Total Platform Cyber Protection

1 Introduction

Traditional security defenses for the computing environment have focused on securing the *border* of the network, as this is the entry place for most attackers. However, even with sophisticated border technologies in place, there are constant attacks and data breaches against our networks. The goal of this work is to survey the state of security for the entire computing platform, in order to identify specific areas that are underserved by the research community and that require additional research and investment.

Modern computing infrastructure serves a wide array of needs, with diverse technologies and implementations: embedded systems, cloud computing, sensor networks, desktops, mobile devices, and industrial control systems. Rather than consider each of these computing platforms independently, in this work we abstracted the computing platform into several layers, each of which are applicable to every computing infrastructure. In particular, we analyzed the security research performed at the following layers of the computing platform: hardware, firmware, bus, hypervisor, operating system, application, and network layers.

The goal of this work is to focus research effort on *securing the entire computing platform.* An attack must effectively target a specific vulnerability in a specific layer of the computing stack, and an attacker uses that vulnerability to establish persistence on the machine, potentially attacking the underlying computing layers of the same machine or leveraging their place in the network to attack other machines. Therefore, if we wish to increase the security of our computing systems, and reduce the number and scope of security breaches, it is essential that we encourage focus on novel ideas, algorithms, and techniques to secure every level of the computing stack.

In this report, we will first discuss the computing stack, and, for every layer, we will describe the layer, identify primary threats against the layer, and summarize the research on attempts to secure that layer, and finally we will determine areas that require further research investment to ensure the security of the layer.

2 Background

Here, we consider the diverse array of computing platforms as a number of abstracted layers: hardware, firmware, bus, hypervisor, operating system, application, and network layers. In this way, we can categorize the research efforts that target each layer.

The layers are arraigned from lowest to highest layer. The lower layer has more control over the computing system, while the upper layers have less control. For instance, the Hardware layer has total control over the physical hardware and memory of the machine, while an application running in the Application layer can only see a portion of the machine's memory (because of the virtual memory used by the Operating System).

A typical compromise, in relation to the proposed layers, is the following: A vulnerability in an application is targeted, thus allowing the attacker full control of the application layer. From here, the attacker can scan the other applications on the network and attempt to exploit those applications, thus spreading horizontally throughout the network. Alternatively, as the application layer is the least privileged and least persistent of the layers, the attack can attempt to further the compromise on the lower layers of the machine. For instance, the attacker can exploit the hypervisor layer to compromise the host hypervisor. From this vantage point, the attacker can transparently extract information from all the other guest Operating Systems. More insidiously, the attacker can even exploit the firmware level, which allows the attacker to persist even after a complete reinstall of the operating system.

In our abstract model, the lower levels have more control over the computing system, yet they have a smaller attacker surface. However, the lower levels can be attacked by exploiting the upper levels. Also, the upper levels trust the abstractions provided by the lower levels. For instance, the application-layer visualization of an industrial control system trusts that the data reported by the hardware layer is accurate. If an attacker is able to control and manipulate the hardware to falsely report the state of the system, the entire integrity of the system can be compromised.

The interconnected nature of the layers means that the security of the entire layer must be a priority. However, at the same time, the security improvement of any layer will have an amplifying effect on the security of the whole system. For instance, improvement to the security of the application layer decreases the chance of an application compromise, which further decreases the risk of the attack propagating throughout the network and through the other layers.

3 Hardware

Hardware underpins all of modern computing. While most software is constructed and written in such a way as to abstract the peculiarities of the underlying hardware, the software cannot execute without hardware. In our abstracted model of the computing stack, hardware is at the bottom, which means that it has complete control of the computing system.

For hardware here, we consider all types of hardware in the computing system: CPUs, SoCs, or any other integrated circuit (IC) that performs computation or communication on behalf of the upper computing layers. This also includes networking cards and other peripheral devices (Bluetooth or USB devices), which may have their own special purpose embedded hardware.

A key difficulty in analyzing the security concerns of hardware is reverse engineering the workings of the hardware. Traditional techniques to reverse engineer hardware require destroying the chip, and if a single hardware component is destroyed to reverse engineer, there is no guarantee that other hardware is identical to the reverse engineered chip.

3.1 Threats

As the lower-most level of the computing stack, all layers implicitly trust that the hardware is executing correctly. Therefore, if an adversary is able to maliciously control the hardware execution, they can compromise all layers of the computing stack. Thus, the security of the hardware is of paramount importance.

The main threat considered by the literature on hardware is the threat of Trojan logic. This is malicious logic that is introduced into the hardware chip by an adversary. Trojan introduction can be done at several points of the hardware lifecycle:

- **Design**. A malicious insider can incorporate a Trojan into the design of the hardware.

- **Third-party IP**. Oftentimes, hardware designers will take advantage of the functionality of third-party components, using them as a black-box. Thus, a third-party can include malicious Trojan logic.
- **Compilation of Design**. When the design of the hardware is compiled, but before it is sent for fabrication, the design could be changed to incorporate the Trojan logic. Note that this differs from Design-time, because the Trojan logic is not included in the design. Consider the case of malware on a hardware designer's machine, which changes the design when it is compiled.
- **Manufacture**. A malicious manufacturer can introduce Trojan logic into hardware after the design is received but before the hardware is fabricated.
- **Delivery**. A malicious hardware component containing Trojan logic can be swapped with a benign component after fabrication but before delivery and installation.

The malicious Trojan logic behavior comes in three different forms:

- **Trigger-based**. The malicious behavior or logic is only expressed when certain conditions are met. Consider a Trojan logic in a CPU, which changes the executing code to Ring 0 (with the most privileges) when a specific sequence of instructions are issued. The trigger can be internal to the chip, such as in the CPU case, or could be external, for instance a certain radio frequency will trigger the malicious code.
- **Constant**. The malicious behavior or logic is continually executed and subtly changes the output of the hardware in a way that compromises security. Consider a Trojan logic of a Wi-Fi card, which leaks the Wi-Fi passphrase by embedding it in the operating parameters of the Wi-Fi radio.

These features of Trojan logic in hardware drive the research that attempts to detect them.

3.2 Research

Most research into hardware Trojans deal with using side-channel analysis to identify the malicious Trojan behavior [4, 10–12, 26, 53, 56, 59, 61, 92, 114, 121, 122, 137, 138]. This research uses power analysis, heat analysis, or other side-effects to determine that the hardware is executing abnormal logic. Unfortunately, this typically requires building a model based on a "golden" or known-good hardware, which is often not accessible. Furthermore, as chips become more complex, the Trojan logic become an increasingly smaller portion of the chip, which in turn reduces the side-channel behavior of the chip significantly.

Other research has demonstrated the attacks that are possible by actually implementing Trojan behavior [91, 93, 102, 122, 182, 222]. This line of research shows that Trojan logic is feasible for an attack.

Of particular practical interest is a description of *actual Trojan logic* discovered in a the ProASIC3 Flash FPGA military chip [173]. While in this case it was not proved that the logic introduced to the chip was intentionally malicious, nevertheless there existed functionality in the hardware that was explicitly denied by the hardware designer.

Another avenue of research is to change the design of the chip itself, in order to improve the testability of the chip or to increase the effectiveness of the side-channel analysis [17, 29, 53, 80, 115, 150, 155, 157, 161, 201]. This line of research attempts to rethink designing the hardware itself. However, a major drawback is that this can increase the cost of the hardware and it must be somehow mandated to be used in COTS components.

JTAG is a main avenue of reverse engineering hardware, so some work has investigated the problem of securing JTAG access to hardware [151, 156]. The main idea is to allow only authenticated users to have access to the JTAG functionality, however this increases the cost and complexity of implemented JTAG.

Another approach is to change the design of the chip to support runtime monitoring [200]. In this way, the hardware itself is monitoring itself, rather than external analysis of side-channels. However, this has all the same drawbacks of changing the design of hardware, in addition to not being robust in the face of a powerful attacker who is able to alter the design.

Recent approaches suggest to use machine-learning techniques to identify the Trojan logic [78]. Another approach attempts to use a formal verification technique in order to prove that no malicious logic was added to the hardware [76].

Finally, there are several surveys conducted on hardware security, which provide a good overview of the field [73–75, 193, 202].

Important problems include how to deal with counterfeit chips, better non-destructive examination methods, continuous monitoring of the integrity not just at boot time, and detection of backdoors and "undocumented features". Hardware design can also be done to aid this process, e.g., having some part of the chip consume more power to make it more identifiable for authentication purposes. Other possibilities include running multiple chips with different implementations side-by-side and comparing results to detect unexpected behavior.

3.3 Suggested Focus Areas

A key problem with hardware security is non-destructive IC examination, identification, and authentication. How to ensure that the chip that is designed is what is actually fabricated, with no additional logic.

An additional neglected area is how to detect malicious Trojan logic in third-party components. This would require some type of static analysis or simulated dynamic analysis of third-party components, which could detect Trojan logic.

Another approach that could be effective is how to properly test the functionality of hardware after it has been fabricated. This could include changing the design of the chip to allow better/more guided fuzz testing of the chip itself, in order to discover the Trojan logic.

Finally, a prevention approach could be taken, to monitoring the upper layers of the computing stack for attempts to exploit the underlying hardware. This can also include changing or altering the firmware layer in such as way that it is not possible to exercise the Trojan logic, potentially using randomization techniques similar to ASLR.

4 Firmware

Firmware is the layer of software that sits and interacts directly with the hardware. Typically, the firmware is controlled from the upper layers by device drivers in the Operating System.

Many different hardware contain firmware, for instance the boot BIOS on a motherboard is a form of firmware, embedded systems often contain specific firmware, hardware peripheral such as network cards contain firmware as well. We consider all such areas as firmware.

The key traits of firmware is that, although it is binary code, it is operating/interacting directly with the hardware, so there are no OS-style abstraction layers such as virtual memory, well-defined system calls, or hardware abstraction. Therefore, each firmware is binary code with is specifically tailed to a specific hardware platform.

Due to the hardware-specific nature of firmware, *reverse engineering* is critical to understanding the firmware. In addition, reverse engineering is particularly difficult because source code is

frequently unavailable, and, vendors desire to protect their intellectual property, therefore, they actively obfuscate their firmware. All this means that, extracting semantically useful information from firmware is a key first step to performing any security analysis of the firmware.

4.1 Threats

As the lowest-level of software code in the computing stack, threats to firmware can compromise the security of the entire machine. Firmware security is concerned with two main threats: (1) vulnerabilities in the firmware code itself which allow an attacker to execute arbitrary code with the capabilities of the firmware and (2) firmware with intentional Trojan (backdoor) capabilities.

Vulnerabilities in the firmware code allow an attacker to execute arbitrary code with the permissions of the firmware. The class of vulnerabilities are not unique to firmware, as they are the same class of vulnerabilities that infect traditional binary software (buffer overflows, logic flaws, etc.). However, firmware typically has full control over the entire computing system, therefore vulnerabilities in firmware are much more severe than those in the application layer. In addition, if an attacker is able to persist their exploit, because the compromise occurs at such a low layer of the computing system, the attacker can persist *even after reinstalling the operating system or application*.

