Modelling and Verification of Security-Oriented Resource Partitioning Schemes

Adwait Godbole*, Leiqi Ye†, Yatin A. Manerkar†, Sanjit A. Seshia*
*University of California Berkeley, Berkeley, USA
{adwait, sseshia}@berkeley.edu
†University of Michigan, Ann Arbor, USA
{yeleiqi, manerkar}@umich.edu

***Abstract*—Side channel attacks such as Spectre and Meltdown exploit on-chip resources such as caches and buffers shared between the victim and the attacker in order to leak secret information from the victim. Previous works aim to mitigate these attacks by partitioning these vulnerable resources and allocating disjoint partitions to mutually untrusting process domains. While disjoint allocation prevents the attacker from gaining direct visibility of victim's partitions, secret information can also be leaked through the book-keeping state implementing the replacement/allocation policy. Proofs of security must reason about the partitions as well as the policy.**

In this work, we develop an abstract formal model for a generic security-oriented resource partitioning scheme, and formulate a corresponding attacker model. We then develop *conditional equality*-based relational invariants that enable unbounded proofs of security of the partitioning scheme with respect to the attacker model. These invariants allow us to reason about the state of the partitioning policy, which, as we discuss, can be more challenging than reasoning about the partitions themselves. We use our framework to model two resource partitioning approaches: DAWG and COLORIS. We demonstrate that using our invariants leads to verification performance improvements over other, more automated, model-checking approaches such as BMC and PDR.

I. Introduction and Example

Transient execution attacks such as Spectre [1], Meltdown [2], and the more recent MDS attacks [3], [4], [5], [6] exploit microarchitectural features such as caches, buffers, and functional units in order to leak secret data (e.g. cryptographic private keys). These features form *side channels* that allow the attacker to observe execution artefacts such as cache accesses, execution time and power consumption. The attacker can infer the secret data based on these observations. For instance, cache-based side channels [7], are based on the fact that victim's accesses to specific cache lines are observable to the attacker through a timing-based side channel [8]. Timing measurements can then be used to reconstruct accessed memory addresses, and consequentially, leak data that these addresses depend on.

While the microarchitectural features exploited by these attacks vary (e.g. caches [1], [2], [9], TLB [10], load, store and line-fill buffers [3], [4], [5], [6]), the central theme is exploiting a cache-like *resource* which is *shared* between the attacker and victim. In order to mitigate these attacks, there are approaches (e.g. [11], [12], [13], [14]) that partition these shared resources, and enforce allocation of *disjoint* partitions to mutually distrusting processes. Disjoint partitions disallow an attacker from affecting (modifying/observing) the victim's partitions, thus eliminating side channels formed by resource entries (e.g. cache lines). However, data leakage may still be possible through the state that implements the replacement/partitioning policy (e.g. [15] demonstrates an attack which leaks data through the Least-Recently-Used replacement policy state). Hence, proofs of security for designs implementing resource partitioning must reason over the partitioning policy in addition to the partition contents themselves.

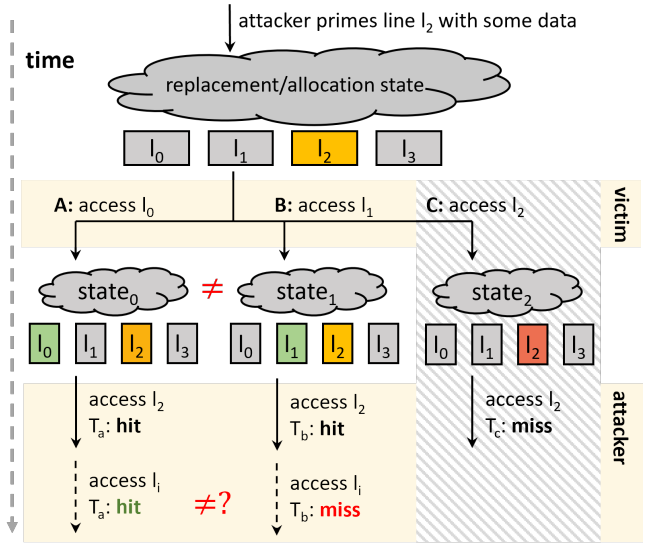

Fig. 1. Example illustrating a Prime+Probe style attack. The grey cloud represents the policy state and rectangles represent the resource partitions (cache lines). (A, B) indicate behaviours permitted by a partitioning scheme which has allocated lines l_0, l_1 to the victim and line l_2 to the attacker. (C, textured) indicates a behaviour which is possible on a non-partitioned cache, but is not possible on a partitioned cache with the above allocation.

