VPsec: Countering Fault Attacks in General Purpose Microprocessors with Value Prediction

Rosario Cammarota
Qualcomm Technologies, Inc.
San Diego, California
ro.c@qti.qualcomm.com

Rami Sheikh
Qualcomm Technologies, Inc.
Raleigh, North Carolina
ralsheik@qti.qualcomm.com

ABSTRACT

Despite their complexity, general purpose microprocessors are susceptible to fault attacks. The state-of-the-art fault attacks rely on a precise understanding of the microprocessor datapath and the instructions critical path, to identify the exact time and location for injecting data faults that affect only targeted instructions in the pipeline. Software-only mitigations are only partially effective to defend against such attacks, whereas existing hardware-assisted mitigations require substantial changes to the microprocessor design. Both types of mitigation introduce significant overheads to the application memory footprint, the microprocessor area, or impact the overall system performance.

We propose a novel hardware-only scheme: Value Prediction for security (*VPSec*). VPsec leverages value prediction in an embodiment and system design to mitigate fault attacks in general purpose microprocessors. Value prediction is an elegant and hitherto mature microarchitectural performance optimization, which aims to predict the data value ahead of the data production with high prediction accuracy and coverage. VPsec leverages the presence of the state-of-the-art value prediction in a general purpose microprocessors, and re-architects it for security. It augments the original value prediction embodiment with fault detection logic and reaction logic to mitigate fault attacks to both the datapath and the value predictor itself. VPsec defines a new mode of execution in which the predicted value is trusted rather than the produced value. From a design perspective, VPsec requires minimal hardware changes (*negligible area impact*) with respect to a baseline that supports value prediction, it has no software overheads (*no increase in memory footprint*), and it retains most of the performance benefits of value prediction. Our evaluation of VPsec demonstrates its efficacy in countering fault attacks as well as its ability to retain the performance benefits of value prediction on cryptographic and non-cryptographic workloads.

CCS CONCEPTS

• **Security and Privacy → Security in Hardware**; • **Security in Hardware → Hardware Attacks and Countermeasures**; • **Hardware Attacks and Countermeasures → Side Channel Analysis and Countermeasures**;

KEYWORDS

Fault attacks; Fault detection; Fault reaction; General Purpose Microprocessors; Pipelining; Value prediction; Computer Security.

ACM Reference Format:
Rosario Cammarota and Rami Sheikh. 2018. VPsec: Countering Fault Attacks in General Purpose Microprocessors with Value Prediction. In *CF '18: CF '18: Computing Frontiers Conference, May 8–10, 2018, Ischia, Italy*. ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3203217.3203276

CF '18, May 8–10, 2018, Ischia, Italy

ACM ISBN 978-1-4503-5761-6/18/05…$15.00
https://doi.org/10.1145/3203217.3203276

1 INTRODUCTION

Yuce et al. [20–22] analyzed and demonstrated that an attacker can form an understanding of how the pipeline of a modern microprocessor works through profiling the critical path of a subset of its instruction set (identifying potential targets for fault attacks.) Profiling via micro-benchmarking is a common, reliable and inexpensive practice, which provides sufficient information for an attacker to engineer a sequence of faults to the data in the processor datapath, and ultimately perform: key extraction from cryptographic implementations, access control circumvention, and control flow subversion, to initiate buffer overflows and more sophisticated return oriented programming attacks. The severity of fault attacks in general purpose microprocessors directly correlates with the increasing importance or criticality of the contents being processed [5]. The threat affects a wide variety of products on the market. For example, premium contents and payments are processed on mobile devices and servers; and metering information are processed on relatively resource constrained devices that use sophisticated processors.[1] In both cases, the valuables are protected by encryption and access control mechanisms which are prone to physical attacks, i.e., side-channel and fault attacks [7].

Value prediction is a performance enhancing technique in which the value(s) produced by an instruction (producer) are predicted before the instruction is executed. Instructions that consume the predicted value(s) (consumers) can speculatively execute before the producer has executed, resulting in higher performance. The prediction is later confirmed when the producer is executed. If the predicted value did not match the produced value (the trusted value), recovery actions take place. Motivated by design simplicity and very high prediction accuracy[2] achieved by state-of-the-art value predictors (above 99%) [19, 27, 28], it is commonplace to use

[1] A class of modern IoT devices embody multicore processors with secure execution environment and vector units, e.g., ARM M7.
[2] Accuracy is defined as the number of correctly predicted dynamic instructions divided by the number of predicted dynamic instructions.

pipeline flushes as the default value misprediction recovery action. The basic idea is to throw away all instructions younger than the value mispredicted instruction, and then re-fetch and re-execute them. The high prediction accuracy and coverage[3] of modern value predictor designs enable the adoption of value prediction in real products.

In a typical fault attack scenario, an attacker injects faults into the underlying processor hardware to temporarily alter the execution of instructions. For example, the attacker can use clock glitches to force the processor to run beyond its nominal operating conditions [20, 21]. We assume that the attacker can inject faults into loaded or computed values, but he/she does not have control over the faulty value, i.e., the attacker can only flip bits at random positions in the data. Applying perturbations to the microprocessor creates faulty data values upon which instructions operate. Then, the attacker observes the fault effect in the output of the running software. The effect of a fault to bypass access control mechanisms or perform control flow subversion [7, 21] concludes with the attacker gaining unauthorized access to sensitive resources. In the case of key extraction, the attacker can break the security of the system by performing a systematic fault analysis method (e.g., Differential Fault Intensity Analysis) on the observed output (both correct and faulty output) to perform key extraction attacks [12]. In pipelined microprocessors, faults need to be carefully crafted, e.g., the intensity of a clock glitch, to avoid affecting all in flight instructions, as illustrated in [20]. Faults are injected by inserting glitches in the clock, such that the intensity of the glitch only influences instructions with the longest critical path (e.g., load instructions.)

