Meng *et al. Cybersecurity* (2018) 1:2
https://doi.org/10.1186/s42400-018-0001-z

Cybersecurity

SURVEY Open Access

Security-first architecture: deploying physically isolated active security processors for safeguarding the future of computing

Dan Meng[1], Rui Hou[1*], Gang Shi[1], Bibo Tu[1], Aimin Yu[1], Ziyuan Zhu[1], Xiaoqi Jia[1] and Peng Liu[2]

Abstract

It is fundamentally challenging to build a secure system atop the current computer architecture. The complexity in software, hardware and ASIC manufacture has reached beyond the capability of existing verification methodologies. Without whole-system verification, current systems have no proven security. It is observed that current systems are exposed to a variety of attacks due to the existence of a large number of exploitable security vulnerabilities. Some vulnerabilities are difficult to remove without significant performance impact because performance and security can be conflicting with each other. Even worse, attacks are constantly evolving, and sophisticated attacks are now capable of systematically exploiting multiple vulnerabilities while remain hidden from detection. Eagering to achieve security hardening of current computer architecture, existing defenses are mostly ad hoc and passive in nature. They are normally developed in responding to specific attacks spontaneously after specific vulnerabilities were discovered. As a result, they are not yet systematic in protecting systems from existing attacks and likely defenseless in front of zero-day attacks.

To confront the aforementioned challenges, this paper proposes *Security-first Architecture*, a concept which enforces systematic and active defenses using *Active Security Processors*. In systems built based on this concept, traditional processors (i.e., *Computation Processors*) are monitored and protected by Active Security Processors. The two types of processors execute on their own physically-isolated resources, including memory, disks, network and I/O devices. The Active Security Processors are provided with dedicated channels to access all the resources of the Computation Processors but not vice versa. This allows the Active Security Processors to actively detect and tackle malicious activities in the Computation Processors with minimum performance degradation while protecting themselves from the attacks launched from the Computation Processors thanks to the resource isolation.

The challenges of building a secure system

In the era of cyberspace, computer security becomes vitally important for protecting personal privacy, business confidence and even national security. However, existing computer security mechanisms have no proven security, and they usually don't protect computer systems against all existing attacks, not to mention the future ones. PRISM (Gellman and Poitras 2013), WannaCry (Ehrenfeld 2017), and recent serious security incidents like Meltdown (Lipp et al. 2018) and Spectre (Kocher et al. 2018) reflect the weakness of existing computer security mechanisms. It is clear that there is an urgent need to enhance computer security using effective, efficient and systematic countermeasures.

Software vulnerabilities are inevitable

Software security vulnerabilities exist because software systems are too complicated. During the past two decades, software has been increasingly woven into the human society, and the scale and complexity of software systems have been increasing significantly. The complexity of most applications goes far beyond the understanding of a single person. For example, the Firefox browser contains 36 million lines of code committed by 6313 contributors over ten years (Mozilla Firefox 2018). Complex code would no

*Correspondence: hourui@iie.ac.cn
[1]Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
Full list of author information is available at the end of the article

wonder have a higher chance of having security vulnerabilities exploitable by attacks (Shin et al. 2011). As mentioned in the June 2017 update, Mozilla fixed a total of 32 bugs of the Firefox browser, among which the most dangerous is a use-after-free vulnerability, CVE-2017-5472, using a destroyed node when regenerating trees. This vulnerability is obviously a severe security vulnerability resulted at least partially from code complexity. Another representative instance is the Linux operating system. The size of Linux kernel is increasing and the function is more and more complicated. As shown in Fig. 1 (Common Vulnerabilities and Exposures 2018), in 2017 the number of newly exposed Linux kernel vulnerabilities hit a historical high of 453, more than twice as many as in 2016. Overflow and memory corruption are classic security vulnerabilities and should have been avoided as much as possible. However, in 2017 the yearly numbers of CVE exposed Linux kernel vulnerabilities of the two types are still the highest, reaching 51 and 26, respectively. Due to great complexity, many critical software systems are extremely difficult to test. Also, the complexity in software has reached beyond the capability of existing verification methodologies. Without whole-system verification, current software systems have no proven security. As the complexity of modern software systems increases continuously, software vulnerabilities are inevitable.

Hardware components are insecure

Similar to the inevitable software vulnerabilities, hardware vulnerabilities broadly exist as well. Unsafe hardware architecture and complex production processes are two main factors of hardware vulnerabilities.

(1) Unsafe hardware architecture

Security risks widely exist in performance-oriented architecture design. In such design, inherent security risks are associated with branch prediction, cache, instruction prefetch and so on. These security risks are well reflected in the following two examples. First, cache is often the target of attacks. Exploiting the time difference between cache miss and hit, side channel attacks (Liu et al. 2015) can successfully speculate code paths. Private keys can be extracted from a code path because the code paths of an encrypted program are deterministic. Second, out-of-order and speculative execution are two fundamental optimization techniques for improving chip-level performance, but from the security viewpoint they can also be viewed as two security vulnerabilities. The recently disclosed Meltdown (Lipp et al. 2018) and Spectre (Kocher et al. 2018) attacks respectively exploited these two vulnerabilities to compromize computers.

(2) Complex production processes

Modern processor chips are produced by complex manufacturing processes. Due to the complexity, product