In this work, we develop an abstract formal model for resource partitioning schemes, and a corresponding attacker model, capturing cache-based timing side channels. With the goal of proving the security of resource-partitioning approaches, we formulate *conditional-equality invariants*, which are a form of relational invariants [16]. These invariants enable unbounded proofs of the system against a non-interference-based formulation of the attacker model.

https://doi.org/10.34727/2023/isbn.978-3-85448-060-0_35

Previous works (e.g. [17], [18]) use pure equality-based relational invariants. While pure equality-based invariants suffice for reasoning over partition contents, the policy state requires more nuanced reasoning which is enabled by conditional equality invariants. We now illustrate this in the context of a Prime+Probe [8] attack (ref. Fig. 1).

Prime+Probe on a Resource Partitioned Cache*:* As a warm-up, we begin by discussing how a Prime+Probe attack operates on an unpartitioned cache. For simplicity, consider a segment of the cache consisting of four cache lines: $\{l_0, \cdots, l_3\}$. These are illustrated at the top of Fig. 1. Initially the attacker *primes* the cache by loading a value into a specific line, say l_2 (highlighted in orange in Fig. 1).

Following this, the victim performs a memory operation leading to a cache access (victim tab in Fig. 1). We consider three scenarios where the victim accesses either l_0, l_1 or l_2 (denoted A, B, and C respectively). In cases (A, B) the victim access (highlighted in green) does not evict the attacker primed line l_2 while in case (C) it does (highlighted in red).

Now, if the attacker accesses (*probes*) the cache on line l_2, there are two possible outcomes: in case (C) the access results in a cache miss (since the victim evicted the attacker's primed data), while cases (A, B) result in a cache hit (since the attacker's data was untouched). This difference manifests in the execution time of the attacker's probing access and allows the attacker to deduce the cache line accessed by the victim earlier. Since the cache line is indexed by the memory address, the attacker can infer this address, and as a consequence, any potentially secret data that the address depend on.

Cache partitioning schemes avoid this by allocating disjoint partitions to untrusting processes. In our example, a partitioning scheme can allocate l_0, l_1 to the victim process and l_2 to the attacker process. Since the victim is not allocated l_2, (C) is not a valid execution for the partitioned cache. Since both remaining executions (A, B) have the same (hit) result on the attacker's probing access, it cannot distinguish between these, and consequently, cannot infer the victim's loaded address.

Leakage through the policy state*:* While disjointness of victim and attacker partitions prevents a hit/miss timing side-channel leakage for the *first* probing access, leakage may still be possible through the replacement/allocation policy state (cloud in Fig. 1). The possible victim accesses, (A, B), may still lead to differing policy states ($\texttt{state}_0$) and ($\texttt{state}_1$). While this difference is not visible on the first probing access, it could potentially manifest after *some* number of attacker accesses.

Policy states *need not be equal*: In order to prove that an arbitrary number of attacker accesses lead to the same (hit/miss) outcome, relational model-checking [19], [16] based approaches (e.g. [18], [17], [20]) develop invariants relating possible resource states (A, B in Fig. 1). While the contents of the attacker-observable partition (l_2) itself must be equal, (else the attacker observes different outcomes), for the policy state, enforcing exact equality may be too strong. Policy states which are not fully equivalent, but still related in *some way* may be sufficient to ensure identical attacker access outcomes. Our model allows us to formulate *conditional* equality-based invariants, that are more nuanced than exact equality, and hence support inductive proofs of non-interference.

Related work*:* We formulate the security of partitioning schemes as a non-interference-based [21] hyperproperty [22]. Our verification approach is based on the well-known translation of non-interference to 2-safety [23], [24]. We verify the resulting 2-safety property with symbolic model checking [25], [26] (e.g. BMC, k-Induction [27], [28], PDR [29]).

There are several approaches that perform verification of non-interference-based properties, both on RTL (e.g. [30], [31]) and on abstract models (e.g. [32], [33], [34]). These approaches focus on proving programs secure against specific vulnerabilities by using techniques such as bounded model checking (e.g. [32]) and fuzzing (e.g. [31]). Our focus, on the other hand, is developing an abstract model specialized for resource partitioning schemes (implemented in either SW/HW) and proving unbounded security through invariants.

Our work is closest to the approaches performing relational symbolic execution/model checking (e.g. [18], [19], [17]). These too verify security-based hyperproperties on the self-composition of the model, by identifying relational invariants. These works consider different models than ours (e.g. [18] uses a simple programming language) or make use of different (often simpler) relational invariants (e.g. [17] considers pure equality-based invariants). Our focus is identifying specialized invariants in the context of resource partitioning-based models (for which pure equality-based invariants are inadequate).