The attack surface for firmware varies depending on where it is used. In the most general case, the attacks can come from the upper layers, specifically the Operating System or Hypervisor layers, as these are the layers of the computing stack that can directly communicate to the firmware. However, other firmware can be attacked either directly through the Hardware layer itself, or through the data being transmitted through the firmware from the hardware. For instance, a network card firmware could be attacked through network traffic that the firmware processes.

Traditional vulnerability analysis is not effective on firmware, due to the specificity of the firmware for the specific hardware and the lack of a well-defined operating system abstraction for the firmware code. In essence, each firmware is a custom binary with little in the way of assumptions that vulnerability analysis can take advantage of.

Trojan capabilities in the firmware itself can allow a system to be remotely controlled or accessed. This is similar to a vulnerability, in that an attacker is able to control the firmware, however the key difference lies in the fact that Trojan behavior will be intentionally obfuscated by an attacker, whereas a vulnerability is accidentally introduced by a benign developer. Therefore, Trojan behavior is more difficult to detect, because there is an active adversary who is intentionally obfuscating the behavior.

Vulnerabilities in the firmware and Trojan capabilities are related because both require *understanding* the behavior and semantics of the firmware code. This is why reverse engineering is such a critical component of securing the threats against the firmware layer: in order to detect either vulnerabilities or Trojan functionality, an automated tool must first understand the semantics of the particular firmware/hardware combination.

4.2 Research

Researchers have attempted to identify real-world attacks on firmware code [15, 42, 44, 57, 132, 183, 197, 217, 218]. These attacks are particularly effective, and researchers have shown that these are able to persist through a reboot and cannot be detected by the upper layers.

Other approaches have looked at how to prevent the tampering of firmware [3, 225, 226]. This protects from the threat of tampering or altering the firmware, however it does not address vulnerable firmware.

Another approach has been to include runtime monitoring into the firmware, so that compromises can be detected [43, 45]. This approach is further complicated by the lack of abstractions in the firmware environment.

Recent work has looked at the problem of updating the design of the firmware so that it has security features, such as memory handling and avoiding information leakage [100, 106]. However, forcing firmware developers to use such features is an open issue.

Very few work has looked at how to reverse engineer and understand the firmware [216]. This problem is of particular interest, as understanding the semantics of the firmware and how it interacts with hardware is a key challenge.

Recent work has looked at how to detect vulnerabilities in firmware [40, 52, 171]. One of these has an excellent website[1] that contains the results of their reverse engineering and vulnerability analysis [40].

4.3 Suggested Focus Areas

Of all the layers, firmware has received the least attention from the research community. Therefore, we believe that focus on this research area will yield significant dividends.

The major issue with firmware vulnerabilities is how to decouple firmware from the hardware, potentially using emulation of the hardware, in order to aid the analysis of the firmware.

In a similar vein, there is need for automated reverse engineering tools, including analysis of memory layouts, code modules, even answering the simple question of what in a firmware binary is code and what is data.

Finally, even if we find vulnerabilities or Trojan logic in a firmware, there is still the unanswered questions of how to patch or update the functionality of the firmware while still guaranteeing the behavior of the firmware. This is particularly complicated due to the real-time requirements of the firmware and the potentially concurrent executions.

5 Bus and Interconnect

The Bus and Interconnect is an abstract layer of the computing system which allows the firmware and hardware to communicate with each other. Different computing architectures have different technical implementations of the bus and interconnect. For instance, the communication network in the automotive system, which has received significant attention from the research community, uses a CAN bus to communicate. Modern motherboards us either a Northbound or Southbound bus to communicate between the CPU and the other peripheral devices, such as USB.

5.1 Threats

Depending on the technical architecture of the bus and interconnect layer, the bus can be used either purely for communication or for other functionality as well. For instance, the CAN bus is only used to allow different hardware components to communicate. However, on modern motherboards, techniques such as Direct Memory Access (DMA) allow a peripheral device that is connected to the bus to directly access the system's memory, without going through the CPU.

In the case of the bus and interconnect being used as a communication medium, then the classic threats are the same security concerns as other mediums of communication: communication integrity and confidentiality. A malicious device on the bus can impersonate another device and send messages on the bus as the impersonated device. On an automotive system, researchers have demonstrated that by compromising a single device attached to the system's CAN bus allows an

[1] `http://firmware.re`

attacker to have complete control over all functions of the automobile, by impersonating messages on the CAN bus. In addition to impersonating messages, an attacker on the bus and interconnect layer can listen to messages on the bus, potentially eavesdropping on sensitive messages.

If the bus and interconnect layer offers advanced functionality, such as the DMA example, then devices that an attacker is able to connect to the bus could take advantage of this functionality. In the case of DMA, this means that any device on the bus has full access to the physical memory of the system, therefore the attacker has complete control over the upper layers of the computing system (hypervisor, operating system, and application).

The difficulty of addressing the threats of the bus and interconnect layer is the diversity and specificity of the specific bus and interconnect technology. Often, these were created without consideration for security needs, therefore, as a communication mechanism they lack authentication from the design. In addition, as the bus and interconnect layer cause different hardware to communicate, all hardware must agree on the protocol to talk on the bus. Therefore, adding security to the bus and interconnect protocol is difficult in legacy systems.

5.2 Research

A major area of research in the Bus and Interconnect layer is work that shows the danger of a bus and interconnect that allows malicious devices to either impersonate devices on the bus or leverage bus capabilities, particularly in the automotive area [23, 31, 83, 107, 158, 162].

The other major area of research is to apply encryption to devices on the bus and interconnect [71, 79, 88, 167, 180, 189]. In this way, malicious devices on the bus cannot impersonate other devices. However, a key problem with this research direction is how to distribute the keys of the devices, which is typically done by distributing them in hardware. Unfortunately, this approach is brittle, as it disallows new devices from accessing the bus and interconnect.

Other research has investigated how to ensure the integrity of devices attached to the bus and interconnect using attestation techniques [118, 119]. These techniques attempt to ensure that only authorized and unmodified devices are attached to the bus and interconnect. However, this research has the problem of how to allow third-party devices to access the bus.

A key problem is how to perform secure updates to devices attached to the bus and the interconnect [113]. This is a challenge that also has similarities with the firmware layer.

In order to discover malicious communication, researchers have utilized data flow tracking techniques to discover how the data flows between devices connect to the bus and interconnect layers [166]. Using these techniques, the researchers are able to identify malicious flows or potential vulnerabilities by analyzing the way that data flows throughout the system.

Other researchers have explored modeling either the system itself or the behavior of the devices on the bus [55, 179]. If the device behavior or the system behavior does not match the learned model, then an alert is triggered. In this way, these techniques are similar to anomaly detection techniques, with similar benefits and drawbacks.

In order to address the issue of devices on the bus able to access capabilities unauthorized, researchers have tackled the similar problem of reducing the Trusted Computing Base, essentially reducing the functionality of the bus and interconnect layer [199, 219, 225]. In this way, if the functionality is not there on the bus, then a malicious device cannot take advantage of it.

Another approach is to leverage hardware features of the underlying bus and interconnect system to reduce the impact of an attack [208, 209]. However, as these hardware functionality must be implemented, it's impact in legacy systems is limited.

Finally, other research in this area has performed surveys of the research on this area [104, 181, 194, 210–212, 223].

5.3 Suggested Focus Areas

Further research in this area should focus on enabling better monitoring of the runtime of devices attached to the bus or interconnect. There is also the need to support logging of the bus or interconnect activity. This will enable to detection of a malicious device on the bus or interconnect.

Another promising area of research is into the automated filtering of the bus communication, potentially allowing an administrator to define a policy for a specific bus or interconnect system. This would be particularly effective if it can be implemented on legacy bus and interconnect systems, with no modification to the devices.

A different approach would be to secure the bus by only allowing authorized devices to attach to the bus. This could be a white-listing based approach or other policy-driven approach. However, the key question is how to identify a device, particularly a third-party device.

While there has been some research on encrypting the communication on the bus, most has focused on pre-distributing the keys to devices on the bus. This approach is brittle, as new devices cannot be added to the bus easily. Therefore, further approaches should be investigated, such as using attribute-based encryption to be able to encrypt and distribute messages on the bus without prior sharing of keys.

6 Hypervisor

The purpose of the hypervisor layer is to provide virtualization techniques for the upper layers of the computing stack. In essence, the Hypervisor layer provides a virtualization or emulation of the lower layers of the computing stack, so that multiple Operating Systems, which each assume that the are the sole users of the underlying hardware, can run concurrently on the same hardware.

For discussing the Hypervisor layer, we use the term *host* to define the hypervisor itself (which, in some cases can be an Operating System) and the *guest* is the Operating System that is running in the hypervisor. A host can run multiple guest Operating Systems.

Hypervisors have a number of benefits, from allowing greater sharing and usage of hardware resources, to improving the isolation of the Operating Systems. The security benefits of using Hypervisors comes from each guest Operating System being isolated from the other. In theory, this means that a compromise of a guest Operating System will be isolated to only that Operating System, and the attack should not spread to other guest Operating Systems.

6.1 Threats

The central threat to the Hypervisor layer is from attackers in a guest Operating System exploiting a vulnerability in the hypervisor itself to gain access to the actual hardware or to propagate their attack to the other guest Operating Systems that are running in the hypervisor. As the Hypervisor layer is essentially emulating the hardware for the guest operating systems, it has a large attack surface. Furthermore, modern hypervisors, in an attempt to increase the performance of the hypervisor, frequently allow guest Operating Systems direct access to the hardware. If the hardware does not provide for full isolation (i.e., it is not hypervisor-aware), then an attacker can leverage this hardware to tamper or extract sensitive information from other guest Operating Systems.

In modern hypervisors, the hypervisors frequently provide isolation from a well-behaved guest Operating System, as the guest Operating System is assumed to be benign. This assumption can be unfounded, as the guest Operating System can be hostile and can attempt to exploit or crash the hypervisor.

Another threat actually stems from the Hypervisor itself being untrusted. In the cloud computing environment, the guest Operating System does not trust the underlying Hypervisor, as it is

controlled by a third-party. In this case, the Hypervisor can spy on the guest or alter the memory of the guest.

6.2 Research

One area of research is to leverage the vantage point of the Hypervisor to monitor the guest for security violations [13, 99], intrusion detections [34, 58, 65, 66, 108], checking the integrity of the guest Operating system [133, 168], and identifying rootkits present in the guest Operating System [70, 95]. This work assumes that the guest can be compromised and that the Hypervisor is trusted, which, due to vulnerabilities in the Hypervisor, is not always true.

A significant problem in monitoring or otherwise inspecting a guest Operating System from the hypervisor layer is the *semantic gap* problem, where the Hypervisor has no outside knowledge of the guest-specific Operating System constructs. Therefore, research has investigated how to overcome this semantic gap [38, 94, 178].

