H&P Chapter 1 Supplement:

Introduction to Security for Computer Architecture Students

Adam Hastings Mohammed Tarek Simha Sethumadhavan

September 30, 2019

For decades, the role of the computer architect has been to design systems that meet requirements, defined primarily by speed, area, and energy efficiency. However, as humans become more and more reliant on computers, it becomes increasingly obvious that our specifications have failed us on one key design requirement – Security, – as evidenced by nearly weekly news of high-profile security breaches.

In the face of such damaging consequences, it becomes clear that the security of our technology must no longer be relegated as an afterthought, but must be a first-order design requirement, from the very beginning—security must be "built-in" rather than "bolted on", so to speak. In particular, it becomes important, even necessary, for security to be built into the lowest levels of the computing stack. It is for this reason that security must become a fundamental part of computer architecture, and must be included in any serious study of the subject.

But First, What is Security?

In its broadest sense, security is about protecting *assets*. An asset is anything of value that is worth protecting, *e.g.* anything from the valuables in a bank vault to a secret message shared between friends to classified government information to objects of personal sentimental value. As long as an asset is valued, its owners should work to secure it. For an asset to be *secure*, security professionals generally think of the asset as maintaining three essential properties:

- **Confidentiality**—Is the asset only accessible to those authorized to access it? In computer systems, maintaining confidentiality means controlling who can *read* some piece of information.
- **Integrity**—Is the asset only modifiable by those authorized to modify it? In computer systems, maintaining integrity means controlling who can *write* some piece of information.
- **Availability**—Is the asset available to be used or retrieved when needed?

These three properties—**C**onfidentiality, **I**ntegrity, and **A**vailability—are often referred to by their initials as the CIA triad. An asset's security is maintained as long as these three properties are met, and is violated when one or more of these properties is breached.

Assets, Vulnerabilities, Threats, and Risks

In the field of security, the words *asset, vulnerability, threat*, and *risk* take on specific meanings, which may differ from their everyday English usage. Given that these words have subtle yet important distinctions which are often confused, we find it useful to define these foundational concepts here:

- **Asset**—Something of value that someone wants to protect.
- **Vulnerability**—A gap or weakness in the security of an asset. That is, an asset is vulnerable if some weakness in its security can be exploited, leading to a violation of confidentiality, integrity, or availability.
- **Threat**—Anything or anyone that has the ability or desire to exploit a vulnerability.
- **Risk**—The intersection between assets, vulnerabilities, and threats. Risk is the amount an asset's owner would suffer if a threat exploited an asset's vulnerability.

How to achieve Security?

If we are to ask computer architects to include security as a first-order design requirement, we must first tell computer architects what secure design looks like. Salzter and Schoeder identify some guidelines for designing secure systems [8]:

1. *Open design*—The security of a system should not rest on the secrecy of the system but on the secrecy of a key or password. In other words, the attacker should gain no advantage even if they know the exact details of the security mechanisms used. This is often referred to as *Kerchoff's principle*. A well-known reformulation of this principle is Shannon's maxim, "the enemy knows the system" [9].

2. *Economy of mechanism*—Keep the design as simple and small as possible. "A simpler design is easier to test and validate" [10], and consequently has fewer bugs that can be exploited by an adversary.

3. *Complete mediation*—Every access to every object must be checked for authority. This greatly reduces the scope of possible attacks.

4. *Separation of privilege*—A protection mechanism that requires two or more keys is more robust and secure than a protection mechanism that requires only one. Separation of privileges improves security by removing single points of failure. This is the principle behind two-factor authentication.

5. *Least common mechanism*—Users shouldn't share system mechanisms except when absolutely necessary, because shared mechanisms might provide unintended communication paths or means of interference [10].

6. *Least privilege*—Every program and every user of the program should operate using the least set of privileges necessary to complete the job. This limits accidental, as well as intentional, abuses of privileges.

7. *Fail-safe defaults*—Base access decisions on permission rather than exclusion. In other words, a whitelist is more conservative, and therefore more secure, than a blacklist.

8. *Psychological acceptability*—Human interfaces to security systems should be easy to use, so that users routinely and automatically apply the protection mechanisms correctly. This principle has also been reinterpreted as the better-known "principle of least astonishment", which states that system behavior should be in accordance with the user's experience, expectations, and mental models [7, 10].