Value prediction has appealing features that can be leveraged for security purposes to recover from fault attacks when produced (a.k.a., computed or loaded) data values are under attack. As opposed to trusting the produced value in value prediction, in security, the predicted value can be used to raise suspicion that the produced value has been tampered with (i.e., faulted.) In fact, under the attack scenarios in [21], in which an attacker can engineer a series of faults on produced data values, a value predictor can effectively prevent the propagation of faulty values to the attacker's advantage. For example, when the value predictor predicts a value that is discrepant with the produced value, the following actions can be engineered to mitigate the fault: (*a*) the predicted value, if highly confident, can be used in place of the faulty value, thus, the fault is corrected; (*b*) otherwise, the producer instruction along with all younger instructions are flushed, and then re-fetched and re-executed. Thus, the correct value is re-produced, and the fault is corrected.

In this work, we present VPsec, a security framework built around the concept of value prediction to counter fault attacks in general purpose microprocessors. VPsec can be applied to any value prediction schema/design. The design of VPsec enhances the original value predictor design with the following elements: (*a*) logic to detect the occurrence of faults in the produced or predicted data values; (*b*) logic to react to the occurrence of faults, by categorizing faults to the datapath or to the value predictor; (*c*) new security-aware recovery actions (reactions), which are triggered in place of the default recovery action when the value predictor is deemed under attack. The VPsec architecture guarantees that an attacker can never leverage the propagation of faults to his/her advantage. Furthermore, we present the design of the VPsec framework. The proposed design leverages state-of-the-art value predictors from [19, 27, 28], and provides the appropriate extensions to handle fault attack scenarios. If an attacker injects potentially successful faults, VPsec guarantees that the output value observed by the attacker will not be correlated with the attacker's fault assumptions (the value is either corrected by VPsec or it is infected when a corrective action cannot be taken.) Thus, VPsec deceives the attacker, as discussed in Section 3.4, without requiring software mitigations. Interestingly, since software mitigations can increase the attack surface, because more instructions are executed, a hardware-only solution like VPsec avoids such undesirable side-effect.

To the best of our knowledge this is the first contribution that proposes the design of a value prediction schema for its application in computer security to mitigate fault attacks. Our evaluation shows that the proposed technique protects the execution of unmitigated cipher suites in `OpenSSL` [1], the industry standard benchmarks `SPEC CPU2006` [3] and `SPEC CPU2017` [4], and other benchmark suites. Furthermore, we show that the proposed design requires minimal changes to the underlying value prediction machinery and it retains most of the performance benefits.

The rest of this contribution is organized as follows: Section 2 discusses the prior art; Section 3 details both the framework of VPsec as well as the proposed design; Section 4 provides the experimental and security evaluation of VPsec; Section 5 discusses system integration and system security aspects of VPsec; finally Section 6 concludes the contribution.

2 PRIOR ART

2.1 Value Prediction

Since the introduction of value prediction in the 90s [15, 23], significant improvements have been made to make value predictors more accurate and amenable to adoption in production hardware. In general, value predictors can be classified into two broad classes:

- **Computation-based Predictors**: In this class of predictors, predicted values are generated by applying a function to the value(s) produced by previous instance(s) of the instruction. Stride predictors [6, 23] are good examples of this class. The prediction is generated by adding a constant (stride) to the previous value.
- **Context-based Predictors**: This class of predictors rely on identifying patterns in the history of a given static instruction to predict the value. Finite Context Method predictors (FCM) [29, 30] are good examples of this class. Typically, such predictors use two structures. The first structure captures the history for the instruction. This history is used to index the second structure, which captures the values.

More recent proposals on context-based value predictors include: VTAGE [27] and D-VTAGE [28]. VTAGE uses several tagged prediction tables that are indexed using a hash of instruction program counter (PC) and different number of bits from the global branch history (context). These tables are backed up by a PC indexed, tagless last-value predictor (LVP). D-VTAGE augments VTAGE with a last value table (LVT) that is located before the first VTAGE table

[3]Coverage is defined as the number of predicted dynamic instructions divided by the number of dynamic instructions.

(VT0). LVT stores the last value (per instruction), while the VTAGE tables store the strides/deltas.

Another interesting class of value predictors advocates for predicting values indirectly via memory address prediction [19, 24]. Such techniques can only be used to predict the values produced by load instructions. The basic idea is to predict the memory address to be referenced by the load instruction, early in the pipeline (e.g., at fetch stage), and then probe the data cache to retrieve the predicted value. We refer to such predictors as indirect value predictors in the text.

State-of-the-art value predictors [19, 27, 28] addressed key practical challenges facing value prediction, and delivered very high prediction accuracy (over 99%) and good coverage, across a wide spectrum of workloads. The baseline value prediction scheme used in this work is described in Section 3.2.

2.2 Mitigations Against Fault Attacks

Both software (primarily) and hardware based mitigations against fault attacks have been studied in the literature. Hardware based mitigations usually duplicate a portion of the hardware blocks (e.g., registers) [21], repeatedly execute a computation, and verify the results from the multiple computations using a specific hardware unit. This type of mitigation is costly in terms of application performance as well as hardware area and complexity.

In contrast, software based mitigations provide more flexibility and portability as they do not require any underlying security hooks in the hardware. Prior art on software based mitigations proposes algorithm level [13, 26] and instruction level [8, 9] mitigations to hinder consistent fault injection. Algorithm level mitigations duplicate the execution of an algorithm and then compare the outcomes of both runs to verify the execution integrity. A fault is detected once a mismatch is signaled. At the Instruction Set Architecture (ISA) level, mitigations operate at a much finer granularity. Such mitigations attempt to counteract faults through duplicating instructions, repeating execution, and comparing the results from the original instruction and the redundant one.

Instruction duplication is believed to be able to reach full error coverage. Unfortunately, such a coverage comes at a cost: a significant performance overhead (e.g., [8] reports a 3.4x performance overhead) and energy increase. Duplication overheads are due to the increase in number of executed instructions, as an instruction needs to execute at least twice. Another side effect of instruction duplication is the increase in register pressure.

To provide comprehensive coverage of the attack surface, software-only mitigations are insufficient [10, 21, 22]. This is because microarchitectural aspects of the processor such as pipeline effects, cache effects, and physical implementation are invisible to the software countermeasures. Thus, faults injected in the hardware make the software-only countermeasures themselves vulnerable to fault injection resulting in an unmitigated software.

