Opportunities and Challenges: Hardware Vulnerability descriptions with Hybrid Logic.

William Casey
wcasey@usna.edu

ABSTRACT

Hardware exploits such as Spectre and Meltdown underscore the increasing need to validate hardware security properties. To account for known exploits we suggest Hybrid logic, an extension of Modal logic, for its concise description of security problems. We further suggest that compiling known exploits into a database would be a helpful tool for testing designs against security properties. We demonstrate a hybrid logic description for Spectre and describe its benefits. We reflect further on the unique challenges of sharing security information about hardware vulnerabilities. We conclude that careful thought is required to achieve responsible disclosure processes for hardware vulnerabilities.

CCS CONCEPTS

• **Security and privacy → Logic and verification**; **Hardware attacks and countermeasures**.

KEYWORDS

Hybrid Logic, Hardware Verification, Vulnerability Management

1 INTRODUCTION

Recent exploits against hardware systems, such as Spectre and Meltdown, have demonstrated the increased need to validate hardware security properties, such as *Non-Interference*[1] (see [5]). These vulnerabilities are complex and subtle and require a comprehensive analysis of nonstandard combinations of hardware and system software tools, unusual use cases, and critically, the side effects they cause (see [6]). Then we are concerned with the lifespan and measurability of any side effects. But how can this be done adequately and safely? We argue that approaches to enumerate hardware exploits within a hybrid logic may prove to be a useful step for hardware designers to enhance security. At the same time, hardware design processes presents distinct challenges for a *responsible disclosure process*, which is difficult to manage even in software projects with low cost for patching.

2 HYBRID LOGIC DESCRIPTIONS OF EXPLOITS

Hybrid logic is similar to but more expressive than Modal logic on Kripke Frames (see [2],[3]). Additional expressivity is acheived with *nominals* the naming of Possible worlds, and then *binding* or using the named worlds within modal logic formulae. This turns out to be rather expressive and concise, as we will next demonstrate on the Spectre style exploit. As in other common approaches to Hardware Verification such as bounded model checking (see [1]), Possible worlds W will include a discrete set comprised of all possible architectural states (registers and their values), along with the relations $\mathcal{R}_k$, indexed by k, to represent reachability withing a k clock cycle transition. That is $w\mathcal{R}_k v$ if and only if there is a k step ISA program which reaches architectural state v in k steps starting from w.

Lets reminding ourselves of the key sequence of Events in Spectre/Meltdown Exploits

(1) Exploitation Code Execution: User code that miss-trains a branch predictor, and then induces a speculative execution path.
(2) Speculative Access: The processor, running user code speculatively, accesses and reads protected kernel memory whose value determines a unique choice of several *cold memory* user object to reload, inducing a cache load side effect. The speculative execution is rejected and no exception is handled. However long the execution window, the effects on cache will have a longer lifespan, and are observable via timing measurements.
(3) Timing Measurement: Quickly, the user code measures how long loads take for various user objects in memory, Only one is *hot* and in chache, providing the user indirect means to infers kernel data values.

Propositions offer behavioral abstraction as formula: $E(w)$ an attacker executes in user space the exploitation code at world w. $S(w)$ that a speculative access (described in step 2 above) occurs at world w. $T(w)$ that timing measurements are conducted at world w.

The exploit sequence above can be described by the existence of worlds labeled w_1, w_2, w_3, so that $w_1\mathcal{R}_k w_2\mathcal{R}_k w_3 \wedge E(w_1) \wedge S(w_2) \wedge T(w_3)$. Hybrid logic formula, using temporal operators for possibility (ie $\Diamond$ over $\mathcal{R}_k$) can state the same, while giving names to intermediate worlds and nominal binding[2].

$$Q = \Diamond \downarrow w_1.@_{w_1} E(w_1) \Diamond \downarrow w_2.@_{w_2} S(w_2) \Diamond \downarrow w_3.@_{w_3} T(w_3).$$

Since $Q \rightarrow interference$, A formal verification might seek to ensure $\mathcal{M} \vDash \neg Q$, as partial evidence of security[3]. In contrast, simulation and Hybrid methods such as black-box fuzz test or concolic execution may also determine an execution sequence that satisfies Q, and identifies various nominal (or named worlds) along the way.

2.1 Benefits

Hybrid logic seems particularly well suited for organizing search goals around known exploitation patterns. For example, if nominal w_1 can be found, that is $Q_1 = \Diamond \downarrow w_1.@_{w_1} E(w_1)$ satisfied. An architectural check point state can be used to focus search on a

[1] Non-Interference rules out execution traces that violate access control such as users reading sensitive kernel process memory

[2] $\Diamond \downarrow x.@_x P$ is interpreted as true when its possible to reach a world, given the name x, where P is true. If no such P satisfying world exists, then the statement is false.