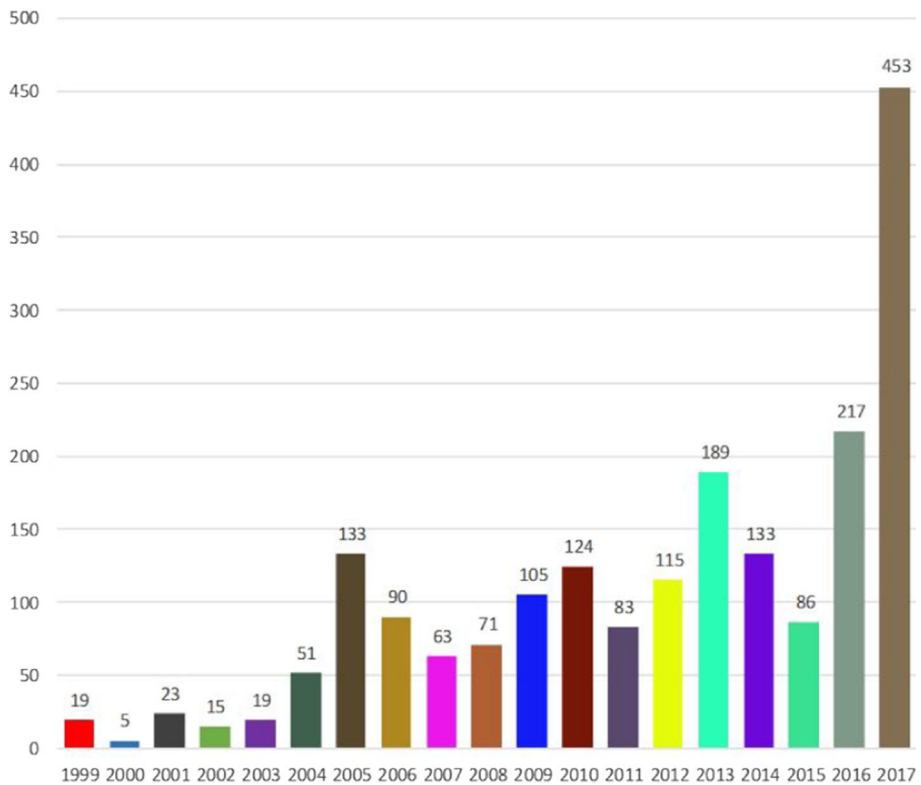

Fig. 1 Vulnerabilities numbers of Linux kernel over years

security cannot be guaranteed. To make sure that the cost is not too high, modern processor chips generally adopt the OEM (Original Equipment Manufacturer) production mode. In this mode, design, layout, manufacturing, packaging and testing are in general separated; each process has different vendors. The design process frequently uses third party IP, standard cell, and a variety of EDA (Electronics Design Automation) tools. Due to globalization, a product often needs to "travel" from one country to another. Figure 2 illustrates the production process of CPU chips and highlights the potential hazards in relevant stages. These uncontrollable factors are likely to introduce additional security risks. On one hand, it is possible for disgruntled or malicious chip designers to implant malicious logic or circuits without being noticed. On the other hand, foundries also have the opportunities to deploy malicious circuits on the chip layout without even modifying the design logic, as shown in the A2 attack (Yang et al. 2016). Finally, real-world manufacturing processes have very limited ability to detect malicious circuits; and very limited ability to verify that circuits are free of malice. This also aggravates the aforementioned hardware security problem.

Attacks are becoming more sophisticated

The complexity of attacks is mainly shown in two aspects:

(1) Width of attack range. Attacks can be launched from any places, including all layers from application software to physical hardware. Many popular attacks are mainly at the software level, such as SQL injection (Halfond et al. 2006) and CSRF (Cross-site request forgery) (Lin et al. 2009). In addition, ARP (Address Resolution Protocol) Spoofing (Whalen 2001), session hijacking and others indicate that the network protocols can be attacked. Furthermore, various vulnerabilities also show that hackers are able to launch hardware-level attacks, such as Rowhammer (Seaborn and Dullien 2015; Kim et al. 2014), Meltdown (Lipp et al. 2018) and Spectre (Kocher et al. 2018).

(2) Variety of attack means. The simple means of attack rely on unintentional yet harmful operations, such as downloading a program bundled with a Trojan. Next, worms and botnets are designed to self-replicate and spread. These attack means automatically exploit software vulnerabilities. The exploitation of software vulnerabilities has also evolved from code injection attacks to code reuse attacks such as ROP (Return-oriented programming) (Checkoway et al. 2010) and JOP (Jump-oriented programming) (Bletsch et al. 2011). Shifting the target from software to hardware, many side channel attacks exploit hardware vulnerability.

Fundamental issues

Building a secure system atop the current computer architecture is fundamentally difficult. The fast-growing complexity in software inevitably leads to exploitable security vulnerabilities. IP (Intellectural Property) reuse and the separated ASIC manufacture procedure with multiple vendors make it impossible to thoroughly inspect the whole design and the manufacture procedures. Some architecture-level vulnerabilities are difficult to remove due to the conflicting interests between performance and security. In the meanwhile, attacks are constantly evolving with new means to outwit existing defenses. To meet this challenge, this paper proposes security-priority architecture, a concept which separates the security tasks from the

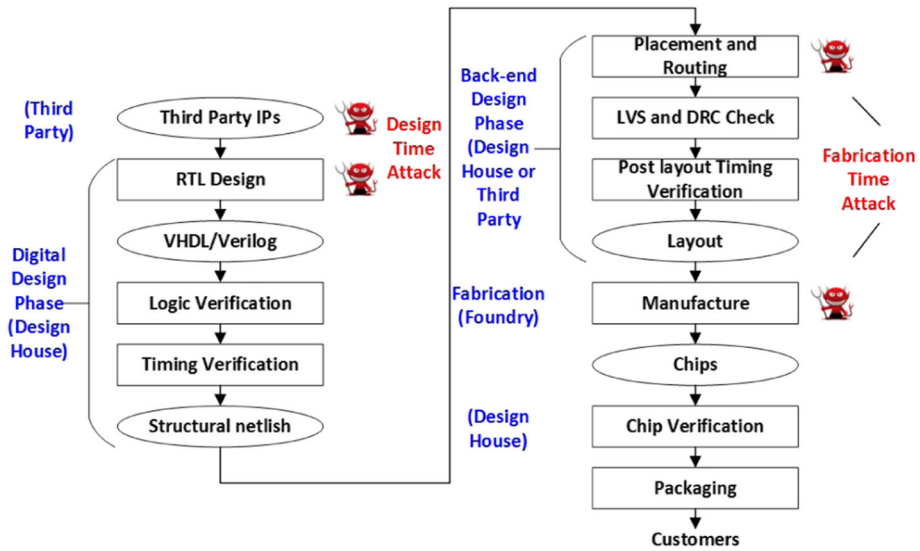

Fig. 2 The common attacks in typical chip design flow (this figure is from (Yang et al. 2016))

normal tasks by physical isolation. The architecture can remedy the deficiencies of existing security mechanisms and provide a new direction worth exploring.