Our contributions are as follows*:* (1) *Formal model for security-oriented resource partitioning*: We develop a formal model for security-oriented resource partitioning schemes with a corresponding attacker model that captures cache-based timing side channels. (2) *Conditional equality invariants*: We formulate relational invariants that are customized to this model and which enable us to reason about the partitions as well as the policy states. (3) *Evaluation*: We use our approach to model two partitioning schemes, DAWG [11] and COLORIS [13], and demonstrate that inductive proofs using our invariants can have much better performance as compared to bounded (BMC) and unbounded (PDR) techniques.

Outline*:* In §II we formulate the resource partitioning model and the corresponding threat model. In §III we develop our conditional-equality invariants that support inductive proofs of security. In §IV we discuss our case studies and experimental results, and §V concludes.

II. Modelling resource partitioning schemes

A. Resource model

Our model captures an abstract shared *resource* (e.g. cache lines, cache-ways, memory pages) that is being partitioned. For simplicity, we assume that there is only a single resource, but, this can easily be extended to multiple resources.

1) Resource: A resource is a collection of *cells*, each cell representing one unit of the resource. In Fig. 1, each line (l_i) is a cell. We assume that cells are indexed by an index set $\mathbb{I}$, and hold a value from the set $\mathbb{V}$. The value $\perp \in \mathbb{V}$ represents

```
1 if (∃i. r(i) = α ∧ d = a(i)) {
2     // Is a hit, let i be the hit
      index
3     p ← f_updH(a, p, d, i)
4 } else {
5     // Not a hit
6     i ← f_evict(a, p, d)
7     r(i) ← α
8     p ← f_updM(a, p, d, i)
9 }
```

Fig. 2. Semantics of an access $\mathsf{d} : \alpha$, from configuration $\mathsf{c} = \langle \mathsf{a}, \mathsf{r}, \mathsf{p} \rangle$.

the NULL value. The map r defines the mapping from cell indices to values that they contain, $\mathsf{r} : \mathbb{I} \to \mathbb{V}$.

2) Domains: A set of protection domains, denoted by $\mathbb{D}$, share the resource. In Fig. 1, the victim and attacker processes are domains. Each cell in the resource is allocated to a domain. This allocation is specified by an *allocation map*, identifying the domain that has access to a cell, $\mathsf{a} : \mathbb{I} \to \mathbb{D}$. We denote the set of possible allocations as $\mathsf{A} = \mathbb{I} \to \mathbb{D}$. For allocation a, we denote the set of all cell indices allocated to d by $\mathsf{a}{\downarrow_{\mathsf{d}}} \subseteq \mathbb{I}$.

3) Policies: We consider an abstract policy with states P. Each policy state $\mathsf{p} \in \mathsf{P}$ is an assignment to *policy elements*, $\mathsf{p} : \mathsf{E} \to \mathbb{V}_p$. In the Prime+Probe example of Fig. 1, the policy elements E are the bits in the replacement/allocation state (e.g. PLRU tree bits for a Tree-PLRU policy [35]).

Modelling the state as a collection of elements, E, as opposed to monolithically, allows us to develop conditional equality invariants in §III.

We identify three functions defining the policy behaviour:

$$\mathsf{f}_{evict} : \mathsf{A} \times \mathsf{P} \times \mathbb{D} \to \mathbb{I}$$
$$\mathsf{f}_{updM} : \mathsf{A} \times \mathsf{P} \times \mathbb{D} \times \mathbb{I} \to \mathsf{P}$$
$$\mathsf{f}_{updH} : \mathsf{A} \times \mathsf{P} \times \mathbb{D} \times \mathbb{I} \to \mathsf{P}$$

The function f_{evict}, given the current allocation, policy state, and the domain performing the access identifies the cell index that is chosen for replacement by the policy. The function f_{updM} identifies the new policy state that results when an access (by a given domain) is a miss, while f_{updH} identifies the state when the access is a hit (the hitting index is a function input).

4) Overall configuration: The overall configuration is a tuple $\mathsf{c} = \langle \mathsf{a}, \mathsf{r}, \mathsf{p} \rangle$ of these components. We denote the set of configurations as C. Initial configurations are of the form $\langle \mathsf{a}, \mathsf{r}_{init}, \mathsf{p}_{init} \rangle$, where a is arbitrary, the resource is empty, $\mathsf{r}_{init} = \lambda \mathsf{i}.\bot$, and $\mathsf{p}_{init} \in \mathsf{P}$ is the initial partitioning state.

5) Resource access semantics: At each step, a domain $\mathsf{d} \in \mathbb{D}$ performs an access operation with argument $\alpha \in \mathbb{V}$, denoted as $\mathsf{d} : \alpha$ (e.g. in a cache resource, the address is the argument). Fig. 2 provides the semantics of an operation $\mathsf{d} : \alpha$, which are determined by the functions $\mathsf{f}_{evict}, \mathsf{f}_{updM}, \mathsf{f}_{updH}$.

