cryptography

MDPI

Article

Improving Performance and Mitigating Fault Attacks Using Value Prediction [†]

Rami Sheikh [1] and Rosario Cammarota [2,*]

[1] Qualcomm Technologies, Inc., Raleigh, NC 27617, USA; ralsheik@qti.qualcomm.com or rmalshei@ncsu.edu
[2] Qualcomm Technologies, Inc., San Diego, CA 92121, USA
[*] Correspondence: ro@ieee.org or rosarioc@qti.qualcomm.com; Tel.: +1-858-845-8178

[†] This paper is an extended version of our paper published in 2018 IEEE International Symposium on Hardware Oriented Security and Trust (HOST), Washington, DC, USA, 30 April–4 May 2018; pp. 235–238.

Received: 24 July 2018; Accepted: 20 September 2018; Published: 23 September 2018

Abstract: We present Value Prediction for Security (`VPsec`), a novel hardware-only framework to counter fault attacks in modern microprocessors, while preserving the performance benefits of Value Prediction (`VP`.) `VP` 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. Instances of `VPsec` leverage the state-of-the-art Value Predictors in an embodiment and system design to mitigate fault attacks in modern microprocessors. Specifically, `VPsec` implementations re-architect any baseline `VP` embodiment with fault detection logic and reaction logic to mitigate fault attacks to both the datapath and the value predictor itself. `VPsec` also defines a new mode of execution in which the predicted value is trusted rather than the produced value. From a microarchitectural design perspective, `VPsec` requires minimal hardware changes (*negligible area and complexity impact*) with respect to a baseline that supports VP, it has no software overheads (*no increase in memory footprint or execution time*), and it retains most of the performance benefits of `VP` under realistic attacks. Our evaluation of `VPsec` demonstrates its efficacy in countering fault attacks, as well as its ability to retain the performance benefits of VP on cryptographic workloads, such as OpenSSL, and non-cryptographic workloads, such as SPEC CPU 2006/2017.

Keywords: modern microprocessors; value prediction; performance; fault attack; fault mitigation

1. Introduction

Galathy [1] and Yuce et al. [2–4] analyzed and demonstrated the application of biased faults to modern, pipelined microprocessors. In this model, an attacker can engineer and inject biased faults (e.g., via clock glitches of a certain intensity) that affect in-flight instructions in the pipeline. The intensity of a fault (e.g., the length of a clock glitch, or the intensity of an electromagnetic beam with EM-FI Transient Probes: https://www.riscure.com/uploads/2017/07/datasheet_em-fi_transient_probe.pdf.) is set such that it affects a few, but not all the instructions in-flight in the pipeline, such as instructions with long critical paths - load instructions, to make the fault observable to the attacker. The effect of biased faults on loaded or computed values enables the attacker to flip bit(s) in the data bytes, where the number of bits flipped is correlated with the intensity of the fault. The correlation of the fault intensity to a byte hamming weight exists, but the attacker has no control over the faulty value (i.e., the attacker cannot set/specify the faulty value). To engineer the fault, an attacker can gain an understanding of the critical path of instructions via profiling with micro-benchmarking. Profiling with micro-benchmarking is a common, reliable and inexpensive practice, which jointly with the information of software maps on the microprocessor pipeline provides sufficient information for an attacker to engineer a short and effective sequence of faults to the data in the processor datapath.

This ultimately allows the attacker to perform: key extraction from cryptographic implementations (Figure 1a), via Differential Fault Analysis [5], or the most sophisticated Differential Fault Intensity Analysis [1]); access control circumvention (Figure 1b), or control flow subversion (Figure 1c), 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 [6]. 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. A class of modernIoT devices embody multicore processors with secure execution environment and vector units, e.g., ARM M7. 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].

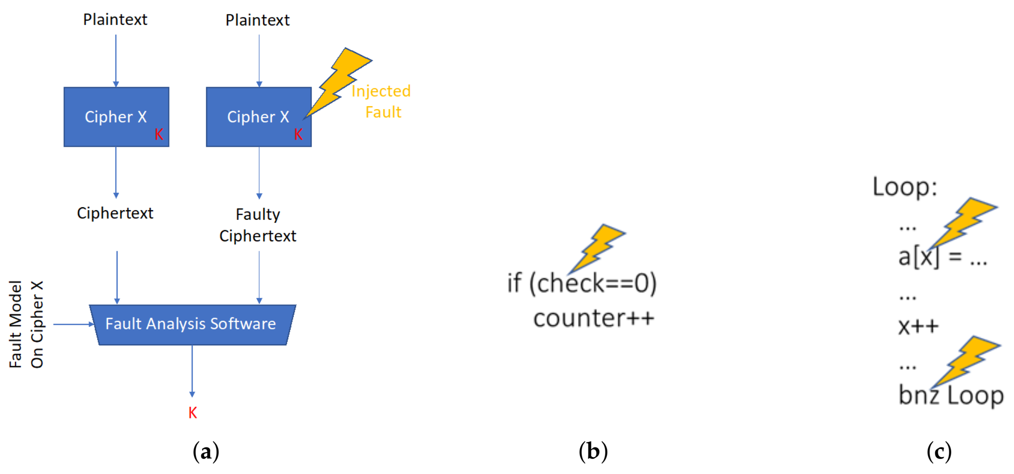


Figure 1. Fault Attack Targets: (**a**) Fault Observation and Key Extraction. (**b**) Access Control Bypass. (**c**) Data Corruption, or Premature Loop Termination.