The state-of-the-art hardware-assisted software mitigation takes advantage of Single Instruction Multiple Data (SIMD) instruction set extensions, which are ubiquitous in modern microprocessors. Rather than duplicating and executing two identical instructions, the authors in [10] proposed to vectorize the original instruction and its replicate using a SIMD instruction. This solution effectively converts operation duplication into data duplication, therefore obtaining fault tolerance with much reduced overhead. Yet, this approach counters only the case of single fault in [21]. In [18], the authors propose a compiler-assisted mitigation to loop trip-count fault attacks, with modest performance and code footprint overhead.

Our work provides a hardware-only mitigation which counters against the attacks described in [21]. The proposed VPsec requires minimal changes to the hardware design of the value prediction machinery. By being a hardware-only solution, it avoids the high overheads associated with the deployment of software mitigations while retaining most of the benefits of value prediction. Finally, as an attacker will target mitigating instructions [21] as well as data values, the fact that VPsec does not require the deployment of software mitigations such as instruction duplications and sanity check [8] reduces the number of possible attacks.

2.3 Soft-Errors Tolerance

Technology scaling impacts the design of future microprocessors as the frequency of transient faults or soft-errors increases. The occurrence of transient faults exhibits statistical properties that can be characterized (via profiling), and then leveraged to design microprocessors that are tolerant to soft-errors in the data. Solutions that leverage the statistical characterization of transient faults, joint value locality and prediction, have been proposed in the literature [16, 25]. These proposals do not consider an adversarial model. That is, faults are not being forcefully injected under the control of an attacker, which limits the applicability of these designs, as opposed to VPsec which directly addresses these conditions. To be more specific, design considerations derived by characterizing transient faults in modern microprocessors lead to the design of mitigations that cannot withstand actual fault-attacks. On one hand, the occurrence of fault attacks does not follow any process-specific distribution. An attacker can attack at any point in time according to his/her attack schema. Design choices in [16, 25] can be circumvented by an attacker skilled in the art. On the other hand, the presence of an intentional attack changes the trust model in the data being processed and predicted, which we account for the design of VPsec.

3 VALUE PREDICTION FOR SECURITY

3.1 Framework

Value Prediction for Security, VPsec, provides a security framework built around the concept of value prediction. The proposed framework includes value prediction and extends any value prediction schema/design to provide an exhaustive coverage against all possible known fault attack scenarios, i.e., faults to produced data values, and faults to predicted data values. VPsec implements a pipeline which includes the following components (refer to Figure 1): (*a*) *value prediction machinery*, which performs value prediction; (*b*) *detection logic*, which compares the predicted and the produced values, and signals the presence of a discrepancy to the reaction logic; (*c*) *reaction logic*, which takes mitigating actions when a discrepancy is observed by the detection logic. A discrepancy between the predicted and the produced values can occur under one of the following two scenarios. First, faults are injected into the datapath

(i.e., a faulty value is produced), or faults are injected into the value predictor (i.e., a faulty predicted value is available). Second, faults are injected into several consecutive instances of the same producer instruction.

While the first scenario represents the basic case for using value prediction as a mitigation against fault attacks to general purpose microprocessors, it also illustrates a fundamental difference between the traditional use of value prediction in high-performance computing, which always trusts the produced value, and VPsec, which does not trust the produced value, and might trust the predicted value, when predictions are available, i.e., when value prediction accuracy and confidence are high. Furthermore, while the default recovery action in traditional value prediction only requires the re-execution of consumer instructions, the default recovery action in VPsec requires the re-execution of the producer instruction as well, as again, the data value is not trusted.

The first scenario is handled as follows. If the accuracy of the predicted value is high (above 99%) and the confidence is high, then a prediction is generated, and the predicted value can be used instead of the produced value. In this case, the reaction logic does nothing. If the accuracy of the predicted value is relatively low (below 99%, but above 90%) or the confidence is low, then a predicted value is not generated and the correction logic initiates recovery actions: flushing, re-fetching, and then re-executing the producer and all younger instructions, effectively re-computing the correct value. In both cases, the fault is masqueraded; in the former case, the fault is corrected on the fly.

For the second scenario, VPsec uses newly introduced Producer Status Registers (PSRs) that track if producer instructions are re-executed. A PSR is 8-bit[4] and it is allocated and initialized to zero when a producer is value predicted. When the producer successfully completes (i.e., commits and updates the architectural state), the PSR is released. The first time a producer instruction is re-executed (due to recovery actions), the value of its PSR is set to its complement, i.e., all bits in the PSR are set to 1. If a producer needs to be re-executed and its PSR value is non-zero, signaling that the previous instance of the producer was faulted, the produced data value is infected by VPsec, as VPsec deems the situation highly abnormal and irreversible, i.e., VPsec cannot correct the occurrence of the fault. Specifically, VPsec defines the following three types of recovery actions (*Reactions*) to mitigate faults.

Reaction 1. When PSR equals zero, and the predicted value has high accuracy and confidence, no action is taken, and the predicted value continues to be used by the consumer instructions.

Reaction 2. Like the conditions for Reaction 1, except that the accuracy is relatively low, or the confidence is low. In this case, we flush the pipeline and re-execute the producer and consumer instructions.

Reaction 3. When PSR is not equal zero, indicating a highly abnormal and irreversible scenario, we infect the computed value with a random number, and propagate the random number through the pipeline. Infection occurs by XOR-ing the produced data value with a random number. Such a reaction produces the wrong program results that the attacker will be observing. The correction of

[4]We use 8-bit, instead of 1-bit, PSRs to protect the PSRs against fault attacks.

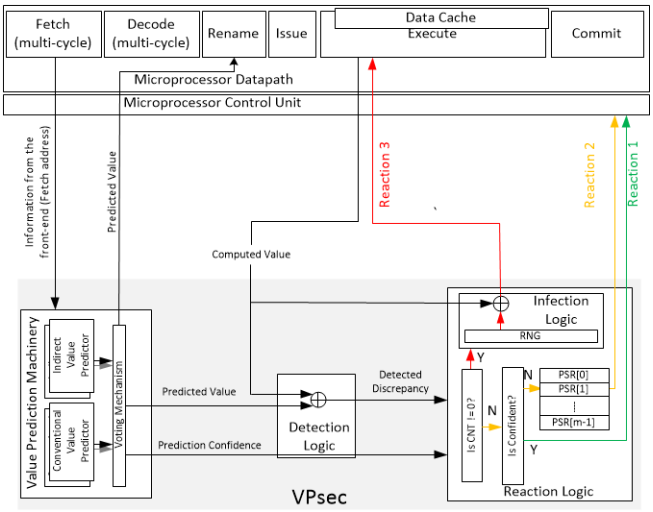

Figure 1: VPsec high-level diagram. CNT represents the PSR value for the value predicted instruction.

such result is assumed to happen at a higher level in the software stack.