In case of a hit (i.e some allocated cell contains the accessed argument), the replacement state is updated according to f_{updH} (line 3). On a miss, the replaced index i is determined (line 6), following which the cell at i, and the replacement state is updated (lines 7, 8). We denote the transition relation (defined in Fig. 2) as $\delta_{\mathsf{acc}} : \mathsf{C} \times \{\mathsf{d} : \alpha \mid \mathsf{d} \in \mathbb{D}, \alpha \in \mathbb{V}\} \to \mathsf{C}$, which for a previous state and operation $\mathsf{d} : \alpha$, gives the resultant state.

6) Executions: A trace of the model is a sequence of configurations, $\pi = \mathsf{c}_0 \cdot \mathsf{c}_1 \cdots$ where c_0 is an initial configuration (§II-A4), and at each step j, some access operation $(\mathsf{d}_j : \alpha_j)$ is performed: $\delta_{\mathsf{acc}}(\mathsf{c}_j, \mathsf{d}_j : \alpha_j) = \mathsf{c}_{j+1}$. For a trace π, we denote $\mathsf{op}_\pi[j]$ as the operation performed at step j. We denote the set of executions of the model as Π.

B. Attacker Model and Security Property

We consider a timing-based attacker that can observe differences between the hit/miss outcomes of its accesses. Hence the hit/miss outcomes should not depend on the arguments (αs) of accesses by other domains. Otherwise, the attacker could infer these arguments, constituting an information leak. Operation $(\mathsf{d} : \alpha)$ results in a hit if $\mathsf{isHit}(\mathsf{c}, \mathsf{d} : \alpha) = \exists \mathsf{i}.\ \mathsf{r}(i) = \alpha \wedge \mathsf{a}(i) = \mathsf{d}$. Configurations $\mathsf{c}_1, \mathsf{c}_2$ are (single-access) indistinguishable to the attacker (with domain $\mathsf{d}^\#$) if the following holds:

$$\phi_{indist}(\mathsf{c}_1, \mathsf{c}_2) \equiv \forall \alpha.\ \mathsf{isHit}(\mathsf{c}_1, \mathsf{d}^\# : \alpha) \iff \mathsf{isHit}(\mathsf{c}_2, \mathsf{d}^\# : \alpha)$$

We formulate security as a non-interference property, where the attacker allocations and accesses are public inputs and the attacker hit/miss outcome is the public output. Non-interference requires that for any two traces, if the public inputs are equal, then so must be the public output. In our setting, two traces π_1, π_2 have the same public inputs (denoted $\pi_1 =_L \pi_2$) if the attacker allocation is identical ($\mathsf{a}(\pi_1[0]){\downarrow_{\mathsf{d}^\#}} = \mathsf{a}(\pi_2[0]){\downarrow_{\mathsf{d}^\#}}$) and attacker accesses are identical:

$$\forall j.\ \forall \alpha.\ \mathsf{op}_{\pi_1}[j] = (\mathsf{d}^\# : \alpha) \iff \mathsf{op}_{\pi_2}[j] = (\mathsf{d}^\# : \alpha)$$

Finally, identical public outputs (hit/miss outcomes) are captured by ϕ_{indist}. Hence, the overall non-interference-based hyperproperty is formulated as:

$$\Phi_{sec} \equiv \forall \pi_1, \pi_2 \in \Pi.\ \ \pi_1 =_L \pi_2 \implies \forall j.\ \phi_{indist}(\pi_1[j], \pi_2[j])$$

III. Invariants for Resource Partitioning Schemes

We aim to prove the hyperproperty Φ_{sec} using relational model checking [16] by developing a relational invariant $\phi_{inv}(\mathsf{c}_1, \mathsf{c}_2)$ (that relates states from the two traces π_1, π_2).

A. Conditions on ϕ_{inv}

We begin by listing conditions on ϕ_{inv}. As the base case, we get (by $\pi_1 =_L \pi_2$):

$$\mathsf{a}_1 \downarrow_{d^\#} = \mathsf{a}_2 \downarrow_{d^\#} \implies \phi_{inv}(\langle \mathsf{a}_1, \mathsf{r}_{init}, \mathsf{p}_{init} \rangle, \langle \mathsf{a}_2, \mathsf{r}_{init}, \mathsf{p}_{init} \rangle) \quad \text{(base)}$$

Next, we want ϕ_{inv} to be inductive, both w.r.t. attacker (Eq. ind-$\mathsf{d}^\#$) and non-attacker (Eq. ind-non-$\mathsf{d}^\#$) accesses. For the attacker accesses, the access argument should be identical.

