A New Doctrine for Hardware Security

Adam Hastings
hastings@cs.columbia.edu
Columbia University
New York, New York

Simha Sethumadhavan
simha@cs.columbia.edu
Columbia University
New York, New York

ABSTRACT

In recent years, high-profile hardware attacks have brought attention to the importance of and inadequate state of hardware security. Hardware security remains an elusive challenge because like other areas of security, it is an abnormal good where the economic laws of the free market fail to produce optimal outcomes. Correcting such marketplace failures is generally the role of government or other regulatory agencies and has been proposed and even implemented in some areas of security. However, little if no comparable work has been done in the realm of computer hardware. One explanation for this is that unlike other areas of security, we lack a comprehensive intellectual framework for discussing and reasoning about hardware security. We find that previous doctrines of security either do not apply or offer an incomplete perspective in this domain. We propose a new doctrine of hardware security based on the idea that achieving security is a burden, and that this burden must be shared between all the players in the game of security. Our doctrine serves as a tool for conceptualizing and understanding how hardware security should be regulated and administered.

KEYWORDS

hardware security, mandated security, economics of security, Spectre, Meltdown, Rowhammer

1 INTRODUCTION

Once niche and arcane, the field of hardware security has recently become one of the most pressing issues in cybersecurity. Physical-level attacks like Rowhammer gave attackers the ability to modify a system's memory at will [1]. And microarchitectural side channel attacks like Spectre and Meltdown have shown how pervasive, dangerous, and hard-to-fix a hardware attack could be [2, 3]. Especially concerning is that these problems, while well-known and publicized, have generally not been fixed. Why?

The answer is unfortunately more complicated than simply a lack of technical solutions. Hardware security, like other types of security, is an abnormal good, where the economic laws of supply and demand do not necessarily produce the most efficient outcome. As we will demonstrate, hardware security succumbs to many of the classic market failures such as information asymmetry, free riding, misaligned incentives, and markets for lemons. We can understand the persistence of serious hardware vulnerabilities as a series of market failures, where market forces do not properly incentivize those who are able to fix the problems from doing so.

Fortunately, there are ways to fix marketplace failures: Regulations and laws, if crafted carefully, can change the incentives that allow the failures to occur in the first place. We've seen this already in other areas of security, which suffer from similar market failures. For example, the European Union's recent GDPR law is a sweeping legislation that regulates consumer data privacy [4]; California's CCPA law followed shortly thereafter and included similar provisions[5]. Legislation to better inform the public on the security risks of consumer products has also been proposed in the United Kingdom and Singapore [6, 7]. And while currently a recommendation for the private-sector and only a requirement for federal agencies, the NIST Cybersecurity Framework provides a framework that organizations can use to identify, protect, detect, respond and recover from cyber threats [8]. Proposals for codified secure software development have been proposed as well [9].

Absent from this recent flurry of legislation, however, is hardware. Due to the seriousness of the problem and the inability of the marketplace to correct itself, we should expect that hardware will soon garner the attention of legislators and policymakers. But what will this legislation look like? Who will it mandate? And what driving principles will be behind these decisions?

We can turn to a doctrine of security, or a codified set of teachings and beliefs from which all principles are derived, to inform policy and legislation. Many doctrines of security have been proposed before [10]. For example, one doctrine says that we should improve security by building systems to be bug-free, while another says that we should improve security by better catching and prosecuting the cybercriminals. As we will see, prior doctrines are fundamentally flawed when applied to the domain of hardware. The key insight is that these prior doctrines do not properly distribute and allocate the burden of security among the players in the game of security. We propose a new doctrine for hardware security, namely the Doctrine of Shared Burdens, that aims to position the intellectual discussion on hardware security as a question of who should should have to "pay" for security, and where.

The rest of the paper is organized as follows. Section 2 provides a background on what makes hardware security a unique challenge from other areas of security, and the market failures that hinder it. We present our Doctrine of Shared Burdens in Section 3. We use our doctrine to examine the shortcomings of prior doctrines of security in Section 4. We examine three case studies in Section 5, and then use our doctrine to inform us on how we should build and design future systems in Section 6. We discuss non-technical obstacles towards achieving our doctrine in Section 7 and conclude in Section 8.

2 MOTIVATION

We define hardware security to be the security of everything at and beneath the level of the hardware-software interface. For example, in a processor we consider hardware security to include the instruction set architecture (ISA) as well as all the microarchitecture that implements it. We consider physical characteristics such as input voltage, heat, and electromagnetic radiation to be in scope as well. We also include peripheral devices to be in scope, including DRAM,

graphics cards, and PCI/e cards, and their architecture and microarchitecture as well. Physical access to systems is also in scope. We do not consider hardware support for software security to be in scope until ISA support is added to support software security.

2.1 Unique Challenges in Hardware Security

Hardware security is different than other domains of security, and doctrines tailored towards more traditional views of security (like software security or system/network administration) may not apply. We highlight four unique challenges in hardware:

2.1.1 Bug Permanence. One key distinction of hardware security is that hardware bugs are much harder—if not outright impossible—to fix. Rarely can a hardware vulnerability be patched in the same way a software vulnerability can. Once a chip has been fabricated in silicon or a system design soldered onto a circuit board, it is more or less immutable. Some minor exceptions exist—microcode and firmware updates offer some legroom, but are limited in what they can repair. A partial solution to this predicament is to use Field Programmable Gate Arrays (FPGAs) where possible, which can be reprogrammed and updated "in the field". If the hardware design programmed into the FPGA is discovered to have a flaw, the design can be updated in the FPGA's reprogrammable fabric. However, most modern FPGAs make heavy use of hardened-silicon accelerators and cores, which generally are not reprogrammable like the FPGA fabric. Bugs discovered in these non-programmable components are equally as difficult to patch in the field as any non-FPGA device. In the worst case, a critical and unpatchable hardware vulnerability means that the only defense is to discard the entire device and purchase a new one without the vulnerability (if such a new device even exists).

2.1.2 High Performance Cost of Patching. Even if a vulnerability is patchable, it is likely to be more detrimental to system performance than the average software patch. Many of the most-commonly exploited software bugs have relatively low-cost fixes—a simple bounds check fixes buffer overflows while setting a pointer to null fixes use-after-free errors. Hardware patches rarely have the same luxury. Because hardware is generally designed to *not* be updated, it is not amenable for repairs in-the-field, and patches tend to not be as lightweight as in software. This means that vulnerabilities in hardware products can be especially painful, as the users will forevermore be burdened with the patch's loss of performance. For example, patches for the recent microarchitectural data sampling (MDS) hardware bugs on Intel CPUs hurt overall performance by 16% according to some estimates [11] and over 40% on certain workloads [12].

2.1.3 More Dangerous. Another important distinction in hardware security is that attacks have the potential to be more pernicious and more devastating than other areas of security. Hardware forms the foundation of any secure computation, and vulnerabilities in this domain can undermine and compromise top security in all other domains. For example, a hardware component called a Root of Trust provides mechanisms for secure boot and cryptographic key storage and access, and functions as an always-trusted kernel which extends trust to other parts of a system. Subversion or infiltration of a system's root of trust can destroy all security guarantees a system can provide. Likewise, a malicious microcode update that tampers with hardware-managed privilege levels can subvert even the most secure operating system and application software.

2.1.4 Stealthier. A fourth, equally disturbing characteristic of hardware exploits is that they can be extremely hard to detect. For example, devices can be tampered with during the manufacturing process in a threat known as supply chain attacks. This is a common concern, because device fabrication and manufacturing is often outsourced to foreign companies. If you hire a manufacturer to build your printed circuit boards (PCBs), how do you know that they didn't insert some kind of backdoor into your device? Especially if the backdoor was some tiny modification, such as a minuscule microchip embedded inside the circuitboard itself, it could be extremely hard to detect. Supply chain attacks can target the silicon inside the chips as well, via malicious modifications to the circuitry known as hardware trojans. Such supply chain attacks may only require a tiny, almost indetectable change to a system, but can have disastrous security consequences. The physical nature of hardware makes this challenge different from other areas of security. Software doesn't need to be manufactured, and its integrity can often be easily determined via a hash, but hardware doesn't have this luxury.

For these reasons, hardware exploits are somewhat of a holy grail to attackers. They have the potential to be more permanent, more dangerous, and harder to catch than traditional software- or network-based attacks. Hardware security is therefore a fundamentally different game than other types of security. Existing doctrines and paradigms may not apply. If our intent is to correct failures in the hardware security marketplace, we need a set of guidelines upon which we can craft sensible policy recommendations. A unique, tailored doctrine is required.

2.2 Market Failures in Hardware Security

A market failure is an economic situation wherein a free market fails to produce the most efficient distribution of goods or services. Market failures are a known issue in security that have been discussed in the past [13]. Despite hardware security's unique characteristics, we find that it succumbs to many of the same market failures as other areas of security. We examine four types of market failures from the perspective of hardware security.