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. Figure 2 illustrates the value prediction operations. Figure 2a shows a dependence chain consisting of four instructions (i1, i2, i3, and i4). The value produced by instruction i1 is consumed by instruction i2, value produced by instruction i2 is consumed by instruction i3, and so on. In the absence of value prediction, the four instructions execute sequentially (i.e., execution rate is 1 instructions-per-cycle). With value prediction, the data dependencies between the instructions can be broken, and the execution rate increases (from 1 to 4 instructions-per-cycle), as illustrated in Figure 2b.

Motivated by design simplicity and very high prediction accuracy (Accuracy is defined as the number of correctly predicted dynamic instructions divided by the number of predicted dynamic instructions) achieved by state-of-the-art value predictors (above 99%) [8–11], it is commonplace to use 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 (illustrated in Figure 2c). The high prediction accuracy and coverage (Coverage is defined as the number of predicted dynamic instructions divided by the number of dynamic instructions) of state-of-art value predictor designs enable the adoption of value prediction in real products.

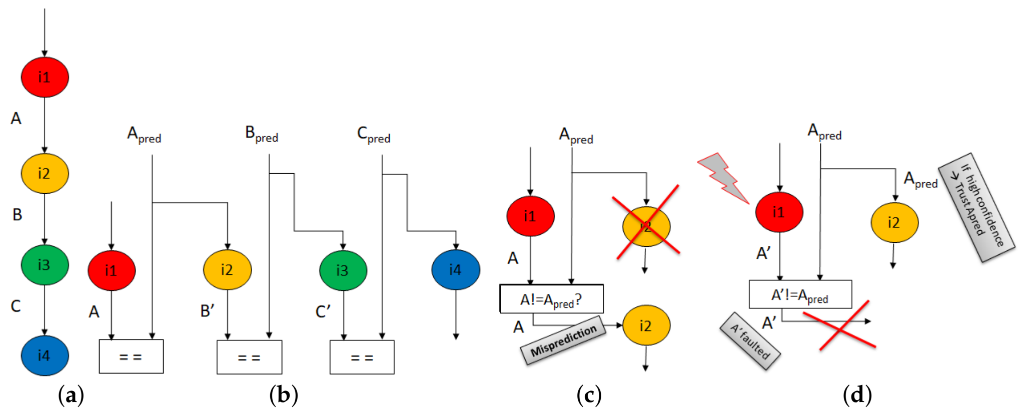


Figure 2. Value prediction operations. (**a**) Base; (**b**) Correct value prediction; (**c**) Incorrect value prediction; (**d**) Reverse trust model (VPsec).

Value prediction has appealing features that can be leveraged for security purposes to recover from fault attacks when computed or loaded data (*a.k.a. produced 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 [1–3], in which an attacker can engineer a series of biased faults on produced data values, a value predictor can effectively prevent the attacker from observing faulty output values. 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 *trusted*, can be used in place of the faulty value, thus, the fault is corrected (illustrated in Figure 2d); (b) otherwise, the producer instruction along with all younger instructions are flushed, and then re-fetched and re-executed. Thus, the correct value is reproduced, and the fault is corrected (similar to Figure 2c).

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 [8–10], 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, or the software outcome is silenced. Thus, `VPsec` instances can be defined to deceive the attacker without requiring the costly mitigation techniques in software. Interestingly, since mitigation techniques in software can increase the attack surface, because more instructions are executed, a hardware-only solution like `VPsec` avoids such undesirable side-effect, and it is friendly to legacy software.

This work extends our previous contributions [12,13], in which we presented the first hardware-only fault mitigation approach that leverages a high-performance microarchitecture feature-value prediction, that allows fault mitigation without degrading performance, and with negligible area and power overheads. This contribution extends the discussions on Value Prediction and Fault Analysis in modern microprocessors, and presents more detailed experimental results for a wide variety of benchmark suites, including cryptographic and non-cryptographic applications. Our detailed evaluation shows that the proposed technique protects the execution of unmitigated cipher suites in `OpenSSL` [14], the industry standard benchmarks `SPEC CPU2006` [15] and `SPEC CPU2017` [16], 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 and Related Art

2.1. Value Prediction

Since the introduction of value prediction in the 90s [17,18], significant improvements have been made to make value predictors more accurate and amenable to adoption in production hardware. In general, *conventional* 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 [17,19] 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 relies on identifying patterns in the history of a given static instruction to predict the value. Finite Context Method predictors (FCM) [20,21] 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 [9], D-VTAGE [10], and EVES (Extended VTAGE Extended Stride) [11]. 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, tag-less last-value predictor (LVP). D-VTAGE augments VTAGE with a last value table (LVT) that is located before the first VTAGE table (VT0). LVT stores the last value (per instruction), while the VTAGE tables store the strides/deltas. EVES predictor (EVES predictor wonall tracks of the first championship value prediction (CVP-1) [22]) tunes VTAGE and augments it with a stride value predictor. We refer to such predictors as conventional value predictors in the text.

Another interesting class of value predictors advocates for predicting values indirectly via memory address prediction [8,23]. 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 [8–11] 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.