[3] Here $\mathcal{M}$ describes a Kripke model for component operation within its instruction set.

second stage goal, to determine a nominal w_2 satisfying: $Q_2 = \Diamond \downarrow w_1.@_{w_1} E(w_1) \Diamond \downarrow w_2.@_{w_2} S(w_2)$ and so on.

Hybrid logic has favorable abstraction; for example $S(w)$, can be achieved in various ways, refined logically for specific kernel address and data object loaded can be done as:

$$S(w) = \Diamond \downarrow v.@_v \left(\bigvee_{j=1}^{N} S_j(v) \right) \Diamond \downarrow w.@_w \left(\bigvee_{k=1}^{n} L_k(w) \right),$$

where $S_j(v)$ is the event that user code reads kernel memory bit j at world v, and $L_k(w)$ is the event that user object k is reloaded into cache at world w. Such logical abstraction is useful, it can mimic patterns in software often referenced from attack descriptions. For attacks, the logic of Q is also likely to be repeated within a loop, where sequential access to bits will offer behavioral qualification of the malicious behavior, which itself can be expressed in Hybrid Logic. Additionally, the correlated behavior of the kernel data (value) read and the choice of cold object reloaded can be inferred, within such a logical refinement.

Also, hybrid logic seems to provide reasoning tools for mitigation, for example, hard memory segmentation is an option to prevent Q_2 from a world w_1, and transactional structures to revert cache state may have some impact on preventing Q from world w_2. In addition, other mitigation might be identified by refining macro-behaviors. Still, another bio-inspired type of behavior modeling and detection, where a system can self-monitor, a hybrid logic proposition can act like an antibody within the immune system to trigger self-protective responses.

3 CONCLUSION, HYBRID LOGIC DATABASES OPPORTUNITIES AND CHALLENGES

Hybrid logic may form efficient and common descriptions for hardware exploits. While there are many benefits of pooling security information related to vulnerabilities, as has been done with software, caution and careful thought is required for hardware security. Hardware differs from software in several aspects, most notably for this discussion is that software is easier to patch. This aspect of software gives rise to various processes of *responsible disclosure* of vulnerability information, fraught with conflicting interests (see [4]). The basic notion is that responsibility means the following: when one finds a vulnerability in software, the discovery should be presented to the software vendor exclusively prior to wider release. This allows the vendor to design and patch software vulnerabilities beforethe information is more widely distributed via software vulnerability databases such as MITRE CVE (see [7]) and NIST (see [8]), where its public awareness can help others defend systems. However, in hardware, the difficulty to patch and the costly commitment to manufacturing processes are obstacles to achieving success from the same approach. Public disclosure of a zero-day exploit could irreparably harm a manufacturer, stressing the importance of trust in the industry. As such, there is more at stake, and some considerable thought is required to design a best practice or process to manage hardware vulnerability information. However, it seems a repertoire of hybrid logic descriptions of known hardware exploits is essential to advance security by testing and verification. Such a list can help designers assert that novel hardware components are immune to known exploits.

The dilemma of revealing hardware vulnerabilities seems particularly difficult, and security information related to hardware vulnerabilities may require careful controls within trusted communities. Various approaches to this dilemma warrant more thought and may broadly encompass various tools from governance to technology. Given the costs and impacts involved, government economic policy or risk amortization tools should be considered. From the technological side, cryptographic and steganographic concepts are a few tools that could play a role in both distribution of security information. This is an important problem that warrants further thought and consideration in the interesection of security researchers and hardware designers.

ACKNOWLEDGMENTS

The author wishes to thank Bud Mishra (Courant Institute NYU) for discussions and suggestions for Hybrid Logic applications.

REFERENCES

[1] Christel Baier and Joost-Pieter Katoen. 2008. *Principles of model checking*. MIT press.
[2] Patrick Blackburn. 2000. Representation, reasoning, and relational structures: a hybrid logic manifesto. *Logic Journal of the IGPL* 8, 3 (2000), 339–365.
[3] Patrick Blackburn, Maarten De Rijke, and Yde Venema. 2001. *Modal logic: graph. Darst.* Vol. 53. Cambridge University Press.
[4] Hasan Cavusoglu, Huseyin Cavusoglu, and Srinivasan Raghunathan. 2005. Emerging Issues in Responsible Vulnerability Disclosure.. In *WEIS*.
[5] Michael R Clarkson and Fred B Schneider. 2010. Hyperproperties. *Journal of Computer Security* 18, 6 (2010), 1157–1210.
[6] Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, et al. 2020. Spectre attacks: Exploiting speculative execution. *Commun. ACM* 63, 7 (2020), 93–101.
[7] MITRE Corporation. [n. d.]. Common Vulnerabilities and Exposures (CVE). https://www.cve.org. Accessed: July 14, 2025.
[8] National Institute of Standards and Technology (NIST). [n. d.]. National Vulnerability Database (NVD). https://www.nist.gov/programs-projects/national-vulnerability-database-nvd. Accessed: July 14, 2025.