2.2.1 Information Asymmetry/Markets for Lemons. A common failure of open and free markets is information asymmetry. In this failure, one party of an economic transaction has more or better information than the other. For example, consider a scenario where a hardware company knows of serious security vulnerabilities in their product but decides that it would be too costly to fix. It would be rational for a self-interested company to not publicly disclose the vulnerability for fear that it would damage their reputation and hurt sales. There is then an imbalance or asymmetry of information (i.e. the presence of the vulnerability) between the company and its customers. Without knowing of the serious vulnerabilities, customers will continue to buy the product to their own detriment. Breaking down this information asymmetry would push customers to purchase safer, competing products instead, and would incentivize the company to patch the vulnerability.

Information asymmetry leads to a related market failure known as the market for lemons [14]. The market for lemons, first explained in the context of the marketplace for used cars, is a situation wherein information asymmetry degrades the quality of goods in the marketplace. Imagine a marketplace of used cars, where some of the cars are of good quality while others are defective (the "lemons"). The car dealers will price the cars accordingly, selling the more valuable good cars at a higher price point and selling the lemons for cheap. But only the car dealers know the difference between the good cars and the lemons, because the buyers don't know enough about cars to distinguish the good from the bad (because of information asymmetry). Buyers, not wanting to purchase a lemon, will be willing to pay a fixed price somewhere between the price of a good car and a lemon. The result is that the dealers will only sell when they possess lemons, because they will be selling a low-value car at a price higher than the car is worth and make a profit. Likewise, dealers will leave the marketplace when they possess good cars, because the dealers won't want to sell a good car for less than it's worth. This introduces a negative feedback loop that pushes the good cars out of the marketplace and floods it with low-quality, defective lemons.

In other words, customers are unwilling to pay a premium for a feature or quality they can't identify. We see the same principle apply to hardware security. Hardware is dizzyingly complex, and the average customer is not technically knowledgeable enough to be able to distinguish secure from insecure products. The effect is that the more secure but more expensive products will be driven out of the marketplace, leaving behind only the cheaper, less-secure products. Widespread security suffers as a result, and the general welfare of society is worse off.

2.2.2 Prisoner's Dilemma. The prisoner's dilemma, borrowed from game theory, is a scenario where it is rational for two self-interested entities to not cooperate, even if it is in their best interest to do so. The classic formulation is as follows: Assume there are two prisoners who are being charged for the same crime. In order to prosecute either prisoner, the prosecutors need the second prisoner to testify against the first. If the two prisoners cooperate by remaining silent and refuse to rat each other out, then there isn't enough evidence to convict either prisoner, and so the two prisoners are only convicted of lesser charges and given a light prison sentence. But the prosecutors, being clever, offer immunity to either prisoner if they testify against the other, and without being testified against themselves. This means that a prisoner who betrays the other or "defects" walks away free while their partner in crime is given a severe prison sentence. If both prisoners betray the other (i.e. "defect") by witnessing against the other, then both prisoners are convicted and are given a moderate prison sentence.

The optimal outcome for the prisoners is achieved when the prisoners cooperate and don't testify against each other. However, the incentives are such that it is rational for both prisoners to defect, making both prisoners worse off than if they had cooperated. We can see similar dynamics in security. Imagine a market for lemons where products are insecure and the customers can't distinguish between safe and unsafe products (but can distinguish between product performance). It is in the best interest of all for the product vendors to agree on some kind of collective action (i.e. cooperate) to fix their products' insecurity. Assume that this collective action towards defense will degrade the products' performance slightly but greatly enhance security. The incentives are then the same as the prisoners in the prisoner's dilemma: It is rational for a product vendor to defect (i.e. not implement the defense and not take the performance hit) in the hopes that the competing vendors *don't* defect, so that the defecting vendor's product has a performance edge over its competitor's products and becomes more competitive in the marketplace. If all the vendors were rational, no one would cooperate to jointly improve security, leaving everyone in a suboptimal state of security. In an unconstrained market, such prisoner's dilemmas can disincentivize those with the ability to improve security from doing so.

2.2.3 Misaligned Incentives. Another marketplace failure occurs when those who responsible for providing security are not those who suffer in case of insecurity. For example, consider a CPU vendor with a hardware flaw that allows attackers to read privileged memory, such as cryptographic keys or sensitive financial information. If a user is attacked, only the user suffers the consequences, and not the CPU vendor whose insecure product allowed the attack to happen in the first place. Combined with the market for lemons principle, this means that the CPU vendor has little incentive to create a more secure product.

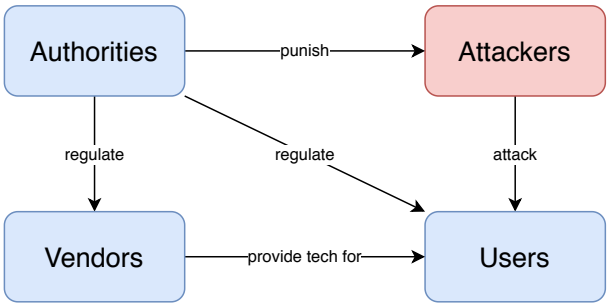
2.2.4 Free Riding. Related to the prisoner's dilemma is the problem of free riding. This market failure occurs when those who benefit from some shared resource rationally do not pay for them. A common example is vaccinations: If a community is widely vaccinated against some disease, it would be rational for an individual to forgo the hassle of vaccination, because the individual is protected by the herd immunity provided by the vaccinated population (i.e. the individual "free rides" off of the efforts of others). Of course, if everyone applied this thinking, no one would get vaccinated, herd immunity would disappear, and disease would spread. We see similar problems in security.

Because security is so dependent on the cyber health of an entire community, it is rational for an individual not to pay their fair contribution towards community security and free ride off of others. For example, if a CPU vendor releases a patch that hurts performance, it is rational for the CPU user to not apply the patch in the hopes that the other community members *do* apply the patch, allowing the user to free ride off of the security benefits of others without themselves taking the hit to performance. This leads to a market failure where no one patches and the community is worse off than if the individuals had patched their systems.

3 DOCTRINE OF SHARED BURDENS

Hardware security nearly always comes at a cost. First, there is the cost of designing and implementing hardware security. This one-time cost of security can come in many forms, such as the cost a vendor incurs when testing a design for vulnerabilities before release. Second, hardware security also often comes with an always-on cost: For example, the cost using a hardware encryption accelerator (in terms of time and energy) is incurred every time a message is encrypted or decrypted. In both cases, security is a burden that must be paid by someone. We introduce a new doctrine

Figure 1: The four players in the game of hardware security, and the relationships between them. The Doctrine of Shared Burdens says that systems should be designed so that the burden of security is distributed between these four players.



of security—the Doctrine of Shared Burdens—which is based on the idea that the burden of security must be distributed between the four players of the security game, namely the Vendors, the Users, the Authorities, and even the Attackers.

3.0.1 The Vendors. The Vendors are the agents who design and and build systems for profit. In our doctrine, the Vendors must bear the burden of ensuring that their products are safe and not easily exploitable. The Vendors pay their burden to security through the cost of validating their products against vulnerabilities, as well as through the opportunity cost of not making products that are more competitive in the marketplace but less secure.

3.0.2 The Users. The Users are the victims of attack. The scope of who can be considered a "User" can range significantly, from a smartphone user to the cybersecurity team at a large organization. In our doctrine, it is the responsibility of the Users to secure their systems as best as possible. Importantly, this means that a User is responsible for the protection of the assets they are entrusted with.

The Users pay for security by incurring the always-on cost of defending their systems. The Users must also uphold their responsibility to security by applying patches and by not disabling security features, or else the Users will end up free riding off of the security efforts of others.

3.0.3 The Authorities. The Authorities are the regulatory bodies that have a degree of authority over the Vendors and the Users. The Authorities have the unique ability to correct the failures in the marketplace for security. The role of the Authorities is often assumed by governments, but not always. For example, self-regulatory organizations (SROs) are non-governmental regulatory groups which have a degree of regulatory authority over certain industries [1] ; SROs and other non-governmental agents can assume the role of an Authority as well. An Authority has the burden of regulating, mandating, and sometimes enforcing the Vendors and the Users to uphold their respective responsibilities to security. This can come in the form of mandates or regulations, e.g. a mandate that Vendors use two-factor authentication in their products. An Authority also has the responsibility of punishing and prosecuting the Attackers when possible.

3.0.4 The Attackers. Finally, the fourth player in the security game of our shared-burden doctrine is the Attacker. The Attacker is the party who perpetrates cybercrime. At first, this may seem odd that the Attacker is one of the security game players that must share the burden of security—After all, we can't expect a criminal to play by the rules set by the other three players. We include the Attacker not as a player who must uphold their responsibility of security to the others, but as a player who pays the price for their decision to attack security. It is a different kind of burden or cost from the other three players, but a cost nonetheless. Our doctrine posits that we should make attacks as expensive as possible for the Attackers in an attempt to discourage them from attacking systems in the first place.