Some of these guidelines may sound obvious but implementing these in an effective manner is a significant challenge. The reasons are varied: for instance, for least privilege it can be hard to determine how much privilege is needed by an instruction in a program given the difficulty with determining the intent of programmers. Similarly, mediating all access can result in significant energy overheads. We will identify some of the implementation challenges as we go through the other chapters.

There are two more guidelines worth keeping in mind when designing systems.

1. *Impossibility of Security* No useful system can be absolutely secure. This is for at least two reasons: 1) we do not know all the ways in which a system can be hacked. Doing so would require a model of human creativity which we do not possess. 2) all computation ultimately results in physical changes. A sufficiently powerful adversary will be able to observe these changes. Given the impossibility the goal can only be to make it sufficiently difficult for an attacker with clearly stated motivations and capabilities.

2. *Full-System property*Security is a full-system property, and all aspects of the system: hardware, software, its configuration and operation have to be secure for a system to be secure.

It is also useful to understand why systems are insecure before aspiring to fix them. Broadly speaking, systems are insecure for one of four reasons:

1. *Insecure design* Security is not a design requirement or security requirements are incomplete due to oversight or intention. An example of this category is the electric grid (which was designed for reliability), or most microcontrollers which do not have any access restriction.

2. *Insecure Implementation* The realization of security intent if flawed. An example would be side channel leaks in cryptographic algorithms that leak information about the keys.

3. *Bad Configuration* Configuration of the security system is flawed. Bad firewall policies in securely designed router, or using incorrect version of TLS protocol are examples of bad configuration.

4. *Improper Use* System is not securely operated. Posting selfies on Instagram with a password written on postits is an example of insecure operation.

As can be seen from the above list, the range of problems that have to be solved for a system to be secure are broad. However, advances in each area can be additive and result in better security.

Approaches Towards Security

Over the years, many different paradigms have been proposed for achieving security. These approaches are outlined below. We will study these in the context in the following chapters.

- *Formal methods*—Construct logical proofs to verify or disprove certain properties about a system. For example, a formal verification of a boolean circuit may prove that the circuit never performs an undesirable action for any input that can be supplied by the adversary. This approach can be powerful, but is often limited by 1) assumptions about the system's environment, 2) the enormous amount of computational power needed to formally verify complex systems, and 3) difficulty in specifying undesirable actions. Also known as "correct by construction" approach to security.
- *Cryptography*—Encrypt sensitive information using cryptosystems. Cryptosystems have been designed in such a way that it is virtually impossible for any adversary to decipher the encrypted information (also known as *ciphertext*) without possessing secret keys. Cryptography forms an essential part of our current best-known practices of protecting confidentiality and integrity. The challenge in using cryptograpic techniques for securing system relate to secure key management, and in many cases, efficient execution.
- *Isolation* —Keep trusted and untrusted components separate from each other, and carefully monitor any interaction between the two. By isolating, and therefore reducing, the scope and size of trusted components, it becomes easier for systems to verify and extend integrity guarantees about potentially untrusted components. Ideal isolation systems aim to enforce a property known as "non-interference" which loosely speaking means that each program should complete execution without being impacted by any factor that can be controlled by an attacker.
- *Meta-data/Information-Flow Approaches*— Everything in a system is an object with two types of entities: data and metadata. As computations are applied to data, shadow computations occur on the corresponding metadata, which detect and report illegitimate data modifications. Some of these systems are also capable of detecting illegal modifications to metadata (to an extent). The exact nature of the shadow computations and metadata is determined by specifics of security policies. Policy violations detected by metadata computations should be handled by higher privileged process. The data in the above definition could also include instructions.
- *Moving Target Approaches*— The observation that begets this approach is the belief that almost all attacks (will) have defenses, and all defenses (will) have attacks. In other words, in a world where attackers and defenders are constantly trying to better each other, moving target approaches minimize the advantage to the attackers by changing how the system/defenses work over time.
- *Diversification* — If each system were distinct, it would force the attacker to prepare bespoke exploits increasing their work and reducing their profits. An extreme example is one where each computer has a different (secret or non-secret) ISA. Since the ISAs are different the same exploitation technique and payload cannot be used blindly stemming mass takeovers.
- *Anomaly detection*—This paradigm is borne out of the philosophy that systems will always be insecure irrespective of the presence of other security mechanisms. The goal is to monitor systems for abnormal or unusual behavior, which may indicate an adversarial attack on a system's security. This can be either *signature based* detection, which checks whether system behavior matches some model of malicious behavior, or *misuse* detection, which checks whether system behavior deviates from some model of benign behavior.