In any of the cases above, the execution of the program continues until the end and it is not delayed by an arbitrary amount of time under the control of the attacker. Overall, the new framework for value prediction is capable of deceiving sophisticated attacks such as the ones described in [21].

3.2 Design

The VPsec design (shown in Figure 1) consists of three components: (*a*) the *value prediction machinery*; (*b*) the *detection logic*; and (*c*) the *reaction logic*. The following text elaborates on each one of these components.

Value Prediction Machinery: The baseline value prediction scheme used in this work is an ensemble of value predictors. The ensemble consists of one or more predictors from each of the following predictor classes: last-value predictors [23], context-based value predictors [27, 28], and indirect value predictors [19, 24]. All predictors are active simultaneously, attempting to predict the data values produced by executing instructions.

In such design, it is possible that multiple predictions can be provided for the same producer instruction. In this case, a voting mechanism is used to select the final prediction. Due to the high accuracy of the predictors in use, we almost never observed a disagreement between the predictions when multiple of them are made. *We mark a value prediction as **confident** when multiple agreeing predictions are supplied by the different value predictors. Moreover, accuracy counters are maintained for each predictor, providing continuous monitoring of the prediction **accuracy** per-predictor.*

Detection Logic: The detection logic collects the prediction from the value prediction machinery, if any prediction exist. Then, it compares the predicted value with the produced value when the producer is executed. The outcome of this comparison, along with the value predictors accuracy and confidence, is communicated to

the reaction logic to flag a discrepancy, i.e., the occurrence of an attack.

Reaction Logic: Upon receiving a discrepancy signal from the detection logic, the reaction logic evaluates the status of the producer instruction (PSR value) and the status of the value predictor (its accuracy and confidence.) One of the three recovery actions (described in Section 3.1) is invoked. It is important to note that when the reaction logic is triggered, the produced value cannot be trusted.

3.3 Overheads

VPsec assumes a general purpose processor that employs several state-of-the-art value predictors in a single embodiment, as described in Section 3.2. Given such a baseline, VPsec adds (1) simple combinatorial logic in the detection and reaction logic blocks, and (2) a set of PSR registers in the reaction logic. In the worst case scenario, VPsec will need to monitor the status of all in-flight instructions in the pipeline, the number of PSR registers required can match the number of entries in the reorder buffer, . Hence, for a single PSR register of n bits (e.g., 8-bit), and a typical reorder buffer with m entries (e.g., 224-entry), the storage required for the PSRs is $n \times m$ (224 bytes). Therefore, we believe that VPsec introduces negligible area and hardware overheads, as well as, it minimally increases the power consumption. At the same time, VPsec reduces the attack surface (by reducing the possible target instructions) and retains the benefits of Value prediction, adding performance benefits even in the presence of an aggressive attacker.

3.4 Modes of Operation

VPsec operates in two modes: a *training* mode, and an *execution* mode. During the training mode, the address and value predictors are trained, and the prediction accuracies are monitored and recorded for each predictor. Similarly, during the execution mode, the accuracies are monitored and compared against the accuracies recorded in the training mode. This comparison enables VPsec to establish trust in the predicted values during execution mode. When the prediction accuracy is high and the confidence in the predicted value is high, in both modes, the predicted value is trusted, and Reaction 1 takes place. The occurrence of Reaction 1 has two benefits: (*a*) it masquerades the occurrence of a fault by correcting the fault with the predicted value; (*b*) it does not incur a performance penalty because no instructions will be re-executed (to the contrary, the execution time can be reduced due to benefiting from value prediction.) It is worth noting that in a traditional fault attack scenario, e.g., the cases indicated in [21], only Reaction 1 is needed to correct the occurrence of data faults.

When the prediction accuracy is relatively low (*according to the value predictor accuracy monitors*) or the confidence in the predicted value is low (*only one value prediction is made despite having multiple value predictors*), the value predictor does not generate a prediction as the predicted value cannot be trusted, and Reaction 2 takes place.

Reactions 1 or 2 can be taken when the PSR value equals zero, indicating that the attack is less severe and that there is the possibility to recover from the fault by correcting the faulty value. When PSR is different from zero, Reaction 3 is taken, the computed value is infected, and the software under attack will output incorrect results to deceive the attacker.

4 EVALUATION

4.1 Environment

The microarchitecture of VPsec presented in Section 3 is faithfully modeled in our internally-developed, cycle-accurate simulator. The parameters of our baseline core are configured as close as possible to those of Intel's Skylake core [11]. Currently there is no publicly disclosed information about a product that deploys value prediction. However, given the enormous advances made in the value prediction space, we foresee value prediction to become a common feature of general purpose microprocessors.

Table 1 shows our baseline core configuration. The value prediction scheme, described in Section 3.2 and implemented in our performance model, supports predicting load instructions only, this is an artifact of our performance model and not a limitation of our proposed framework (VPsec). We restrict our evaluation and analysis to load instructions only. Load instructions have the longest critical path, and therefore, they are the easiest attack targets. Non-load instructions are not handled directly, but they can potentially be handled indirectly as they can influence future load instructions. Table 2 summarizes the focus of our evaluation.

Value prediction, just like any other prediction scheme, requires training time in which no predictions are made. This training manifests as a certain fraction of instructions not being value predicted. Training usually takes place during the initial phases of the workload.

4.2 Methodology

Evaluation is carried out in two parts. First, we evaluate the proposed value prediction design (described in Section 3.2) using benchmarks from the following benchmark suites: `SPEC CPU2017`[4], `SPEC CPU2006` [3], `OpenSSL`[1], `SPMV` [2], and `Terasort`. Our evaluation demonstrates the accuracy, coverage, and confidence of the proposed value prediction scheme. Moreover, we demostrate the effect of injecting faults to cover the different attack scenarios described earlier.

