Security Considerations for Next-Generation Operating Systems for Cyber-Physical Systems *

Bryan C. Ward*, Richard Skowyra*, Samuel Jero*, Nathan Burow*, Hamed Okhravi*, Howard Shrobe†, and Roger Khazan*

*MIT Lincoln Laboratory, †MIT CSAIL

1 Introduction

Cyber-physical systems (CPSs) are increasingly targeted in high-profile cyber attacks. Examples of such attacks include Stuxnet, which targeted nuclear centrifuges; Crashoverride, and Triton, which targeted power grids; and the Mirai botnet, which targeted internet-of-things (IoT) devices such as cameras to carry out a large-scale distributed denial-of-service (DDoS) attack. Such attacks demonstrate the importance of securing current and future cyber-physical systems. Therefore, next-generation operating systems (OSes) for CPS need to be designed to provide security features necessary, as well as be secure in and of themselves.

CPSs are designed with one of three broad classes of OSes: (a) bare-metal applications with effectively no operating system, (b) embedded systems executing on impoverished platforms running an embedded or real-time operating system (RTOS) such as FreeRTOS, or (c) more performant platforms running general-purpose OSes such as Linux, sometimes tuned for real-time performance such as through the PREEMPT_RT patch. In cases (a) and (b), the OS, if any, is very minimal to facilitate improved resource utilization in real-time or latency-sensitive applications, especially running on impoverished hardware platforms. In such OSes, security is often overlooked, and many important security features (*e.g.* process/kernel memory isolation) are notably absent. In case (c), the general-purpose OS inherits many of the security-related features that are critical in enterprise and general-purpose applications, such as virtual memory and address-space layout randomization (ASLR). However, the highly complex nature of general-purpose OSes can be problematic in the development of CPSs, as they are highly non-deterministic and difficult to formally reason about for cyber-physical applications, which often have real-time constraints. These issues motivate the need for a next generation OS that is highly capable, predictable and deterministic for real-time performance, but also secure in the face of many of the next generation of cyber threats.

In order to design such a next-generation OS, it is necessary to first reflect on the types of threats that CPSs face, including the attacker intentions and types of effects that can be achieved, as well as the type of access that attackers have. While threat models are not the same for all CPSs, it is important to understand how the threat models for CPSs compare to general-purpose or enterprise computing environments. We discuss these issues next (Sec. 2), before providing insights and recommendations for approaches to incorporate in next-generation OSes for CPS in Sec. 3.

2 Threat Models

The strategy chosen by an adversary depends heavily on their goal. The cybersecurity community frequently partitions these into three categories, referred to as the *Confidentiality*, *Integrity*, and *Availability* (CIA) triad. Cyber-physical systems must also be concerned with an additional category: *Safety*. CIAS simultaneously refers to guarantees that a system defender seeks to provide, and properties that an attacker seeks to subvert. As part of a larger campaign, adversaries may target any number and combination of these categories.

Confidentiality ensures that data of interest is only able to be read by parties authorized to read that data. Thus, an attacker focused on breaking confidentiality seeks to learn sensitive information that they are not authorized to know (*e.g.*, passwords, financial information, or trade secrets). While confidentiality attacks are often an end-goal in general-purpose computing (e.g. theft of credit card data), they may act as a stepping-stone in the CPS domain. For example, an attacker seeking to cause physical or economic harm may attempt to impersonate an authorized operator by stealing authentication credentials via a confidentiality attack, enabling them to later issue commands that put a process plant in a dangerous or low-yield state.

Integrity ensures that data of interest cannot be undetectably modified or created by unauthorized parties. An attacker focused on breaking integrity may seek to inject forged information, corrupt existing data, or otherwise deceive the system. In the CPS domain, integrity attacks may manifest as spoofed or modified sensor values in order to mis-represent the state of a physical process, or corrupted messages from a controller to an actuator that causes over- or under-compensation. For example, an attacker could inject messages on an automotive CAN bus (*e.g.*, via a malicious OBDII interface) that falsely indicate the brake pedal has been depressed, causing an unexpected and potentially dangerous vehicle deceleration.

Availability ensures the system is functioning and able to perform its task. Denial-of-service (DoS) attacks target availability and seek to render a target unable to provide its intended service. Depending on the system under attack, DoS attacks can range from flooding a shared bus with spurious messages to prevent critical sensor/actuator messages from being transferred, to frequency-shifting attacks on power grids that induce automated load shedding [3].