Further work in this line has approached the problem of how to monitor the hypervisor itself by leveraging hardware features [9].

In order to build a secure Hypervisor, the lower layers of the computing system must also be trusted. Therefore, research into establishing a Trusted Computing Base, which validates the integrity of the lower levels for the guest Operating System [5, 160, 172] is important for ensuring the security of the Hypervisor.

One problem of using the Hypervisor as a security primitive is that it is part of the Trusted Computing Base for the upper layers of the computing platform. Therefore, attempts have been made to reduce the trusted base of the Hypervisor itself [39, 135, 170, 206].

Other research has looked at creating a secure Hypervisor [16, 153, 185, 198] with the possibility of adding functionality to the Hypervisor to support closed guest Operating Systems [64], offering applications running in the guest Operating System to hide information from the guest Operating System [8, 34, 89, 90, 186, 187, 214, 221], secure information flow between the guest Operating System drivers and I/O devices [35], or designing the hypervisor so that it can prove that it did not tamper with the guest Operating System [72].

Another approach to improve the security of the Hypervisor is to remove the duplicated components from the guest Operating System layer [103, 126], which effectively reduces the attack surface of the Operating System layer.

Another interesting approach is to run traditional network middleware boxes, such as firewalls and Intrusion Detection Systems, as a guest Operating System of a Hypervisor [131].

Finally, there have been attempts to introduce traditional application-level security features to Hypervisors, in order to reduce or prevent vulnerabilities, such as control-flow integrity [204] or manual vulnerability analysis [146].

6.3 Focus Areas

As the hypervisor is typically a monolithic trusted entity, research should be performed to reduce the trust in the hypervisor. One avenue could be to have fine grained access control within the hypervisor, so that different policies can be set to the guests hosted by the hypervisor.

In a similar area, we believe that further research into reducing the attack surface of the hypervisor is necessary. Specifically, the hypervisor can be customized based on the applications and workload necessary.

Hypervisors, despite their obvious benefits, have not been applied to real-time systems. Further research must be done to allow reliable and robust hypervisors in a constrained real-time computing environment.

Testing hypervisors is currently a difficult and essentially manual process. There is currently no equivalent to fuzz testing a hypervisor implementation, possibly because there are no models for attacks against a hypervisor. Further research in increasing the testability and automated analysis of hypervisor reliability would improve security.

7 Operating Systems

The Operating System layer is responsible for ensuring that multiple applications are able to concurrently run, without interfering with each other. In addition, the Operating System layer is also responsible for providing a unified abstraction of the underlying hardware layer for the application layer. Therefore, in terms of security, the Operating System layer has more privileges than the application layer, as the Operating System controls the execution and scheduling of processes, the access control policies that allow one application to hide information/data from the other processes.

7.1 Threats

The main threat to the Operating System layer is an attacker compromising the security of the Operating System. There are two main avenues to how this compromise can occur: (1) an attacker exploits a vulnerability in the Operating System from the Application layer or (2) an attacker compromises a lower layer (for instance, modifying the Operating System code directly to the hardware's Hard Drive).

Vulnerabilities in the Operating System allow an attacker to have complete control over the other application running in the operating system. In addition, the attacker can use their vantage point at the operating system to attempt to escalate their attack to the hypervisor or lower layers, such as the firmware. Vulnerabilities in the Operating System result from programming errors that allow an attacker to gain control of the Operating System code or data. These vulnerabilities are traditional memory corruption vulnerabilities or even logic flaws.

A main concern in Operating System security is the presence of rootkits, which are malicious modifications to the Operating System which an attacker uses to achieve persistence on the system. In addition, attackers use rootkits to hide from other applications on the system. For instance, the rootkit will modify the operating system code such that the attacker's processes and files are hidden from the administrator or defensive software installed on the Operating System, such as an anti-virus solution.

Direct access to the hardware can allow an attacker to compromise the Operating System. It is in this way that an attacker can propagate their compromise from the lower layers of the computing platform to the upper layers. The attacker has similar capabilities as when exploiting a vulnerability in the Operating system.

7.2 Research

Research into Operating System security has a long and rich history, which has looked at design flaws in Operating Systems [7]. Further research looked at reducing the permissions of the *root* users, essentially compartmentalizing the privilege of the Operating System [97, 112].

Rootkits attempt to persist the attacker's compromise of the operating system and allow them to hide their malicious behavior from other applications on the system. Therefore, rootkit detection is an important research topic in Operating System security [18,144,147,148,169,203,215]. Another approach is to prevent the rootkit from altering the Operating System kernel [62, 70, 152, 205].

Vulnerability analysis techniques have been used to discover vulnerabilities in the Operating System kernel before a hacker is able to use and exploit those vulnerabilities. Static analysis techniques have been used [32, 37].

Other vulnerability defenses include defenses for specific vulnerability classes, such as buffer overflows [46]. Randomization techniques attempt to alter the environment of the operating system in such as way that the system continues to function but the attacker's exploit fails [67, 120]. Control-flow integrity techniques have been applied to Operating System code as well [116].

Other research into Operating System security has looked at how to protect applications from a malicious or compromised Operating System [117, 140, 190].

As driver code executes in kernel space, and are a frequent source of vulnerabilities [96], researchers have looked into how to make drivers more robust [63, 86] or isolated from the rest of the kernel [24].

Attack papers show that it is possible to compromise the kernel without injecting any additional code, using Return Oriented Programming techniques [87].

A completely different approach is to formally prove the correctness of the Operating System kernel [105].

Updating Operating Systems with security updates is frequently difficult, as the Operating System must be restarted. However, researchers have explored how to apply an update to the Operating System without rebooting the system [6].

Some work has been done on removing the operating system entirely [128] or trimming/tailoring parts of the Operating System [85, 109–111, 127]. This has the security benefit of reducing the attack surface of the Operating System.

7.3 Suggested Focus Areas

We believe that, while significant research has been done on the Operating System layer, there still remains work to be done on this area. In particular, as the Operating System is frequently in contact with malicious applications, it must be hardened from attack.

Therefore, we believe that further research into the line of trimming or tailoring the operating system is important, such that the attack surface is significantly reduced.

We also believe that non-control data-flow attacks will be the new wave of attacks against Operating Systems. Therefore, we need new analysis techniques, both static and dynamic, to discover these vulnerabilities that do not require altering the control flow of the Operating System.

A critical component of the Operating System is to handle the execution of concurrent applications. Therefore, attacks that take advantage of the concurrency in the Operating System are unexplored. Research must be done to create new models for these attacks.

8 Application Layer

The application layer is the layer of the computing stack that performs that actual useful computation or offers the computing service. It is essentially at the top of the computing stack, as the lower layers of the computing stack provide abstractions so that applications are easier to write and the upper layer (networking) is used by the application layer to communicate to other applications.

At the application layer, we consider all the following types of "applications" under the application layer umbrella: GUI applications, command-line applications, server batch process applications, web applications, or mobile applications. Also, all programming languages used to develop the applications are also in scope, from compiled languages such as C or C++ to scripting languages such as Python and Ruby. In short, all types of computing application are included in the application layer.

Due to abstractions provided by the lower layers, applications typically have less privilege than the lower layers of the computing stack. This is due to the abstractions provided to the application layer, for instance an application cannot access the underlying hard drive directly, but must use the Operating System's provided interface.

Applications developed are typically very complex to understand completely. This is due in part to the reliance on the abstractions provided by the lower computing layers (as these abstractions must be fully understood to understand how the application uses them). Another aspect is that applications frequently use shared libraries or other middleware, which as also considered to be part of the application layer. In this way, multiple different applications can all use the same underlying library. As the application's behavior depends on the behavior of *all* used libraries and abstractions, this means that understanding even a simple application is dependent on the complexity of the used abstractions.

8.1 Threats

Applications, by definition must be accessible by users in order to perform their useful computation. Due to this accessibility, the application layer, of all the other computing layers, is the entry point for attacks or exploits.

At a high level the threats to the application layers is an attacker able to influence the system to perform computation of their choosing. There are two main vectors for an attacker to influence the system: (1) software vulnerabilities in an application that can allow an attacker to exploit the vulnerability to either perform arbitrary computation or otherwise influence the application and (2) a malicious application (typically described as a Trojan application) that the attacker tricks the user into installing and running on the system, usually by using social engineering techniques.

Vulnerabilities are flaws or bugs in the application (or any software for that matter) that allows an attacker to compromise the security of the application. The key difference from normal flaws or bugs in software is that vulnerabilities allow the program to deviate from the behavior. These vulnerabilities range the gamut of possible causes such as programming errors or configuration mistakes. They can be in the actual, custom application software or in any of the libraries that the application uses. Some vulnerabilities are specific to a given programming language, while others are more broad and can any affect application. Examples of common vulnerabilities are: buffer overflow, heap overflow, format string, off-by-one, integer overflow, time-of-check-time-of-use, cross-site scripting, SQL injection, and logic flaws.

Trojan applications attempt to control or manipulate the system by tricking a user into executing them. They frequently do this by using social engineering techniques to entice the user to execute them. The application may appear innocuous, such as a flashlight application on a mobile device, but may in fact include malicious functionality that the user does not want.

Using either of these two techniques, the attackers goal is to move from an external position to having influence on the executing system. The best scenario for the attacker is to completely control the execution of the application. From here, the attacker can now arbitrarily access any resource that the application can access. For instance, this could mean reading sensitive data, changing sensitive data, deleting sensitive data, or using the vantage point of the application to further the compromise to the lower levels of the computing platform or even to other applications on the network.

Applications have a large and complex attack surface, which increases the chances of a developer accidentally introducing a vulnerability. The attack surface of the application is increased by the transitive closure of the libraries that the application uses (and that those libraries use). A vulnerability in a library is a vulnerability in the application that uses that library. Therefore, the

complexity and increased attack surface of applications are a threat, as they allow increases in vulnerabilities.

8.2 Research

Security of the application layer has received significant attention from the research community. Because of this, we have divided the research into the following categories: attack surface reduction, prevent control-flow hijacking, logic and implementation flaws, mitigation and recovery, modeling and monitoring execution, and protection against malicious code.

8.2.1 Attack surface reduction

The idea behind this subarea of application layer security research is to reduce the attack surface of the application. This line of research views vulnerabilities as a probability, and by reducing the attack surface, the odds of an attacking being able to exploit a vulnerability successfully is reduced as well.

Attempts have been made to measure an application's attack surface [84, 129, 130, 159]. This has the benefit of attempting to quantify the attack surface of an application, which in itself is a difficult problem.

Several research works attempt to reduce the attack surface. Some approaches try to statically analyzing for deadcode, then restrict the execution of the deadcode during runtime [110]. Other ways attempt to leverage compile-time configurability to reduce the attack surface [111, 159, 192]. A different approach is to reduce the attack surface by partitioning the application into different privilege levels [14, 213]. Other approaches attempt to restrict the communication between applications, in order to reduce opportunities for the attacker to input data to the system [98].