$$\forall \alpha.\ \phi_{inv}(\mathsf{c}_1, \mathsf{c}_2) \implies \phi_{inv}(\delta_{\mathsf{acc}}(\mathsf{c}_1, \mathsf{d}^\# : \alpha), \delta_{\mathsf{acc}}(\mathsf{c}_2, \mathsf{d}^\# : \alpha)) \quad \text{(ind-}\mathsf{d}^\#\text{)}$$

$$\forall \mathsf{d}_1, \mathsf{d}_2 \neq \mathsf{d}^\#.\ \forall \alpha_1, \alpha_2.\ \phi_{inv}(\mathsf{c}_1, \mathsf{c}_2) \implies \phi_{inv}(\delta_{\mathsf{acc}}(\mathsf{c}_1, \mathsf{d}_1 : \alpha_1), \delta_{\mathsf{acc}}(\mathsf{c}_2, \mathsf{d}_2 : \alpha_2)) \quad \text{(ind-non-}\mathsf{d}^\#\text{)}$$

Finally, we want the invariant to imply indistinguishability for an attacker access (ϕ_{indist}):

$$\forall \mathsf{c}_1, \mathsf{c}_2.\ \phi_{inv}(\mathsf{c}_1, \mathsf{c}_2) \implies \phi_{indist}(\mathsf{c}_1, \mathsf{c}_2) \qquad \text{(indist)}$$

It is straightforward to see that if some ϕ_{inv} satisfies Eqns. base, ind-$\mathsf{d}^{\#}$, ind-non-$\mathsf{d}^{\#}$, indist, we get a proof of Φ_{sec}.

B. Shape of ϕ_{inv} invariants

In this section, we specialize the form of the invariant ϕ_{inv} considered. We require that ϕ_{inv} enforce (a) that cells allocated to $\mathsf{d}^{\#}$ are identical, and (b) that contents of the $\mathsf{d}^{\#}$-allocated cells are equal. Formally, we require $\phi_{inv} \supseteq \phi_1$, where,

$$\phi_1 \equiv (\mathsf{a}_1{\downarrow_{\mathsf{d}^{\#}}} = \mathsf{a}_2{\downarrow_{\mathsf{d}^{\#}}}) \wedge (\forall \mathsf{i}.\ \mathsf{i} \in \mathsf{a}_1{\downarrow_{\mathsf{d}^{\#}}} \implies \mathsf{r}_1(\mathsf{i}) = \mathsf{r}_2(\mathsf{i}))$$

Note that $\phi_1 \implies \phi_{indist}$. While ϕ_1 constrains allocations and their contents such that single-step indistinguishability is guaranteed, different policy states $(\mathsf{p}_1, \mathsf{p}_2)$ may lead to ϕ_1 not being inductive. Hence ϕ_{inv} additionally needs to relate the policy states from the two traces. However, unlike the resource contents $(\mathsf{r}_1, \mathsf{r}_2)$, the policy states may not be fully equivalent, and yet the scheme may be secure. Hence the invariant ϕ_{inv} must be more nuanced when relating c_1 with c_2.

In order to develop more nuanced invariants we make use of the fact that the policy state is composed of elements (§II-A3). We constrain that p_1, p_2 must only agree on some elements from E. The choice of these elements depends on the attacker-allocated indices ($\mathsf{a}{\downarrow_{\mathsf{d}^{\#}}}$) and is identified by a filtering function $\mathit{filter} : 2^{\mathbb{I}} \mapsto 2^{\mathsf{E}}$. The equality of $\mathsf{p}_1, \mathsf{p}_2$ is conditioned to only the elements in $\mathit{filter}(\mathsf{a}{\downarrow_{\mathsf{d}^{\#}}})$. The invariant ϕ_{inv} is defined as:

$$\phi_{inv} \equiv \phi_1 \wedge \forall e \in \mathit{filter}(\mathsf{a}{\downarrow_{\mathsf{d}^{\#}}}).\ \mathsf{p}_1(e) = \mathsf{p}_2(e)$$

The first term (ϕ_1) enforces equivalence of the $\mathsf{d}^{\#}$-allocated cells and their contents while the second enforces equality of the *filter*-identified elements of the policy state.

Importantly, the set of elements in $\mathit{filter}(\mathsf{a}{\downarrow_{\mathsf{d}^{\#}}})$ is not known statically since $\mathsf{a}\downarrow_{\mathsf{d}^{\#}}$ can be arbitrary (we want to verify security for arbitrary allocations). Hence conditional equality-based invariants cannot be subsumed by pure equality based relational invariants.