We view the "cost" an Attacker pays as the opportunity cost they incur by choosing to attack a system, as well as the consequences they may face as a result of this decision. We can consider the opportunity cost to be the amount of effort or resources an Attacker may need to expend to be successful. Our doctrine dictates that we should build defenses that require a high cost for the Attacker to overcome. Importantly, our doctrine also says that we should do so without overburdening the User, who must pay for the always-on cost of defense. In other words, our defenses should be deliberately asymmetric against the Attackers and should offload the "cost" of security from the Users to the Attackers as much as possible. One such example is cryptography, which imposes a minor overhead cost to Users with the proper keys, but an enormous cost for an Attacker to break if they don't possess the necessary keys. Finally, we view the risk an Attacker assumes when breaking the law as another form of "cost" the Attacker pays. Our doctrine says that we should make Attackers pay for their actions by improving the Authorities' ability to catch and prosecute the Attackers and increasing the risks and punishments of attacking. Both of these costs paid by the Attackers are intended to discourage and deter attacks.

4 DIFFERENCES FROM PRIOR DOCTRINES

We find that existing doctrines of cybersecurity doctrines are insufficient because they misplace the burden of security among the players in the security game or do not apply to the domain of hardware security. We examine four doctrines of security promulgated by Mulligan and Schneider [10].

4.1 Doctrine of Prevention

The Doctrine of Prevention states that security should be achieved by eliminating bugs and vulnerabilities from systems. While a worthwhile goal, Mulligan and Schneider point out that such a doctrine is impractical to achieve. First, even a vulnerability-free system can still be overcome by social engineering or insider malfeasance. Second, even if we ignore the human element, the computational cost of proving a design to be free of vulnerabilities is often impractical [10]. Mulligan and Schneider position this in terms of software security but the arguments largely hold for hardware security as well, which is why most hardware designs are not formally verified. However, even a formally verified design does not guarantee

[1] A well-known and exemplar SRO is the Financial Industry Regulatory Authority, Inc. (FINRA) which regulates and arbitrates all stock market operations in the United States. FINRA is a non-governmental organization comprised of the very members it regulates. FINRA is authorized by the U.S. Securities and Exchange Commission to enforce the rules and regulations of the the securities industry.

	 *Vendors*	 *Users*	 *Authorities*	 *Attackers*
Doctrine of Prevention	✔			
Doctrine of Risk Management		✔		
Doctrine of Cybersecurity as a Public Good			✔	
Doctrine of Deterrence				✔
Doctrine of Shared Burdens	✔	✔	✔	✔

Table 1: Prior doctrines focus on assigning the burden of security to a single party. Our new proposed doctrine of shared burdens argues that security must be seen as a burden to shared among all the players in the game of security.

a secure product. Formal proofs rest on assumptions about how a design will be used and the environment under which it will be operated, and are only valid if these assumptions are met. For example, Sawada and Hunt formally verified a relatively advanced processor [15], but an attacker can still hack such a system by getting it to violate one of its assumptions or by targeting something that the proof considers out-of-scope, such as side channels.

We add that the Doctrine of Prevention also fails because it assumes that the Vendors are sufficiently motivated to build secure products in the first place. Failures in the marketplace for hardware security show that this is not always the case. The Doctrine of Shared Burdens clarifies that an Authority is needed to regulate and sometimes coerce the Vendors into creating secure products.

4.2 Doctrine of Risk Management

The Doctrine of Risk Management takes a more pragmatic approach to security by acknowledging that vulnerabilities and attacks are inevitable. Rather than trying to build perfectly secure systems, the Doctrine of Risk Management puts security into terms of probabilities and expectations. According to this doctrine, security administrators should prioritize finding and fixing the vulnerabilities that are 1) most likely to be exploited, and 2) most likely to cause harm if exploited. This doctrine posits that security ought to be seen as an investment against future attacks and financial losses, and that the "right" level of security is whatever is best for an organization's bottom line.

Mulligan and Schneider present this doctrine from the perspective of software or network security. The claim is that this doctrine fails because there is a lack of accurate and publicly available information on threats and attacks, making it very difficult to quantitatively reason about the risks of cyber attack and build useful actuarial models. The consequence is that security practitioners are rarely able to make metric-driven decisions on how to best secure their systems, and instead must resort to an ad hoc and qualitative approach to this doctrine.

The Doctrine of Risk Management doesn't work for hardware security, and not just because the risks are hard to measure. Unlike other areas of security—say, network security, where a system administrator (a User) has a high degree of control over their network's security policies and defenses—hardware security is effectively limited by what the Vendors are willing to do to fix the vulnerabilities in their products. Consider the microarchitectural data sampling (MDS) bugs found in Intel processors in 2019 [16–18]. The recommended fix is to disable Hyper-Threading, which is a performance technique that lies at the root of the MDS vulnerabilities. However, Intel has decided that MDS is not a serious enough risk to warrant disabling Hyper-Threading. We note that the performance cost of disabling Hyper-Threading could be as much as 40% for certain workloads [12]. In other words, Intel has assumed the role of deciding the what level of risk is tolerable on behalf of its customers, denying Users the ability to decide this for themselves.

This example is far from unique in the realm of hardware security. The result is that even if the Users were technically-minded and concerned about security, they don't have the ability to configure what they deem to be the appropriate level of risk for themselves or their organizations. An advanced User could disable Hyper-Threading themselves via the operating system, but such a feat is far beyond the abilities of the average User. Thus the Doctrine of Risk Management is often not even an option in hardware security.

The Doctrine of Risk Management also falls short in its allocation of the burden of security. The doctrine requires the Users to assume the full cost of security. Market failures and inefficiencies promise that this strategy will always be suboptimal, as Users are subject to free riding and typically don't know enough about security to prevent markets for lemons. The doctrine underburdens the Vendors, who are free to sell products known to be insecure, and it underburdens the Attackers, who face no repercussions for their actions.

A related doctrine to the doctrine of risk management (or perhaps a different flavor) is cyberinsurance. Rather than internalize the risk of attack, an organization can choose to offload the risk of attack to an cyberinsurance policy underwriter. While only a third of US companies have purchased some kind of cyberinsurance, the marketplace for cyberinsurance is expected to grow [19] Such policies offer financial protection against potential attacks and third-party litigation in exchange for insurance premiums. Cyberinsurance, as with internalized risk management, suffers from the same lack of credible data on which to build actuarial models [19]. Pricing cyberinsurance policies becomes a somewhat of a guessing game, which is suboptimal. Cyberinsurance also shares the same well-known economic issues as other types of insurance: *Adverse selection* occurs when those with high risk for losses are more likely

to purchase insurance than those with a low risk for losses. When an insurer's customers become more likely to experience losses, they have to raise premiums to cover the excessive payouts, pricing out the low-risk customers and further exacerbating the problem. Additionally, *moral hazards* occur when people or organizations with insurance tolerate more risk than they would have without insurance, because negative consequences of risky behavior will be borne by the insurer rather than the insuree.

4.3 Doctrine of Deterrence

The third doctrine of cybersecurity that Mulligan and Schneider highlight aims to discourage crime by improving authorities' ability to catch and prosecute the Attackers. This doctrine is hard enough to achieve in software and network security—cybercrime forensics are limited in their ability to assign blame, mostly because our current internet infrastructure is poorly equipped to handle attribution. This can be partially offset with robust logging, but prosecution remains difficult as it often crosses international borders.

The problem becomes even more challenging in the domain of hardware security. Recent microarchitectural attacks such as Spectre [2] and Meltdown [3] demonstrated exploits that are essentially silent to the User. How can a User catch a cybercriminal red-handed if the User has no way of telling that they are being attacked in the first place? The lack of threat of prosecution means that there is little to deter the Attackers.

Even if better attribution were possible, this doctrine fails because it aims to put the entire cost of security onto the Attacker, and ignores the responsibilities the Authorities, Vendors, and Users must play in achieving security. Rather than leaving the security holes open and prosecuting the Attackers later, it would be more efficient for an Authority to hold the Vendors more accountable and require them to close the security holes in the first place.

4.4 Doctrine of Cybersecurity as a Public Good

Mulligan and Schneider propose viewing security as a public good because security is non-rivalrous and non-excludable. Cybersecurity is non-rivalrous because one person benefiting from security does not prevent others from benefiting from security, and it is non-excludable because the user of a secure system cannot be easily excluded from the benefits of the security [10]. Using this framework, Mulligan and Schneider turn to another well-studied public good—public health—as an exemplar.