Finally, other approaches attempt to quantify the attack surface and measure the actual reduction [109].

8.2.2 Prevent control-flow hijacking

Modern attackers, exploiting binary applications, such as browsers, media players, or PDF readers, attempt to hijack the control-flow of the application. If an attacker successfully hijacks the control-flow of the application, then she has complete control over the execution of the application. Many vulnerability classes enable control-flow hijacking, specifically memory corruption vulnerabilities such as buffer overflows, heap overflows, and format strings [188].

The goal of this line of research is to prevent the attacker from successfully hijacking the control-flow of the application. In this way, this line of research can be seen as a mitigation technique.

Significant progress in this area started with the idea of control-flow integrity, essentially ensuring dynamically at runtime that the control-flow of the application does not deviate [1, 2]. The first implementations were deemed infeasible, so later research attempted to improve the efficiency of control-flow integrity [22, 195, 219, 219, 220]

Researchers have advanced the state of attacks against control-flow hijacking. The most advanced of these is return-oriented programming (ROP), in which an attacker reuses bits of code from the original application, so that the attacker can hijack the control-flow of the application without introducing new code [21, 154, 196]. Researchers have also improved these ROP techniques [25, 30, 33, 69, 101, 165] and applied them to different architecture [49].

Other research has looked at defending ROP attacks [36, 48, 50, 68, 77, 81, 82, 124, 125, 139, 141, 141, 142, 175, 207] or detecting ROP attacks [149]. Metastudies looked at the effectiveness of these defenses [164, 174].

Finally, other research has shown that even with perfect control-flow integrity, due to the imprecisenes of statically creating the control-flow graph, an attacker can still hijack the control flow [27, 28, 51, 163].

8.2.3 Logic and implementation flaws

Not all security vulnerabilities are the result of memory corruption or other programming language-based flaws. Another kind of vulnerability class are called logic (or implementation) flaws. These flaws occur when the logic of the application is faulty. Consider an ecommerce application that allows a coupon to be reapplied to an order multiple times, driving the price to zero. Or if a user is able to enter a negative quantity for the purchase of an item, and the application uses that negative quantity to refund the purchase price!

Logic vulnerabilities are particularly difficult for automated vulnerability analysis because they are deviations from the system's specification and the system's implementation. Very often, a system's specification exists only in the minds of the developers. Therefore, logic flaws are specific to every application, and are not simply the result of faulty data flows as most other vulnerability classes are.

Research in logic flaws has typically focused on detecting a certain type of logic flaw, rather than detecting logic flaws generally. Early work looked at detecting state violations in web applications [41].

Other classes such as preventing access control vulnerabilities [47] or statically detecting them [184]. Privilege escalation vulnerabilities [134], data disclosure vulnerabilities [136]

Parameter tampering vulnerabilities are another type of logic flaw [19, 20]

Execution After Redirect are another type of logic flaw vulnerabilities in web applications [54, 143]

Single sign-on vulnerabilities [224].

Work on detecting logic vulnerabilities directly started with attempting to infer security specifications of an application [123, 191]. Other approaches dynamically execute the application to infer likely invariants, then static analysis attempts to find paths that violation those invariants [60]. Further work tried to find missing security (authorization) checks automatically, without prior knowledge of the checks [176, 177].

Recent work has looked at detecting logic flaw vulnerabilities in web applications in a black-box manner—that is, no knowledge of the application's source code [145].

8.2.4 Mitigation and recovery

8.2.5 Modeling and monitoring execution

8.2.6 Protection against malicious code

8.3 Focus Areas

Much of the prior research focuses on just looking at memory corruption bugs, but we need to look at how the same sort of advanced protections can be applied to other classes of bugs, i.e., what is next? This includes: input validation, logic problems, leaking secrets, performance degradation, denial-of-service, command (e.g., SQL) injection, and concurrency attacks.

It is clear that an large amount of research effort has looked at securing the application layer. However, even so, there are areas that we believe are underserved by the research community and provide a significant opportunity to solve.

1. The next generation of vulnerabilities (e.g., logic flaws)

2. Need good tools for analysis of race conditions / concurrency attacks

This focus of this layer includes the runtime environments and abstractions below a high-level programming language that convert it into bytecode.

8.4 Focus Areas

1. Dealing with bloat and reducing the attack surface
2. Instrumentation is easier at this layer, but faster ways to observe execution are needed
3. Controlling easily misued middleware APIs (confused deputy)

References

[1] M. Abadi, M. Budiu, U. Erlingsson, and J. Ligatti. Control-Flow Integrity. *ACM Conference on Computer and Communications Security*, 2005.

[2] M. Abadi, M. Budiu, Ú. Erlingsson, and J. Ligatti. Control-flow integrity principles, implementations, and applications. *ACM Transactions on Information and System Security*, 13(1):1–40, 2009.

[3] F. Adelstein, M. Stillerman, and D. Kozen. Malicious code detection for open firmware. *Proceedings of the Annual Computer Security Applications Conference*, 2002.

[4] D. Agrawal, S. Baktir, D. Karakoyunlu, P. Rohatgi, and B. Sunar. Trojan Detection Using IC Fingerprinting. *Proceedings of the IEEE Symposium on Security and Privacy*, 2007.

[5] W. Arbaugh, D. Farber, and J. Smith. A secure and reliable bootstrap architecture. *Proceedings. 1997 IEEE Symposium on Security and Privacy (Cat. No.97CB36097)*, 1997.

[6] J. Arnold and F. Kaashoek. Ksplice: Automatic Rebootless Kernel Updates. *Proceedings of the 4th ACM European Conference on Computer Systems*, pages 187–198, 2009.

[7] C. R. Attanasio, P. W. Markstein, and R. J. Phillips. Penetrating an operating system : a study of VM / 370 integrity. *IBM Systems Journal*, 1:102–116, 1973.

[8] A. M. Azab, P. Ning, and X. Zhang. SICE : A Hardware-Level Strongly Isolated Computing Environment for x86 Multi-core Platforms. *Environment*, pages 375–388, 2011.

[9] A. M. Azab and N. C. Skalsky. HyperSentry : Enabling Stealthy In-context Measurement of Hypervisor Integrity. *Measurement*, pages 38–49, 2010.

[10] M. Banga, M. Chandrasekar, L. Fang, and M. S. Hsiao. Guided test generation for isolation and detection of embedded trojans in ics. *Proceedings of the 18th ACM Great Lakes symposium on VLSI (GLSVLSI)*, page 363, 2008.

[11] M. Banga and M. S. Hsiao. A Region Based Approach for the Identification of Hardware Trojans. *International Workshop on Hardware-Oriented Security and Trust (HOST'08)*, pages 40–47, 2008.

[12] C. Bao, D. Forte, and A. Srivastava. Temperature Tracking: Towards Robust Run-time Detection of Hardware Trojans. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 0070(c):1–1, 2015.

[13] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. *ACM SIGOPS Operating Systems Review*, 37(5):164, 2003.

[14] A. Barth, A. P. Felt, P. Saxena, and A. Boodman. Protecting Browsers from Extension Vulnerabilities. *Ndss*, 147:1315–1329, 2010.

[15] Z. Basnight, J. Butts, J. Lopez, and T. Dube. Firmware modification attacks on programmable logic controllers. *International Journal of Critical Infrastructure Protection*, 6(2):76–84, 2013.

[16] A. Belay, A. Bittau, and A. Mashtizadeh. Dune: safe user-level access to privileged cpu features. *OSDI '12: Proceedings of the 10th USENIX conference on Operating Systems Design and Implementation*, pages 335–348, 2012.

[17] S. Bhunia, M. Abramovici, D. Agrawal, M. S. Hsiao, J. Plusquellic, M. Tehranipoor, and P. Bradley. Protection against hardware trojan attacks: Towards a comprehensive solution. *IEEE Design and Test*, 30(3):6–17, 2013.

[18] A. Bianchi, Y. Shoshitaishvili, C. Kruegel, and G. Vigna. Blacksheep: Detecting Compromised Hosts in Homogeneous Crowds. *ACM conference on Computer and communications security (CCS)*, page 341, 2012.

[19] P. Bisht, T. Hinrichs, N. Skrupsky, R. Bobrowicz, and V. Venkatakrishnan. NoTamper: Automatic Blackbox Detection of Parameter Tampering Opportunities in Web Applications. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 607–618. ACM, 2010.

[20] P. Bisht, T. Hinrichs, N. Skrupsky, and V. Venkatakrishnan. WAPTEC : Whitebox Analysis of Web Applications for Parameter Tampering Exploit Construction. In *Proceedings of the 18th ACM Conference on Computer and Communications Security (CCS 2011)*, Chicago, IL, 2011.

[21] T. Bletsch, X. Jiang, and V. Freeh. Jump-Oriented Programming: A New Class of Code-Reuse Attack *. pages 1–20, 2010.

[22] T. Bletsch, X. Jiang, and V. Freeh. Mitigating code-reuse attacks with control-flow locking. *Acsac*, page 353, 2011.

[23] A. Bonkoski, R. Bielawski, and J. A. Halderman. Illuminating the Security Issues Surrounding Lights-Out Server Management. *Presented as part of the 7th USENIX Workshop on Offensive Technologies*, 2013.

[24] S. Boyd-Wickizer and N. Zeldovich. Tolerating malicious device drivers in Linux. *Proceedings of the 2010 USENIX Annual Technical Conference*, 2010.

[25] E. Buchanan, R. Roemer, H. Shacham, and S. Savage. When good instructions go bad: generalizing return-oriented programming to RISC. *CCS '08 Proceedings of the 15th ACM conference on Computer and communications security*, pages 27–38, 2008.

[26] Y. Cao, C.-H. Chang, and S. Chen. Cluster-based distributed active current timer for hardware Trojan detection. *IEEE Transactions on Information Forensics and Security*, pages 1010–1013, 2013.

[27] N. Carlini, A. Barresi, E. T. H. Zürich, M. Payer, D. Wagner, T. R. Gross, E. T. H. Zürich, N. Carlini, A. Barresi, D. Wagner, and T. R. Gross. Control-Flow Bending : On the Effectiveness of Control-Flow Integrity. *USENIX Security*, 2015.

[28] N. Carlini and D. Wagner. ROP is Still Dangerous: Breaking Modern Defenses. *23rd USENIX Security Symposium (USENIX Security 14)*, pages 385–399, 2014.

[29] R. S. Chakraborty, S. Paul, and S. Bhunia. On-demand transparency for improving hardware Trojan detectability. In *2008 IEEE International Workshop on Hardware-Oriented Security and Trust, HOST*, pages 48–50, 2008.

[30] S. Checkoway, L. Davi, A. Dmitrienko, a.R. Sadeghi, H. Shacham, and M. Winandy. Return-oriented programming without returns. *CCS'10 Proceedings of the 17th ACM conference on Computer and communications security*, pages 559–572, 2010.