Remark 1. *It was observed in [8], and confirmed in our experiments with VPsec, that the instructions being predicted by the two value predictor families (namely,* conventional *and* indirect *predictors) can have significant overlap in some workloads, and very little to no overlap in other workloads. This observation is leveraged by VPsec to counter fault attacks.*

2.2. Fault Analysis

In this section, as examples, we use traditional attacks to the AES encryption and Standard RSA digital signature to explain the fault attack process and analysis. These are representative examples, even though specificity of faults attack and analysis is dependent on the fault model (in this manuscript we essentially consider random byte faults) and the cipher.

The AES algorithm operates on 128-bit blocks of plaintext, and produces an output ciphertext of 128-bit. The input block is arranged into a 4×4 state matrix, each entry representing one byte of the 128 bits, and the algorithms consists of **r** rounds, each round modifies the state matrix and the output of round **r** is the ciphertext (FIPS-197, Advanced Encryption Standard (AES):https://nvlpubs.nist.gov/nistpubs/fips/nist.fips.197.pdf). The first $\mathbf{r}-\mathbf{1}$ rounds have four main operations, SubByte, ShiftRows, MixColumns, and AddRoundKey. The last round omits the MixColumns operation. The number of rounds depends on the key size. For example, the standards define $\mathbf{r} = \mathbf{10}$ for a key size of 128 bits.

Differential Fault Analysis (DFA), as originally introduced by Biham and Shamir in [5], obtains the information of the secret key based on the fault-free ciphertext and the faulty ciphertext under a certain fault model (refer to Figure 1a, in which Cipher X is AES.) The objective of an attacker performing DFA on AES is to recover enough information on the last round key to be able to mount an exhaustive search to recover the AES key. In general, not all vulnerable locations in a block cipher are equal from a fault attack perspective [24]. In the basic attack, the principle is to induce a fault resulting in a differential of one byte at the input of the last MixColumns. By guessing four bytes of the last round key, the attacker tests if the corresponding differential at the output of the last MixColumns corresponds to a one-byte differential at its input. If the fault is injected in the the first column of this state then the faulty ciphertext will differ from the correct ciphertext in bytes at the positions 0, 7, 10, and 13. The four byte differential results is then compared with the 1020 (4×255) elements contained in the list of all possible differentials at the output of one column of the MixColumns transformations, assuming one-byte difference at its input.

Standard RSA digital signatures involves a modular exponentiation with the signer private key on a fixed size input (the hash of the message to sign). To perform fault attacks on standard RSA signature it is necessary to gather a sufficient number of message/faulty signature pairs, by inducing faults on the register that contains an intermediate value.

The application of a fault at a certain stage in the computation allows to use the verification operation to guess the portion of the exponent from the application of the fault onward in a left-to-right implementation of modular exponentiation. The whole exponent is gradually recovered from the most to the least significant bit by repeating the previous analysis on different faulty signatures. In each analysis, several bits from the exponent are recovered.

Remark 2. *In both examples (AES encryption and Standard RSA digital signature,) a certain cipher implementation needs to execute for a sufficient number of cycles (until the 8th round of AES encryption,) or times (for Standard RSA digital signature,) to allow key recovery after fault injection. Specifically, in the case of AES, the cipher executes fault-free until the MixColumns in rounds $r-2$. As it is illustrated and remarked later on, the fault-free execution of AES until the point of fault is sufficient to train a value predictor embodiment to perform confident prediction, hence* VPsec *is capable of masquerading the faulty value upon the occurrence of a fault.*

In the case of Standard RSA signature, key recovery via fault-attack is an incremental process in which only a small portion of the exponent can be recovered. Indeed, the application of faults on intermediate results during Standard RSA computation will not return faulty signatures after the value prediction embodiment is trained to produce confident predictions.

2.3. Mitigations Against Fault Attacks

Both software (primarily) and hardware-based mitigation techniques against fault attacks have been studied in the literature. Hardware-based mitigation techniques usually duplicate a portion of the hardware blocks (e.g., registers) [3], 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 mitigation techniques provide more flexibility and portability as they do not require any underlying security hooks in the hardware. Prior art on software-based mitigation techniques proposes algorithm level [25,26] and instruction level [27,28] mitigation to hinder consistent fault injection. Algorithm level mitigation 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, mitigation techniques operate at a much finer granularity. Such mitigation techniques 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., [27] reports a 3.4× 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 mitigation techniques are insufficient [3,4,29]. 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 [29] 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 [3]. In [30], 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 [3]. The proposed VPsec [12,13] requires minimal changes to the hardware design of the value prediction machinery. By virtue of being a hardware-only solution, it avoids the high overheads associated with the deployment of software mitigation while retaining most of the benefits of value prediction. Finally, as an attacker will target instructions [3] as well as data values, the fact that VPsec does not require the deployment of software mitigation such as instruction duplication and sanity check [27] reduces the number of possible attacks.

2.4. 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 [31,32]. 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 mitigation techniques 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 [31,32] 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 possible fault attack scenarios, i.e., faults to produced data values, and faults to predicted data values. In the `VPsec` framework, the concept of trust in the predicted value is introduced. A predicted value is trusted if and only if two or more of the value predictors in the value prediction embodiment supply matching, confident predictions. `VPsec` implements a pipeline which includes the following components (refer to Figure 3): (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 is trusted, i.e., it 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 (We use 8-bit, instead of 1-bit, PSRs to protect the PSRs against fault attacks) 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 is trusted, 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 to zero, indicating a highly abnormal and irreversible scenario, `VPsec` generates an exception and takes an action to misguide the attacker. For example, the taken action can be defined as to infect the computed value with a random number, and propagate the infected value through the pipeline as illustrated in Figure 3. In this case, 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. However Reaction 3 is defined, a corrective action is delegated to the firmware handling the exception or to the higher-software layers. 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.