Weaknesses of current countermeasures

Security has always been a secondary consideration in computer system designs in the past two decades because the priority is always given to performance, power and cost (area). This subordination leads to potential security risks and weak defense mechanisms.

Existing architecture-level defenses are usually resulted from passively responding to specific attacks

Several micro-architecture level defense mechanisms have been integrated into commercial chips. Here, we discuss three representative types of architecture-level defenses.

Memory overflow defense : Memory corruption bugs enable attackers to maliciously change a program's behavior (van der Veen et al. 2012; Szekeres et al. 2013). Applications written in low-level languages like C/C++ are prone to these kinds of bugs due to the lack of memory safety. Since it is difficult to figure out all potential memory overflow bugs, an efficient way to enforce memory safety is to add extra hardware protection mechanisms. For example, Intel recently released a new ISA extension, named as Memory Protection Extensions (MPX) (Oleksenko et al. 2017). To facilitate memory safety checking, application developers are allowed to insert *boundary record instructions* to store the boundaries of protected memory regions, like arrays, at the places where these data structures are defined, and insert *boundary check instructions* to ensure that the access to these data structures are inside their valid ranges. Unfortunately, MPX suffers from high performance overhead which limits its adoption.

Pointer integrity defense : The Pointer Authentication (PA) mechanism is added to ARMv8.3-A to prevent memory corruption attacks (Qualcomm Technologies 2017). PA guarantees the integrity of pointers by binding each pointer with a Pointer Authentication Code (PAC). As the actual address space in 64-bit architectures is less than 64 bits, PA places the PAC to the unused bits in the pointer value to minimize the size and performance impact. PACs are computed by a lightweight cryptography algorithm (Avanzi 2017) and added to the pointer values by extended `PAC` instructions. The integrity of pointers is verified and restored by the `AUT` instructions. To restrict the accesses to pointers in special context, PA allocates keys for instruction pointers, data pointers and general-purpose instructions. The keys are still managed by software.

Control-flow integrity defense : Control-flow integrity (CFI) (Abadi et al. 2005; Davi 2015; Burow et al. 2017) is considered as one general and promising defense against code-reuse attacks (Shacham 2007; Bletsch et al. 2011; Carlini and Wagner 2014; Schuster et al. 2015). CFI restricts the control-flow of an application program to valid execution traces; and the program's predefined Control-Flow Graph (CFG) tells what is valid. Forward-edge control-flow represents transfers caused by indirect jumps and function calls. Backward-edge control-flow represents transfers caused by function return instructions. Intel has an ISA-level CFI extension, named as Control-flow Enforcement Technology (CET) (Intel Corporation 2016), which protects forward-edge CFI by indirect branch tracking and ensures backward-edge CFI by hardware shadow stacks. However, CET suffers from two problems. One is the difficulty in defining a complete and precise legal CFG (Evans et al. 2015), because some informaion is only available at runtime, such as the target of an indirect jump. The other is the limited size of the hardware shadow stack. The stack has to rely on the operating system (OS) for handling context switches and deeply nested function calls (Frantzen and Shuey 2001).

In summary, a common characteristic of three representative types of architecture-level defenses is that they seek to protect the chip or system from a specific vulnerability. In other words, existing architecture-level defenses are usually resulted from passively responding to specific attacks. As a consequence, such patch-like approaches are usually not generic and cannot handle attacks which exploit zero-day vulnerabilities.

Trusted computing alone is not secure

Trusted computing ensures that executable binaries are not tampered (David et al. 2008; Trusted Computing 2008). The foundation for trusted computing relies on a dedicated chip, which is defined as a Trusted Platform Module (TPM), to measure the integrity of executable binaries by verifying their hash values. To implement this measuring idea, the TPM is designed as a coprocessor that helps the platform software to verify itself using a predefined sequence as well as a cryptographic engine that accelerates encryption, digital signatures, and hashing.

The TPM was initially used as the static root of trust for measurement (SRTM) (Trusted Computing Group 2003). SRTM utilizes the TPM to verify the integrity of booting processes. If a chain of trust is established during a booting process, the boundary of trust can be extended to include more than one level of software within the system. To set the system into a clean state without rebooting, researchers propose dynamic root of trust for measurement (DRTM). Two DRTM implementations are as follows. Intel develops the Trusted Execution Technology (TXT) (Intel Corporation 2006) to securely launch

software (such as the hypervisor and security kernel) at arbitrary time. AMD offers similar capabilities with its Secure Virtual Machine (SVM) extensions.

However, the security requirements are far beyond guaranteeing the integrity of executable binaries. Even if the executable binaries are unchanged, there are still security risks as the control flow integrity (CFI) and data flow integrity (DFI) can be exploited by malware. Dedicated mechanisms implemented in processor chips could be very helpful in defending such kind of advanced attacks.

Logical isolation suffers from information leak through physical side-channels

ARM TrustZone (Wojtczuk and Rutkowska 2017) and Intel SGX (McKeen et al. 2013) are similar technologies which adopt logical isolation to provide a trusted execution environment for security-sensitive data and code. For performance reasons, secure and non-secure programs (or worlds) still share substantial physical hardware resources, including the on-chip cache hierarchy, TLB and others. This sharing leads to the risk of side-channel information leakage.

ARM TrustZone adds a NS bit in the memory system to divide the processor into two worlds: a normal world and an isolated secure world. The non-secure world cannot directly access the resources used by the secure world (ARM Limited 2009). The two worlds communicate with each other through a security monitor. To improve the system performance, caches are not flushed during world switches. However, this allows cache lines from the secure world to be evicted by the cache lines from the normal world and vice versa. Such evictions can be exploited as a cache side-channel to leak security-sensitive information. Exploiting the cache side-channel by evicting the secure cache lines cached in the shared cache belonging to the normal world, a prime+probe attack (Zhang et al. 2016) was able to infer the full AES128 secret key in 2.5 seconds from the normal world kernel or 14 minutes from a user space Android application. Allowing non-secure and secure cache lines to co-exist in caches may also result in cache incoherence behavior. This incoherence can be exploited to install rootkit, which evades the memory introspection mechanisms (Zhang et al. 2016).