According to Mulligan and Schneider, it is the responsibility of an Authority (namely the government) to ensure and administer public health, through activities such as public education, disease prevention, and disease control. Public health is a mature model of how and where an Authority's obligation to protect the population can supersede individual liberties. Using this as a framework, Mulligan and Schneider define the goals of the public goods cybersecurity doctrine as (i) providing public cybersecurity, and (ii) managing insecurity in a way that balances individual rights and public welfare.

This doctrine improperly assigns the burden of security to the Authorities alone. Along this line of thought, the Doctrine of Cybersecurity as a Public Good was called into question by Weber [20]. Weber argues that to apply the public health model to cybersecurity may require a level of Authority coercion far beyond what the society is currently willing to accept. For example, we see little precedent for something like a government-enforced cyber-quarantine or cyber-vaccinations, and have little reason to believe a government can (or should) take on the full responsibility of security.

We add that the sheer complexity of hardware is also a severe hindrance to effective Authority-administered security. Only the Vendors and industry experts know how best to secure their products; Authority intervention would inevitably be heavy-handed and misguided. Allowing the Vendors to secure their own devices, but holding them liable for their products' security would be more efficient. The role of the Authorities should be to regulate industries and correct market failures, but not to administer security wholesale.

This doctrine also falls short in its framing of the problem, as it fails to hold the Attackers responsible. The doctrine lacks the notions of punishment and deterrence. For example, communicable diseases such as viruses cannot be punished for their "actions", and are not deterred by the threat of prosecution. A full-picture view of security needs to consider the Attacker as an active participant in the struggle for security, and particularly as a self-interested participant who is motivated by the rewards of hacking but deterred by its drawbacks, such as the real or perceived risk of being caught and punished. A complete doctrine of security needs to include the notion of deterrence in one way or another.

5 CASE STUDIES

We examine three recent high-profile problems in hardware security—Rowhammer, Spectre, and Meltdown—as case studies that illustrate how our doctrine can inform us on how we should allocate the burden of security.

5.1 Rowhammer

Rowhammer is a problem found in modern Dynamic Random Access Memory (DRAM), the technology behind main memory in virtually all computing devices. We use it as a case study because it exemplifies an end-to-end application of our Doctrine of Shared Burdens, from initial discovery to what we believe to be its solution. It provides an excellent model of how the burden of defense can be distributed between Vendors, Users, Authorities, and Attackers.

5.1.1 The Attack. DRAM is a victim of its own success. For the last forty years, its transistors have shrank tremendously, allowing for an exponential increase in density (bits stored per unit area). Rowhammer is an unintended consequence of this tremendous density.

As DRAM cells (essentially just a transistor and a capacitor, capable of storing a single bit) got smaller and smaller, two things happened. First, they became more delicate and more susceptible to losing the data they stored. And second, as DRAM components became more tightly packed together, they started electromagnetically interfering with each other. In 2014, it was shown that this interference could be reliably harnessed to alter the contents of the data stored in DRAM by repeatedly accessing the same row of DRAM memory (henceforth known as "hammering"), wherein the

fluctuations in voltage on the DRAM's internal wires could flip the values of the bits stored in nearby DRAM cells [1]. And in 2015, this primitive was demonstrated to enable a working exploit [21].

Rowhammer is a serious hardware vulnerability because it breaks basic integrity guarantees in computer systems by allowing Attackers to modify unauthorized memory locations, enabling an entire new class of attacks. Rowhammer has demonstrated dangerous potential, and can be leveraged to achieve privilege escalation [21], cross-VM attacks [22], and even as a side channel to read privileged data [23]. To make matters worse, there's no easy fix—bits can flip faster than a doubled DRAM refresh rate can fix, and are more numerous than error correcting codes (ECC) can correct [1].

5.1.2 Who Should Fix Rowhammer? How the DRAM industry has handled the rowhammer problem is an interesting and illuminating case study in how overall security is often a function of the distribution of burdens between the Vendors, the Users, the Authorities, and the Attackers. We begin by looking at the balance of burdens immediately after the rowhammer problem was identified. Since ECC couldn't fix the problem, the only available rowhammer defense a User could employ was to increase the DRAM refresh rate[2], thus reducing how long a DRAM row could be "hammered" before being automatically refreshed. Unfortunately, simply doubling the refresh rate doesn't fix the problem—the authors estimate that, in the worst case, the refresh rate would need to increase by a factor of *seven* in order to fully mitigate bit flips [1]. Since refresh is already such a expensive operation (in terms of both DRAM latency and energy), the overhead of such a defense would come at a tremendous cost. By increasing the refresh rate, it is ultimately the Users who pay for security, who suffer from slower memory that consumes more energy. Through the perspective of our doctrine of shared burdens, we see that the cost of security is placed solely on the Users, and that the other players are not shouldering their fair share of the burden.

Rowhammer is a flaw in DRAM products, and should be the responsibility of the Vendors to correct. However, it is not immediately clear *which Vendor* should be responsible for fixing the problem: For DRAM to be used, it requires an external memory controller—typically located on the same chip as the CPU—to issue commands such as reads, writes, and refreshes. And of the various rowhammer defenses promoted after the vulnerability became known, some advocated for rowhammer to be fixed by the memory controller whiles others advocated that the problem should be fixed within the DRAM chips themselves. Since the DRAM chips and memory controllers are made by different companies, who should be responsible for fixing the problem?

If the memory controllers vendors and DRAM vendors operated completely unconstrained, it is reasonable to believe that neither side would voluntarily take on the burden of fixing rowhammer. Each side could rightfully claim that it is the responsibility of the other side to fix the problem. However, the memory controller vendors and the DRAM vendors do *not* operate wholly unconstrained. Both sides belong to JEDEC, a DRAM industry trade organization. JEDEC decides and defines standards for DRAM technologies, including the interface between DRAM devices and memory controllers, which the JEDEC members must then follow. Importantly, JEDEC members don't join because they like being told how their products should behave; Rather, JEDEC members join because it is in their own self interest to do so: Standardization increases cross-compatibility between DRAM and the devices that use it, effectively opening up a DRAM vendor's products to a wider consumer base. For a DRAM vendor *not* to comply with JEDEC standardization would essentially be a death sentence, as no memory controller vendor would want their product to be reliant on a single DRAM vendor. We see then that JEDEC has a high degree of authority over the DRAM industry, and can act in the role of an Authority in our doctrine of shared burdens.

Takeaway #1: Trade organizations and standards committees can fulfill the role of an Authority.

But JEDEC is a non-governmental organization, and is not guaranteed to make decisions that are in the best interest of widespread security. After all, JEDEC is comprised of self-interested companies; If these companies collectively decided against a standardized rowhammer defense, it is plausible that rowhammer would remain unsolved. Indeed, if JEDEC was comprised solely of DRAM and memory controller vendors, this might be the case. But in addition to DRAM and memory controller vendors, JEDEC also contains a significant number of DRAM *consumers*, in industries ranging from cloud computing and automotive to aerospace and defense. Since these consumers, who play the role of the User rather than the Vendor, stand to lose more in case of DRAM insecurity than the Vendors, it is likely that their presence in JEDEC has influenced the standardization committees towards seriously addressing the rowhammer problem.

Takeaway #2: Consumer (User) interest groups may be needed to motivate an Authority into acting on their behalf.

5.1.3 Attempt # 1: TRR. One of the first concerted efforts to address rowhammer problem was Targeted Row Refresh (TRR), an optional mode of operation defined in JEDEC's 2014 LPDDR4 standard (the fourth generation of low-power DRAM) [24]. And while not part of the DDR4 (fourth generation DRAM, intended for higher-performance applications than LPDDR4) standard, some DRAM vendors opted to include TRR in some of their later DDR4 products as well. TRR wasn't a prescriptive order telling DRAM vendors how to fix rowhammer, but was more of a contract between memory controller and DRAM device on the high-level actions that should be taken in the presence of excessive row activations. Patents give hints on how each vendor may have internally implemented TRR [25–28], but ultimately the architecture used commercial devices is largely unknown. While TRR was a first step towards rowhammer protection, it unfortunately was doomed to fail—it could only refresh a limited number of DRAM rows per DRAM refresh window, and was only designed to refresh the rows physically adjacent to excessively activated (i.e. hammered) rows. This allowed rowhammer attacks with multiple targets to overwhelm TRR's ability to fix the hotspots, and did not address the issue of rowhammers affecting more than just the nearest physically adjacent row [29]. Given that the LPDDR4 standard was released only a few months after the rowhammer bug was announced, it is not surprising and perhaps

[2]DRAM is *dynamic*, meaning that the DRAM cells will lose their contents unless they are periodically refreshed.

even expected that the DRAM industry's first attempt at fixing the rowhammer problem was not without flaw.

