuration can either be:
  - $\bullet \ (\langle \psi(\mathtt{D},\bot,\bot),\rho,\mathtt{k_s}\rangle: F, ([[\![\mathtt{E}]\!]_{\rho,w} \mapsto [\![\mathtt{F}]\!]_{\rho,w}], \overline{(\mu,w\diamond\tau)}),\bot): S,$
  - $(\operatorname{err}, \bot) : S, \operatorname{or}$
  - unsafe.

We take these three possible outcomes give us the elements of the set Z. Notice that from (H2), it must be the case that  $b_{ms} = \bot$ . From now on, the proof follows the same strategy adopted for the previous point.

- Case I = x := \*E. The proof goes by cases on m.
  - CASE  $m = \top$ . If this is the case, by introspection of the definition of  $\psi(\cdot, \top, e)$ , we deduce that  $\psi(x := *E; D, m, e) = fence; x := *E; \psi(D, \bot, \top)$ . From (H3), and by introspection of the rules, we deduce that d = st and that the rule for showing the first transition is [Fence],

after that, another transition applies with the directive  $d' \in \{st\} \cup \{ld \ i \mid i \in \mathbb{N}\}$ . This means that the whole reduction has this shape:

$$w \vdash_{\sigma} (\langle \mathtt{fence}; x := \mathtt{*E}; \psi(\mathtt{D}, \bot, \top), \rho, \mathtt{k_s} \rangle : F, (\mu, w \diamond \tau), b_{ms}) : S \xrightarrow[\mathtt{st}]{o} S'' \xrightarrow[\mathtt{d}']{o} z \text{ for some } o, z.$$

Depending on the rule that is applied to show the last transition, the final configuration can either be:

- $(\langle \psi(\mathtt{D},\bot,\top), \rho[x \leftarrow \overline{(\mu,w \diamond \tau)}(\llbracket \mathtt{E} \rrbracket_{\rho,w})], \mathtt{k}_{\mathtt{s}} \rangle : F, (\epsilon, \overline{(\mu,w \diamond \tau)}), \bot) : S$ , if the rule is [SLOAD] or [SLOAD-STEP],
- $(err, \bot) : S$ , if the rule is [SLOAD-ERROR] or
- unsafe, if the rule was [SLOAD-UNSAFE].

We take these three possible outcomes give us the elements of the set Z. Notice that because of the existence of the first reduction step—which, in turn, is a consequence of (H3)— it must be the case that  $b_{ms} = \bot$ . Therefore, S'' satisfies  $\Psi$ . Finally, we are required to show that  $(\langle \psi(D, \bot, \top), \rho[x \leftarrow (\mu, w \diamond \tau)(\llbracket E \rrbracket_{\rho,w})], k_s \rangle : F, (\epsilon, (\mu, w \diamond \tau)), \bot) : S$  satisfies (C1) and (C2). (C1) holds trivially, because the mis-speculation flag of the target configuration is  $\bot$ . Also (C2) can be established trivially, as it follows from (H2), and the observation that the buffer is empty and the mis-speculation flag is  $\bot$ .

- Case  $m = \bot$ . If this is the case, by introspection of the definition of  $\psi(\cdot, \bot, e)$ , we deduce that  $\psi(x := *E; D, \bot, e) = x := *E; \psi(D, \neg e, e)$ . From (H3), and by introspection of the rules, we deduce that the rule for showing the first transition is one among [SLOAD], [SLOAD-STEP], [SLOAD-ERROR], or [SLOAD-UNSAFE]; the directive  $d \in \{st\} \cup \{ld \ i \mid i \in \mathbb{N}\}$ . Depending on the rule that is applied, the target configuration can either be:
  - $(\langle \psi(\mathsf{D}, \neg e, e), \rho[x \leftarrow v], \mathsf{k}_s \rangle : F, (\mu, w \diamond \tau), b_{ms} \vee f) : S$ , where  $(\mu, w \diamond \tau)^i(\llbracket \mathsf{E} \rrbracket_{\rho, w}) = v, f$  for some  $i \in \mathbb{N}$ , if the rule is [SLOAD] or [SLOAD-STEP],
  - (err,  $\perp$ ): S, if the rule is [SLOAD-ERROR] or
  - unsafe, if the rule was [SLOAD-UNSAFE].