Intel SGX is designed to increase the security of software through an "inverse sandbox" mechanism. In this approach, rather than attempting to identify and isolate all the malware on the platform, legitimate software can be sealed inside an enclave and protected from attacks by the malware, irrespective of the privilege level of the latter. In other words, this allows an application to create a secure enclave at the CPU level which is protected from the OS upon which it is running. Data inside an enclave is allowed to be accessed only by the codes located in the same enclave. Moreover, the content of an enclave is encrypted when stored in the memory. Even if an attacker has the ability to snoop the memory bus, she cannot get any useful information. Although the OS cannot directly access the memory region used by an enclave, to simplify the deployment and improve the performance, SGX still leaves the OS in charge of setting up the page tables used by enclaves. By observing an application's page faults and page table attributes, a malicious OS can infer part of the memory access pattern of the application. By hacking the page fault handler of the OS, attackers can track the access patterns of the enclave at page size granularity. Xu et al. (2015) implemented a controlled-channel attack on SGX-enabled platforms to reveal the input-dependent control transfers and data accesses of the target program. A similar but stealthier attack can be launched utilizing the access bit inside the page table entry (Wang et al. 2017). Leveraging the contention in memory resources including caches and TLBs etc., Wang et al. implemented a traditional prime+probe attack which is more fine-grained and powerful on SGX-enabled platforms.

Current architecture-level security subsystems are not really secure

Most of current server chips include a subsystem dedicated for system management. The subsystem is basically a tiny computer-within-a-computer normally embedded directly in the chip along with the host processor cores. Typical examples of such subsystems include Intel Management Engine (ME) (Datenschutz and Pataky 2017; Bogowitz and Swinford 2004), AMD Platform Security Processor (PSP) (Advanced Micro Devices 2018b; Wikimedia Foundation 2018) and Power On Chip Controller (OCC) (Sinharoy et al. 2015). The functions of these subsystems include managing the boot process, keeping the system under thermal limits, and running based on user input modes and parameters. The subsystems can also be used to monitor the system for identifying suspicious activities or events, and make appropriate responses. As security subsystems usually contain a rich set of hardware-based cryptographic primitives, they process security related functions, such as secure boot, secure updates, secure debug and secure communication, significantly faster than software-only solutions.

However, current designs (of these subsystems) need improvement in terms of performance and security. Firstly, complicated logic is expected to run on these subsystems to detect and handle potential malicious behaviors, even unknown attacks. These security inspection workloads can have strong hardware performance requirements, but popular designs use tiny embeded processor cores, such as ARM M series processors. Hence, there is a performance gap between the required workloads and the actual computation power. Secondly, the

security of such a subsystem itself is a concern. As the subsystem has accessves to everything, it might provide an attacker with a powerful backdoor if compromised. The subsystem consists of layers of quite complex software. For example, the Intel ME runs a hidden MINIX OS (Datenschutz and Pataky 2017; Bogowitz and Swinford 2004). Various software components also run on the ME, such as the Intel Active Management Technology (AMT) (Bogowitz and Swinford 2004). On other hand, the complexity provides an attacker with a good chance to compromise the subsystem. On the other hand, the subsystem is implemented with embedded processors (Sinharoy et al. 2015; Datenschutz and Pataky 2017; Advanced Micro Devices 2018b) which usually lack the common security hardening such as stack cookies, No-eXecute (NX) flags, or address space layout randomization (ASLR). Hence, there is also a security gap between the security requirements and the current security hardening. It has been reported that the vulnerabilities in Intel ME and AMD PSP allow attackers to gain administrative capabilities (Intel Security 2017; Advanced Micro Devices 2018a).

Security-first architecture

Architecture overview

According to the analysis in "Weaknesses of current countermeasures" section, the design of security mechanism has the trend which is evolving from passive to active. For example, existing architecture-level defenses are usually resulted from passively responding to specific attacks. And the techniques like trust computing and SGX/trustzone are of much more active defenses, which provide generic secure execution check or environment instead of targeting at specific attacks. However, these active defence mechanisms still suffer from some problems. Firstly, since security is usually not the first priority design goal, logical isolation is preferred than physical isolation, which causes side-channel information leakages. And secondly, even for independent secure environment like AMD PSP or Intel ME, there is a performance gap between the required security workloads and the actual computation power. More seriously, existing active mechanisms usually need to be invoked by other compute units such as CPU. Therefore, once the computer has been compromised, these active security mechanisms are possibly evaded.

Motivated by above analysis, we propose security-first architecture. And our design philosophy obeys following principles:

- Security should be considered at the stage of architecture design and in the context of the whole system instead of the processor chip alone.
- Security should be one of the major goals of architecture design, and it should be paid as least the same amount attention as paid to performance and energy efficiency.
- Minimum performance impacts on the computational workloads are expected.
- Minimum side-channel information leakage is expected.
- Efficiently support the execution of diverse security tasks.
- Security unit should have the highest priority and is independent on computation.

Figure 3 illustrates our proposed security-first architecture. In existing commercial systems, tasks in different security modes or with different privilege levels are being performed consuming one set of physical resources, including processor cores and shared cache. This in principle would result in side-channel information leakages. As a comparison, security-first architecture explicitly differentiates the role of a processor in the same node into two types: conventional processors named as *Computation Processors (CP)*, and newly designed "dedicated to security" processors named as *Active Security Processors (ASP)*. *CPs* are monitored and protected by these *ASPs*. The two types of processors execute on their own resources, including memory, disks, network and I/O devices. The *ASPs* are provided with dedicated channels to access all the resources of the *CPs* but not vice versa. Such uni-directional information flows lead to physical isolation for computation and security systems. The functionalities of computation and security are physically decoupled. Most of the security tasks or functions shall be assigned to *ASP*, and all other tasks are still kept on *CP* as what commercial systems are currently doing. Due to physical isolation, the likelihood to suffer from side-channel attacks will be significantly reduced for the ASP system.