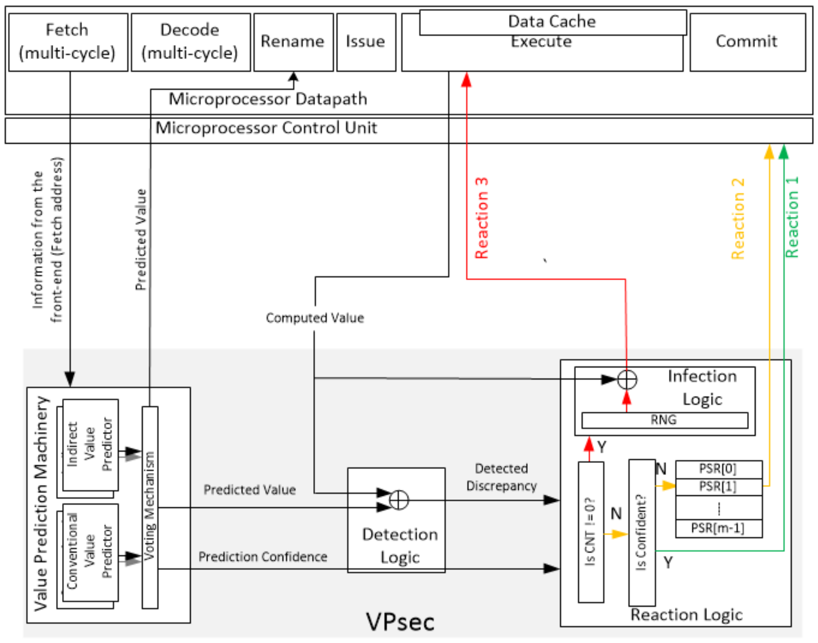


Figure 3. VPsec high-level diagram. CNT represents the PSR value for the value predicted instruction.

3.2. Design

The `VPsec` design 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. In this Section, we describe a specific instance of `VPsec`, as illustrated in Figure 3.

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 [17], context-based value predictors [9–11], and indirect value predictors [8,23]. All predictors are active simultaneously, attempting to predict the data values produced by executing instructions.

In such designs, 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. Reaction 3 is defined to infect the corrupted value.

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 combinational 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 accuracy is monitored and recorded for each predictor. Similarly, during the execution mode, the accuracy is monitored and compared against the accuracy recorded in the training mode for each predictor. 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 (on 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 [3], 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 [33]. 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.

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

Component	**Configuration**
Branch Prediction	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 Issue through Commit Width	4 instr./cycle 9 instr./cycle (9 execution lanes: 3 support load-store operations, and 6 generic)
ROB/IQ/LDQ/STQ Physical RF	224/97/72/56 (modeled after Intel Skylake) 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 2. Evaluation Methodology.

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		

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. Such phases are usually of little to no interest to the attacker.

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` [16], `SPEC CPU2006` [15], `OpenSSL` [14], `SPMV` [34], and `Terasort`. Our evaluation demonstrates the accuracy, coverage, and confidence of the proposed value prediction scheme. Moreover, we demonstrate the effect of injecting faults to cover the different attack scenarios described earlier.

Table 3 shows a list of our benchmarks. 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 [35], except for short-running benchmarks, we simulate the first 100 million instructions, or until the benchmark completes.

Table 3. Applications used in our evaluation.

Benchmark Suite	Applications
OpenSSL	aes128, aes256, rsa, sha1, sha256
Terasort	multiple input datasets
SPMV	csc, csr
SPEC2K6-FP	milc, namd, dealII, soplex, povray, lbm, sphinx3
SPEC2K6-INT	perlbench, bzip2, gcc, mcf, gobmk, hmmer, sjeng, libquantum, h264ref, omnetpp, astar, xalanbmk
SPEC2017-INT	perlbench, gcc, mcf, omnetpp, xalanbmk, x264, deepsjeng, leela, exchange2, xz

4.3. Value Prediction

In this section, we evaluate the value prediction scheme described in Section 3.2 using the workloads listed earlier. Figure 4a 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 the figure, the prediction accuracy of each one of the used value predictors is well above 99% [8–11].

4.4. VPsec

Figure 4b 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 4a) 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.