We take these three possible outcomes give us the elements of the set Z. Finally, we are required to show that  $(\langle \psi(D, \neg e, e), \rho[x \leftarrow v], \mathbf{k}_s \rangle : F, (\mu, w \diamond \tau), b_{ms} \vee f) : S$  satisfies (C1) and (C2). to this aim, we go by cases on e:

- CASE  $e = \bot$ . If this is the case, by introspection of the definition of  $\psi(\cdot, \top, e)$ , we deduce that  $\psi(D, \top, e)$  cannot be a possibly unsafe command. This shows (C1). The claim (C2) is a direct consequence of (H2).
- CASE  $e = \top$ . From (H2), we deduce that  $\mu = \epsilon$ , therefore, by introspection of the definition of  $(\mu, w \diamond \tau)^i(\llbracket \mathbf{E} \rrbracket_{\rho, w})$ , we deduce that  $f = \bot$ , so (C1) is trivial, and (C2) follows from (H2), and from  $f \vee b_{ms} = \bot \wedge \mu = \epsilon$ .
- CASE I = call  $E(\vec{F})$ ; D. By introspection of the definition of  $\psi(\cdot, m, e)$ , we deduce that  $\psi(\text{scall } E(\vec{F}), m, e) = \text{fence}$ ; scall  $E(\vec{F})$ ;  $\psi(D, \bot, \top)$ . From (H3), and by introspection of the rules, we deduce that the rule for showing the first transition is [FENCE], after that, another transition applies with the directive st. This means that the whole reduction has this shape:

$$w \vdash_{\sigma} (\langle \mathtt{fence}; \mathtt{scall} \ \mathtt{E}(\vec{\mathtt{F}}); \psi(\mathtt{D}, \bot, \top), \rho, \mathtt{k}_{\mathtt{s}} \rangle : F, (\mu, w \diamond \tau), b_{ms}) : S \xrightarrow[\mathtt{st}]{o} S'' \xrightarrow[\mathtt{d'}]{o} z \text{ for some } o, z.$$

Depending on the rule that is applied to show the last transition, the final configuration can either be:

- $(\langle \psi(\mathtt{D}', \top, \bot), \rho_0[\vec{x} \leftarrow \llbracket \vec{\mathsf{F}}_{\rho,w} \rrbracket], \mathtt{k}_s \rangle : \langle \psi(\mathtt{D}, \top, \bot), \rho, \mathtt{k}_s \rangle : F, (\epsilon, \overline{(\mu, w \diamond \tau)}), \bot) : S$ , if the rule is [SCALL] or [SLOAD-STEP],
- (err,  $\perp$ ): S, if the rule is [SCALL-ERROR] or
- unsafe, if the rule was [SCALL-STEP-UNSAFE].

We take these three possible outcomes give us the elements of the set Z. Notice that because of the existence of the first reduction step—which, in turn, is a consequence of (H3)—it must

be the case that  $b_{ms} = \bot$ . Therefore, S'' satisfies  $\Psi$ . Finally, we are required to show that  $(\langle \psi(\mathsf{D}', \top, \bot), \rho_0[\vec{x} \leftarrow \llbracket \vec{\mathsf{F}}_{\rho,w} \rrbracket], \mathsf{k}_{\mathtt{s}} \rangle : \langle \psi(\mathsf{D}, \top, \bot), \rho, \mathsf{k}_{\mathtt{s}} \rangle : F, (\epsilon, \overline{(\mu, w \diamond \tau)}), \bot) : S$  satisfies (C1) and (C2). (C1) holds trivially, because the mis-speculation flag of the target configuration is  $\bot$ . For (C2), we are required to establish:

- 1.  $\overline{(\mu, w \diamond \tau)} = w \diamond \tau'$  for some  $\tau'$  such that  $\psi, \sigma \vdash \mathsf{twf}_{w,s}(\tau')$ .
- 2. all the frames in the stack of the final configuration carry commands  $C_0, \ldots, C_k$  such that  $\Sigma(C_0, \top, \bot), \ldots, \Sigma(C_k, \top, \bot)$ .
- 3.  $\bot \Rightarrow \top \land \bot \Rightarrow \epsilon = \epsilon$
- 4.  $dom(\epsilon) \subseteq \underline{w}(Arrld)$

The proof of point (1) is analogous to the one we gave for the case of memory stores. Point (2) is a consequence of (H2), and follows by introspection of the target configuration, points (3) and (4), where we already substituted  $m, e, b_{ms}$ , and the value of the buffer in the target configuration, are trivial.