The workloads used in our evaluation are compiled to the ARM ISA using `GNU GCC` with `-O3` level optimization. We use 100-million instruction simpoints [17], except for short-running benchmarks, we simulate the first 100 million instructions, or until the benchmark completes.

4.3 Value Prediction

In this section, we evaluate the value prediction scheme described in Section 3.2 using the workloads listed earlier. Figure 2 shows the speedup (i.e., improvement in Instructions Per Cycle (IPC)) and coverage of the proposed value prediction scheme. For example, in the case of `OpenSSL`, on average 88.7% of loads are value predicted. Though not shown in Figure 2, the prediction accuracy of each one of the used value predictors is well above 99% [19, 27, 28].

Pipeline Stage	Configuration
Branch Prediction	BP: state-of-art TAGE predictor
Memory Hierarchy	`Block size`: 64B (L1), 128B (L2 and L3) `L1`: split, 64KB each, 4-way set-associative, 3-cycle access latency `L2`: unified, private, 1MB, 8-way set-associative, 16-cycle access latency `L3`: unified, shared, 8MB, 16-way set-associative, 32-cycle access latency `Memory`: 200-cycle access latency `Stride-based prefetchers`
Fetch through Rename Width	4 instr./cycle
Issue through Commit Width	9 instr./cycle (9 execution lanes: 3 support load-store operations, and 6 generic)
ROB/IQ/LDQ/STQ	224/97/72/56 (modeled after Intel Skylake)
Physical RF	348
Indirect Value Predictors (via Address Prediction)	`Stride-based`: 64k-entry, direct-mapped, indexed with pc only `Context-based`: 64k-entry, direct-mapped, use 32-bit load-path history
Conventional Value Predictors	Context-based: 7 tables, 64k-entry each, direct-mapped, use global branch histories of {0 "last-value", 5, 9, 17, 23, 39, 57}
VPsec PSRs	224×8 *bit*

Table 1: Baseline core configuration equipped with three value predictors.

Instruction Type	Number of Predictions made		
	0	1	≥ 2
Load	Outside the scope	Reaction 2	Reaction 1
non-Load	Can be handled indirectly as non-load instructions can influence future load instructions		

Table 2: Evaluation Methodology.

4.4 VPsec

Figure 3 shows the percentage of value predicted load instructions for which: only one value prediction is obtained from the value prediction machinery, or multiple predictions are obtained. For example, in the case of `OpenSSL`, on average 56.1% of the value predicted loads (88.7% in Figure 2) are covered by a single prediction, for which the prediction is not considered confident. Upon detecting the occurrence of a fault (detection logic), the reaction logic shall execute Reaction 2, that is, the producer load and consumer instructions shall be re-fetched and re-executed. For the remaining predicted loads, two (or more) predictions with high accuracy are available. Thus, upon detecting the occurrence of a fault (detection logic), the reaction logic shall execute Reaction 1, that is, the effect of the fault is corrected.

Admittedly, each time Reaction 2 is taken, there can be a performance penalty which is paid due to re-executing the producer load and the consumer instructions. Meanwhile, each time Reaction 1 is taken, not only the effect of a fault is corrected, but also there is a performance advantage due to the early execution of the consumer instructions, which operate on a predicted value with high confidence. The penalty due to Reaction 2 on load instructions depends on the locality of the workload when the producer load is re-executed. In the worst case, very unlikely, the re-execution of the producer load instruction may incur a cache miss and result in re-loading the data from main memory. In the best case, very likely, the re-execution of the producer load instruction will a hit in the L1 cache.

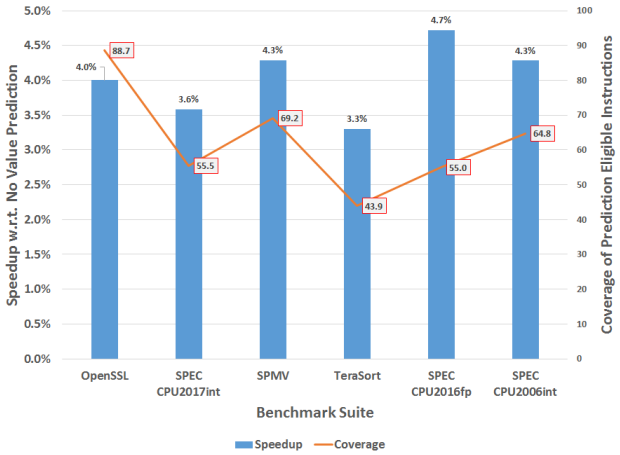

Figure 2: Speedup (bars, primary y-axis) and coverage (line, secondary y-axis) of the value prediction scheme. The accuracy of each predictor is over 99% (not shown).

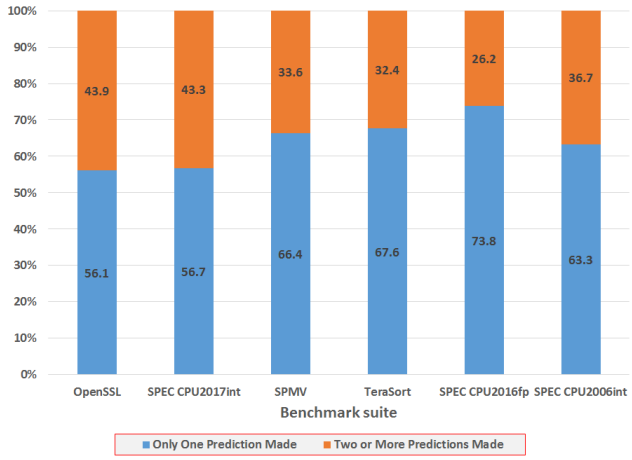

Figure 3: Breakdown of value predictions based on the number of predictions made.