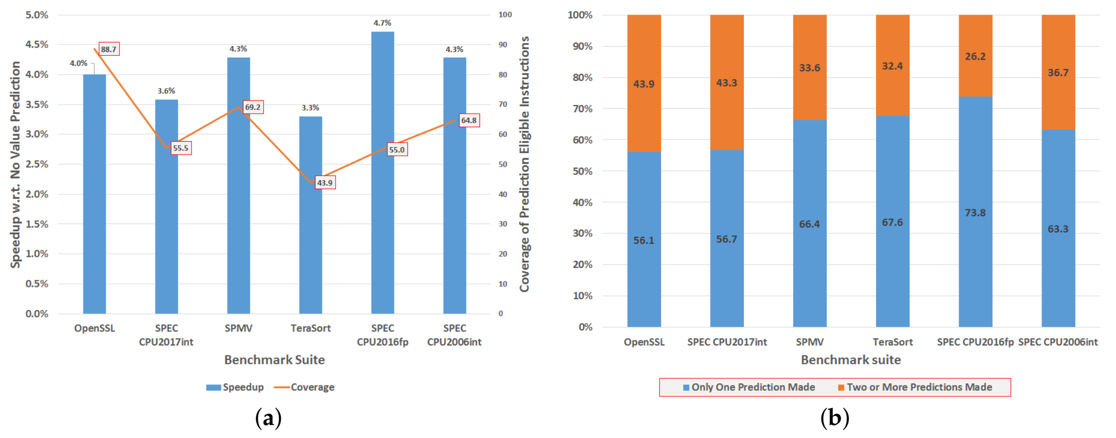
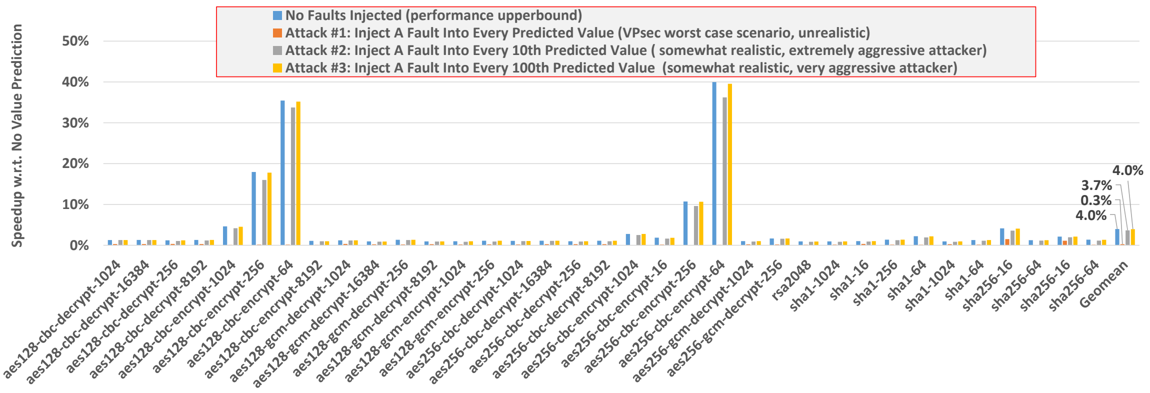


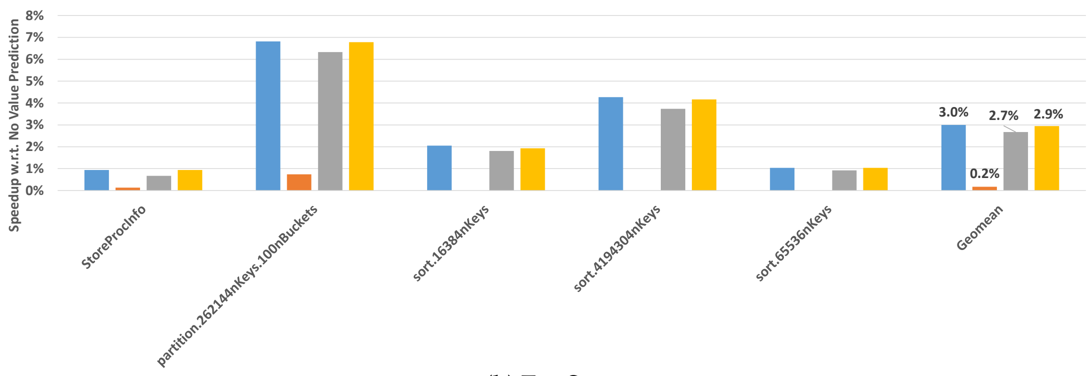
Figure 4. (**a**) Speedup (bars, primary y-axis) and coverage (line, secondary y-axis) of the value prediction scheme. The accuracy of each predictor is over 99% (not shown). (**b**) Breakdown of value predictions based on the number of predictions made.

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.

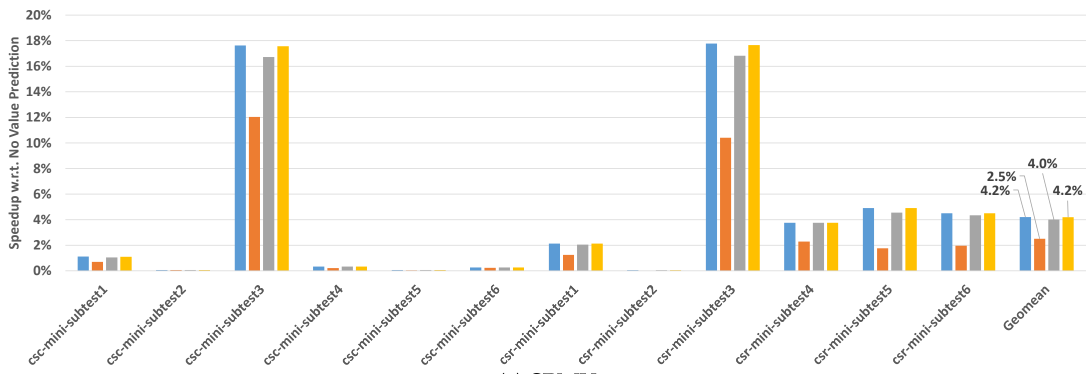
To evaluate the performance impact due to the execution of Reaction 2, we assume the following extreme attack scenarios, in which an attacker faults: each value predicted load (Attack #1), every 10th predicted load (Attack #2), and every 100th predicted load (Attack #3). The attacker can inject biased faults in loaded and computed values. In the evaluation we assume injection of biased faults in the loaded values, as loaded values have the largest critical section. [4] Figures 5 and 6 show the performance impact with respect to a baseline with no value prediction (and no attacks). Table 4 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% in `IPC`, with an average of 4%.