Despite its shortcomings, one of the DDR4 vendors (Vendor C in the TRRespass paper) seems to have leveraged TRR successfully enough to completely protect against many variants of rowhammer [29]. However, because vendors A and B also implement TRR yet still fall susceptible to rowhammer, we must conclude that the difference between success and failure is not the protocol defined in the standard itself, but the private, proprietary, and vendor-dependent architecture used to implement TRR. In short, the LPDDR4 standard alone does not offer sufficient protection against rowhammer. JEDEC was presumably aware of this, which is why they introduced a new rowhammer defense in the next generation of DRAM.

5.1.4 Attempt # 2: RFM. In early 2020, JEDEC released the LPDDR5 standard. In LPDDR5, JEDEC replaced TRR with Refresh Management (RFM) [30]. In RFM, the DRAM device counts activate commands per bank, and issues a refresh once a threshold number of activations has been reached. RFM requires a degree of coordination between the memory controller and DRAM device, and distributes the burden of defense between the DRAM vendors and the memory controller vendors. While LPDDR5 devices are just now entering the market and researchers have not yet been able to experimentally evaluate RFM, the tightness of the standard suggests that RFM will provide a very high level of protection against rowhammer attacks, possibly even eliminating the rowhammer problem altogether. And while the DDR5 standard hasn't yet been released, we suspect that it will include a very similar if not identical rowhammer defense.

RFM is an asymmetrical defense that punishes the Attacker but not the User. In the LPDDR5 standard, RFM is defined to have the memory controller count the number of times a row of DRAM memory is accessed via an `ACT` (activate) command. If the number of activations exceeds some threshold, the memory controller issues a special type of refresh command (`RFM`) which applies fine-grained, selective refreshes to "hot" regions in the DRAM (essentially regions of DRAM cells that have repeatedly been accessed since the last time the region was refreshed). The DRAM chips are specially designed to account for this special type of refresh and still maintain performance. Therefore, if an Attacker tries to rowhammer a region of DRAM by repeatedly activating rows in some region of DRAM, RFM will automatically refresh the victim region, nullifying any advances the Attacker had made. The standard is flexible enough to allow the DRAM and memory controller vendors to "tune" the defense so that the RFM mechanisms spring into action before an Attacker is able to make any headway in flipping DRAM bits. Yet at the same time, the RFM standard is designed not to burden the User (the DRAM owner and user) with excessive, additional refreshes, minimizing the always-on cost of rowhammer defense. Then if RFM is properly implemented, we expect that it will successfully shift the burden of security away from the User and onto the Attacker. RFM only burdens systems with its full weight in the presence of anomalous behavior.

Takeaway #3: The burden of security can be offloaded from the User to the Attacker by punishing anomalous behavior.

The DRAM industry's response to the rowhammer problem, from its initial discovery to what seems to be a viable solution (RFM), shows how the burden of security can be allocated among the four players of the security game. This case study also provides examples of how the thorny non-technical challenges can be overcome, namely how responsibility over a community-wide problem can be allocated and distributed in a self-regulatory body such as JEDEC. Finally, it is worth pointing out that JEDEC, a non-governmental organization with no coercive power, was able to get its members to agree to collectively take on the burden of securing their products against rowhammer. We see that standards can play a huge role in achieving security by getting Vendors to agree to take on some of the cost of security. Perhaps other community-wide security problems can be solved if we leverage standardization to make it in the best interest of those who have the ability to fix longstanding problems to do so.

5.2 Spectre

Another serious open threat to hardware security is Spectre [2]. Announced in early 2018, Spectre demonstrated that speculative execution, a performance-enhancing feature in modern processors, could be exploited in a dangerous new type of attack. Unlike Rowhammer, Spectre is still very much an open problem that has little in the way of usable and deployable solutions.

5.2.1 The Attack. Spectre targets speculative execution. Speculative execution in processors can be defined as any action that is taken preemptively and on the expectation (but not guarantee) that a program's execution will follow one path and not some other path. Modern processors employ many types of speculation as an effective means of improving performance. In the canonical Spectre attack, the type of speculation targeted was branch prediction, which works as follows: When a processor is executing a program, it frequently encounters branches in the execution path. Branches can come in many forms. They can be *conditional*, where a particular execution path is taken if some set of conditions are met and not taken if the conditions are not met (an `if` statement is a simple example of this). Branches can also be *indirect*. Unlike conditional branches, which explicitly tell the CPU the address at which to start executing if the condition holds, an indirect branch instead tells the CPU where the address is located. As it turns out, both types of branches are highly predictable using on-line machine learning algorithms built into the hardware. CPUs can achieve dramatic performance improvements if they can correctly predict branch direction, because they can start speculatively executing the branch before the program's actual branch direction is known. If the branch prediction was correct, then the speculatively executed instructions are confirmed to be correct, and the program is further along in its execution than if the CPU has waited until the direction of the branch was known. But if the branch prediction was wrong, the speculatively executed instructions are incorrect, and must be purged from the CPU.

It is in this way that Spectre takes advantage of speculative execution. Spectre maliciously mistrains the branch predictor to purposely mispeculate and access out-of-bounds data. Before Spectre, such mispeculation wasn't thought to be a security problem, because the CPU invalidates speculated instructions once it realizes it mispredicted. But Spectre showed that the results of mispeculated, out-of-bounds instructions could be exfiltrated even when

invalidated. Spectre exfiltrates data through cache timing side channels such as Flush+Reload [31, 32] or Evict+Reload [33], although other microarchitectural side channels could theoretically be used as well. While originally demonstrated to attack branch prediction, Spectre attacks can target many of the types of speculation found in modern CPUs.

Spectre is a serious threat to security at large. It has the potential to read arbitrary memory locations, including cryptographic keys. But perhaps the most troubling to security researchers is that there is no real solution to the problem. Computer architects have not yet found a way to engineer themselves out of the problem, and there is no widely accepted solution.

5.2.2 Who Should Fix Spectre? The only fail-safe defense currently available is to turn off speculation altogether. This is comparable to the early days of the rowhammer vulnerability, where the only available solution was to raise the refresh rate to intolerable levels. Much like the rowhammer problem was several years ago, the burden of defense against Spectre rests far too heavily on the Users of systems, and not nearly enough on the Vendors, Authorities, or Attackers. Spectre defenses will require a rebalancing of these roles and responsibilities before a solution can be achieved.

As with rowhammer, we can first look to the Vendors to take upon more of the burden of security. Unlike rowhammer, which had multiple parties partially to blame, with Spectre we know exactly who needs to fix the problem: the CPU vendors. Unfortunately, there is no JEDEC-like organization among CPU vendors to standardize what a defense should look like. In the terms of our doctrine of shared burdens, there is no Authority that can exert its authority and get the CPU vendors to build Spectre defenses into their products. Likewise, there are no consumer advocate groups that have enough influence to motivate the CPU vendors to fix the problem either. Finally, information asymmetry prevents the market from correcting the problem: Due to the sheer complexity of CPUs, consumers won't be able to evaluate and properly price the value of a Spectre defense, and won't pay a premium for a feature (Spectre security) they can't identify, causing a market for lemons. This puts the CPU vendors in a prisoner's dilemma. It would be beneficial for all if the vendors cooperate and agree to jointly fix Spectre in their products, but the threat of defection from competing Vendors makes this an irrational proposition. In other words, it is rational for the CPU Vendors to *not* fix Spectre, at the expense of security as a whole. Unconstrained and financially unmotivated, we shouldn't expect the CPU vendors to take upon the burden of fixing Spectre without first applying some kind of exogenous pressure.

> **Takeaway #4:** Spectre stands to remain unresolved, because the CPU vendors are stuck in a prisoner's dilemma and there is no Authority to correct this market failure.

Even if a JEDEC-like organization did exist for CPU vendors and was able to coordinate a standardized defense, what would such a defense look like? The defense would have to be specific enough to fix the problem but general enough to allow for variations in implementations between the different Vendors. One way we can foresee such an approach would be a tax on performance or energy. This approach balances the burden between Vendors and Users, with the Vendors paying to implement the defense and the Users paying (in terms of performance or energy) for the always-on overhead.

In our search for a solution to Spectre, we must also consider the balance of the burden of security between the Attacker and User. For defenses to be accepted by the Users, we need the always-on, recurring cost of defense to be tolerably low. Likewise, in accordance with our doctrine of distributed burdens, we want the burden of defense to be asymmetrically placed onto the Attackers. We would like defenses that can flare up when anomalous behavior is detected. We can consider mispeculation to be the anomalous behavior. However, since mispeculation is a regular occurrence even in benign program execution, we need a defense that doesn't needlessly punish programs for mispeculating. We can look to the adaptive lockout mechanisms on phones and laptops as an example: Perhaps the "punishment" meted out by a Spectre defense should scale with the number of mispeculations within some time frame, where punishment could be something like the speed at which a CPU allows a process to execute.