[31] S. Checkoway, D. Mccoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, and T. Kohno. Comprehensive Experimental Analyses of Automotive Attack Surfaces. In *Proceedings of the USENIX Security Symposium*, pages 6–6, 2011.

[32] H. Chen, Y. Mao, X. Wang, D. Zhou, M. F. Kaashoek, and N. Zeldovich. Linux Kernel Vulnerabilities : State-of-the-Art Defenses and Open Problems. In *Proceedings of the Second Asia-Pacific Workshop on Systems*, 2011.

[33] H. Chen and B. Zang. CFIMon: Detecting violation of control flow integrity using performance counters. *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2012)*, pages 1–12, 2012.

[34] P. M. Chen, P. M. Chen, B. D. Noble, and B. D. Noble. When Virtual Is Better Than Real. *HotOS*, 2008.

[35] Y. Cheng, X. Ding, and R. H. Deng. DriverGuard: Virtualization-Based Fine-Grained Protection on I / O Flows. *ACM Transactions on Information and System Security*, 16(2), 2013.

[36] Y. Cheng, Z. Zhou, M. Yu, X. Ding, and R. H. Deng. ROPecker : A Generic and Practical Approach for Defending Against ROP Attacks. *Ndss*, (February):23–26, 2014.

[37] A. Chou, J. Yang, B. Chelf, S. Hallem, and D. Engler. An empirical study of operating systems errors. *ACM SIGOPS Operating Systems Review*, 35(5):73, 2001.

[38] M. Christodorescu, R. Sailer, D. L. Schales, D. Sgandurra, and D. Zamboni. Cloud Security Is Not (Just) Virtualization Security A Short Paper. *Security*, pages 97–102, 2009.

[39] P. Colp, M. Nanavati, J. Zhu, W. Aiello, G. Coker, T. Deegan, P. Loscocco, and A. Warfield. Breaking Up is Hard to Do: Security and Functionality in a Commodity Hypervisor. *SOSP*, 24(1):189–202, 2011.

[40] A. Costin, J. Zaddach, A. Francillon, and D. Balzarotti. A Large-Scale Analysis of the Security of Embedded Firmwares. *USENIX Security 2014*, 2014.

[41] M. Cova, D. Balzarotti, V. Felmetsger, and G. Vigna. Swaddler: An Approach for the Anomaly-based Detection of State Violations in Web Applications. In *Proceedings of the 10th international conference on Recent advances in intrusion detection*, volume 4637 of *LNCS*, pages 1–20. Springer, 2007.

[42] A. Cui, M. Costello, and S. J. Stolfo. When Firmware Modifications Attack : A Case Study of Embedded Exploitation. In *20th Annual Network Distributed System Security Symposium*, 2013.

[43] A. Cui, J. Kataria, and S. J. Stofo. From Prey To Hunter: Transforming Legacy Embedded Devices Into Exploitation Sensor Grids. In *Proceedings of the Annual Computer Security Applications Conference (ACSAC)*, 2010.

[44] A. Cui and S. J. Stolfo. A quantitative analysis of the insecurity of embedded network devices: Results of a wide-area scan. *Proceedings of the Annual Computer Security Applications Conference (ACSAC)*, pages 97–106, 2010.

[45] A. Cui and S. J. Stolfo. Defending embedded systems with software symbiotes. *Proceedings of the Symposium on Recent Advances in Intrusion Detection (RAID)*, 6961 LNCS:358–377, 2011.

[46] M. Dalton, H. Kannan, and C. Kozyrakis. Real-World Buffer Overflow Protection for Userspace & Kernelspace. *Proceedings of the USENIX Security Symposium*, pages 395–410, 2008.

[47] M. Dalton, C. Kozyrakis, and N. Zeldovich. Nemesis: Preventing Authentication & Access Control Vulnerabilities in Web Applications. In *Proceedings of the 18th conference on USENIX security symposium*, SSYM'09, pages 267–282, Berkeley, CA, USA, 2009. USENIX Association.

[48] L. Davi, a. R. Sadeghi, and M. Winandy. Dynamic integrity measurement and attestation: Towards defense against return-oriented programming attacks. *Proceedings of the ACM Conference on Computer and Communications Security*, pages 49–54, 2009.

[49] L. Davi, A. Dmitrienko, M. Egele, F. Thomas, T. Holz, R. Hund, S. Nurnberger, and A.-R. Sadeghi. MoCFI: A framework to mitigate control-flow attacks on smartphones. *NDSS 2012 (19th Network and Distributed System Security Symposium)*, 2012.

[50] L. Davi, A. Sadeghi, and M. Winandy. ROPdefender: A detection tool to defend against return-oriented programming attacks. *Asiaccs*, pages 1–22, 2011.

[51] L. Davi, A.-R. Sadeghi, D. Lehmann, and F. Monrose. Stitching the gadgets: On the ineffectiveness of coarse-grained control-flow integrity protection. *USENIX Security*, 2014.

[52] D. Davidson, T. Ristenpart, and W. Madison. FIE on Firmware : Finding Vulnerabilities in Embedded Systems using Symbolic Execution. *USENIX Security 2013*, 2013.

[53] A. Davoodi, M. Li, and M. Tehranipoor. A Sensor-Assisted Self-Authentication Framework for Hardware Trojan Detection. *IEEE Design & Test*, 30(5):74–82, 2013.

[54] A. Doupé, B. Boe, C. Kruegel, and G. Vigna. Fear the EAR : Discovering and Mitigating Execution After Redirect Vulnerabilities. In *Computer and Communications Security (CCS)*, CCS '11, pages 251–261. ACM Press, 2011.

[55] U. Drolia, Z. Wang, Y. Pant, and R. Mangharam. AutoPlug: An automotive test-bed for electronic controller unit testing and verification. *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC*, pages 1187–1192, 2011.

[56] D. Du, S. Narasimhan, R. S. Chakraborty, and S. Bhunia. Self-referencing: A scalable side-channel approach for hardware trojan detection. *Cryptographic Hardware and Embedded Systems Workshop (CHES)*, 6225 LNCS:173–187, 2010.

[57] L. Duflot, Y. A. Perez, and B. Morin. What if you can't trust your network card? *Proceedings of the Symposium on Recent Advances in Intrusion Detection (RAID)*, 6961 LNCS:378–397, 2011.

[58] G. Dunlap, S. King, and S. Cinar. ReVirt: Enabling intrusion analysis through virtual-machine logging and replay. *ACM SIGOPS Operating . . .* , 36(SI):211–224, 2002.

[59] J. Dworak, A. Crouch, J. Potter, A. Zygmontowicz, and M. Thornton. Don't Forget to Lock your SIB: Hiding Instruments using P1687 1. pages 1–10, 2013.

[60] V. Felmetsger, L. Cavedon, C. Kruegel, and G. Vigna. Toward Automated Detection of Logic Vulnerabilities in Web Applications. In *Proceedings of the USENIX Security Symposium*, Washington, DC, aug 2010.

[61] D. Forte, C. Bao, and A. Srivastava. Temperature tracking: An innovative run-time approach for hardware Trojan detection. *IEEE/ACM International Conference on Computer-Aided Design, Digest of Technical Papers, ICCAD*, pages 532–539, 2013.

[62] F. Gadaleta, N. Nikiforakis, Y. Younan, and W. Joosen. Hello rootKitty: A lightweight invariance-enforcing framework. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 7001 LNCS:213–228, 2011.

[63] V. Ganapathy, M. J. Renzelmann, A. Balakrishnan, M. M. Swift, and S. Jha. The design and implementation of microdrivers. *ACM SIGPLAN Notices*, 43(3):168, 2008.

[64] T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, and D. Boneh. Terra: A virtual machine-based platform for trusted computing. *Operating Systems Review (ACM)*, 37:193–206, 2003.

[65] T. Garfinkel and M. Rosenblum. A Virtual Machine Introspection Based Architecture for Intrusion Detection. *Proc. Network and Distributed Systems Security . . .* , 1:253–285, 2003.

[66] T. Garfinkel and M. Rosenblum. A Virtual Machine Introspection Based Architecture for Intrusion Detection. *Proc. Network and Distributed Systems Security . . .* , 1:253–285, 2003.

[67] C. Giuffrida, A. Kuijsten, and A. Tanenbaum. Enhanced operating system security through efficient and fine-grained address space randomization. *21th USENIX conference on Security*, page 16, 2012.

[68] E. Gökta\c s, E. Athanasopoulos, M. Polychronakis, H. Bos, and G. Portokalidis. Size Does Matter: Why Using Gadget-Chain Length to Prevent Code-Reuse Attacks is Hard. *23rd USENIX Security Symposium (USENIX Security 14)*, pages 417–432, 2014.

[69] E. Göktas, E. Athanasopoulos, H. Bos, and G. Portokalidis. Out of control: Overcoming control-flow integrity. *Ieee S&P*, 2014.

[70] M. Grace, Z. Wang, D. Srinivasan, J. Li, X. Jiang, Z. Liang, and S. Liakh. Transparent protection of commodity OS kernels using hardware virtualization. *Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering*, 50 LNICST:162–180, 2010.

[71] B. Groza, S. Murvay, A. Van Herrewege, and I. Verbauwhede. LiBrA-CAN: A lightweight broadcast authentication protocol for controller area networks. *Proceedings of the International Conference on Cryptology and Network Security (CANS)*, 7712 LNCS:185–200, 2012.

[72] L. Gu, A. Vaynberg, B. Ford, Z. Shao, and D. Costanzo. Certikos: A certified kernel for secure cloud computing. *Proceedings of the Second Asia-Pacific Workshop on Systems*, (0362-1340):3:1—-3:5, 2011.

[73] U. Guin, D. DiMase, and M. Tehranipoor. A Comprehensive Framework for Counterfeit Defect Coverage Analysis and Detection Assessment. *Journal of Electronic Testing*, 2014.

[74] U. Guin, D. Dimase, and M. Tehranipoor. Counterfeit integrated circuits: Detection, avoidance, and the challenges ahead. *Journal of Electronic Testing: Theory and Applications (JETTA)*, 30(1):9–23, 2014.

[75] U. Guin, K. Huang, D. Dimase, J. M. Carulli, M. Tehranipoor, and Y. Makris. Counterfeit integrated circuits: A rising threat in the global semiconductor supply chain. *Proceedings of the IEEE*, 102(8):1207–1228, 2014.

[76] X. Guo, R. G. Dutta, Y. Jin, F. Farahmandi, and P. Mishra. Pre-Silicon Security Verification and Validation: A Formal Perspective. In *EEE/ACM Design Automation Conference (DAC)*, 2015.

[77] A. Gupta, S. Kerr, M. S. Kirkpatrick, and E. Bertino. Marlin: A fine grained randomization approach to defend against ROP attacks. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 7873 LNCS:293–306, 2013.

[78] S. K. Haider, C. Jin, M. Ahmad, D. M. Shila, O. Khan, and M. van Dijk. HaTCh: A Formal Framework of Hardware Trojan Design and Detection. Technical report, 2015.