(**a**) OpenSSL

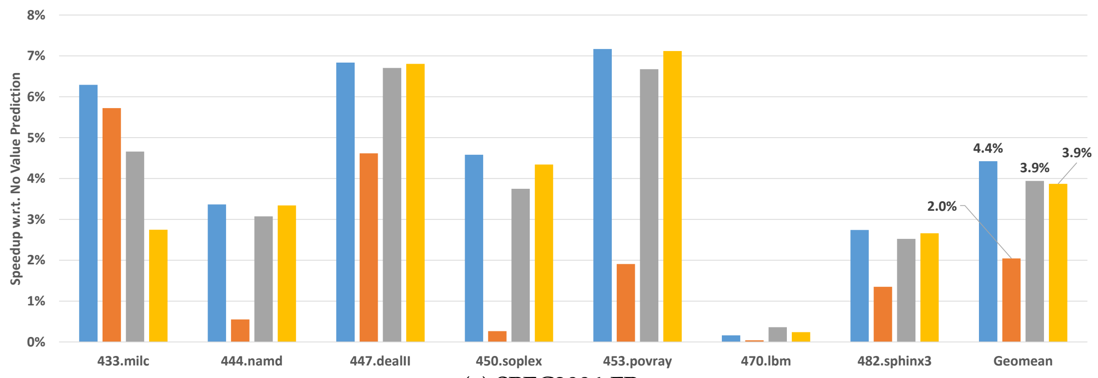


(**b**) TeraSort

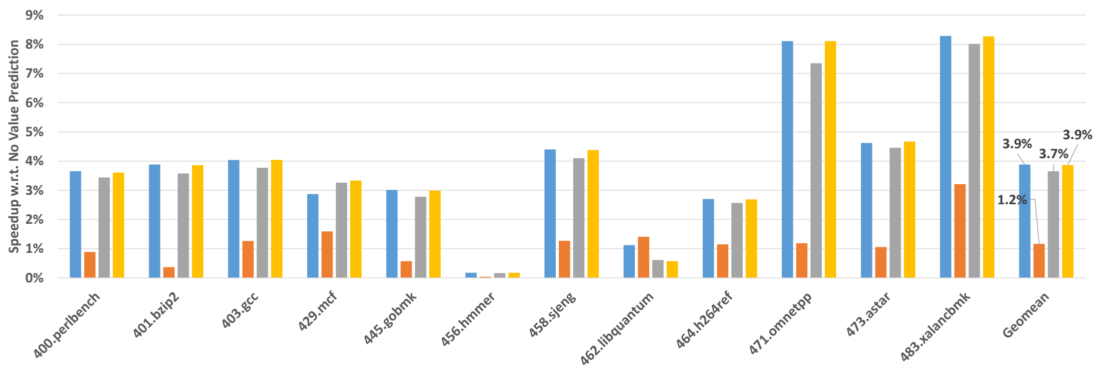


(**c**) SPMV

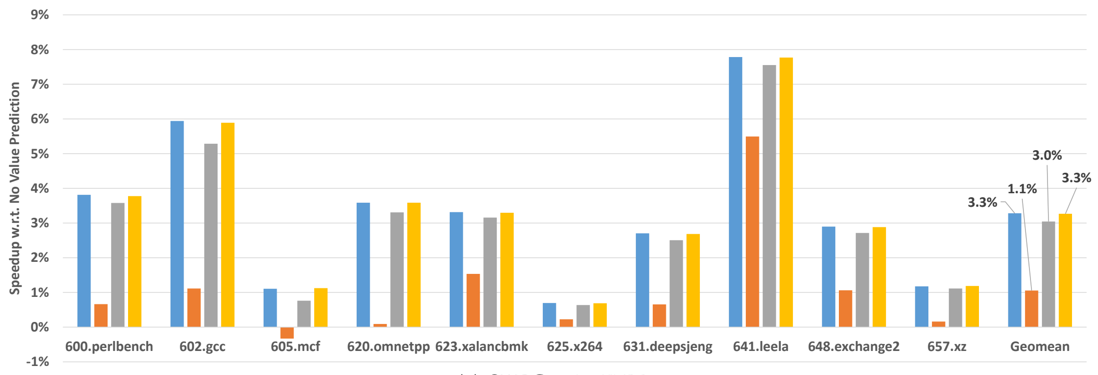
Figure 5. VPsec: evaluation of different attack scenarios on non-SPEC workloads.



(**a**) SPEC2006-FP



(**b**) SPEC2006-INT



(**c**) SPEC2017-INT

Figure 6. VPsec: evaluation of different attack scenarios on SPEC workloads. Same legend as Figure 5a.

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 Figures 5 and 6, the workloads still exhibit performance improvements which nearly match the performance improvement achieved by the no-attack scenario. Similar results can be observed for the other workloads, Terasort and SPMV in Figure 5b,c and SPEC CPU in Figure 6.