Safety ensures the system does not cause physical harm. It is unique to the cyber-physical domain and not part of the tradi-

This material is based upon work supported by the Assistant Secretary of Defense for Research and Engineering under Air Force Contract No. FA8702-15-D-0001. Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Assistant Secretary of Defense for Research and Engineering.

tional CIA triad, largely because general-purpose computers are unlikely to cause direct physical harm under cyber-attack. In the CPS domain, however, physical processes are governed by software control systems that if attacked can endanger human life. Attacks on safety may leverage any combination of integrity violations (forged or corrupted messages that cause control loop destabilization), denial of service (denying a bus to stop a system from leaving a dangerous state), or theft of secrets (stolen credentials used to initiate an unsafe operation).

2.1 Access Vectors

In the past, CPS were seen (rightly or not) as largely disconnected from the larger Internet. The growing popularity of Internet of Things (IoT) devices and cellular-connected automobiles has put an end to this perception. Remote attackers are now a genuine threat to CPS. Nonetheless, even for isolated, air-gapped networks, hardware ports (*e.g.*, USB or OBDII) and privileged control interfaces can serve as useful vectors by which an attacker can compromise a system. Therefore, any cybersecurity defense must consider the threat model and access vector of the attacker.

Remote Attackers can compromise a system by supplying maliciously formed input over a legitimate communications channel. This input is designed to exploit a bug in message parsing or handling logic, which causes the system to enter an attacker-dictated state. Malicious input includes sensor data, control commands, and low-level network-packet data. It is important to note that the attack surface extends beyond the application listening on that channel. The OS managing the network hardware, drivers that encode/decode data, and support libraries handling message processing are all potential targets for exploitation.

Hardware ports (*e.g.* USB) can allow a system to be compromised even if it is isolated from attacker-accessible networks. Due to the generality of the protocol, an OS queries a newly connected USB device to determine its function. Malicious devices can mis-represent themselves, causing, for example, a thumb drive to be treated as a keyboard. The malicious drive can then log or inject keystrokes, as though it were a user. Other hardware ports, such as Thunderbolt, allow a high-throughput operating mode for Direct Memory Access (DMA), which bypasses the OS and memory permissions and isolation. A malicious Thunderbolt device could thus potentially overwrite the entire OS with code of its choosing, corrupt a target application, *etc.*

Privileged control interfaces may be leveraged by an attacker to put a CPS into an unsafe or unrecoverable state. Even if these interfaces are not intended to be remotely accessible, attackers may still be able to acquire access. Unintended bridging of the cellular network to automobile CAN buses, for example, has been demonstrated by exploiting a vulnerability in the car's infotainment package [6]. In networked CPSs, automated malware can seek out and steal the authentication credentials of privileged users, and interact with software-based control interfaces using this elevated access. Malicious insiders can also issue such commands. While not a cybersecurity problem directly, insider attacks demonstrate that it remains necessary to architect a CPS such that it is resilient to commands that may place it in an unsafe or unrecoverable state.

2.2 Common Attack Techniques

While cyberattacks can leverage myriad techniques in an attempt to compromise a system, there are a few strategies that are heavily relied on due to the systemic, widespread vulnerabilities that they can exploit. Understanding these techniques can enable CPS designers to better architect new systems, and to understand where their efforts should be focused in defending extant systems.

Memory corruption exploits leverage software bugs in compiled languages such as C and C++. These bugs enable an attacker, via a malicious input, to corrupt program state that is adjacent in memory to the object storing the attacker-provided input. Once memory corruption has occurred, an attacker can completely subvert the running process, read any data in memory, and execute arbitrary code of their choosing. Memory corruption is particularly powerful in the embedded and bare-metal domains, where memory isolation mechanisms like virtual memory do not exist. This can allow a corrupted process to corrupt other running processes, and potentially the OS itself.

Command injection exploits target logical bugs in applications which can receive a mix of data and control messages. The attacker provides an input, which is intended to contain only data, but the vulnerable application treats some or all of the message as control commands. A well-known example is SQL injection, which uses the ';' character to denote the end of a command. By appending a semi-colon and a command on to the end of a data field, attackers can induce an SQL database to execute that command. Command injection is also frequently leveraged against complex file formats (*e.g.* streaming audio/video), in which one part of a file is used to inform the process about how to interpret another part of that file.

Message forgery and corruption exploits the lack of confidentiality and integrity checks on messages sent over a communications channel (*e.g.* a shared bus). Attackers either craft and inject a message of their choosing (including source and destination addresses), or capture, modify, and re-inject an originally benign message. Such attacks can corrupt sensor data, execute control commands, cause unexpected device actuation, etc. Cryptographic protections can eliminate forgery and corruption attacks entirely, but are often difficult to implement on resource-starved systems. This is especially true in real-time contexts, where the added latency can cause deadline misses.