IV. Experimental Evaluation

We evaluate our modelling and verification approach on two case studies based on previously proposed partitioning schemes: (1) DAWG [11] and (2) COLORIS [13]. We cast both of these into our formal model (§II-A). We then formulate conditional equality-invariants by manually identifying the filter function (§III-B) and then perform verification w.r.t. the property Φ_{sec} (§II-B). We discuss details of modelling and verification in §IV-A and §IV-B respectively. Our experimentation is performed on a server machine running on an Intel i7-13700k processor at 5.2 GHz with 20GB of RAM. Our case study examples including models, invariants, and proof scripts can be found at `https://github.com/lichye/sec_resource_partitioning`.

TABLE I
DAWG Verification Run Times

Policy	Verification approach	Runtime
PLRU	k-Ind ($k = 10$)	2.789s
	BMC ($d = 12$)	42.6s
	BMC ($d = 20$)	145m43s
	PDR	24m10s
NRU	k-Ind ($k = 10$)	2.674s
	BMC ($d = 12$)	1m12s
	BMC ($d = 20$)	179m16s
	PDR	Timeout

A. Dynamically Allocated Way Guard (DAWG)

DAWG [11] proposes a technique for secure *way partitioning* of set associative structures, and develops an implementation of a way-partitioned cache. It allows privileged software to allocate *cache-ways* to processes based on resource utilization, and aims to provide isolation between cache-ways allocated to mutually untrusting processes.

1) Eviction Policy: We implement both Pseudo-Least Recently Used (PLRU) and Not Recently Used (NRU) eviction policies in DAWG. Similar to standard PLRU or NRU policies, DAWG's PLRU and NRU policies include metadata that is used to determine the victim way and to record the order in which addresses are accessed. PLRU employs a pointer to a metadata tree, while NRU uses access bits. The metadata tree and access bits form the policy elements (E) in our model in the case of PLRU and NRU respectively. In order to ensure isolation, DAWG constrains updates to the metadata and the identification of victim ways to the allocated partitions. Correspondingly, the hardware implementation must ensure that accesses performed by one domain do not modify metadata visible to another domain, as this could alter the hit/miss outcomes in other domains, potentially leading to timing-based information leakage.

2) Formal Modelling and Verification: We consider an 8-way DAWG cache design and implement Verilog modules for PLRU and NRU policies. Since DAWG performs way-partitioning, each way is a cell in our model, and hence, $|\mathbb{I}| = 8$ for both cases. For the PLRU policy, the policy elements contain the PLRU-tree bits, $|\mathsf{E}_{PLRU}| = 7$ and for NRU they are the access bits $|\mathsf{E}_{NRU}| = 8$. We perform a self-composition of this module and formulate Φ_{sec} as a safety property over this self-composition. For this, we use the standard encoding of 2-safety as a safety property [23]. We use the Yosys [36] based SymbiYosys (SBY) [37] model checker with Boolector [38] and ABC [29] as backend solvers to verify this property.

3) Result and Analysis: We apply three approaches: k-induction and BMC (using SMTBMC), and PDR (using ABC). For k-induction, we formulate invariants as discussed in §III-B. In Table I, we present the proof runtimes observed for the PLRU and the NRU policies. We observe that both PDR proof runtimes and BMC runtimes (for larger depths) are significantly higher than a k-inductive proof with our invariants. Our approach shows a significant speedup of 15.2x to 4022x in the runtime.

B. COLORIS

COLORIS [13] performs cache partitioning based on page colouring. In COLORIS, the OS kernel assigns colours to each memory page based on its address bits. Consequently, memory pages with different colours map to different cache sets in a physically indexed cache. By allocating different colours to each process COLORIS aims to improve performance in scenarios where cache contention occurs.

1) Formal Modelling and Verification: While COLORIS performs colour-based partitioning, it does allow colours to be shared between processes under certain scenarios. Hence, in full generality, it may allocate non-disjoint partitions to mutually untrusting processes, which puts it at risk of security leaks. However, in our experiments, we assume that the allocated colours are in fact disjoint, making the scheme secure.

We develop a model for a 4-way associative cache with a page allocation scheme, implemented in UCLID5 [39]. UCLID5 allows us to develop a model that partially abstracts the replacement policy by using uninterpreted functions to model policy functions (§II-A3), while constraining the policy to enforce disjointness of allocations. This abstraction allows us to verify an arbitrary policy that enforces disjointness.

2) Result and Analysis: We use both BMC and induction-based approaches to verify the model in UCLID5. For the proof by induction, we formulate Eqns. base, ind-$\mathsf{d}^{\#}$, ind-non-$\mathsf{d}^{\#}$, and indist as separate proofs using UCLID5. The cumulative runtime of the inductive proof (summing individual proof runtimes) is 16m33s. On the other hand, a BMC proof of security does not terminate even for a depth of three.