`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).

Table 4. VPsec: percentage average, minimum, and maximum speedups. (avg/[min, max]).

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]
`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 CPU2006fp`	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 CPU2006int`	3.9/[0.2, 8.3]	1.2/[0.0, 3.2]	3.7/[0.2, 8.0]	3.9/[0.2, 8.3]
`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]

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 [9]. Alternative proposals [8] 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)(https://nvlpubs.nist.gov/nistpubs/fips/nist.fips.197.pdf) exhibit both forms of predictability (address and value). The main loop of an AES implementation iterates for several 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.

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 [8] 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 a few limitations, and can be improved. First, it does not address the situation of fault attacks in earlier stages of the pipeline to skip instructions. This is inherent in that `VPsec` leverages value prediction. Second, 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)(https://www.globalplatform.org/mediaguidetee.asp), 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 life-cycle.

Third, 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 4b). 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 instance of `VPsec` that we evaluated in this paper, and that it do not jeopardize the validity of the concepts, findings and conclusions presented.

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 (ARM TrustZone: https://developer.arm.com/technologies/trustzone). 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 [36].

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 [3] 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 5. Cases (1), (2) and (4) are handled properly by VPsec in these 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 3 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.

Table 5. VPsec Operating Scenarios.

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

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 accuracy for each predictor 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, VPsec did not incur Reaction 3 (Recall that Reaction 3 is incurred for low confidence yet incorrect predictions, a scenario that is even more likely than high confidence yet incorrect prediction, i.e., Case 3).

5.3. Relevance to Recently Discovered Attacks

The Spectre attack appeared in two variants [36]. 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 [36].

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 microarchitectural 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.

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.

Author Contributions: The authors equally contributed to the architecture design of VPsec. R.S. developed VPsec in the simulation environment, and evaluated it. He elaborated on all the details on Value Prediction in the manuscript. R.C. had the original idea of the VPsec framework. He developed a security architecture centered around Value Prediction. He elaborated on all the details on fault analysis in the manuscript.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Ghalaty, N.F.; Yuce, B.; Taha, M.; Schaumont, P. Differential Fault Intensity Analysis. In Proceedings of the 2014 Workshop on Fault Diagnosis and Tolerance in Cryptography, Busan, Korea, 23 September 2014.
2. Yuce, B.; Ghalaty, N.F.; Schaumont, P. Improving Fault Attacks on Embedded Software Using RISC Pipeline Characterization. In Proceedings of the 2015 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC), St. Malo, France, 13 September 2015.
3. Yuce, B.; Ghalaty, N.F.; Santapuri, H.; Deshpande, C.; Patrick, C.; Schaumont, P. Software Fault Resistance is Futile: Effective Single-Glitch Attacks. In Proceedings of the 2016 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC), Santa Barbara, CA, USA, 16 August 2016.
4. Yuce, B.; Ghalaty, N.F.; Deshpande, C.; Santapuri, H.; Patrick, C.; Nazhandali, L.; Schaumont, P. Analyzing the Fault Injection Sensitivity of Secure Embedded Software. *ACM Trans. Embed. Comput. Syst.* **2017**, *16*, 95:1–95:25. [CrossRef]
5. Biham, E.; Shamir, A. Differential fault analysis of secret key cryptosystems. In *Annual International Cryptology Conference*; Springer: Berlin/Heidelberg, Germany, 1997.
6. Semiconductor Research Corporation. 2017 Research Opportunities, an Industry Vision and Guide: Security and Privacy. 2017. Available online: https://www.semiconductors.org/clientuploads/Research_Technology/SIA%20SRC%20Vision%20Report%203.30.17.pdf (accessed on 23 September 2018).
7. Bar-El, H.; Choukri, H.; Naccache, D.; Tunstall, M.; Whelan, C. The Sorcerer's Apprentice Guide to Fault Attacks. *Proc. IEEE* **2006**, *94*, 370–382. [CrossRef]
8. Sheikh, R.; Cain, H.W.; Damodaran, R. 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), Cambridge, MA, USA, 14–18 October 2017.
9. Perais, A.; Seznec, A. Practical data value speculation for future high-end processors. In Proceedings of the 20th IEEE International Symposium on High Performance Computer Architecture (HPCA), Orlando, FL, USA, 15–19 February 2014.
10. Perais, A.; Seznec, A. BeBoP: A cost effective predictor infrastructure for superscalar value prediction. In Proceedings of the 2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA), Burlingame, CA, USA, 7–11 February 2015; pp. 13–25.
11. Seznec, A. *Exploring Value Prediction with the EVES Predictor*; First Championship Value Prediction, CVP; ACM Los Angeles, CA, USA, 2018.
12. Sheikh, R.; Cammarota, R.; Ruan, W. Value prediction for security (VPsec): Countering fault attacks in modern microprocessors. In Proceedings of the 2018 IEEE International Symposium on Hardware Oriented Security and Trust (HOST), Washington, DC, USA, 30 April–4 May 2018.
13. Cammarota, R.; Sheikh, R. VPsec: Countering Fault Attacks in General Purpose Microprocessors with Value Prediction. In Proceedings of the 15th ACM International Conference on Computing Frontiers, Ischia, Italy, 8–10 May 2018.
14. OpenSSL, Cryptography and SSL/TLS Toolkit. Available online: http://www.openssl.org (accessed on 23 September 2018).
15. Standard Performance Evaluation Corporation. The SPEC CPU 2006 Benchmark Suite. 2006. Available online: https://www.spec.org/cpu2006/ (accessed on 23 September 2018).
16. Standard Performance Evaluation Corporation. The SPEC CPU 2017 Benchmark Suite. 2017. Available online: https://www.spec.org/cpu2017/ (accessed on 23 September 2018).
17. Mendelson, A.; Gabbay, F. *Speculative Execution Based on Value Prediction*; Technical Report; Technion: Haifa, Israel, 1996.
18. Lipasti, M.H.; Wilkerson, C.B.; Shen, J.P. Value Locality and Load Value Prediction. *ACM SIGPLAN Not.* **1996**, *31*, 138–147. [CrossRef]
19. Eickemeyer, R.J.; Vassiliadis, S. A load-instruction unit for pipelined processors. *IBM J. Res. Dev.* **1993**, *37*, 547–564. [CrossRef]
20. Sazeides, Y.; Smith, J.E. *Implementations of Context-Based Value Predictors;* Technical Report; University of Wisconsin-Madison: Madison, WI, USA, 1997.
21. Sazeides, Y.; Smith, J.E. The Predictability of Data Values. In Proceedings of the Thirtieth Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), Research Triangle Park, NC, USA, 1–3 December 1997.