- CASE I = if E then  $C_1$  else  $C_2$  fi; D. By introspection of the definition of  $\psi(\cdot, m, e)$ , we deduce that

$$\psi(\text{if E then C}_1 \text{ else C}_2 \text{ fi}; \mathtt{D}, m, e) = \text{if E then } \psi(\mathtt{C}_1, \top, \bot) \text{ else } \psi(\mathtt{C}_2, \top, \bot) \text{ fi}; \psi(\mathtt{D}, \top, \bot).$$

From (H3), and by introspection of the rules, we deduce that the rule for showing the first transition is either [SIF] [SIF-BRANCH]. This means that the reduction takes the following form:

$$(\langle \mathtt{if} \ \mathtt{E} \ \mathtt{then} \ \psi(\mathtt{C}_1, \top, \bot) \ \mathtt{else} \ \psi(\mathtt{C}_2, \top, \bot) \ \mathtt{fi}; \psi(\mathtt{D}, \top, \bot), \rho, \mathtt{k}_\mathtt{s} \rangle : F, (\mu, w \diamond \tau), b_{ms}) : S \xrightarrow{\mathsf{br} \ b} \\ (\langle \psi(\mathtt{C}_i, \top, \bot); \psi(\mathtt{D}, \top, \bot), \rho, \mathtt{k}_\mathtt{s} \rangle : F, (\mu, w \diamond \tau), b_{ms}) : S \ \mathtt{for} \ \mathtt{some} \ b, i )$$

No err or unsafe state can be reached, so we take as witness a set Z that contains the target configuration shown above and some other dummy configurations for err and unsafe. We are required to show that

$$(\langle \psi(\mathsf{C}_i, \top, \bot); \psi(\mathsf{D}, \top, \bot), \rho, \mathsf{k}_s \rangle : F, (\mu, w \diamond \tau), b_{ms}) : S$$

satisfies (C1) and (C2). The validity of (C1) comes from introspection of the function  $\psi(\cdot, \top, \bot)$ . To assess the validity of (C2) we need show the validity of:

- 1.  $w \diamond \tau =_{\mathsf{FunId}} w \diamond \tau''$ , which is a direct consequence of (H2)
- 2.  $F = \langle C_0, \rho_0, b_0 \rangle : \ldots : \langle C_k, \rho_k, b_k \rangle$  for  $C_0, \ldots, C_k$  such that  $\Sigma(C_0, \top, \bot), \ldots, \Sigma(C_k, \top, \bot)$ , that trivially follows from (H2), as the frame stack remains unchanged.

- 3.  $\Sigma(\psi(C_i, \top, \bot); \psi(D, \top, \bot))$ , that is trivial.
- 4.  $dom(\mu) \subseteq \underline{w}(Arrld)$ , which is a consequence of (H2).
- Case I = while E do C od. Analogous to the case of conditional statements.
- CASE  $\vec{Q} \neq \epsilon$ . This remaining case is analogous to the case where  $\vec{Q} = \epsilon$  and  $C \neq \epsilon$ , as the premises (H1), (H2), (H3) also hold for the first element of  $\vec{Q}$ .

**Lemma 16.** For every  $\sigma = (\tau, \gamma, \xi)$ , if

$$w \vdash_{\psi(\sigma)} ((\mathtt{P}, \rho, \mathtt{k}_{\mathtt{S}}), (\mu, (w \diamond \tau')), b_{ms}) : S \xrightarrow{O}_{D} ^{n} S',$$

with:

- $S' \notin \{ \text{unsafe}, (\text{err}, b'_{ms}) : S'' \},$
- bt  $\notin D$ ,

76

- $\Psi(((P, \rho, k_s), (\mu, (w \diamond \tau')), b_{ms}) : S),$
- $\psi, \sigma \vdash \mathsf{twf}_{w,s}(((\mathsf{P}, \rho, \mathsf{k}_s), (\mu, (w \diamond \tau')), b_{ms}) : S),$

then we have  $\Psi(S')$ .

*Proof.* Notice that, from the premises of this lemma, assumptions (H1) and (H2) of Lemma 15 hold. With this in mind, we show the claim by induction on n.