5.3 Meltdown

Meltdown is a hardware-based attack that was announced at the same time as Spectre [3]. Meltdown shares some similarities with Spectre, and uses some of the same mechanisms as the Spectre attack, and therefore is sometimes conflated with Spectre or thought to be a rooted in the same underlying problem. However, while superficially similar, Spectre and Meltdown are fundamentally different problems and must be thought of as such. Seeing Meltdown through the lens of our doctrine requires its own interpretation, independent of Spectre. From this case study we see an example of where market forces can incentivize a Vendor to fix some types of hardware security problems *without* the need for an Authority, and examine the circumstances that make this possible.

5.3.1 The Attack. Meltdown is a consequence of an optimization found in many processors. The optimization (and its deleterious effects) stem from the way that some processors handle faults. A fault is an exception raised by the hardware when code tries to do something unallowed or undefined, such as a division by zero, or in the case of Meltdown, an attempt to read privileged kernel data from an unprivileged process. When such a fault occurs, the processor will typically halt or kill the offending process. Prior to Meltdown, the way many processors handled faults would best be described as "lazy", meaning that the they wouldn't deal with the faults for as long as possible. More specifically, vulnerable processors wouldn't kill such an unauthorized memory access until just before the faulting instruction retires and updates the architectural state of the program. Presumably, some CPU vendors chose to build their fault handling this way because it enabled some kind of optimization somewhere else in the processor. At first glance, this seems like a perfectly reasonable thing to do—eventually the fault is caught, and before the faulty access is able to update the program, so where's the harm? The problem is that between the illegal memory access and the exception being raised, for a brief period of time the unauthorized memory access resides somewhere in the processor's microarchitectural state. Meltdown is a way of exfiltrating this secret memory value in the small time window between unauthorized access and fault handling.

Meltdown leverages a performance-enhancing technique known as *out-of-order execution* to exfiltrate the secret value. Out-of-order execution is a performance-enhancing technique used in processors wherein instructions are allowed to execute as soon as their operands are available rather than being required to wait to execute in program order, and is permitted insofar as the instruction reordering still preserves program correctness. In a Meltdown attack, malicious out-of-order instructions use the secret value obtained by the unauthorized memory access *before* the exception is handled, which can affect microarchitectural structures such as the L1 data cache. Like Spectre, Meltdown then uses a cache side channel timing such as Prime+Probe or Flush+Reload to leak the secret.

In the canonical Meltdown attack, the target is kernel memory. Kernel memory is typically mapped into the virtual address space of every process as a performance enhancement technique—it allows for kernel memory pages to remain in memory and in the translation lookaside buffer (TLB) when the operating system undergoes a context switch. Since Meltdown provides a way for an unprivileged process to read privileged data, all of kernel memory becomes readable. And because the kernel itself typically contains the virtual-to-physical mappings of all of physical memory, it becomes possible to read any memory location from inside any unprivileged user space process.

5.3.2 Who Should Fix Meltdown? Clearly, Meltdown is a serious problem in desperate need of a solution. What is not immediately clear is *how* it should be fixed, and who should pay for the cost of defense. Like Rowhammer, there are multiple parties who could implement a solution to Meltdown. We now use our doctrine of shared burdens to help us understand the problem and how to fix it.

Let's first look at the defense originally proposed by the Meltdown authors: KPTI [34]. Kernel page table isolation (KPTI, also formerly known as KAISER) actually predates Meltdown, as it was originally intended to solve another problem, namely a kernel side-channel attack against KASLR (kernel address space layout randomization), itself a defense against memory safety exploits in the operating system's kernel. As it turns out, KPTI defends against Meltdown as well. KPTI essentially removes the kernel from each processes' address space, thus denying Meltdown its attack surface. KPTI had previously been deployed as a Linux patch, and was implemented in Windows and OS X patches during a responsible disclosure period before Meltdown was announced. Despite the patches Meltdown was not completely fixed, as KPTI still leaves a residual attack surface. It also came with a hefty performance overhead for Users, who were stuck paying (in terms of performance) to defend a product that was initially advertised as secure. Like Rowhammer and Spectre, the first available solution was costly to the Users and allows the Vendors to avoid responsibility.

Another problem with relying on KPTI to fix Meltdown is that it places the burden of defense on the operating system vendors, who were suddenly asked to fix a problem they didn't create. This is unfair, as the Doctrine of Shared Burdens says that Vendors should be held responsible for the security of their own products. Upon a close examination of Meltdown, it is very clear that the problem does not come from the OS vendors but rather the CPU vendors: Meltdown is possible because some CPU vendors bypassed a security domain (privileged data accesses from unprivileged processes), which is a violation of an architectural security principle. In other words, Meltdown was not an out-of-the-blue, completely unexpected and unprecedented attack like Spectre; Rather Meltdown may best be described as simply a bug, and clearly the CPU vendors should be held responsible.

> **Takeaway #5:** Fixing Meltdown is the responsibility of the CPU vendors whose products were insecurely designed.

Since the CPU industry lacks an Authority that can set rules and mediate problems, we may expect a prisoner's dilemma similar to Spectre that prevents CPU vendors from fixing the problem. However, Meltdown arose under certain circumstances that has allowed the free market to partially fix the problem on its own. We highlight two circumstances—endogenous to the marketplace—that have helped fix Meltdown. First, at least in the x86 marketplace which dominates desktop and server computing, Meltdown was isolated to only one CPU vendor—Intel—while its main competitor, AMD, was unaffected. And second, Meltdown (and Spectre) received an enormous amount of publicity at the time, unprecedented for a hardware vulnerability. Meltdown was covered by major mainstream news organizations, and it became known far outside the niche domain of hardware security. This undoubtedly broke down the information asymmetry between Intel and its consumers, who now knew of a problem that while they maybe didn't fully understand, were definitely aware that Intel's products were vulnerable. Consumers then could then knowingly choose between a Meltdown-susceptible processor or a Meltdown-free processor. Clearly, it was in Intel's best financial interest to fix their processors as fast as possible to make them more competitive in the marketplace. And in late 2018, that's exactly what Intel did. Intel announced its new Whiskey Lake architecture, which among other things, had in-silicon fixes to the problem.

> **Takeaway #6:** Market forces can sometimes fix problems on their own without the need for a coercive Authority.

While this may seem like a success of the free market, there are some notable caveats that need to be addressed. The primary issue is that without an Authority to mediate vulnerability disclosure, there are tremendous incentives for Vendors to delay known vulnerabilities and downplay their risks once they are known. We see this in the case of Meltdown and particularly in the later but related MDS attacks [16, 17]. In both cases, the vulnerability was known to CPU vendors (in this case, Intel) for a very long time—over a year in the case of the MDS attacks—before the vulnerability was publicly disclosed. This is very different from software security, where the process from vulnerability discovery to patch is typically 90 days or less. This tremendous delay between vulnerability discovery and vulnerability defense hugely exacerbates the information asymmetry between Vendor and User, as the Vendor is selling a known insecure product to the User without the User's knowledge, potentially for many months if not longer. There was an even larger gap between when Intel first learned of Meltdown and when they first announced their intent to fix it. To fix these problems, the intervention of an Authority may be needed.

We propose the use of a self-regulatory organization (SRO) to act as an Authority and improve vulnerability response. Such an SRO

needs to be comprised of the members it regulates, who are the only ones who understand the sheer complexity of modern hardware designs and how to best regulate them. Under this approach, vulnerability researchers would no longer have to wait on the Vendor's terms before announcing discovered vulnerabilities, and would no longer have to try to talk the Vendors into fixing the discovered problems. A SRO could act as a mediator between vulnerability researchers and Vendors, and could wield the authority necessary to bring the Vendors to make meaningful change.

Takeaway #7: Authorities such as SROs need to mediate the disclosure of vulnerabilities.

6 ARCHITECTING SYSTEMS TO IMPLEMENT A DOCTRINE OF SHARED BURDENS

The current generation of hardware vulnerabilities foreshadow a grim future in which these vulnerabilities become the norm rather than the exception. Trends of increased system integration and increased product complexity, enabled by Moore's law and propelled by consumer expectations, promise that more vulnerabilities are yet to be introduced and discovered. This suggests that it would be advantageous to build our doctrine into our systems rather than only using it once the vulnerabilities are discovered. In this section we suggest how future systems could be architected to better implement a doctrine of shared burdens.

6.1 Architectures for Swift Justice

Consider the scenario where a Vendor discovers a patchable hardware vulnerability in their product. The Vendor begins taking the necessary steps to patch the problem, but discovers that the patch will cause a significant hit to performance when pushed to customers. If the vulnerability was externally discovered, a Vendor may rationally decide to delay the patch release until the end of the disclosure embargo period, and may even extend the embargo period in an attempt to forestall the patch and its performance penalties and the subsequent loss of competitiveness in the marketplace. If the vulnerability was internally discovered, a rational Vendor may not even disclose the vulnerability at all. Both cases, instigated by the Vendor's rational self-interest, are to the detriment of security overall. What can we do to prevent this scenario from happening? What can we do to force the Vendor to enhance security in a timely manner when there is a performance cost?