In practice, none of these approaches are secure by themselves, and must be combined to achieve reasonable security guarantees.
Exercise: Discuss what aspects of "Reasons for Insecurity" each of the above methods target.

Trends and Concepts in Hardware Security

Many of the various approaches towards security have appeared in practical defenses. Figure 1 provides a timeline of architectural support for security, as found in commercial products. After a very active interest in security during the early ages of digital computing, there appears to have been a hiatus in the 80s and 90s coinciding with the rise of RISC systems. Most security features on the timeline fall into one of four categories:

- *Virtualization*—Giving code the illusion that it in an environment other than the one it is actually being executed on. This can be in the form of hardware support for virtual memory (which isolate processes from each other) or hardware support for virtual machines (which isolate entire operating systems from each other). More in Chapter 2.
- *Tagging*—Equivalent to the metadata-based approach to security described above. Memory locations are "tagged" with metadata, which can signal things like data types or permission levels.

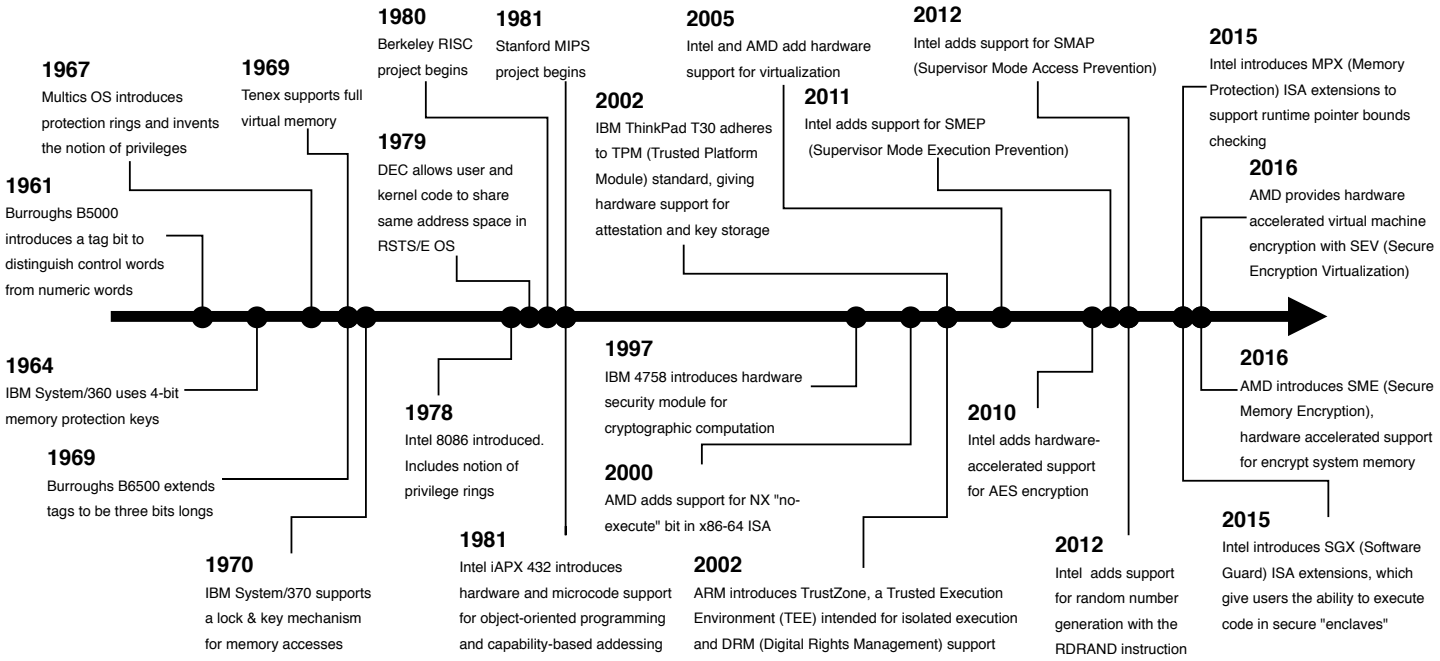

Figure 1: Sixty years of hardware support for security

- *Attestation*—Providing systems the means to attest or verify the integrity of their components. This typically begins with a *root of trust*, a small, trusted set of hardware that can 1) verify its own integrity, and 2) use this to extend and verify the integrity of other system components.
- *Acceleration*—Adding hardware support to reduce the runtime overheads of security features. Typical examples include adding dedicated hardware accelerators to speed up encryption and decryption computations.