- Case n = 0. In this case, the claim is trivial.
- Case n = n' + 1. We go by cases on n'.
  - Case n'=0. In this case n=1, therefore, we can apply Lemma 15, which concludes the claim.
  - Case n' > 0. In this case we have  $n' \ge 2$ , therefore, there is a stack of configuration T that is reached in 1 or 2 steps from the initial configuration such that  $\psi, \sigma \vdash \mathsf{twf}_{w,s}(T)$  and  $\Psi(T)$ . If this configuration is final, the conclusion is trivial because it must be the case that T = S', otherwise, we apply the IH.

**Lemma 17.** For every  $\sigma = (\tau, \gamma, \xi)$ , if

$$w \vdash_{\psi(\sigma)} ((\gamma(s), \rho, k_s), (\mu, (w \diamond \tau')), b_{ms}) \xrightarrow{O} ^n S,$$

with  $S \notin \{\text{unsafe}, (\text{err}, b'_{ms}) : S''\}, \ \tau' =_{\text{FunId}} \tau, \ \text{dom}(\mu) \subseteq \underline{w}(\text{ArrId}), \ and \ \text{bt} \notin D, \ then \ we \ have \ \Psi(S).$ 

*Proof.* Notice that by definition of  $\psi(\sigma)$ , the body of syscall are translated with initial flags  $m = \top$  and  $e = \bot$ , this means that the initial configuration looks like:

$$((\psi(\mathsf{C}, \top, \bot), \rho, \mathtt{k}_{\mathtt{s}}), (\mu, (w \diamond \tau')), b_{ms}).$$

By introspection of the definition of  $\psi(\cdot, \top, \bot)$ , we deduce that  $\psi(\mathsf{C}, \top, \bot)$  cannot be a possibly unsafe command, therefore  $\Psi(((\psi(\mathsf{C}, \top, \bot), \rho, \mathtt{k}_s), (\mu, (w \diamond \tau')), b_{ms}))$  holds. We also have

$$\psi, \sigma \vdash \mathsf{twf}_{w,s}(((\psi(\mathsf{C}, \top, \bot), \rho, \mathsf{k}_s), (\mu, (w \diamond \tau')), b_{ms}))$$

for the assumptions on  $\mu$  and  $\tau'$ . The claim is an application of Lemma 16.

**Lemma 18.** The transformation  $\psi$  imposes speculative kernel safety.

*Proof.* More precisely, we need to show that: said  $\sigma = (\tau, \gamma, \xi)$ , for every buffer  $\mu$  with  $dom(\mu) \subseteq \underline{w}(Arrld)$  and store  $\tau' =_{Fun} \psi(\tau)$ , if

$$w \vdash_{\zeta(\sigma)} ((\gamma(\mathtt{s}), \rho, \mathtt{k}_{\mathtt{s}}), (\mu, (w \diamond \tau')), b_{ms}) \xrightarrow{O} \flat^* \mathsf{unsafe},$$

then

$$w \vdash_{\zeta(\sigma)} ((\gamma(\mathtt{s}), \rho, \mathtt{k}_\mathtt{s}), \overline{(\mu, (w \diamond \tau'))}) \to^* \mathsf{unsafe}.$$

We start by observing that for Lemma 10, we can assume without lack of generality that bt  $\notin D$ . By introspection on the semantics, the very last configuration preceding the unsafe state, carried one of the commands \*E := F, x := \*E, call  $E(F_1, \ldots, F_k)$ , scall  $E(F_1, \ldots, F_k)$ , or scall  $E(F_1, \ldots, F_k)$ , and the last transition can be reached with any of the rules [SLOAD-UNSAFE], [SSTORE-UNSAFE], [SCALL-STEP-UNSAFE], or [SCALL-UNSAFE]. From Lemma 17, we can exclude this last case, and we know that, when the command is any of the other three, the mis-speculation flag of the last configuration before unsafe is  $\bot$ . Therefore, the conclusion of this proof is analogous to that of Lemma 8: by cases on the rule that has been used to show the last transition and with a combination of Lemmas 9 and 14 we show that using the speculative semantics, the configuration  $((\gamma(s), \rho, k_s), \overline{(\mu, (w \diamond \tau'))})$  can reach a configuration that satisfies the premises of rules [LOAD-UNSAFE], [STORE-UNSAFE], or [CALL-UNSAFE] respectively, and therefore, it also reaches the unsafe state.