The current paradigm is that if a Vendor releases a performance-sapping patch that it the burden of the Users to accept this cost as part of owning technology. But according to our Doctrine of Shared Burdens, this scenario should be interpreted as the Vendor offloading the cost of an insecure and flawed design onto the User. In economic terms, the Vendors is externalizing the cost of their own insecurity. We see this as an injustice facing Users today.

Current non-technical fixes to this problem are largely unsatisfactory. Users could invoke the legal power of an Authority in a class action lawsuit to correct this problem. While this mechanism *could* motivate the Vendors to improve the security of their products, it is not timely or efficient enough to be considered justice: There is little if not no precedent of such class action lawsuits in the realm of computer hardware, and legal battles would be protracted and expensive. Meanwhile, all users of the defective product are continuously paying for the cost of insecurity in the form of the performance-sapping patch. The lack of swift action denies justice to the Users. As the maxim goes, justice delayed is justice denied.

To enable swift justice, we envision and present the use of performance-security contracts that can be automatically settled between Vendor and User without the explicit intervention of the Authority. The scheme would work as follows:

(1) Authorities would require all Vendors to provide performance-security contracts to users.
(2) Authorities would require all internal and external vulnerabilities to be registered with the Authority as soon as they become known to the Vendors. Non-compliance will lead to stiff penalties (including license to operate) as working rules of the Authorities.
(3) Authorities will start monitoring the development of patches for the vulnerability. As soon as the patches are available they are pushed out to the users.
(4) Once the patches are deployed, the Users' hardware starts monitoring the performance drop seen by the user based on the patch. Note that the performance loss will be different per User and highly dependent on User workloads.
(5) A rebate or credit will be given to the Users based on the measured performance loss. This rebate/credit can be converted to real money if desired based on conversion rates set by the Authorities, a function that can be performed by experts who price products based on what users value most.

The above scheme is fair and equitable to Users and Vendors, and enhances the state of security in a timely manner: If a User does not turn on the computer, or does not use workloads that are impacted by the fix, then the Vendor does not have to provide credits. If the Vendors delay patches, then Authorities can impose penalties. The inclusion of Vendors in the Authorities (along with Consumer Advocates) ensures that Authorities do not abuse their power.

To enable proper implementation, and checks and balances in the above system, a number of technical challenges have to be resolved.

(1) Computer hardware designers need to determine how to measure the overhead of defense. This raises two questions: What is the overhead? And, how exactly should it be measured? There are number of options of answering the first question: energy, performance, responsiveness, or even user frustration as measured by a camera [35], or some combination thereof based on the market segment. The infrastructure necessary for these measurements is more or less available in most systems. The second question is likely more challenging, as it requires us to estimate the overhead of the defense without running the insecure version of the system. This estimation can be based on mechanistic models of hardware, or based on data-driven models trained on a large number of cases run with and without the defense.
(2) Once a suitable measurement framework is determined, computer architects need to find ways to ensure the integrity of the reported data; Otherwise, a malicious User could alter or fabricate the cost of defense in hopes of a higher credits

from the Vendor. We expect that cryptography and tamper-resistant hardware will play a role in achieving this constraint.

(3) There is also a need to continuously audit the defenses themselves. Once a hardware vendor pushes out a defense, how do the Authorities and Users know that the Vendor's final product actually implements the defense in a satisfactory manner? How do we know that a Vendor has actually taken on the burden of defense, and hasn't instead forgone the defense in favor of higher run time performance? Or if the Vendor uses an adaptive, best-effort strategy to implement a defense, how do we know that the best-effort sufficiently deters attackers? Likewise, how do we know that Users have not turned off their defenses to gain higher performance?

One might question if these concerns are realistic or warranted. Indeed, one needs only to look to the automotive industry for a recent example of how systems can be engineered to cheat audits. In 2015, it was revealed that Volkswagen automobiles were knowingly falsifying their self-reported exhaust emissions levels. By lying about emissions levels, Volkswagen's automobiles could achieve a better fuel efficiency while still appearing to conform to exhaust emissions regulations, thus giving Volkswagen a competitive edge in the marketplace. If we intend to audit the presence of security, we must anticipate that Vendors (and Users) will try to do the same.

We suggest building auditing support in hardware to mitigate this concern. Periodically, the Authorities would run attacks on hardware to determine if the defenses are working as provisioned. The hardware must be designed in a way that executing the (test) attack does not interfere with the normal performance of the hardware to avoid ethical issues. Design of such auditing mechanism requires test lanes, test memory, etc., to be reserved in the hardware for the attacks. Minimizing the cost of such infrastructure is a significant challenge.

6.2 Punitive Architectures

We should expect cybercrime to continue for as long as the criminals are able to act with near impunity [36]. This dynamic is unlikely to change anytime soon, as there is currently no notion of a "punitive" architecture in computer hardware. We describe an architecture as punitive if it either 1) includes hardware support that will enable criminal attackers to be prosecuted by law enforcement channels, or 2) deters attacks by delaying and preventing attackers from causing harm.

What is the use case for such a system? Consider a cloud system. If a VM is carrying out a microarchitectural attack on another co-located instance, it would be valuable to log information on the VM, the processes are running in both VMs, and the data collected so it can be used for prosecution if such a need arises. Of course, this seems unachievable without compromising privacy to some degree. To address this, any loss of privacy due to implementing this solution should first be disclosed in clear terms to the Users. Second, this level of privacy loss may not be as unprecedented as it may first appear, and may be more acceptable to Users than one might initially expect. For instance, companies usually monitor large amounts of internal traffic and behavioral data to detect insider threats, analogous to security cameras in a building. The privacy loss of such a scheme is likely comparable to such monitoring frameworks.

If evidence collection towards attribution is unachievable because of privacy concerns, we must think of other ways to design deterrence into our systems. If the attacker has specific signature moves, it could be possible to isolate the attacker and move them to a different, quarantined machine. Going back to the cloud example, if one instance is suspected to be attacking another co-located instance, the attacker instance can be migrated to another machine where there is no co-location [37]. This other machine needs to be a bit slower so the attacker does not use this strategy to get higher performance.

Another possible solution is to slow down parts of the computer that are used more by Attackers than everyday Users. For instance, high precision clocks are rarely needed in normal programs yet are used very commonly in attacks. It should be possible to slow down these clocks intentionally to frustrate and delay attackers [38, 39]. We refer to these as asymmetrical defenses.

In our goal of asymmetrical defenses, we also see the need for smaller, faster, and more accurate anomaly detectors that can identify the Attackers' behavior and frustrate their efforts. To minimize the always-on cost of defense to the Users, we need to design our defenses to be scalable so that the full weight of defense is only applied to a system during periods of highly suspicious behavior. Only once these two goals are met can our systems truly find the right balance between always-on defenses (which burden the User) and adaptive defenses (which burden the Attacker). Such asymmetrical defenses should be a top design priority in future hardware.

7 NON-TECHNICAL ISSUES

A holistic approach to our Doctrine of Shared Burdens requires more than just technical solutions to improve security. In this section, we outline some non-technical measures may be needed as well.

As a first example, we point out that our technical solution (architectures for swift justice) does not offer explicit incentives to solve the prisoner's dilemma towards solving security problems. Specifically, our schemes do not offer any incentives for companies to be first movers towards making their systems more secure, especially when security comes at a cost to performance. Non-technical solutions are needed for this problem.

To move the market, we suggest that the Authorities could introduce a mandate that *requires a certain fraction of the advertised insecure peak performance and/or operational energy to be set aside for security measures.* This mandate should be satisfied by all Vendors but gives individual Vendors complete flexibility in implementing how this set-aside should be used for security. This type of mandate has several advantages:

(1) It can help Vendors innovate in terms of security without worrying what they will lose out in terms of performance if their competitor spends less on security. In a manner of speaking this levels the playing field for security.

(2) It is not prescriptive in telling the Vendors exactly which problems to fix, and allows Vendors to select specific solutions that demand attention based on their own estimate of risk and demand.
(3) It allows Vendors to address 0-day vulnerabilities on their own and removes the incentive to hide internally-discovered vulnerabilities, since any performance loss from patches can be "absorbed" by the set aside performance dedicated towards security.
(4) It identifies the features the Vendors consider to be security features and creates an audit trail for the Authorities to monitor (using solutions described in the previous section). In other words, it breaks down some barriers towards information asymmetry by disclosing how Vendors respond to threats.