Active Security Processor (ASP)

A physically isolated *ASP* is a key component for security-first architecture. On one hand, the security of security tasks and mechanisms is guaranteed due to phys-

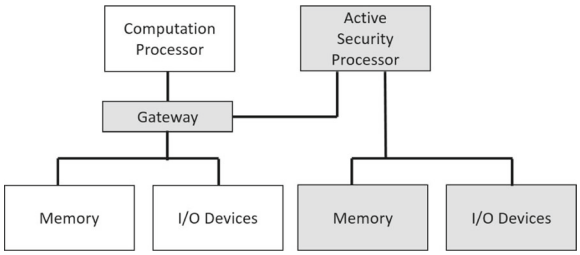

Fig. 3 Security-first architecture

ical isolation. On the other hand, the performance impacts on the execution of computational workloads are minimum due to the transfer the security tasks from CPs to ASPs. This section introduces the detailed ASP design.

At architecture level, *ASP* is designed to have the highest privileges. Once the system is reset, it starts to work on firstly building the trusted boot chain. During the execution phase, *ASP* is designed to have the following (composable) capabilities to support diverse active security tasks:

- Actively access (read and write) the host memory. Since many defence mechanisms need to do in-depth analysis of host memory, *ASP* should efficiently access the contents of host memory. One intuitive idea is to integrate DMA engine into *ASP*. However, such a method can only monitor the memory contents, and would fail to monitor each memory access request issued by *CP*. In some cases, such a method even discards some dangerous memory requests (e.g., illegal access according to a specific rule). To address these limitations, one enhanced solution is to employ a store-and-forward design. However, such advanced feature inevitable increases the memory access latency. Thus in security-first architecture it is optional regarding whether to to support fine-grained memory request monitoring or not.
- Bridge the host I/O devices and *CP*. The feature of bridging (or I/O switch hub) enables *ASP* to analyze the I/O traffic and actively manage the I/O devices (e.g., to physically connect or dis-connect host I/O devices). Considering that the PCIe protocol is widely used to connect I/O devices and processor chip in commercial systems, an embedded PCIe switch module is integrated into *ASP*.
- Actively access (read and write) micro-architecture states or events inside *CP*. The micro-architectural execution data, including committed instruction trace, cache miss, branch mis-prediction, and value change of status registers, provide many valuable information items for detecting malicious behavior. It is challenging to collect these information since it is difficult to send these high volume data from the domain of *CP* to *ASP*. In security-first architecture, many micro-architecture event collection agents are embedded into each major module in *CP*, and the collected information will be sent to an on-chip pre-processing engine via dedicated on-chip event monitoring fabric. The pre-processing engine will filter unnecessary or repeated information, and send the compressed valuable information to *ASP* for further analysis.

Another architecture-level feature is the asymmetric memory space, which enables the physical isolation between the Computation domain and the Security domain. CP and ASP have their own memory, chip sets and I/O devices. And the inter-domain communications are unidirectional. That is to say, *ASP* can access all resources including the internal status of *CP*, the memory and I/O devices in the Computation domain, while *CP* cannot access any information within the Security domain.

At micro-architecture level, security-first architecture adopts the classic general-purpose processor hardware optimizations including Out-of-Order execution and speculative technique to meet the performance and flexibility requirements of diverse security tasks. Besides these, unique hardware optimizations like heterogeneous computing are also considered in the micro-architecture design of *ASP*. For instance, typical operations like encryption/de-encryption, and pattern matching are selected as acceleration targets via domain-specific hardware accelerators.

Security mechanisms on ASP

The security tasks safeguard the execution of computational workloads. In general, three kinds of security tasks are running on *ASP*:

Malicious behavior active online detection and security inspection. Malicious behaviors like ROP and Flush+Reload attacks usually have unique patterns, which can be observed by analyzing the execution trace and other relevant micro-architectural events sent from *CP*. Thus, active online detection logic is deployed on *ASP* to recognize such malicious patterns. In addition, it is also allowed to specify actionable rules for security inspection. For example, the host memory may be scanned at specific frequency; and the contents of specified memory region where kernel data structures are located may be monitored to avoid rootkit attacks. *ASP* also has the active capability to capture each I/O operation, to discard illegal I/O operations, or even to close the related I/O channel according to the rules.

Secure computation offloading. *ASP* can provide a secure execution environment for legitimate applications. Selected codes and data are allowed to be encrypted offline in advance, and the encrypted codes and data can be sealed as a secure region with a signature. Once the secure region is dispatched to *ASP* via dedicated APIs, the internal codes and data will be decrypted and the integrity will also be verified by checking the signature. Then the codes start to run after necessary environment is initialized. During the execution, only the codes located in the same secure region can access its internal data. Since the

ASP is physical isolated from *CP*, the codes and data are not visible to any application or OS running on *CP*. The secure region is similar to an enclave of Intel SGX, and the difference is that the secure region runs on physical isolated *ASP*. Such mechanism provides a better protection against side-channel attacks.

Active Trusted computing. Traditional trust computing has two limitations. First, the core part of existing trusted computing model called Trusted Platform Module(TPM) is a passive chip intended to serve as a hardware root of trust for trusted infrastructure leading to software applications that are trusted. The TPM cannot actively initiate a trusted measurement, and it needs to be invoked by other compute units such as CPU. Therefore, once the computer has been compromised, the trusted measurement process is capable of being evaded. Second, although provably correct chains of trust from BIOS through several levels of operating systems is theoretically sound, the measurement technique used in conjunction with the TPM is vulnerable because it lacks of runtime changes detection methods. Here we introduce our active trusted model for computer devices, which can solve those two problems above. Our active trusted model consists of static trusted chain and dynamic trust chain. The static trusted measurement are performed when the computer is being started or a software application is being executed. For the integrity of the static trusted chain to remain intact, operating system kernel, modules, libraries and all the applications must be measured before its being executed. The dynamic trusted measurement is performed when the computer is running. It measures lots of factors such as the code segment and static data at the runtime. *ASP* is considered as the trust foundation of the active trusted model, and it can actively initiate a trusted measurement to establish the chain of trust that consists of multiple trust dependencies. Moreover, *ASP* can be used as the root of both static trusted chain and dynamic trusted chain.