#### A.2.4 Technical Observations on the $\theta$ Transformation

**Lemma 19.** For every system  $\sigma = (\tau'', \gamma, \xi) \in \text{im}(\theta)$ , directive  $d \neq \text{bt}$ , and speculative stack of configurations  $(\langle C, \rho, k_s \rangle : F, (\mu, w \diamond \tau), b_{ms}) : S \text{ such that:}$ 

$$(H1) \ \theta, \sigma \vdash \mathsf{twf}_{w,\mathtt{s}}((\langle \mathtt{C}, \rho, \mathtt{k}_\mathtt{s} \rangle : F, (\mu, w \diamond \tau), b_{ms}) : S),$$

$$(H2) \neg (w \vdash_{\sigma} (C, \rho, k_s) : F(\mu, w \diamond \tau) b_{ms} : S \downarrow_d),$$

there is a set Z containing:

• a speculative stack of configurations  $(\langle C', \rho', k_s \rangle : F', (\mu', w \diamond \tau'), b'_{ms}) : S'$  and an observation of such that

(C1) 
$$\theta, \sigma \vdash \mathsf{twf}_{w,s}((\langle \mathsf{C}', \rho', \mathsf{k}_s \rangle : F', (\mu', w \diamond \tau'), b'_{ms}) : S'),$$

- a speculative stack (err,  $\perp$ ): S',
- the configuration unsafe,

and either:

$$w \vdash_{\sigma} (\langle C, \rho, k_s \rangle : F, (\mu, w \diamond \tau), b_{ms}) : S \xrightarrow{o} z \text{ with } z \in Z,$$
 (C2A)

$$w \vdash_{\sigma} (\langle \mathtt{C}, \rho, \mathtt{k}_{\mathtt{s}} \rangle : F, (\mu, w \diamond \tau), b_{ms}) : S \xrightarrow[d]{o} S'' \xrightarrow[\mathtt{st}]{o} z \text{ with } z \in Z \text{ and } S'' \in \Psi,$$
 (C2B)

or

$$(\langle \mathtt{C}, \rho, \mathtt{k}_{\mathtt{s}} \rangle : F, (\mu, w \diamond \tau), b_{ms}) : S \xrightarrow{o} S'' \text{ and } S'' \xrightarrow{o'} T \Rightarrow \bot$$
 (C2C)

*Proof.* The proof goes by cases on C:

- CASE  $C = \epsilon$ . In this case, from (H3) and by introspection of the rules, we deduce that the only directive that is compatible with this command is st and that the transition rule that is applied must be [SPOP]. By introspection of this rule, we also deduce that F is not empty, and therefore from (H1), we deduce that  $F = \langle \theta(C_1), \rho_1, k_s \rangle : \ldots : \langle \psi(C_k), \rho_k, k_s \rangle$ , and therefore the source configuration can only reach the target stack of configuration is  $(F, (\mu, w \diamond \tau), \bot) : S$  which we put in Z, the other elements of Z are not important for this part of proof, so they can be witnessed by dummy values. This stack satisfies (C1), and (C2A). Claim (C1) is a direct consequence of (H1). Claim (C2A) can be verified by introspection of the
- Case C = I; D. In this case, we go by cases on the first instruction.
  - CASE I = skip. We start by observing that  $\theta(\text{skip}; D) = \text{skip}; \theta(D)$ . From (H3) and by introspection of the rules, we deduce that the only directive that is compatible with this command is st and that the transition rule that is applied must be [SSKIP]. The target stack of configuration is  $(\langle \theta(D), \rho, b \rangle : F, (\epsilon, w \diamond \tau), \bot) : S$ , which we put in Z together with other dummy values that are not important for this part of the proof. It is trivial to see that the target stack satisfies (C1), and (C2A).
  - Case I = x := E. This case is analogous to the previous one, although the transition modifies the register file, it does not influence the proof.
  - CASE I = \*E := F. By introspection of the definition of  $\theta(\cdot)$ , we deduce that  $\theta(*E := F; D) = *E := F; fence; <math>\theta(D)$ . From (H2), and by introspection of the rules, we deduce that the first transition is shown with [SSTORE], [SSTORE-ERROR], or [SSTORE-UNSAFE], and the target configuration can either be:
    - ( $\langle \text{fence}; \psi(D), \rho, k_s \rangle : F, ([\llbracket E \rrbracket_{\rho,w} \mapsto \llbracket F \rrbracket_{\rho,w}], w \diamond \tau), \bot) : S$ . Notice that, by applying the [FENCE] rule, this configuration reaches: ( $\langle \psi(D), \rho, k_s \rangle : F, (\epsilon, \overline{([\llbracket E \rrbracket_{\rho,w} \mapsto \llbracket F \rrbracket_{\rho,w}], w \diamond \tau)}), \bot) : S$ .
    - $(\operatorname{err}, \bot) : S$ , or