Another largely non-technical issue is the need for widespread security education among engineers. Since a computer engineer's ineptitude can cause harm to a User, it may be needed to require engineers to pass some kind of licensing exam proving they are competent at secure design, similar to the way a lawyer or medical doctor must pass exams administered by an Authority before they are able to practice in their field. Due to the ever-changing nature of security and threats, such a licensing will likely need to be periodically renewed so that engineers are up-to-date on the latest and best practices.

8 CONCLUSION

The prognosis for hardware security is grim. The rate at which serious vulnerabilities are discovered far outpaces the computer hardware industry's ability (or motivation) to fix the problems. We expect that the severity of the problem will only increase without some sort of widespread action. In this paper, we presented a new doctrine for hardware security that aims to advance the discourse on what these kinds of widespread actions should look like and where they should come from. Without a conceptual framework for the decision makers and regulators to use, we run the risk of crafting policy that misappropriates the burden of security and misses the bigger picture of how security should be administered.

REFERENCES

[1] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu, "Flipping bits in memory without accessing them: An experimental study of dram disturbance errors," *ACM SIGARCH Computer Architecture News*, vol. 42, no. 3, pp. 361–372, 2014.
[2] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher *et al.*, "Spectre attacks: Exploiting speculative execution," in *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2019, pp. 1–19.
[3] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, A. Fogh, J. Horn, S. Mangard, P. Kocher, D. Genkin *et al.*, "Meltdown: Reading kernel memory from user space," in *27th {USENIX} Security Symposium ({USENIX} Security 18)*, 2018, pp. 973–990.
[4] "General data protection regulation," 2016. [Online]. Available: https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32016R0679
[5] "California consumer privacy act of 2018," 2018. [Online]. Available: https://leginfo.legislature.ca.gov/faces/billTextClient.xhtml?bill_id=201720180AB375
[6] "Government to strengthen security of internet-connected products," January 2020. [Online]. Available: https://www.gov.uk/government/news/government-to-strengthen-security-of-internet-connected-products
[7] "Cybersecurity labeling scheme." [Online]. Available: https://www.csa.gov.sg/programmes/cybersecurity-labelling
[8] "Framework for improving critical infrastructure cybersecurity, version 1.1," 2018. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/CSWP/NIST.CSWP.04162018.pdf
[9] "Mitigating the risk of software vulnerabilities by adopting a secure software development framework (ssdf)," 2020. [Online]. Available: https://csrc.nist.gov/publications/detail/white-paper/2020/04/23/mitigating-risk-of-software-vulnerabilities-with-ssdf/final
[10] D. K. Mulligan and F. B. Schneider, "Doctrine for cybersecurity," *Daedalus*, vol. 140, no. 4, pp. 70–92, 2011.
[11] M. Larabel, "The performance impact of mds / zombieload plus the overall cost now of spectre/meltdown/l1tf/mds," Phoronix, May 2019. [Online]. Available: ThePerformanceImpactOfMDS/ZombieloadPlusTheOverallCostNowOfSpectre/Meltdown/L1TF/MDS
[12] "Additional mitigations for speculative execution vulnerabilities in intel cpus," Apple.
[13] R. Anderson, "Why information security is hard-an economic perspective," in *Seventeenth Annual Computer Security Applications Conference*. IEEE, 2001, pp. 358–365.
[14] G. A. Akerlof, "The market for "lemons": Quality uncertainty and the market mechanism," in *Uncertainty in economics*. Elsevier, 1978, pp. 235–251.
[15] J. Sawada and W. A. Hunt, "Verification of fm9801: An out-of-order microprocessor model with speculative execution, exceptions, and program-modifying capability," *Formal Methods in System Design*, vol. 20, no. 2, pp. 187–222, 2002.
[16] S. van Schaik, A. Milburn, S. Österlund, P. Frigo, G. Maisuradze, K. Razavi, H. Bos, and C. Giuffrida, "RIDL: Rogue in-flight data load," in *S&P*, May 2019.
[17] C. Canella, D. Genkin, L. Giner, D. Gruss, M. Lipp, M. Minkin, D. Moghimi, F. Piessens, M. Schwarz, B. Sunar, J. Van Bulck, and Y. Yarom, "Fallout: Leaking data on meltdown-resistant cpus," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 2019.
[18] M. Schwarz, M. Lipp, D. Moghimi, J. Van Bulck, J. Stecklina, T. Prescher, and D. Gruss, "Zombieload: Cross-privilege-boundary data sampling," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 753–768.
[19] S. Romanosky, L. Ablon, A. Kuehn, and T. Jones, "Content analysis of cyber insurance policies: how do carriers price cyber risk?" *Journal of Cybersecurity*, vol. 5, no. 1, p. tyz002, 2019.
[20] S. Weber, "Coercion in cybersecurity: What public health models reveal," *Journal of Cybersecurity*, vol. 3, no. 3, pp. 173–183, 2017.
[21] M. Seaborn and T. Dullien, "Exploiting the dram rowhammer bug to gain kernel privileges," *Black Hat*, vol. 15, p. 71, 2015.
[22] Y. Xiao, X. Zhang, Y. Zhang, and R. Teodorescu, "One bit flips, one cloud flops: Cross-vm row hammer attacks and privilege escalation," in *25th {USENIX} Security Symposium ({USENIX} Security 16)*, 2016, pp. 19–35.
[23] A. Kwong, D. Genkin, D. Gruss, and Y. Yarom, "Rambleed: Reading bits in memory without accessing them," in *41st IEEE Symposium on Security and Privacy (S&P)*, 2020.
[24] "Low power double data rate 4 (lpddr4)," JEDEC Solid State Technology Association, Standard, August 2014.
[25] B. I. Jung and S. Y. Kim, "Memory device, memory system, and operating methods thereof," Feb. 9 2016, uS Patent 9,257,169.
[26] S.-C. Yoon, B.-Y. Kim, and J.-B. Park, "Memory and memory system including the same," Jul. 19 2016, uS Patent 9,396,786.
[27] D. H. Hong and S. I. Park, "Smart refresh device," Apr. 12 2016, uS Patent 9,311,984.
[28] J. Lin and M. Garrett, "Handling maximum activation count limit and target row refresh in ddr4 sdram," Mar. 7 2017, uS Patent 9,589,606.
[29] P. Frigo, E. Vannacci, H. Hassan, V. van der Veen, O. Mutlu, C. Giuffrida, H. Bos, and K. Razavi, "TRRespass: Exploiting the Many Sides of Target Row Refresh," in *S&P*, May 2020, best Paper Award. [Online]. Available: Paper=https://download.vusec.net/papers/trrespass_sp20.pdfSlides=https://download.vusec.net/slides/trrespass_sp20.pdfWeb=https://www.vusec.net/projects/trrespassCode=https://github.com/vusec/trrespass
[30] "Low power double data rate 5 (lpddr5)," JEDEC Solid State Technology Association, Standard, January 2020.
[31] D. Gullasch, E. Bangerter, and S. Krenn, "Cache games–bringing access-based cache attacks on aes to practice," in *2011 IEEE Symposium on Security and Privacy*. IEEE, 2011, pp. 490–505.
[32] Y. Yarom and K. Falkner, "Flush+ reload: a high resolution, low noise, l3 cache side-channel attack," in *23rd {USENIX} Security Symposium ({USENIX} Security 14)*, 2014, pp. 719–732.
[33] D. Gruss, R. Spreitzer, and S. Mangard, "Cache template attacks: Automating attacks on inclusive last-level caches," in *24th {USENIX} Security Symposium ({USENIX} Security 15)*, 2015, pp. 897–912.
[34] D. Gruss, M. Lipp, M. Schwarz, R. Fellner, C. Maurice, and S. Mangard, "Kaslr is dead: long live kaslr," in *International Symposium on Engineering Secure Software and Systems*. Springer, 2017, pp. 161–176.
[35] J. Grafsgaard, J. B. Wiggins, K. E. Boyer, E. N. Wiebe, and J. Lester, "Automatically recognizing facial expression: Predicting engagement and frustration," in *Educational Data Mining 2013*, 2013.
[36] R. Anderson, C. Barton, R. Bölme, R. Clayton, C. Ganán, T. Grasso, M. Levi, T. Moore, and M. Vasek, "Measuring the changing cost of cybercrime," 2019.

[37] J. Demme, M. Maycock, J. Schmitz, A. Tang, A. Waksman, S. Sethumadhavan, and S. Stolfo, "On the feasibility of online malware detection with performance counters," *ACM SIGARCH Computer Architecture News*, vol. 41, no. 3, pp. 559–570, 2013.

[38] R. Martin, J. Demme, and S. Sethumadhavan, "Timewarp: rethinking timekeeping and performance monitoring mechanisms to mitigate side-channel attacks," in *2012 39th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2012, pp. 118–129.

[39] D. Kohlbrenner and H. Shacham, "Trusted browsers for uncertain times," in *25th {USENIX} Security Symposium ({USENIX} Security 16)*, 2016, pp. 463–480.