Implementation options

In this section, three typical system-level implementations are discussed. They all follow the design philosophy of security-first architecture, and reflect different design considerations in terms of system overhead, complexity and performance. ASP can be implemented as an additional IP module on a chipset, an extra core on a processor chip, or a separate processor chip.

(1) Implementation inside a chipset. In Fig. 4, the ASP is located at the entry point of the chipset as an integrated IP and is attached to the existing on-chip fabric of the chipset. This implementation option makes it convenient for the ASP to analyze I/O traffic and effectively shutdown/re-open the I/O channel when requested. Following the principles of asymmetric memory space, the

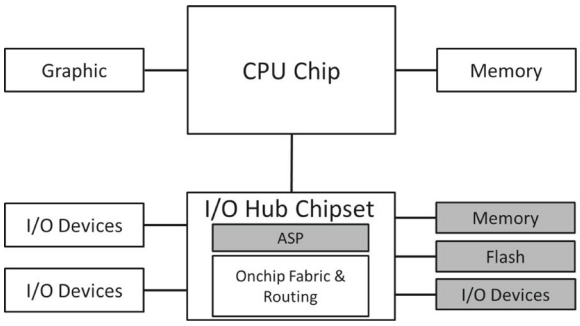

Fig. 4 Implement ASP in Chipsets

ASP has its own memory, disk, and I/O devices. An embedded DMA engine is used to scan the host memory on the CPU side. However, the chipset is normally I/O bounded. Without the ASP, the original chipset has an already complex internal architecture, which contains an on-chip routing fabric and several controllers for various I/O devices. It is difficult to increase the number of I/O pins in an I/O bounded chipset. The memory bandwidth for ASP is thus possibly limited. For this reason, Intel ME borrows memory from the CPU but this unfortunately introduces security risk. Moreover, due to the limited area available in the already crowded chipset, the ASP is left with a tiny processor which lacks the calculation power required by active defenses.

(2) Implementation inside a processor chip. As shown in Fig. 5, the ASP Core is integrated into the processor chip as an extra core. This is the most efficient way of implementing an ASP to monitor and analyze the on-chip micro-architecture behaviors on the CPU side. On-chip monitoring enables the timely identification and response to elusive attacks and malicious behaviors. On-chip monitoring also avoids transferring a large amount of micro-architecture (behavior) information off the chip, which is actually very challenging. Since the ASP is also utilized as an extra IP, it has similar performance and memory bandwidth limitations as the chipset implementation has.

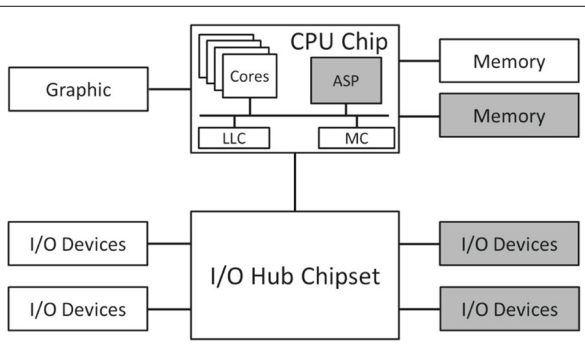

Fig. 5 On-chip implementation of ASP

(3) Implementation on a separate chip. As depicted in Fig. 6, the ASP is deployed as a separate chip. The ASP controls all system resources, monitors all CPU off-chip requests in fine granularity and collects all the CPU micro-architectural events. Compared to the other two implementations, in this implementation option the ASP has the largest asymmetric memory bandwidth. It is even possible to use additional hardware accelerators or GPUs for accerlating the security tasks. For example, online real-time deep neural network learning tasks can be boosted by using additional accelerators, which could greatly enhance the ability of monitoring and recognizing elusive attacks. In addition, the abundent computing power of the ASP and the isolated physical environment make it efficient to handle offloaded jobs such as protecting specific codes and data, and running encryption and decryption operations.

Architecture comparisons

This section compares security-first architecture with other architecture-level security techniques.

Many existing architecture-level security mechanisms have their specific uses. For examples, TPM is used to provide a chain of trust and measure the integrity of executable binaries. In contrast, the security-first architecture proposes a generic security platform with strong support for various defense mechanisms.

Given the big challenges mentioned in "Weaknesses of current countermeasures" section, "patching" hardware and the OS kernel in response to any new attack is a passive way of defense. In contrast, active defense mechanisms enabled by security-first architecture monitor suspicious behaviors at multiple layers ranging from chip to software applications, dynamically recognize the illegal operations, and physically shutdown or re-open the I/O channels.

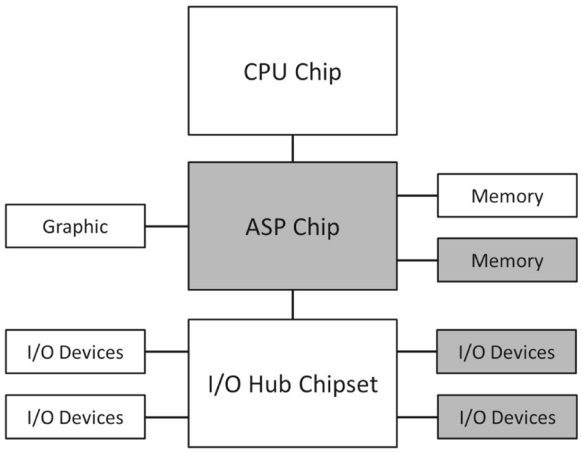

Fig. 6 Board-level implementation of ASP

SGX and Trustzone suffer from various side-channel attacks. In contrast, security-first architecture is immune to most if not all side-channel attacks thanks to its physical isolation. Different from the logic isolation designs in Trustzone and SGX, physical isolation significantly reduces side-channel information leakage. Numerous security mechanisms, such as ASLR at OS level, and CFI and buffer overflow protection at chip level, can be directly integrated into ASP (and the security tasks running on the ASP). Moreover, the security inspection agent running on the ASP can be thoroughly verified as the execution environment of ASP is fairly enclosed. This further enhances the security of ASP.