[79] A. V. Herrewege, D. Singelee, and I. Verbauwhede. CANAuth - A Simple , Backward Compatible Broadcast Authentication Protocol for CAN bus. In *ECRYPT Workshop on Lightweight Cryptography*, 2011.

[80] M. Hicks, M. Finnicum, S. T. King, M. M. K. Martin, and J. M. Smith. Overcoming an untrusted computing base: Detecting and removing malicious hardware automatically. *Proceedings - IEEE Symposium on Security and Privacy*, pages 159–172, 2010.

[81] J. Hiser, A. Nguyen-Tuong, M. Co, M. Hall, and J. W. Davidson. ILR: Where'd My Gadgets Go? *2012 IEEE Symposium on Security and Privacy*, pages 571–585, 2012.

[82] A. Homescu, M. Stewart, P. Larsen, S. Brunthaler, and M. Franz. Microgadgets: Size Does Matter In Turing-complete Return-oriented Programming. *USENIX Workshop on Offensive Technologies*, 2012.

[83] T. Hoppe, S. Kiltz, and J. Dittmann. Security threats to automotive CAN networksPractical examples and selected short-term countermeasures. In *Proceedings of the International Conference on Computer Safety, Reliability, and Security (SAFECOMP)*, volume 96, pages 11–25, 2008.

[84] M. Howard, J. Pincus, and J. M. Wing. Measuring Attack Surfaces. page 24, 2003.

[85] J. Howell, B. Parno, and J. R. Douceur. How to Run POSIX Apps in a Minimal Picoprocess. *2013 USENIX Annual Technical Conference*, pages 321–332, 2013.

[86] T. Hruby, D. Vogt, H. Bos, and A. S. Tanenbaum. Keep net working - On a dependable and fast networking stack. *Proceedings of the International Conference on Dependable Systems and Networks*, 2012.

[87] R. Hund and F. C. Freiling. Return-Oriented Rootkits : Bypassing Kernel Code Integrity Protection Mechanisms. *Usenix Sec*, pages 383–398, 2009.

[88] K. Jiang, P. Eles, and Z. Peng. Co-design techniques for distributed real-time embedded systems with communication security constraints. *Design, Automation Test in Europe Conference Exhibition (DATE), 2012*, pages 947–952, 2012.

[89] S. Jin, J. Ahn, S. Cha, and J. Huh. Architectural support for secure virtualization under a vulnerable hypervisor. *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture - MICRO-44 '11*, page 272, 2011.

[90] S. Jin and J. Huh. Secure MMU: Architectural support for memory isolation among virtual machines. *Proceedings of the International Conference on Dependable Systems and Networks*, pages 217–222, 2011.

[91] Y. Jin, N. Kupp, and Y. Makris. DFTT: Design for Trojan test. *2010 IEEE International Conference on Electronics, Circuits, and Systems, ICECS 2010 - Proceedings*, pages 1168–1171, 2010.

[92] Y. Jin and Y. Makris. Hardware Trojan detection using path delay fingerprint. *2008 IEEE International Workshop on Hardware-Oriented Security and Trust, HOST*, pages 51–57, 2008.

[93] Y. Jin and D. Sullivan. Real-Time Trust Evaluation in Integrated Circuits. In *Design, Automation and Test in Europe Conference and Exhibition (DATE)*, 2014.

[94] S. T. Jones, A. C. Arpaci-dusseau, and R. H. Arpaci-dusseau. Antfarm: Tracking Processes in a Virtual Machine Environment. *Proceedings of the USENIX Annual Technical Conference - USENIX'06*, pages 1–14, 2006.

[95] S. T. Jones, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau. VMM-based hidden process detection and identification using Lycosid. *Proceedings of the fourth ACM SIGPLAN/SIGOPS international conference on Virtual execution environments - VEE '08*, page 91, 2008.

[96] A. Kadav and M. M. Swift. Understanding modern device drivers. *ACM SIGARCH Computer Architecture News*, 40(1):87, 2012.

[97] P.-h. Kamp and R. N. M. Watson. Jails : Confining the omnipotent root. *Proceeding of the 2nd International System Administration and Networking Conference*, pages 3–14, 2000.

[98] D. Kantola, E. Chin, W. He, and D. Wagner. Reducing Attack Surfaces for Intra-Application Communication in Android. *SPSM*, 2012.

[99] P. A. Karger, M. E. Zurko, D. W. Bonin, A. H. Mason, and C. E. Kahn. A Retrospective on the VAX VMM Security Kernel. *IEEE Transactions on Software Engineering*, 17, 1991.

[100] B. Kauer. OSLO: improving the security of trusted computing. *Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium*, pages 16:1—-16:9, 2007.

[101] M. Kayaalp, M. Ozsoy, N. Abu-Ghazaleh, and D. Ponomarev. Branch regulation: Low-overhead protection from code reuse attacks. *2012 39th Annual International Symposium on Computer Architecture (ISCA)*, pages 94–105, 2012.

[102] S. King, J. Tucek, A. Cozzie, C. Grier, W. Jiang, and Y. Zhou. Designing and Implementing Malicious Hardware. *Leet*, pages 1–18, 2008.

[103] A. Kivity, D. Laor, G. Costa, and P. Enberg. OSvOptimizing the Operating System for Virtual Machines. *Proceedings of the 2014 USENIX Annual Technical Conference*, pages 61–72, 2014.

[104] P. Kleberger, T. Olovsson, and E. Jonsson. Security aspects of the in-vehicle network in the connected car. *IEEE Intelligent Vehicles Symposium, Proceedings*, (Iv):528–533, 2011.

[105] G. Klein, K. Elphinstone, G. Heiser, J. Andronick, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, T. Sewell, H. Tuch, and S. Winwood. seL4: Formal verification of an OS kernel. *Proceedings of the ACM SIGOPS 22nd Symposium on Operating System Principles*, pages 207–220, 2009.

[106] P. Koeberl, S. Schulz, A.-R. Sadeghi, and V. Varadharajan. TrustLite: A Security Architecture for Tiny Embedded Devices. *Proceedings of the European Conference on Computer Systems (EuroSys)*, pages 1–14, 2014.

[107] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. Mccoy, B. Kantor, D. Anderson, H. Shacham, and S. Savage. Experimental Security Analysis of a Modern Automobile. *Proceedings of the IEEE Symposium on Security and Privacy*, pages 1–16, 2010.

[108] K. Kourai and S. Chiba. HyperSpector: virtual distributed monitoring environments for secure intrusion detection. ... *conference on Virtual execution environments*, pages 197–207, 2005.

[109] A. Kurmus, S. Dechand, and R. Kapitza. Quantifiable Run-time Kernel Attack Surface Reduction. *Proceedings of the Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)*, 2014.

[110] A. Kurmus, A. Sorniotti, and R. Kapitza. Attack surface reduction for commodity OS kernels. In *Proceedings of the Fourth European Workshop on System Security - EUROSEC '11*, pages 1–6, 2011.

[111] A. Kurmus, R. Tartler, D. Dorneanu, B. Heinloth, V. Rothberg, A. Ruprecht, W. Schroder-Preikschat, D. Lohmann, and R. Kapitza. Attack Surface Metrics and Automated Compile-Time Kernel Tailoring. pages 1–63, 2013.

[112] A. Kurmus and R. Zippel. A Tale of Two Kernels: Towards Ending Kernel Hardening Wars with Split Kernel. *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, pages 1366–1377, 2014.

[113] U. E. Larson and D. K. Nilsson. Securing vehicles against cyber attacks. *Proceedings of the 4th annual workshop on Cyber security and informaiton intelligence research developing strategies to meet the cyber security and information intelligence challenges ahead - CSIIRW '08*, page 1, 2008.

[114] J. Y. Lee, S. W. Jung, and J. Lim. Detecting Trapdoors in Smart Cards Using Timing and Power Analysis. *Testing of Communicating System*, pages 275–288, 2005.

[115] J. Li and J. Lach. At-speed delay characterization for IC authentication and Trojan horse detection. *2008 IEEE International Workshop on Hardware-Oriented Security and Trust, HOST*, pages 8–14, 2008.

[116] J. Li, Z. Wang, T. Bletsch, D. Srinivasan, M. Grace, and X. Jiang. Comprehensive and efficient protection of kernel control data. *IEEE Transactions on Information Forensics and Security*, 6(4):1404–1417, 2011.

[117] Y. Li, J. McCune, J. Newsome, A. Perrig, B. Baker, and W. Drewry. MiniBox: A Two-Way Sandbox for x86 Native Code. *2014 USENIX Annual Technical Conference (USENIX ATC 14)*, pages 409–420, 2014.

[118] Y. Li, J. M. McCune, and A. Perrig. SBAP: Software-Based Attestation for Peripherals. *Trust and Trustworthy Computing*, 2010.

[119] Y. Li, J. M. Mccune, and A. Perrig. VIPER : Verifying the Integrity of PERipherals' Firmware. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, volume 22, pages 3–16, 2011.

[120] Z. Lin, R. D. Riley, and D. Xu. Polymorphing software by randomizing data structure layout. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 5587 LNCS:107–126, 2009.

[121] Y. Liu, K. Huang, and Y. Makris. Hardware Trojan Detection through Golden Chip-Free Statistical Side-Channel Fingerprinting. *Proceedings of the The 51st Annual Design Automation Conference on Design Automation Conference - DAC '14*, pages 1–6, 2014.

[122] Y. Liu, Y. Jin, and Y. Makris. Hardware Trojans in wireless cryptographic ICs: Silicon demonstration & detection method evaluation. *IEEE/ACM International Conference on Computer-Aided Design, Digest of Technical Papers, ICCAD*, pages 399–404, 2013.

[123] B. Livshits, a. V. Nori, S. K. Rajamani, and a. Banerjee. Merlin: Specification Inference for Explicit Information Flow Problems. *Programming Language Design and Implementation*, pages 75–86, 2009.

[124] K. Lu, D. Zou, W. Wen, and D. Gao. Packed, Printable, and Polymorphic Return-Oriented Programming. *Recent Advances in Intrusion Detection*, 2011.

[125] D. Lu, Kangjie, and Zou, Dabi, and Wen, Weiping, and Gao. deRop: Removing Return-Oriented Programming from Malware. *Annual Computer Security Applications Conference (ACSAC)*, pages 363–372, 2011.

[126] a. Madhavapeddy and T. Leonard. Jitsu: Just-in-time summoning of unikernels. *USENIX Symposium on Networked Systems Design and Implementation*, 2015.

[127] A. Madhavapeddy, R. Mortier, C. Rotsos, D. Scott, B. Singh, T. Gazagnaire, S. Smith, S. Hand, and J. Crowcroft. Unikernels: Library Operating Systems for the Cloud. *ASPLOS*, 2013.

[128] A. Madhavapeddy, R. Mortier, R. Sohan, T. Gazagnaire, S. Hand, T. Deegan, D. Mcauley, and J. Crowcroft. Turning Down the LAMP : Software Specialisation for the Cloud. *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, 85(23):7, 2010.