• unsafe.

We  $Z = \{\text{unsafe}, (\text{err}, \bot) : S, (\langle \psi(\mathtt{D}), \rho, \mathtt{k}_{\mathtt{s}} \rangle : F, (\epsilon, ([\llbracket \mathtt{E} \rrbracket_{\rho, w} \mapsto \llbracket \mathtt{F} \rrbracket_{\rho, w}], w \diamond \tau)), \bot) : S\}$ . If the target configuration is  $(\text{err}, \bot) : S$  or unsafe, there is nothing to show. In the first case, we must establish (C1) and (C2B). For (C2B), we have:

$$(\langle \mathtt{fence}; \psi(\mathtt{D}), \rho, \mathtt{k}_\mathtt{s} \rangle : F, ([\llbracket \mathtt{E} \rrbracket_{\rho, w} \mapsto \llbracket \mathtt{F} \rrbracket_{\rho, w}], w \diamond \tau), \bot) : S \xrightarrow{\circ}_{\mathsf{st}} \\ (\langle \psi(\mathtt{D}), \rho, \mathtt{k}_\mathtt{s} \rangle : F, (\epsilon, \overline{([\llbracket \mathtt{E} \rrbracket_{\rho, w} \mapsto \llbracket \mathtt{F} \rrbracket_{\rho, w}], w \diamond \tau)}), \bot) : S, )$$

and we are required to show that  $\Psi((\langle \mathtt{fence}; \psi(\mathtt{D}), \rho, \mathtt{k}_{\mathtt{s}} \rangle : F, ([\llbracket \mathtt{E} \rrbracket_{\rho,w} \mapsto \llbracket \mathtt{F} \rrbracket_{\rho,w}], w \diamond \tau), \bot) : S)$  holds, but this is trivial. For (C1), we must establish

$$\theta, \sigma \vdash \mathsf{twf}_{w,s}((\langle \psi(\mathtt{D}), \rho, \mathtt{k}_{\mathtt{s}} \rangle : F, (\epsilon, \overline{([[\mathtt{E}]_{\rho,w} \mapsto [\mathtt{F}]_{\rho,w}], w \diamond \tau)}), \bot) : S).$$

Crucially, to establish (C1) we need to observe that  $\theta, \sigma \vdash \mathsf{twf}_{w,s}((\epsilon, \overline{([[\mathbb{E}]]_{\rho,w} \mapsto [\![\mathbb{F}]\!]_{\rho,w}]}, w \diamond \tau)))$  holds. To this aim, we observe that for the premises of [SSTORE], it must be the case that  $[\![\mathbb{E}]\!]_{\rho,w} \in \underline{w}(\mathsf{Arrld}_k)$ , therefore, the claim is a direct consequence of Remark 4. The other conditions required for (C1) follow directly by introspection of the configuration and from (H1).

- CASE I = x := \*E. By introspection of the definition of  $\theta(\cdot)$ , we deduce that  $\theta(x := *E; D) = x := *E; \theta(D)$ . From (H2), and by introspection of the rules, we deduce that the rule for showing the first transition is one among [SLOAD], [SLOAD-STEP], [SLOAD-ERROR], or [SLOAD-UNSAFE]; the directive  $d \in \{st\} \cup \{ld \ i \mid i \in \mathbb{N}\}$ . Depending on the rule that is applied, the target configuration can either be:
  - $(\langle \theta(\mathbb{D}), \rho[x \leftarrow v], \mathbf{k}_s \rangle : F, (\epsilon, w \diamond \tau), f) : S$ , where  $(\epsilon, w \diamond \tau)^i(\llbracket \mathbf{E} \rrbracket_{\rho, w}) = v, f$  for some  $i \in \mathbb{N}$ , if the rule is [SLOAD] or [SLOAD-STEP],
  - (err,  $\perp$ ): S, if the rule is [SLOAD-ERROR] or
  - unsafe, if the rule was [SLOAD-UNSAFE].