To evaluate the performance impact due to the execution of Reaction 2, we assume the following extreme attack scenarios, in which an attacker faults: each value predicted load (Attack #1), every 10th predicted load (Attack #2), and every 100th predicted load (Attack #3). Figure 4 shows the performance impact with respect to a baseline with no value prediction (and no attacks). Table 3 reports both the average speedup and the range of speedups (indicating the minimum and maximum speedups of benchmarks within each benchmark suite.) In the case of `OpenSSL`, when no attack is performed, value prediction speeds up the execution of the benchmarks by up to 40%, with an average of 4%.

Benchmark suite	No Attack	Attack #1	Attack #2	Attack #3
OpenSSL	4.0/ [1.0, 40.0]	0.3/ [0.0, 1.5]	3.7/ [0.8, 36.2]	4.0/ [0.9, 39.6]
SPEC CPU2017int	3.3/ [0.7, 7.8]	1.1/ [-0.3, 5.5]	3.0/ [0.6, 7.6]	3.3/ [0.7, 7.8]
SPMV	4.2/ [0.1, 17.8]	2.5/ [0.0, 12.0]	4.0/ [0.1, 16.8]	4.2/ [0.1, 17.7]
TeraSort	3.0/ [0.9, 6.8]	0.2/ [0.0, 0.7]	2.7/ [0.7, 6.3]	2.9/ [0.9, 6.8]
SPEC CPU2006int	4.4/ [0.2, 7.2]	2.0/ [0.0, 5.7]	3.9/ [0.4, 6.7]	4.0/ [0.2, 7.1]
SPEC CPU2006fp	3.9/[0.2, 8.3]	1.2/[0.0, 3.2]	3.7/[0.2, 8.0]	3.9/[0.2, 8.3]

Table 3: VPsec: percentage average, minimum, and maximum speedups. (avg/[min, max])

In the most extreme scenario, in which an attacker launches an attack on each value predicted load, we observe no performance degradation as VPsec can correct 43.9% of the attacks (Reaction 1), while incurring the recovery action penalty on only 56.1% of the attacks (Reaction 2). Interestingly, the benefits of value prediction make up for the introduced re-execution overheads. While unrealistic, this scenario estimates the worst-case overheads that OpenSSL can experience. Under more realistic, yet very aggressive attack scenarios, as shown in Figure 4, the workloads still exhibit performance improvements which nearly match the performance improvement achieved by the no-attack scenario.

VPsec effectively tolerated the presence of realistic to extreme attack scenarios without incurring performance penalties for the benchmarks, even though the number of single predictions (i.e., unconfident predictions that trigger Reaction 2) is slightly higher than the number of multiple predictions (i.e., confident predictions that trigger Reaction 1).

In our experiments, we did not observe Reaction 3 get triggered. Reaction 3 is a very unlikely event, as we discuss in Section 5.2.

4.5 VPsec Rationale

We frame the discussion in this section around cryptographic algorithms, though the observations presented are equally applicable to non-cryptographic algorithms as well.

Value prediction relies on uncovering patterns in the values produced by the program instructions. Recent proposals for value prediction demonstrate remarkable ability for identifying and exploiting complex value patterns [27]. Alternative proposals [19] advocate for predicting the values produced by load instructions by leveraging patterns in the memory addresses being referenced.

Once sufficient confidence is established in these address or value patterns, they get used to predict future program values. When combined, value predictability and address predictability, can complement and strengthen one another. For example, Cryptographic algorithms, e.g., NIST standard compliant implementations of the Advanced Encryption Standard (AES) [5] exhibit both forms of predictability (address and value). The main loop of an AES implementation iterates for a number of rounds, which depends on the cryptographic strength of the AES instance, e.g., 10 rounds for 128 bit (key) algebraic strength of the block cipher. For each round, the algorithm updates the state table, during the steps of byte substitution, shift row, mix columns and add round key. These steps are simply loops over the elements of the AES state, i.e., the number of bytes in the state, which is 16.

[5]https://nvlpubs.nist.gov/nistpubs/fips/nist.fips.197.pdf

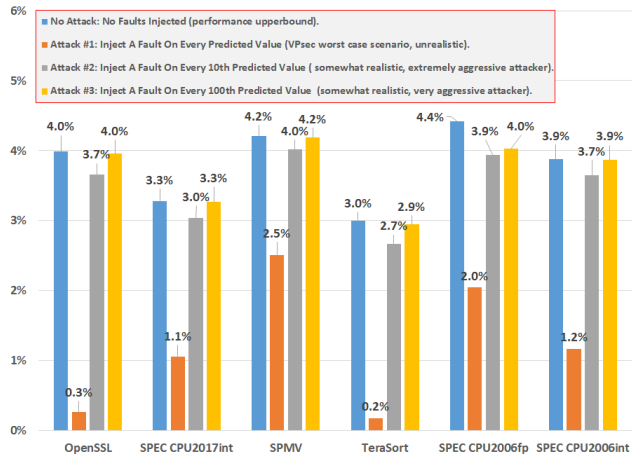

Figure 4: VPsec: evaluation of different attack scenarios.

Observe that the memory access patterns for the inner loops do repeat, and therefore values are easily predictable via address prediction. Our evaluation in Section 4 demonstrate that schemes like [19] are capable of predicting these patterns with very high accuracy and significant coverage.

Similarly, many of the non-load operations performed within the cryptographic algorithms are predictable. For instance, the number of times a loop iterates (a.k.a. *loop trip-count*) repeats across multiple executions of the loop and therefore is very predictable.

VPsec builds on all the recent advances in value prediction to deliver a security solution that can protect against fault attacks for a wide range of applications: cryptographic and non-cryptographic.

4.6 VPsec Limitations and Improvements

Admittedly, the implementation of VPsec evaluated has few limitations, and can be improvemed.

The training window discussed in Section 3.4 can be a window of vulnerability, as an attacker can inject faults during training time. The predictors in VPsec can be trained offline (i.e., pre-trained), to eliminate the the need to train online. In practice, for software executed in Trusted Execution Environment (TEE)[6], e.g., cryptographic algorithm implementations, address and value patterns can be very stable (discussed in Section 4.5.) This is in part because of security standards requirements on the implementation, and in part because of best practices in secure software development lifecycle.