22. First Championship Value Prediction. 2018. Available online: https://www.microarch.org/cvp1 (accessed on 23 September 2018).
23. González, J.; González, A. Speculative Execution via Address Prediction and Data Prefetching. In Proceedings of the 11th International Conference on Supercomputing (ICS), Vienna, Austria, 7–11 July 1997.
24. Li, Y.; Gomisawa, S.; Sakiyama, K.; Ohta, K. An Information Theoretic Perspective on the Differential Fault Analysis against AES. Cryptology ePrint Archive. 2010. Available online: https://eprint.iacr.org/2010/032 (accessed on 23 September 2018).
25. Karri, R.; Kuznetsov, G.; Goessel, M. Parity-Based Concurrent Error Detection of Substitution-Permutation Network Block Ciphers. In *International Workshop on Cryptographic Hardware and Embedded Systems*; Springer: Berlin/Heidelberg, Germany, 2003; pp. 113–124.
26. Medwed, M.; Schmidt, J.M. A Generic Fault Countermeasure Providing Data and Program Flow Integrity. In Proceedings of the 5th Workshop on Fault Diagnosis and Tolerance in Cryptography, Washington, DC, USA, 10 August 2008.
27. Barenghi, A.; Breveglieri, L.; Koren, I.; Pelosi, G.; Regazzoni, F. Countermeasures against fault attacks on software implemented AES: effectiveness and cost. In Proceedings of the 5th Workshop on Embedded Systems Security (WESS), Scottsdale, Arizona, 24 October 2010.
28. Patrick, C.; Yuce, B.; Ghalaty, N.F.; Schaumont, P. Lightweight Fault Attack Resistance in Software Using Intra-Instruction Redundancy. In *International Conference on Selected Areas in Cryptography*; Springer: Cham, Switzerland, 2016.
29. Chen, Z.; Shen, J.; Nicolau, A.; Veidenbaum, A.; Ghalaty, N.F.; Cammarota, R. CAMFAS: A Compiler Approach to Mitigate Fault Attacks via Enhanced SIMDization. In Proceedings of the 2017 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC), Taipei, Taiwan, 25 September 2017.
30. Proy, J.; Heydemann, K.; Berzati, A.; Cohen, A. Compiler-Assisted Loop Hardening Against Fault Attacks. *ACM Trans. Archit. Code Optim.* **2017**, *14*, 36:1–36:25. [CrossRef]
31. Li, X.; Yeung, D. Exploiting Value Prediction for Fault Tolerance. In Proceedings of the 3rd Workshop on Dependable Architectures, Lake Como, Italy, 8 November 2008.
32. Pomeranz, I.; Vijaykumar, T.N. FaultHound: value-locality-based soft-fault tolerance. In Proceedings of the 42nd Annual International Symposium on Computer Architecture (ISCA), Portland, OR, USA, 13–17 June 2015.
33. Doweck, J.; Kao, W.F.; Lu, A.K.Y.; Mandelblat, J.; Rahatekar, A.; Rappoport, L.; Yoaz, A. Inside 6th-Generation Intel Core: New Microarchitecture Code-Named Skylake. *IEEE Micro* **2017**, *37*, 52–62. [CrossRef]
34. SpMV Benchmark. Available online: http://bebop.cs.berkeley.edu/spmvbench/ (accessed on 23 September 2018).
35. Perelman, E.; Hamerly, G.; Calder, B. Picking Statistically Valid and Early Simulation Points. In Proceedings of the 12th International Conference on Parallel Architectures and Compilation Techniques (PACT), New Orleans, LA, USA, 27 September–1 October 2003.
36. Kocher, P.; Genkin, D.; Gruss, D.; Haas, W.; Hamburg, M.; Lipp, M.; Yarom, Y. Spectre Attacks: Exploiting Speculative Execution. *arXiv* **2018**, arXiv:1801.01203.

© 2018 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (http://creativecommons.org/licenses/by/4.0/).