We take the above-mentioned configurations as the elements of Z, and we observe that we are in the case where (C2A) holds. Finally, we are required to show that  $(\langle \theta(D), \rho[x \leftarrow v], k_s \rangle : F, (\epsilon, w \diamond \tau), f) : S$  satisfies (C1). By introspection of the definition of  $(\mu, w \diamond \tau)^i(\llbracket E \rrbracket_{\rho, w})$ , we deduce that  $f = \bot$ , so (C1) is a direct consequence of (H1).

- CASE I = call  $E(\vec{F})$ ; D. By introspection of the definition of  $\theta(\cdot)$ , we deduce that  $\theta(\text{scall }E(\vec{F})) = \text{scall }E(\vec{F})$ ;  $\psi(D)$ . From (H2), and by introspection of the rules, we deduce that another transition applies with the directive st. This means that the whole reduction has this shape:

$$w \vdash_{\sigma} (\langle \mathtt{scall} \ \mathtt{E}(\vec{\mathtt{F}}); \theta(\mathtt{D}), \rho, \mathtt{k_s} \rangle : F, (\epsilon, w \diamond \tau), \bot) : S \xrightarrow[d]{o} z \text{ for some } o, z.$$

Depending on the rule that is applied to show the step, the target final configuration can either be:

- $(\langle \theta(\mathsf{D}'), \rho_0[\vec{x} \leftarrow [\vec{\mathsf{F}}_{\rho,w}]], \mathsf{k}_s \rangle : \langle \theta(\mathsf{D}), \rho, \mathsf{k}_s \rangle : F, (\epsilon, w \diamond \tau), \bot) : S$ , if the rule is [SCALL] or [SLOAD-STEP],
- (err,  $\perp$ ): S, if the rule is [SCALL-ERROR] or
- unsafe, if the rule was [SCALL-STEP-UNSAFE].

Therefore, we are in the case where the target configuration is reached in one step (C2A). We take these three possible outcomes give us the elements of the set Z. Finally, we are required to show that  $(\langle \theta(D'), \rho_0[\vec{x} \leftarrow [\vec{F}_{\rho,w}]], k_s \rangle : \langle \theta(D), \rho, k_s \rangle : F, (\epsilon, w \diamond \tau), \bot) : S$  satisfies (C1), namely we are required to establish

$$\theta, \sigma \vdash \mathsf{twf}_{w,s}((\langle \theta(\mathsf{D}'), \rho_0 | \vec{x} \leftarrow \llbracket \vec{\mathsf{F}}_{\rho,w} \rrbracket], \mathsf{k}_s \rangle : \langle \theta(\mathsf{D}), \rho, \mathsf{k}_s \rangle : F, (\epsilon, w \diamond \tau), \bot) : S).$$

This conclusion is a direct consequence of (H1).

- CASE I = if E then  $C_1$  else  $C_2$  fi; D. By introspection of the definition of  $\theta(\cdot)$ , we deduce that  $\theta(\text{if E then } C_1 \text{ else } C_2 \text{ fi}; D) = \text{if E then fence}; \theta(C_1) \text{ else fence}; \theta(C_2) \text{ fi}; \theta(D).$ 

From (H2), and by introspection of the rules, we deduce that the rule for showing the first transition is either [SIF] [SIF-BRANCH]. This means that the first reduction step takes the following form:

Depending on the directive that has been supplied two cases arise:

- CASE  $d \in \{\text{st, br } [\![\![ E]\!]_{\rho,w} \}$ . If this is the case, then we have  $b_{ms} = \bot$ , therefore with the [Fence] rule, the configuration

$$(\langle \theta(\mathbf{C}_i; \mathbf{D}), \rho, \mathbf{k}_{\mathrm{s}} \rangle : F, (\epsilon, w \diamond \tau), b_{ms}) : S$$

is reached in two steps. By introspection of this configuration, and from (H1), we conclude that it satisfies (C1).