[129] P. K. Manadhata, K. Tan, R. Maxion, and J. Wing. An Approach to Measuring A System's Attack Surface. Technical Report August, 2007.

[130] P. K. Manadhata and J. M. Wing. Measuring the Attack Surfaces of SAP Business Applications. Technical report, 2008.

[131] J. Martins, M. Ahmed, C. Raiciu, V. Olteanu, M. Honda, R. Huici, and B. Felipe. ClickOS and the Art of Network Function Virtualization. *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, pages 459—-473, 2014.

[132] J. Maskiewicz, B. Ellis, J. Mouradian, and H. Shacham. Mouse Trap: Exploiting Firmware Updates in USB Peripherals. *Proceedings of the USENIX Workshop on Offensive Technologies (WOOT)*, 2014.

[133] J. M. McCune, Y. Li, N. Qu, Z. Zhou, A. Datta, V. Gligor, and A. Perrig. Trust visor: Efficient TCB reduction and attestation. *Proceedings - IEEE Symposium on Security and Privacy*, 2009:143–158, 2010.

[134] M. Monshizadeh, Prasad Naldurg, and V. N. Venkatakrishnan. MACE : Detecting Privilege Escalation Vulnerabilities in Web Applications. *CCS*, 2014.

[135] D. G. Murray, G. Milos, and S. Hand. Improving Xen security through disaggregation. *Proceedings of the fourth ACM SIGPLAN/SIGOPS international conference on Virtual execution environments - VEE '08*, page 151, 2008.

[136] D. Muthukumaran, D. O. Keeffe, C. Priebe, D. Eyers, B. Shand, and P. Pietzuch. FlowWatcher : Defending against Data Disclosure Vulnerabilities in Web Applications. *Ccs*, pages 603–615, 2015.

[137] S. Narasimhan, X. Wang, D. Du, R. S. Chakraborty, and S. Bhunia. TeSR: A robust temporal self-referencing approach for Hardware Trojan detection. *2011 IEEE International Symposium on Hardware-Oriented Security and Trust, HOST 2011*, (c):71–74, 2011.

[138] S. Narasimhan, W. Yueh, X. Wang, S. Mukhopadhyay, and S. Bhunia. Improving IC security against trojan attacks through integration of security monitors. *IEEE Design and Test of Computers*, 29(5):37–46, 2012.

[139] K. Onarlioglu, L. Bilge, A. Lanzi, D. Balzarotti, and E. Kirda. G-Free: Defeating Return-Oriented Programming through Gadget-less Binaries. *Proceedings of the 26th Annual Computer Security Applications Conference on - ACSAC '10*, page 49, 2010.

[140] K. Onarlioglu, C. Mulliner, W. Robertson, and E. Kirda. PrivExec: Private execution as an operating system service. In *Proceedings - IEEE Symposium on Security and Privacy*, pages 206–220, 2013.

[141] V. Pappas, M. Polychronakis, and A. D. Keromytis. Smashing the Gadgets: Hindering Return-Oriented Programming Using In-place Code Randomization. *2012 IEEE Symposium on Security and Privacy*, pages 601–615, 2012.

[142] V. Pappas, M. Polychronakis, and A. D. Keromytis. Transparent ROP Exploit Mitigation Using Indirect Branch Tracing. *USENIX Security*, 2013.

[143] P. Payet, A. Doupé, and C. Kruegel. EARs in the Wild : Large-Scale Analysis of Execution After Redirect Vulnerabilities. In *SAC*, pages 1792–1799, 2013.

[144] B. D. Payne, M. Carbone, M. Sharif, and W. Lee. Lares: An architecture for secure active monitoring using virtualization. *Proceedings - IEEE Symposium on Security and Privacy*, pages 233–247, 2008.

[145] G. Pellegrino and D. Balzarotti. Toward Black-Box Detection of Logic Flaws in Web Applications. *Network and Distributed System Security Symposium (NDSS)*, (February):23–26, 2014.

[146] D. Perez-Botero, J. Szefer, and R. B. Lee. Characterizing hypervisor vulnerabilities in cloud computing servers. *Proceedings of the 2013 international workshop on Security in cloud computing - Cloud Computing '13*, (May):3, 2013.

[147] N. L. Petroni, T. Fraser, J. Molina, and W. a. Arbaugh. Copilot - a Coprocessor-based Kernel Runtime Integrity Monitor. *USENIX Security Symposium*, pages 179–194, 2004.

[148] N. L. Petroni and M. Hicks. Automated detection of persistent kernel control-flow attacks. *ACM conference on Computer and communications security (CCS) (2007)*, pages 103–115, 2007.

[149] M. Polychronakis and A. D. Keromytis. ROP payload detection using speculative code execution. *Proceedings of the 2011 6th International Conference on Malicious and Unwanted Software, Malware 2011*, pages 58–65, 2011.

[150] J. J. V. Rajendran, A. K. Kanuparthi, R. Karri, S. K. Addepalli, M. Zahran, and G. Ormazabal. Securing Processors Against Insider Attacks : A Circuit-Microarchitecture Co-Design Approach. *IEEE Design & Test*, (April):35–44, 2013.

[151] X. Ren, G. Tavares, and R. D. S. Blanton. Detection of Illegitimate Access to JTAG via Statistical Learning in Chip. In *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 109–114, 2015.

[152] R. Riley, X. Jiang, and D. Xu. Guest-transparent prevention of kernel rootkits with VMM-based memory shadowing. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 5230 LNCS:1–20, 2008.

[153] J. S. Robin and C. E. Irvine. Analysis of the Intel Pentium's ability to support a secure virtual machine monitor. In *Proceedings of the USENIX Security Symposium*, volume 8, page 10, 2000.

[154] R. Roemer, E. Buchanan, H. Shacham, and S. Savage. Return-Oriented Programming : Systems , Languages , and Applications. *ACM Transactions on Information and System Security (TISSEC)*, V:1–36, 2012.

[155] J. D. Rolt, G. D. Natale, M.-l. Flottes, and B. Rouzeyre. Thwarting Scan-Based Attacks on Secure-ICs With On-Chip Comparison. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 22(4):947–951, 2014.

[156] K. Rosenfeld and R. Karri. Attacks and defenses for JTAG. *IEEE Design and Test of Computers*, 27(1):36–47, 2010.

[157] K. Rosenfeld and R. Karri. Security-aware SoC test access mechanisms. *Proceedings of the IEEE VLSI Test Symposium*, pages 100–104, 2011.

[158] I. Rouf, R. Miller, H. Mustafa, T. Taylor, S. Oh, W. Xu, M. Grutese, W. Trappe, and I. Seskar. Security and privacy vulnerabilities of in-car wireless networks: A tire pressure monitoring system case study. *Proceedings of the USENIX Security Symposium*, 39(4):11–13, 2010.

[159] A. Ruprecht, B. Heinloth, and D. Lohmann. Automatic feature selection in large-scale system-software product lines. *ACM SIGPLAN Notices*, 50(3):39–48, 2014.

[160] R. Sailer, X. Zhang, T. Jaeger, and L. V. Doorn. Design and Implementation of a TCG-based Integrity Measurement Architecture. *USENIX Security Symposium*, page 17, 2004.

[161] H. Salmani, M. Tehranipoor, and J. Plusquellic. New design strategy for improving hardware trojan detection and reducing trojan activation time. *Hardware-Oriented ...*, pages 66–73, 2009.

[162] F. L. Sang, É. . Lacombe, V. Nicomette, and Y. Deswarte. Exploiting an I/OMMU vulnerability. *Proceedings of the 5th IEEE International Conference on Malicious and Unwanted Software (MALWARE)*, pages 7–14, 2010.

[163] F. Schuster, T. Tendyck, C. Liebchen, L. Davi, A.-R. Sadeghi, and T. Holz. Counterfeit Object-oriented Programming On the Difficulty of Preventing Code Reuse Attacks in C++ Applications. *Proceedings - IEEE Symposium on Security and Privacy*, 2015.

[164] F. Schuster, T. Tendyck, J. Pewny, A. Maaß, M. Steegmanns, M. Contag, T. Holz, and G. Horst. Evaluating the Effectiveness of Current Anti-ROP Defenses. *Research in Attacks, Intrusions and Defenses*, pages 1–22, 2014.

[165] E. Schwartz, T. Avgerinos, and D. Brumley. Q: Exploit hardening made easy. *Usenix Sec*, 8(3):25, 2011.

[166] H. Schweppe and Y. Roudier. Security and privacy for in-vehicle networks. *2012 IEEE 1st International Workshop on Vehicular Communications, Sensing, and Computing, VCSC 2012*, pages 12–17, 2012.

[167] H. Schweppe, Y. Roudier, and B. Weyl. Car2X communication: securing the last meter. *Vehicular Technology Conference*, pages 1–5, 2011.

[168] A. Seshadri, M. Luk, N. Qu, and A. Perrig. SecVisor: A Tiny Hypervisor to Provide Lifetime Kernel Code Integrity for Commodity OSes. In *Proceedings of the ACM SIGOPS Symposium on Operating Systems Principles*, volume 41, pages 335–350, 2006.

[169] M. Sharif, W. Lee, W. Cui, and A. Lanzi. Secure in-vm monitoring using hardware virtualization. ... *of the 16th ACM conference on ...*, page 11, 2009.

[170] T. Shinagawa, H. Eiraku, K. Tanimoto, K. Omote, S. Hasegawa, T. Horie, M. Hirano, K. Kourai, Y. Oyama, E. Kawai, K. Kono, S. Chiba, Y. Shinjo, and K. Kato. BitVisor: a thin hypervisor for enforcing i/o device security. *VEE 09 Proceedings of the 2009 ACM SIGPLANSIGOPS international conference on Virtual execution environments*, (Vee):121–130, 2009.

[171] Y. Shoshitaishvili, R. Wang, C. Hauser, C. Kruegel, and G. Vigna. Firmalice - Automatic Detection of Authentication Bypass Vulnerabilities in Binary Firmware. *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, (February):8–11, 2015.

[172] L. Singaravelu, C. Pu, H. Härtig, and C. Helmuth. Reducing TCB complexity for security-sensitive applications. *ACM SIGOPS Operating Systems Review*, 40(4):161, 2006.

[173] S. Skorobogatov and C. Woods. Breakthrough silicon scanning discovers backdoor in military chip. *Cryptographic Hardware and Embedded Systems Workshop (CHES)*, 7428 LNCS(September):23–40, 2012.

[174] R. Skowyra, K. Casteel, H. Okhravi, N. Zeldovich, and W. Streilein. Systematic Analysis of Defenses Against Return-Oriented Programming. *Research in Attacks, Intrusions, and Defenses*, pages 82–102, 2013.

[175] K. Z. Snow, F. Monrose, L. Davi, A. Dmitrienko, C. Liebchen, and A. Sadeghi. Just-In-Time Code Reuse: On the Effectiveness of Fine-Grained Address Space Layout Randomization. *2013 IEEE Symposium on Security and Privacy*, pages 574–588, 2013.