V. Conclusion

In this work, we develop a formal model for resource partitioning schemes and a corresponding attacker model that captures information leakage through timing-based side-channel attacks. We develop conditional equality-based relational invariants that enforce equality of state elements conditioned on some dynamic preconditions, and are more expressive than pure equality-based invariants. These invariants can support inductive proofs of security for resource partitioning schemes against a non-interference-based characterization of the attacker model. We model two partitioning schemes using our approach and demonstrate that conditional equality invariant-based proofs, while requiring manual specification of the invariants, can be much faster than other model-checking approaches. For future work, it would be interesting to develop an algorithm for automated synthesis of these invariants by utilizing their structure. Abstract models and invariants, such as the one we propose, that are specialized to design features, can provide scalable and trustworthy security guarantees.

Acknowledgements

This work was supported in part by Intel under the Scalable Assurance program, DARPA contract FA8750-20-C0156 and NSF grant 1837132.

References

[1] Paul C. Kocher, Daniel Genkin, Daniel Gruss, Werner Haas, Michael Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. Spectre attacks: Exploiting speculative execution. *2019 IEEE Symposium on Security and Privacy (SP)*, pages 1–19, 2019.

[2] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul C. Kocher, Daniel Genkin, Yuval Yarom, and Michael Hamburg. Meltdown: Reading kernel memory from user space. In *USENIX Security Symposium*, 2018.

[3] Stephan van Schaik, Alyssa Milburn, Sebastian Österlund, Pietro Frigo, Giorgi Maisuradze, Kaveh Razavi, Herbert Bos, and Cristiano Giuffrida. RIDL: Rogue In-Flight Data Load. *2019 IEEE Symposium on Security and Privacy (SP)*, pages 88–105, 2019.

[4] Stephan van Schaik, Marina Minkin, Andrew Kwong, Daniel Genkin, and Yuval Yarom. CacheOut: Leaking Data on Intel CPUs via Cache Evictions. *2021 IEEE Symposium on Security and Privacy (SP)*, pages 339–354, 2021.

[5] Michael Schwarz, Moritz Lipp, Daniel Moghimi, Jo Van Bulck, Julian Stecklina, Thomas Prescher, and Daniel Gruss. ZombieLoad: Cross-Privilege-Boundary Data Sampling. *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019.

[6] Claudio Canella, Daniel Genkin, Lukas Giner, Daniel Gruss, Moritz Lipp, Marina Minkin, Daniel Moghimi, Frank Piessens, Michael Schwarz, Berk Sunar, Jo Van Bulck, and Yuval Yarom. Fallout: Leaking Data on Meltdown-resistant CPUs. *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019.

[7] Qian Ge, Yuval Yarom, David Cock, and Gernot Heiser. A survey of microarchitectural timing attacks and countermeasures on contemporary hardware. *Journal of Cryptographic Engineering*, 8:1–27, 2016.

[8] Fangfei Liu, Yuval Yarom, Qian Ge, Gernot Heiser, and Ruby B. Lee. Last-Level Cache Side-Channel Attacks are Practical. *2015 IEEE Symposium on Security and Privacy*, pages 605–622, 2015.

[9] Michael Schwarz, Martin Schwarzl, Moritz Lipp, and Daniel Gruss. NetSpectre: Read Arbitrary Memory over Network. *ArXiv*, abs/1807.10535, 2019.

[10] Ben Gras, Kaveh Razavi, Herbert Bos, and Cristiano Giuffrida. Translation Leak-aside Buffer: Defeating Cache Side-channel Protections with TLB Attacks. In *USENIX Security Symposium*, 2018.

[11] Vladimir Kiriansky, Ilia A. Lebedev, Saman P. Amarasinghe, Srinivas Devadas, and Joel S. Emer. DAWG: A Defense Against Cache Timing Attacks in Speculative Execution Processors. *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 974–987, 2018.

[12] Ghada Dessouky, Alexander Gruler, Pouya Mahmoody, Ahmad-Reza Sadeghi, and Emmanuel Stapf. Chunked-Cache: On-Demand and Scalable Cache Isolation for Security Architectures. *ArXiv*, abs/2110.08139, 2021.

[13] Ying Ye, Richard West, Zhuoqun Cheng, and Ye Li. COLORIS: A dynamic cache partitioning system using page coloring. *2014 23rd International Conference on Parallel Architecture and Compilation (PACT)*, pages 381–392, 2014.

[14] Yao Wang, Andrew Ferraiuolo, Danfeng Zhang, Andrew C. Myers, and G. Edward Suh. SecDCP: Secure dynamic cache partitioning for efficient timing channel protection. *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, 2016.

[15] Wenjie Xiong and Jakub Szefer. Leaking Information Through Cache LRU States. *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 139–152, 2020.

[16] Gilles Barthe, Juan Manuel Crespo, and César Kunz. Relational Verification Using Product Programs. In *World Congress on Formal Methods*, 2011.