In terms of performance, first, since ASP has its own memory, disk and I/O resources, it does not occupy the resources belonging to the computation processors; therefore, the performance impact caused by ASP on the computation jobs is minimumized. Second, more powerful processor cores are preferred for ASP in security-first architecture, in order to meet the performance requirements of active defense jobs. This is an important difference between security-first architecture and the existing architecture-level security subsystems like AMD PSP and Intel ME.

Conclusion

Building a secure system atop the current computer architecture is fundamentally difficult. The fast-growing complexity in software inevitably leads to exploitable security vulnerabilities. IP (Intellectual Property) reuse and the separated ASIC manufacture procedure with multiple vendors make it impossible to thoroughly inspect the whole design and the manufacture procedures. Some architecture-level vulnerabilities are difficult to remove due to the conflicting interests between performance and security. In the meanwhile, attacks are constantly evolving with new means to outwit existing defenses. To meet this challenge, this paper proposes security-priority architecture, a concept which separates the security tasks from the normal tasks by physical isolation. In systems built based on this concept, traditional processors (i.e., *Computation Processors*) are monitored and protected by *Active Security Processors*. The two types of processors execute on their own physically-isolated resources, including memory, disks, network and I/O devices. The physically isolated *Active Security Processors* are a key component for security-first architecture, which are provided with dedicated tightly-coupled channels to access all the resources of the Computation Processors but not vice versa. This allows the Active Security Processors to actively detect and tackle malicious activities in the Computation Processors with minimum performance degradation while protecting themselves from the attacks launched from the Computation Processors thanks to the resource isolation.

This paper introduces three implementations of *Security-first Architecture*, and makes comparasions with other classic security mechanisms. *Security-first Architecture* can remedy the deficiencies of existing security mechanisms and provide a new direction worth exploring.

Authors' contributions

DM proposed the concept and made the key design of security-first architecture. RH drafted the paper. GS, BT, AY, ZZ, XJ and RH are in charge of different aspects of this architecture and the implementations of its key mechanisms. PL polished the whole paper. All authors read and approved the final manuscript.

Competing interests

The authors declare that they have no competing interests.

Publisher's Note

Author details

[1]Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China. [2]Pennsylvania State University, Old Main, 16801 State College, PA, USA.

Received: 4 January 2018 Accepted: 17 April 2018
Published online: 05 June 2018

References

Abadi M, Budiu M, Erlingsson U, Ligatti J (2005) Control-flow integrity. In: Proceedings of the 12th ACM Conference on Computer and Communications Security. ACM, Alexandria. pp 340–353

Advanced Micro Devices Inc (2018) An update on AMD processor security. https://www.amd.com/en/corporate/speculative-execution

Advanced Micro Devices Inc (2018) Full security solutions that locks you down, not in. https://www.amd.com/en/technologies/security

ARM Limited (2009) Building a secure system using TrustZone technology. http://infocenter.arm.com/help/topic/com.arm.doc.prd29-genc-009492c/PRD29-GENC-009492C_trustzone_security_whitepaper.pdf

Avanzi R (2017) The QARMA block cipher family. IACR Transactions on Symmetric Cryptology 1:4–44

Bletsch T, Jiang X, Freeh VW, Liang Z (2011) Jump-oriented programming: a new class of code-reuse attack. In: Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security. ACM, Hong Kong. pp 30–40

Bogowitz B, Swinford T (2004) Intel® active management technology reduces it costs with improved PC manageability. Technol@ Intel Mag. https://www.intel.com/content/dam/www/public/us/en/documents/solution-briefs/optimize-management-and-security-of-client-devices-solution-brief.pdf

Burow N, Carr SA, Nash J, Larsen P, Franz M, Brunthaler S, Payer M (2017) Control-flow integrity: Precision, security, and performance. ACM Comput Surv 50(1)

Carlini N, Wagner D (2014) ROP is still dangerous: Breaking modern defenses. In: Proceedings of the 23rd USENIX Conference on Security Symposium. ACM, San Diego. pp 385–399

Checkoway S, Davi L, Dmitrienko A, Sadeghi A-R, Shacham H, Winandy M (2010) Return-oriented programming without returns. In: Proceedings of the 17th ACM conference on Computer and communications security. ACM, Chicago. pp 559–572

Common Vulnerabilities and Exposures (2018) Linux kernel: Vulnerability statistics. https://www.cvedetails.com/product/47/Linux-Linux-Kernel.html?vendor_id=33

Datenschutz HT, Pataky D (2017) Intel management engine. https://bitkeks.eu/docs/intelme-report.pdf/

David C, Kent Y, Ryan C, David S, Leendert D (2008) A practical guide to trusted computing. IBM Press, first ed., Boston

Davi LV (2015) Code-Reuse Attacks and Defenses. PhD thesis. Technische Universität, Darmstadt

Ehrenfeld JM (2017) Wannacry, cybersecurity and health information technology: A time to act. J Med Syst 41(7):101

Evans I, Long F, Otgonbaatar U, Shrobe H, Rinard M, Okhravi H, Sidiroglou-Douskos S (2015) Control jujutsu: On the weaknesses of fine-grained control flow integrity. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security. ACM, Denver. pp 901–913

Frantzen M, Shuey M (2001) Stackghost: Hardware facilitated stack protection. In: Proceedings of the 10th Conference on USENIX Security Symposium. ACM, Washington. pp 5–5

Gellman B, Poitras L (2013) US intelligence mining data from nine U.S. internet companies in broad secret program. The Washington Post. https://www.sanders.senate.gov/newsroom/must-read/us-intelligence-mining-data-from-nine-us-internet-companies-in-broad-secret-program

Halfond WG, Viegas J, Orso A (2006) A classification of SQL-injection attacks and countermeasures. In: Proceedings of the IEEE International Symposium on Secure Software Engineering, vol 1. IEEE, Washington. pp 13–15

Intel Corporation (2006) LaGrande technology preliminary architecture specification. http://kib.kiev.ua/x86docs/SDMs/315168-002.pdf