[176] S. Son and K. S. Mckinley. Fix Me Up : Repairing Access-Control Bugs in Web Applications. *Proceedings of the 2013 Network and Distributed System Security Symposium (NDSS 2013)*, 2013.

[177] S. Son, K. S. McKinley, and V. Shmatikov. RoleCast: finding missing security checks when you do not know what checks are. *Oopsla*, pages 1069–1084, 2011.

[178] D. Srinivasan, Z. Wang, X. Jiang, and D. Xu. Process Out-Grafting : An Efficient Out-of-VM Approach for Fine-Grained Process Execution Monitoring Categories and Subject Descriptors. *CCS*, pages 363–374, 2011.

[179] P. Stewin. A primitive for revealing stealthy peripheral-based attacks on the computing platform's main memory. *Proceedings of the Symposium on Recent Advances in Intrusion Detection (RAID)*, 8145 LNCS:1–20, 2013.

[180] P. Stewin. Enhanced BARM Authentic Reporting to External Platforms. Technical Report September, 2014.

[181] I. Studnia, V. Nicomette, E. Alata, Y. Deswarte, M. Kaaniche, and Y. Laarouchi. Survey on security threats and protection mechanisms in embedded automotive networks. *Proceedings of the International Conference on Dependable Systems and Networks*, 2013.

[182] C. Sturton, M. Hicks, D. Wagner, and S. T. King. Defeating UCI: Building stealthy and malicious hardware. *Proceedings of the IEEE Symposium on Security and Privacy*, pages 64–77, 2011.

[183] J. Stüttgen, S. Vömel, and M. Denzel. Acquisition and analysis of compromised firmware using memory forensics. *Digital Investigation*, 12:S50–S60, 2015.

[184] F. Sun, L. Xu, and Z. Su. Static Detection of Access Control Vulnerabilities in Web Applications. In *Proceedings of the 20th USENIX conference on Security (USENIX 2011)*, 2011.

[185] J. Szefer, E. Keller, R. B. Lee, and J. Rexford. Eliminating the Hypervisor Attack Surface for a More Secure Cloud Categories and Subject Descriptors. *Proceedings of the 18th ACM conference on Computer and Communications Security (CCS'11)*, pages 401–412, 2011.

[186] J. Szefer and R. Lee. A case for hardware protection of guest vms from compromised hypervisors in cloud computing. . . . *Computing Systems Workshops (ICDCSW), . . .* , (June):248–252, 2011.

[187] J. Szefer and R. B. Lee. Architectural support for hypervisor-secure virtualization. *ACM SIGARCH Computer Architecture News*, 40:437, 2012.

[188] L. Szekeres, M. Payer, T. Wei, and D. Song. SoK: Eternal war in memory. *Proceedings - IEEE Symposium on Security and Privacy*, pages 48–62, 2013.

[189] C. Szilagyi and P. Koopman. Low cost multicast authentication via validity voting in time-triggered embedded control networks. *Proceedings of the 5th Workshop on Embedded Systems Security - WESS '10*, pages 1–10, 2010.

[190] R. Ta-min and L. Litty. Splitting Interfaces: Making Trust Between Applications and Operating Systems Configurable. *Osdi*, (Vm):279–292, 2006.

[191] L. Tan, X. Zhang, X. Ma, W. Xiong, and Y. Zhou. AutoISES: Automatically Inferring Security Specifications and Detecting Violations. In *Proceedings of the USENIX Security Symposium*, pages 379–394, jul 2008.

[192] R. Tartler, A. Kurmus, and A. Ruprecht. Automatic OS kernel TCB reduction by leveraging compile-time configurability. *Proceedings of the . . .* , 2012.

[193] M. Tehranipoor and F. Koushanfar. A survey of hardware trojan taxonomy and detection. *IEEE Design and Test of Computers*, 27(1):10–25, 2010.

[194] P. R. Thom and C. A. Maccarley. A Spy Under the Hood: Controlling Risk and Automotive EDR. *Risk Management*, pages 22–26, 2008.

[195] C. Tice, T. Roeder, P. Collingbourne, S. Checkoway, Ú. Erlingsson, L. Lozano, and G. Pike. Enforcing Forward-Edge Control-Flow Integrity in GCC & LLVM. *23rd USENIX Security Symposium*, pages 941–955, 2014.

[196] M. Tran, M. Etheridge, T. Bletsch, X. Jiang, V. Freeh, and P. Ning. On the expressiveness of return-into-libc attacks. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 6961 LNCS:121–141, 2011.

[197] A. Tsow, M. Jakobsson, L. Yang, and S. Wetzel. Warkitting: The Drive-by Subversion of Wireless Home Routers. *Journal of Digital Forensic Practice*, 1(3):179–192, 2006.

[198] A. Vasudevan, S. Chaki, Limin Jia, J. McCune, J. Newsome, and A. Datta. Design, Implementation and Verification of an eXtensible and Modular Hypervisor Framework. *2013 IEEE Symposium on Security and Privacy*, pages 430–444, 2013.

[199] A. Vasudevan, J. McCune, J. Newsome, A. Perrig, and L. van Doorn. CARMA: A Hardware Tamper-resistant Isolated Execution Environment on Commodity x86 Platforms. *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security*, pages 48–49, 2012.

[200] A. Waksman and S. Sethumadhavan. Tamper evident microprocessors. *Proceedings - IEEE Symposium on Security and Privacy*, pages 173–188, 2010.

[201] A. Waksman and S. Sethumadhavan. Silencing Hardware Backdoors. *Proceedings of the IEEE Symposium on Security and Privacy*, pages 49–63, 2011.

[202] X. Wang, M. Tehranipoor, and J. Plusquellic. Detecting malicious inclusions in secure hardware: Challenges and solutions. *2008 IEEE International Workshop on Hardware-Oriented Security and Trust, HOST*, pages 15–19, 2008.

[203] Y.-M. Wang, D. Beck, B. Vo, R. Roussev, and C. Verbowski. Detecting stealth software with Strider GhostBuster. *International Conference on Dependable Systems and Networks (DSN)*, 2005.

[204] Z. Wang and X. Jiang. HyperSafe: A lightweight approach to provide lifetime hypervisor control-flow integrity. *Proceedings - IEEE Symposium on Security and Privacy*, pages 380–395, 2010.

[205] Z. Wang, X. Jiang, W. Cui, and P. Ning. Countering kernel rootkits with lightweight hook protection. *Proceedings of the 16th ACM conference on Computer and communications security - CCS '09*, page 545, 2009.

[206] Z. Wang, C. Wu, M. Grace, and X. Jiang. Isolating commodity hosted hypervisors with HyperLock. *Proceedings of the 7th ACM european conference on Computer Systems - EuroSys '12*, page 127, 2012.

[207] R. Wartell, V. Mohan, K. W. Hamlen, Z. Lin, and W. C. Rd. Binary stirring: Self-randomizing instruction addresses of legacy x86 binary code. *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 157–168, 2012.

[208] R. Wojtczuk and J. Rutkowska. Following the White Rabbit : Software attacks against Intel (R) VT-d technology. Technical report, 2011.

[209] M. Wolf and T. Gendrullis. Design, Implementation, and evaluation of a vehicular hardware security module. In *Proceedings of the International Conference on Information Security and Cryptology (ICISC)*, volume 7259 LNCS, pages 302–318, 2012.

[210] M. Wolf, A. Weimerskirch, and C. Paar. Security in Automotive Bus Systems. In *Proceedings of the Workshop on Embedded Security in Cars*, pages 1–13, 2004.

[211] M. Wolf, A. Weimerskirch, and T. Wollinger. State of the art: Embedding security in vehicles. *Eurasip Journal on Embedded Systems*, 2007, 2007.

[212] A. Wright. Hacking cars. *Communications of the ACM*, 54(11):18, 2011.

[213] R. Xu, H. Saïdi, R. Anderson, and H. Sadi. Aurasium: Practical Policy Enforcement for Android Applications. *Proceedings of the 21st USENIX conference . . .* , page 27, 2012.

[214] J. Yang and K. G. Shin. Using hypervisor to provide data secrecy for user applications on a per-page basis. In *Proceedings of the ACM SIGPLAN/SIGOPS international conference on Virtual execution environments (VEE)*, pages 71–80, 2008.

[215] H. Yin, P. Poosankam, S. Hanna, and D. Song. HookScout: Proactive binary-centric hook detection. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 6201 LNCS:1–20, 2010.

[216] J. Zaddach, L. Bruno, A. Francillon, and D. Balzarotti. Avatar: A Framework to Support Dynamic Security Analysis of Embedded Systems' Firmwares. *Proceedings of the 2014 Network and Distributed System Security Symposium*, (February):23–26, 2014.

[217] J. Zaddach and A. Costin. Embedded Devices Security and Firmware Reverse Engineering. *Black Hat USA*, page 9, 2013.

[218] J. Zaddach, A. Kurmus, D. Balzarotti, E.-O. Blass, A. Francillon, T. Goodspeed, M. Gupta, and I. Koltsidas. Implementation and implications of a stealth hard-drive backdoor. *Proceedings of the 29th Annual Computer Security Applications Conference on - ACSAC '13*, pages 279–288, 2013.

[219] C. Zhang, T. Wei, Z. Chen, L. Duan, L. Szekeres, S. McCamant, D. Song, and W. Zou. Practical control flow integrity and randomization for binary executables. *Proceedings - IEEE Symposium on Security and Privacy*, pages 559–573, 2013.

[220] C. Zhang, T. Wei, Z. Chen, L. Duan, L. Szekeres, S. McCamant, D. Song, and W. Zou. Practical control flow integrity and randomization for binary executables. *Proceedings - IEEE Symposium on Security and Privacy*, pages 559–573, 2013.

[221] F. Zhang, J. Chen, H. Chen, and B. Zang. Cloudvisor: retrofitting protection of virtual machines in multi-tenant cloud with nested virtualization. *SOSP '11: Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, pages 203–216, 2011.

[222] F. Zhang, H. Wang, K. Leach, and A. Stavrou. A Framework to Secure Peripherals at Runtime. In *Proceedings of the European Symposium on Research in Computer Security (ESORICS)*, pages 1–20, 2014.

[223] Y. Zhao. Telematics: Safe and fun driving. *IEEE Intelligent Systems and Their Applications*, 17(1):10–14, 2002.

[224] Y. Zhou and D. Evans. SSOScan: Automated Testing of Web Applications for Single Sign-On Vulnerabilities. *23rd USENIX Security Symposium (USENIX Security 14)*, pages 495–510, 2014.

[225] Z. Zhou, J. Fan, N. Zhang, and R. Xu. Advance and Development of Computer Firmware Security Research. *Proceedings of the 2009 International Symposium on Information Processing (ISIP)*, 2(1), 2009.

[226] Z. Zhou and R. Xu. {BIOS} Security Analysis and a Kind of Trusted {BIOS}. pages 427–437, 2007.