- Case  $d = \text{br } \neg \llbracket \texttt{E} \rrbracket_{\rho,w}$ , by introspection of the rules we observe that  $b_{ms} = \top$ , therefore (C2C)
- Case I = while E do C od. Analogous to the case of conditional statements.

**Lemma 20.** For every  $\sigma = (\tau, \gamma, \xi) \in \text{im}(\theta)$ , if

$$w \vdash_{\sigma} ((\mathtt{P}, \rho, \mathtt{k}_{\mathtt{s}}), (\mu, (w \diamond \tau')), b_{ms}) : S \xrightarrow{O}_{D} ^{n} S',$$

with:

- $S' \notin \{ \text{unsafe}, (\text{err}, b'_{ms}) : S'' \},$
- bt  $\notin D$ ,
- $\theta, \sigma \vdash \mathsf{twf}_{w,s}(((\mathsf{P}, \rho, \mathsf{k}_s), (\mu, (w \diamond \tau')), b_{ms}) : S),$

then we have  $\Psi(S')$ .

*Proof.* Notice that, from the premises of this lemma, assumptions (H1) and (H2) of Lemma 15 hold. Moreover, we also observe that

$$\theta, \sigma \vdash \mathsf{twf}_{w,s}(((P, \rho, k_s), (\mu, (w \diamond \tau')), b_{ms}) : S) \Rightarrow \Psi(S)$$
 (\*)

With this in mind, we show the claim by induction on n.

- Case n=0. In this case, the claim is a trivial consequence of (\*).
- Case n = n' + 1. We go by cases on n'.
  - Case n'=0. In this case n=1, therefore, we can apply Lemma 15, which concludes the claim.
  - CASE n' > 0. In this case we have  $n' \geq 2$ , therefore, there is a stack of configuration T that is reached in 1 or 2 steps from the initial configuration such that  $\theta, \sigma \vdash \mathsf{twf}_{w,s}(T)$ . If this configuration is final, the conclusion is trivial because it must be the case that T = S', otherwise, we apply the IH.

**Lemma 21.** For every  $\sigma = (\tau, \gamma, \xi) \in \text{im}(\theta)$ , if

$$w \vdash_{\sigma} ((\gamma(s), \rho, k_s), (\mu, (w \diamond \tau')), b_{ms}) \xrightarrow{O} \uparrow^n S,$$

with  $S \notin \{\text{unsafe}, (\text{err}, b'_{ms}) : S''\}$ ,  $\tau' =_{\mathsf{Funld}} \tau$ ,  $\mathsf{dom}(\mu) \subseteq \underline{w}(\mathsf{Arrld})$ , and  $\mathsf{bt} \notin D$ , then we have  $\Psi(S)$ .

80

*Proof.* Notice that by definition of  $\theta(\sigma)$ , the body of syscall are translated with an initial fence instruction, this means that the initial configuration looks like:

$$((\texttt{fence}; \theta(\texttt{C}), \rho, \texttt{k}_s), (\mu, (w \diamond \tau')), b_{ms}).$$

The proof goes by cases on n.

- Case n = 0. Trivial.
- CASE n > 0. By introspection of the semantics, we deduce that  $b_{ms} = \bot$ , and that the first rule that is applied is [Fence]. Therefore the first transition is:

$$w \vdash_{\sigma} ((\mathtt{fence}; \theta(\mathtt{C}), \rho, \mathtt{k}_{\mathtt{s}}), (\mu, (w \diamond \tau')), \bot) \rightarrow ((\theta(\mathtt{C}), \rho, \mathtt{k}_{\mathtt{s}}), (\epsilon, \overline{(\mu, (w \diamond \tau'))}), \bot),$$

so we conclude with an application of Lemma 20. In particular, the premise

$$\theta, \sigma \vdash \mathsf{twf}_{w,s}(((\theta(\mathtt{C}), \rho, \mathtt{k}_s), (\epsilon, \overline{(\mu, (w \diamond \tau'))}), \bot))$$

is discharged as follows:

• By introspection of the configuration, we observe, mis-speculation flag is  $\bot$ , and that the write buffer and the frame stack are empty.

• By applying Remark 4, we observe that  $\overline{(\mu,(w\diamond \tau'))}=_{\mathsf{Funld}} \tau'$ 

**Lemma 22.** The transformation  $\theta$  imposes specuative kernel safety.

*Proof.* The proof is analogous to those of Lemmas 8 and 18