Why Computer Architects Should Study Security

Now that we have established some foundations of security, we provide two brief arguments for why security should be a fundamental part of any serious study of computer architecture.

Reason #1: Increased Demands Require Novel Architectures and ISAs

As Amdahl's law famously shows, those wishing to speed up a task should focus on improving the portions of the task the occur most often. Making the common case fast has subsequently become a common theme throughout the history of computer architecture. Whereas security features at the architectural level were once a "rare" request, they are becoming a top priority in today's environment. Consequently, as security features move from the uncommon case towards the common case, we must redesign our systems to appropriately handle this change in demand. This change in system design will result in changes at not only the microarchitectural level, but likely at the ISA level. One recent example of this is the memory protection extensions (Intel MPX) to the x86 architecture, which checks runtime pointer references for spatial memory errors such as buffer overflows [1]; Another example is the pointer authentication extensions to the ARMv8-A architecture, which verify the integrity of runtime pointers through the use of cryptographic message authentication codes [6]. These examples, and the many more to follow, demonstrate that future generations of computer architects will need to understand the fundamentals of security to meet the new requirements of future designs.

Reason #2: Architecture and Security are both a Study of Tradeoffs

A primary goal of Computer Architecture (and engineering in general) is figuring out how to best achieve some design requirement given a limited budget of resources. As such, this is a main theme of the Hennessy and Patterson textbook. Various performance metrics—such as speed, area, and energy efficiency—are presented as a key aspect of modern system design, which unsurprisingly are often at odds with each other: Build a faster system, and it uses more energy; Build a smaller, simpler design, and it runs slower.

As security becomes a first-order design requirement, it must be considered alongside speed, area, and energy efficiency as one of the fundamental tradeoffs in computer architecture—Build a more secure system and performance will suffer; Eschew security altogether and you will achieve smaller, faster designs, but at what cost? Architects are then faced with a very tough question: What is the *right* amount of security to include in a design? Since computer architecture is, in many ways, essentially a study of tradeoffs, security is a natural fit to include in architecture class.

References

[1] Introduction to Intel® Memory Protection Extensions, 2013. URL: https://software.intel.com/en-us/articles/introduction-to-intel-memory-protection-extensions.

[2] The Cost of Malicious Cyber Activity to the U.S. Economy, Feb 2018. URL: https://www.whitehouse.gov/wp-content/uploads/2018/03/The-Cost-of-Malicious-Cyber-Activity-to-the-U.S.-Economy.pdf.

[3] Information on the Capital One Cyber Incident, Aug 2019. URL: https://www.capitalone.com/facts2019/.

[4] Paul Dreyer, Therese Jones, Kelly Klima, Jenny Oberholtzer, Aaron Strong, Jonathan William Welburn, and Zev Winkelman. Estimating the global cost of cyber risk. *Methodology and Examples. RAND*, 2018.

[5] Sam Kottler. February 28th DDoS Incident Report, Mar 2018. URL: https://github.blog/2018-03-01-ddos-incident-report/.

[6] Hans Liljestrand, Thomas Nyman, Kui Wang, Carlos Chinea Perez, Jan-Erik Ekberg, and N Asokan. Pac it up: Towards pointer integrity using arm pointer authentication. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 177–194, 2019.

[7] Jerome H Saltzer and M Frans Kaashoek. *Principles of computer system design: an introduction*. Morgan Kaufmann, 2009.

[8] Jerome H Saltzer and Michael D Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308, 1975.

[9] Claude E Shannon. Communication theory of secrecy systems. *Bell system technical journal*, 28(4):656–715, 1949.

[10] Richard E Smith. A contemporary look at saltzer and schroeder's 1975 design principles. *IEEE Security & Privacy*, 10(6):20–25, 2012.