Intel Corporation (2016) Control-flow enforcement technology preview. https://software.intel.com/sites/default/files/managed/4d/2a/control-flow-enforcement-technology-preview.pdf

Intel Security Center (2017) Intel active management technology, intel small business technology, and intel standard manageability escalation of privilege. https://security-center.intel.com/advisory.aspx?intelid=INTEL-SA-00075&languageid=en-fr

Kim Y, Daly R, Kim J, Fallin C, Lee JH, Lee D, Wilkerson C, Lai K, Mutlu O (2014) Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors. ACM SIGARCH Computer Architecture News 42(3):361–372

Kocher P, Genkin D, Gruss D, Haas W, Hamburg M, Lipp M, Mangard S, Prescher T, Schwarz M, Yarom Y (2018) Spectre attacks: Exploiting speculative execution. ArXiv e-prints. https://spectreattack.com/spectre.pdf

Lin X, Zavarsky P, Ruhl R, Lindskog D (2009) Threat modeling for CSRF attacks. In: International Conference on Computational Science and Engineering. IEEE, Vancouver. Vol. 3. pp 486–491

Lipp M, Schwarz M, Gruss D, Prescher T, Haas W, Mangard S, Kocher P, Genkin D, Yarom Y, Hamburg M (2018) Meltdown. ArXiv e-prints. https://meltdownattack.com/meltdown.pdf

Liu F, Yarom Y, Ge Q, Heiser G, Lee RB (2015) Last-level cache side-channel attacks are practical. In: Proceedings of the IEEE Symposium on Security and Privacy. IEEE, San Jose. pp 605–622

McKeen F, Alexandrovich I, Berenzon A, Rozas CV, Shafi H, Shanbhogue V, Savagaonkar UR (2013) Innovative instructions and software model for isolated execution. In: Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy. ACM, Tel-Aviv. pp 10:1–10:1

Mozilla Firefox (2018) Project summary. https://www.openhub.net/p/firefox

Oleksenko O, Kuvaiskii D, Bhatotia P, Felber P, Fetzer C (2017) Intel MPX explained: An empirical study of intel MPX and software-based bounds checking approaches. arXiv preprint arXiv:1702.00719. https://arxiv.org/pdf/1702.00719.pdf

Qualcomm Technologies Inc (2017) Whitepaper: Pointer Authentication on ARMv8.3. https://www.qualcomm.com/documents/whitepaper-pointer-authentication-armv83

Schuster F, Tendyck T, Liebchen C, Davi L, Sadeghi AR, Holz T (2015) Counterfeit object-oriented programming: On the difficulty of preventing code reuse attacks in C++ applications. In: Proceedings of the IEEE Symposium on Security and Privacy. IEEE, San Jose. pp 745–762

Seaborn M, Dullien T (2015) Exploiting the DRAM rowhammer bug to gain kernel privileges. In: Black Hat. UBM, Las Vegas. pp 7–9

Shacham H (2007) The geometry of innocent flesh on the bone: Return-into-libc without function calls (on the x86). In: Proceedings of the 14th ACM Conference on Computer and Communications Security. ACM, Alexandria. pp 552–561

Shin Y, Meneely A, Williams L, Osborne J (2011) Evaluating complexity, code churn, and developer activity metrics as indicators of software vulnerabilities. IEEE Trans Softw Eng 37(6):772–787

Sinharoy B, Swanberg R, Nayar N, Mealey B, Stuecheli J, Schiefer B, Leenstra J, Jann J, Oehler P, Levitan D, Eisen S, Sanner D, Pflueger T, Lichtenau C, Hall W, Block T (2015) Advanced features in IBM POWER8 systems. IBM J Res Dev 59(1):1–1

Szekeres L, Payer M, Wei T, Song D (2013) Sok: Eternal war in memory. In: Proceedings of the IEEE Symposium on Security and Privacy. pp 48–62

Trusted Computing GroupAdministration (2008) Trusted Platform Module (TPM) summary. https://trustedcomputinggroup.org/trusted-platform-module-tpm-summary/
Trusted Computing Group Incorporated (2003) TCG specification architecture overview. https://www.trustedcomputinggroup.org/wp-content/uploads/TCG_1_4_Architecture_Overview.pdf
van der Veen V, dutt Sharma N, Cavallaro L, Bos H (2012) Memory errors: the past, the present, and the future. In: Proceedings of the 15th ACM International Conference on Research in Attacks, Intrusions, and Defenses. Springer, Amsterdam. pp 86–106
Wang W, Chen G, Pan X, Zhang Y, Wang X, Bindschaedler V, Tang H, Gunter CA (2017) Leaky cauldron on the dark land: Understanding memory side-channel hazards in SGX. In: Proceedings of the ACM SIGSAC Conference on Computer and Communications Security. ACM, Dallas. pp 2421–2434
Whalen S (2001) An introduction to ARP spoofing. Node99 [Online Document]. http://www.madchat.fr/reseau/arp/intro_to_arp_spoofing.pdf
Wikimedia Foundation Inc (2018) AMD Platform Security Processor. https://en.wikipedia.org/wiki/AMD_Platform_Security_Processor
Wojtczuk R, Rutkowska J (2017) SoC and CPU system-wide approach to security. https://www.arm.com/products/security-on-arm/trustzone
Xu Y, Cui W, Peinado M (2015) Controlled-channel attacks: Deterministic side channels for untrusted operating systems. In: Proceedings of the IEEE Symposium on Security and Privacy. IEEE, San Jose. pp 640–656
Yang K, Hicks M, Dong Q, Austin T, Sylvester D (2016) A2: Analog malicious hardware. In: Proceedings of the IEEE Symposium on Security and Privacy. IEEE, San Jose. pp 18–37
Zhang N, Sun K, Shands D, Lou W, Hou YT (2016) Truspy: Cache side-channel information leakage from the secure world on ARM devices. IACR Cryptol ePrint Arch:980
Zhang N, Sun H, Sun K, Lou W, Hou YT (2016) Cachekit: Evading memory introspection using cache incoherence. In: Proceedings of the IEEE European Symposium on Security and Privacy. IEEE, Saarbrücken. pp 337–352