[17] Weikun Yang, Yakir Vizel, Pramod Subramanyan, Aarti Gupta, and Sharad Malik. Lazy Self-composition for Security Verification. In *International Conference on Computer Aided Verification*, 2018.

[18] Lesly-Ann Daniel, Sébastien Bardin, and Tamara Rezk. Binsec/Rel: Efficient Relational Symbolic Execution for Constant-Time at Binary-Level. *2020 IEEE Symposium on Security and Privacy (SP)*, pages 1021–1038, 2019.

[19] Gian Pietro Farina, Stephen Chong, and Marco Gaboardi. Relational Symbolic Execution. *Proceedings of the 21st International Symposium on Principles and Practice of Declarative Programming*, 2017.

[20] Hristina Palikareva, Tomasz Kuchta, and Cristian Cadar. Shadow of a Doubt: Testing for Divergences between Software Versions. *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, pages 1181–1192, 2016.

[21] Joseph A. Goguen and José Meseguer. Unwinding and Inference Control. *1984 IEEE Symposium on Security and Privacy*, pages 75–75, 1984.

[22] Michael R. Clarkson and Fred B. Schneider. Hyperproperties. *2008 21st IEEE Computer Security Foundations Symposium*, pages 51–65, 2008.

[23] Tachio Terauchi and Alexander Aiken. Secure Information Flow as a Safety Problem. In *Sensors Applications Symposium*, 2005.

[24] Gilles Barthe, P. D'Argenio, and Tamara Rezk. Secure information flow by self-composition. *Proceedings. 17th IEEE Computer Security Foundations Workshop, 2004.*, pages 100–114, 2004.

[25] Kenneth L. McMillan. Symbolic model checking. In *International Conference on Computer Aided Verification*, 1993.

[26] Edmund M. Clarke and David E. Long. Model checking, abstraction, and compositional verification. 1993.

[27] Per Bjesse and Koen Claessen. SAT-Based Verification without State Space Traversal. In *Formal Methods in Computer-Aided Design*, 2000.

[28] Mary Sheeran, Satnam Singh, and Gunnar Stålmarck. Checking Safety Properties Using Induction and a SAT-Solver. In *Formal Methods in Computer-Aided Design*, 2000.

[29] Alan Mischenko et al. Berkeley ABC tool. https://github.com/berkeley-abc/abc, 2022.

[30] Mohammad Rahmani Fadiheh, Dominik Stoffel, Clark W. Barrett, Subhasish Mitra, and Wolfgang Kunz. Processor Hardware Security Vulnerabilities and their Detection by Unique Program Execution Checking. *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 994–999, 2018.

[31] Jaewon Hur, Suhwan Song, Dongup Kwon, Eun-Tae Baek, Jangwoo Kim, and Byoungyoung Lee. DifuzzRTL: Differential Fuzz Testing to Find CPU Bugs. *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1286–1303, 2021.

[32] Kevin Cheang, Cameron Rasmussen, Sanjit A. Seshia, and Pramod Subramanyan. A Formal Approach to Secure Speculation. *2019 IEEE 32nd Computer Security Foundations Symposium (CSF)*, pages 288–28815, 2019.

[33] Marco Guarnieri, Boris Köpf, José Francisco Morales, Jan Reineke, and Andrés Sánchez. Spectector: Principled Detection of Speculative Information Flows. *2020 IEEE Symposium on Security and Privacy (SP)*, pages 1–19, 2020.

[34] Musard Balliu, Mads Dam, and Roberto Guanciale. InSpectre: Breaking and Fixing Microarchitectural Vulnerabilities by Formal Analysis. *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020.

[35] David A. Patterson and John L. Hennessy. Computer Organization and Design, Fifth Edition: The Hardware/Software Interface. 2013.

[36] Clifford Wolf, Johann Glaser, and Johannes Kepler. Yosys-A Free Verilog Synthesis Suite. https://github.com/YosysHQ/yosys, 2013.

[37] Claire Wolf, et. al. Symbiyosys. https://github.com/YosysHQ/sby, 2022.

[38] Robert Brummayer and Armin Biere. Boolector: An Efficient SMT Solver for Bit-Vectors and Arrays. In *International Conference on Tools and Algorithms for Construction and Analysis of Systems*, 2009.

[39] Elizabeth Polgreen, Kevin Cheang, Pranav Gaddamadugu, Adwait Godbole, Kevin Laeufer, Shaokai Lin, Yatin A. Manerkar, Federico Mora, and Sanjit A. Seshia. UCLID5: Multi-modal Formal Modeling, Verification, and Synthesis. In *34th International Conference on Computer Aided Verification (CAV)*, volume 13371 of *Lecture Notes in Computer Science*, pages 538–551. Springer, 2022.