Performance of VPsec can be further improved by reducing the occurrences of Reaction 2, which takes place when a single prediction is supplied by the value prediction machinery, despite having three predictors (refer to Figure 3). Such a reduction in the number of Reaction 2 invocations can be achieved by increasing the number of value predictors in VPsec.

It is important to note that the discussion in this Section is relevant to the implementation of VPsec we evaluated in this paper, and that it do not jeopardize the validity of the concepts, findings and conclusions presented.

[6]https://www.globalplatform.org/mediaguidetee.asp

	High Confidence	Low Confidence
Correct Prediction	(1) no re-execution, no infection (Reaction 1)	(2) re-execution, but no infection (Reaction 2)
Incorrect Prediction (Misprediction)	(3) no re-execution, no infection (Reaction 1, upper correction SW invoked)	(4) Infection (Reaction 3, upper SW notified)

Table 4: VPsec Operating Scenarios.

5 SYSTEM AND SYSTEM SECURITY DISCUSSION

5.1 VPsec in The Context of A System on Chip

VPsec is an embodiment composed of state-of-the-art value predictors, being used for multiple purposes: performance improvement (default use case: performance feature), and attack mitigation (new use case: security feature). VPsec can be configured to enable or disable the performance and security features.

When integrated in an SoC, the security feature of VPsec is meant to act when secure software executes within an implementation of the Global Platform TEE, e.g., to protect long term secret keys from being extracted using fault attacks, of which ARM TrustZone, for example, is one of such implementations of the TEE[7]. Outside the context of TEE, VPsec will operate as a traditional value predictor, enabling the performance feature.

The Value predictors in VPsec are context tagged. When VPsec starts its execution the value predictors do not carry the context of previous untrusted executions. Conversely, when the TEE completes its execution, the resources available to VPsec are cleared up and released. Therefore, and as elaborated more in Section 5.3, VPsec is resilient to attack scenarios similar to Spectre variant 2 [14].

5.2 System Security

In this section, we focus on the case when software executes security services in the system TEE. In such a case, an attacker capable of the state-of-the-art attacks [21] cannot observe the results of his/her injected faults, as VPsec corrects or masks out the faulty values before they become visible to the attacker.

The possible operating scenarios of VPsec are summarized in Table 4. Cases (1), (2) and (4) are handled properly by VPsec in both the cases, when an attack occurs or when no attack takes place. In cases (1) and (2) the software produces correct output via Reaction 1 and 2. In case (4) the software produces incorrect results, as VPsec infects the data, and a signal indicating that an infection had occurred (as consequence of an attack) is raised to the higher level of software to handle the case (action not shown in Figure 1 and outside the scope of this work.) As a result, for all the attack scenarios of interest to VPsec, an attacker is either deceived or deterred. With Reaction 1, the occurrence of a fault is first detected and then corrected. With Reactions 1 and 2, we can potentially observe performance benefits by virtue of using value prediction. With Reaction 3, the occurrence of an irreversible fault is countered, e.g., simultaneous faults to the instructions and the value predictor are deterred. In this case, additional recovery actions can be put in place in the upper layers of software implementing a security service, which is beyond the scope of this work.

Case (3) is a remote but conceivable case, which we report for completeness. In case (3) the value predictor itself is highly confident in the predicted value, but incorrect (a.k.a., mispredicted). The occurrence of Case (3) would produce the wrong program output even without the occurrence of an attack. This case is highly unlikely in the presence of multiple predictors, and the probability of this happening approaches zero as the number of value predictors increases. A loose upper bound on the probability of (3) to occur can be computed assuming that the occurrence of misprediction is equally likely to happen on each predicted value. That is, $Pr[(3)\ occurs] < (1 - max\_accuracy)^{nvp}$, where nvp is the number of predictors in the embodiment, and $max\_accuracy$ is the maximum of the accuracies of the predictors in the embodiment. The estimation above is pessimistic as it assumes that the probability of mispredicting is equally distributed across all the predicted instructions. A practical confirmation of the unlikelihood of scenario (3) is provided by our experimental results, for which even with only 3 value predictors, see Table 1, VPsec did not incur Reaction 3.

5.3 Relevance to Recently Discovered Attacks

The Spectre attack appeared in two variants [14]. In Spectre, variant 1 (bounds check bypass), and variant 2 (branch target injection), the conditional and indirect branch predictors are manipulated to steer the program speculation in a specific path that enables extracting information from other running processes. Such attacks are hard to fix, but also quite hard to exploit [14].

Admittedly, value predictors can expose a new variant of Spectre, but this variant can be mitigated using a similar technique as the one used to patch Spectre variant 2, e.g., by tagging prediction tables with Address Space Identifier (`ASID`), and using that information as part of the prediction logic. As Value predictor can be fixed against this new variant of Spectre, so does VPsec.

Because of the high-accuracy and practicality of recent value prediction implementations, we expect value prediction to be a commonplace in future generations of general purpose microprocessors. Thanks to the authors of Spectre, we have the possibility to analyze and fix value prediction against similar attacks. We leave the detailed analysis of this issue as future work. It is worth noticing, however, that VPsec is not designed to protect any form of micro-architectural side-channel attacks, as Meltdown and Spectre. However, it does protect against fault attacks to modern microarchitectures.

6 CONCLUSIONS

This work proposes VPsec, a novel hardware-only schema which leverages value prediction to detect, correct or counter fault attacks in general purpose microprocessors.

To the best of our knowledge this is the first contribution that proposes a framework which enhances value prediction, a performance improvement technique in high-performance microprocessors, for its use in computer security, to mitigate fault attacks. The design of VPsec demonstrates its efficacy in countering fault attacks to modern microprocessors with negligible changes to the original value prediction design and no associated software overhead.

[7] ARM TrustZone: https://developer.arm.com/technologies/trustzone

Furthermore, our evaluation shows that VPsec not only provides protection to the execution of unmitigated cipher suites in `OpenSSL` and industry standard benchmarks such as `SPEC CPU2017`, but also provides performance improvements by virtue of using value prediction.

ACKNOWLEDGMENTS

The authors would like to thank their colleagues: Rashid Attar, Gregory Bullard, Nicholas Yu, Greg Wright, Derek Hower, Eric Rotenberg, Arthur Perais, and Nahid Ghalaty, as well as the reviewers for comments that greatly improved the manuscript. This research was supported by Qualcomm Technologies, Inc. Any opinions, findings, and conclusions or recommendations expressed herein are those of the authors and do not necessarily reflect the views of Qualcomm Technologies, Inc.

REFERENCES

[1] [n. d.]. OpenSSL, Cryptography and SSL/TLS Toolkit. ([n. d.]). http://www.openssl.org
[2] [n. d.]. SpMV Benchmark. ([n. d.]). http://bebop.cs.berkeley.edu/spmvbench/
[3] Standard Performance Evaluation Corporation. 2006. The SPEC CPU 2006 Benchmark Suite. (2006). https://www.spec.org/cpu2006/
[4] Standard Performance Evaluation Corporation. 2017. The SPEC CPU 2017 Benchmark Suite. (2017). https://www.spec.org/cpu2017/
[5] Semiconductor Research Corporation. 2017. 2017 Research Opportunities, An Industry Vision and Guide: Security and Privacy. (2017). https://www.semiconductors.org/clientuploads/Research_Technology/SIA%20SRC%20Vision%20Report%203.30.17.pdf
[6] Eickemeyer and Vassiliadis. 1993. A load-instruction unit for pipelined processors. *IBM Journal of Research and Development* 37, 4 (1993), 547–564.
[7] Bar-El et al. 2006. The Sorcerer's Apprentice Guide to Fault Attacks. *Proc. IEEE* 94, 2 (2006), 370–382.
[8] Barenghi et al. 2010. Countermeasures against fault attacks on software implemented AES: effectiveness and cost. In *Proceedings of the 5th Workshop on Embedded Systems Security, WESS 2010, Scottsdale, AZ, USA, October 24, 2010*. 7.
[9] Conor et al. 2016. Lightweight Fault Attack Resistance in Software Using Intra-Instruction Redundancy. *IACR Cryptology ePrint Archive* 2016 (2016), 850.
[10] Chen et al. 2017. CAMFAS: A Compiler Approach to Mitigate Fault Attacks via Enhanced SIMDization. In *2017 Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2017, Taipei, Taiwan, September 25, 2017*. 57–64.
[11] Doweck et al. 2017. Inside 6th-Generation Intel Core: New Microarchitecture Code-Named Skylake. *IEEE Micro* 37, 2 (2017), 52–62.
[12] Ghalaty et al. 2014. Differential Fault Intensity Analysis. In *2014 Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2014, Busan, South Korea, September 23, 2014*. 49–58.
[13] Karri et. al. 2003. Parity-Based Concurrent Error Detection of Substitution-Permutation Network Block Ciphers. In *Cryptographic Hardware and Embedded Systems - CHES 2003, 5th International Workshop, Cologne, Germany, September 8-10, 2003, Proceedings*. 113–124.
[14] Kocher et al. 2018. Spectre Attacks: Exploiting Speculative Execution. *CoRR* abs/1801.01203 (2018). arXiv:1801.01203 http://arxiv.org/abs/1801.01203
[15] Lipasti et al. 1996. Value Locality and Load Value Prediction. In *ASPLOS-VII Proceedings - Seventh International Conference on Architectural Support for Programming Languages and Operating Systems, Cambridge, Massachusetts, USA, October 1-5, 1996*. 138–147.
[16] Nitin et al. 2015. FaultHound: value-locality-based soft-fault tolerance. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture, Portland, OR, USA, June 13-17, 2015*. 668–681.
[17] Perelman et al. 2003. Picking Statistically Valid and Early Simulation Points. In *12th International Conference on Parallel Architectures and Compilation Techniques (PACT 2003), 27 September - 1 October 2003, New Orleans, LA, USA*. 244–255.
[18] Proy et al. 2017. Compiler-Assisted Loop Hardening Against Fault Attacks. *TACO* 14, 4 (2017), 36:1–36:25.
[19] Sheikh et al. 2017. Load value prediction via path-based address prediction: avoiding mispredictions due to conflicting stores. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2017, Cambridge, MA, USA, October 14-18, 2017*. 423–435.
[20] Yuce et al. 2015. Improving Fault Attacks on Embedded Software Using RISC Pipeline Characterization. In *2015 Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2015, Saint Malo, France, September 13, 2015*. 97–108.
[21] Yuce et al. 2016. Software Fault Resistance is Futile: Effective Single-Glitch Attacks. In *2016 Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2016, Santa Barbara, CA, USA, August 16, 2016*. 47–58.
[22] Yuce et al. 2017. Analyzing the Fault Injection Sensitivity of Secure Embedded Software. *ACM Trans. Embedded Comput. Syst.* 16, 4 (2017), 95:1–95:25.
[23] Gabbay and Gabbay. 1996. *Speculative Execution based on Value Prediction*. Technical Report. EE Department TR 1080, Technion - Israel Institue of Technology.
[24] González and González. 1997. Speculative Execution via Address Prediction and Data Prefetching. In *Proceedings of the 11th international conference on Supercomputing, ICS 1997, Vienna, Austria, July 7-11, 1997*. 196–203.
[25] Li and Yeung. 2008. Exploiting Value Prediction for Fault Tolerance. WDA'08. (2008).
[26] Medwed and Schmidt. 2008. A Generic Fault Countermeasure Providing Data and Program Flow Integrity. In *Fifth International Workshop on Fault Diagnosis and Tolerance in Cryptography, 2008, FDTC 2008, Washington, DC, USA, 10 August 2008*. 68–73.
[27] Perais and Seznec. 2014. Practical data value speculation for future high-end processors. In *20th IEEE International Symposium on High Performance Computer Architecture, HPCA 2014, Orlando, FL, USA, February 15-19, 2014*. 428–439.
[28] Perais and Seznec. 2015. BeBoP: A cost effective predictor infrastructure for superscalar value prediction. In *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*. 13–25.
[29] Sazeides and Smith. 1997. *Implementations of Context-Based Value Predictors*. Technical Report. University of Wisconsin-Madison.
[30] Sazeides and Smith. 1997. The Predictability of Data Values. In *Proceedings of the Thirtieth Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 30, Research Triangle Park, North Carolina, USA, December 1-3, 1997*. 248–258.