Side channels are a popular attack technique in general-purpose computing to break confidentiality. However, the applicability of such attacks in CPS systems given the attacker access vectors should be carefully considered before developing defenses. For example, cache-based side channels are relevant threats to consider in a cloud-computing environment, in which a malicious co-tenant may infer secret information from another co-tenant (*e.g.* secret key). Many CPS applications do not share common processing hardware between the victim and a potential attacker process, and therefore such threats are less pertinent. However, an attacker may infer confidential information via other side channels in CPS systems, which they may be able to read more easily (*e.g.*, network timing).

3 Recommendations

We next elaborate on several recommendations to address these threats to CPS within the OS.

Memory Safety and Operating Systems. Given the prevalence of memory-corruption bugs and their ease of exploitation particularly on impoverished embedded platforms that lack standard OS features (*e.g.*, virtual memory or ASLR), memory safety can pro-

vide tangible benefit in securing CPS systems. Memory-safe programming languages provide the spatial (*e.g.*, preventing buffer overflows) and temporal (*e.g.*, preventing use-after-free bugs) safety guarantees using a combination of compile-time checks, language restrictions (*e.g.*, lack of pointer arithmetic), and runtime checks. Given the often limited resources in CPS systems, memory-safe languages that do not have a large language runtime and are designed for system programming (*e.g.* Rust) provide some of the most viable options. In addition, because of the lack of OS isolation primitives in many classes of CPS systems (the typical user/kernel space isolation), OSes that are themselves developed in a memory-safe language (*e.g.*, Tock OS [5] developed in Rust) provide a more complete protection. To our knowledge, no RTOS is developed in a memory-safe language.

Isolation. Legacy code or binaries for which the source code is no longer available may preclude porting an entire system to a memory-safe language. In such cases, isolation can provide a weaker alternative. Isolation ensures that modules of a CPS system are separated in such a way that they can only interact along well-defined and often limited interfaces. A popular approach for realizing isolation is using a separation kernel, such as a microkernel, that implements strong non-interference among the processes that run atop it. seL4 [4], for example, can provide such a separation kernel. Further, seL4's implementation is formally proved to match its specification on a number of platforms given a set of assumptions about the hardware and the hard-coded assembly code. seL4 and similar microkernels have even been ported to resource-limited, embedded systems (such as Raspberry Pi) that are used frequently in CPS architectures.

Recovery. Unlike traditional computing systems, cyber-physical systems control real physical devices that may be capable of catastrophic and deadly damage if operated unsafely. Availability is also a serious concern in many cases, like drones and other vehicles, where real-time corrections are needed to keep the physical system safe and stable. As a result, the crucial issue of how to mitigate and recover from a detected attack must be addressed. Simply crashing the program, as frequently done in general-purpose systems to prevent integrity and confidentiality compromise, is not acceptable.

This area of recovery and restoration after an attack has not yet been extensively studied. Such a recovery function needs to be able to restore software to its uncorrupted state. More challengingly, it needs to ensure that the entire system, including hardware and software components are in an understood and consistent state, possibly via reinitialization. Finally, all of this needs to be done within the real-time constraints of the system. Such work has many parallels with mixed-criticality scheduling [1], as well as fault-tolerant computing. However, the models and assumptions are different. For example, cyber attacks are not the result of a stochastic process, so the threat model must be carefully considered, as well as how such threats and their detection are incorporated into the recovery process.

Minimization of privileges and trust. Another way to protect CPS systems is to minimize privileges of system components and trust among components. In the CPS domain, privileges and trust be viewed through the lens of both the cyber and physical domains. Therefore, control processing, which interfaces with physical hardware devices (*e.g.*, sensors and actuators), may be crucial to the safety, integrity and availability of the system, and thus must be trusted more than other userspace applications in general-purpose systems. Components in CPS systems should only be able to read or modify exactly the state, either cyber (*i.e.*, in memory) or physical (*i.e.*, through sensors or actuators) required to accomplish its functionality.

Accomplishing this is harder than it sounds because standard ring-privilege models used in many processors are broad and hierarchical. This results in many components with the same or overlapping privileges and, therefore, ability to modify the same state. As a result, many different components could be compromised and used to cause the same impact. Instead, privileges should be fine-grained and orthogonal, such that different components have non-overlapping privileges. This restricts the set of components that can be used to cause some impact. Fine-grained and orthogonal permissions are not common in today's systems, but tagged architectures and capabilities are both promising ways to provide these features. Other software-based approaches have proposed decomposing applications into sub-components based on the permissions required, and peripheral devices accessed [2].

One powerful idea enabled by orthogonal privileges is the low-privileged operating system, which is actually more restricted than userspace in its permissions. Certainly, it would still have ability to do memory management and context switching, which traditionally require high privilege, but it would be restricted in other ways, like being unable to examine userspace memory without explicit permission. Such a design would reduce the impact of an attacker who exploited a bug in the OS.

4 Conclusion

The physical nature of CPS gives rise to unique OS design considerations both for performance (*e.g.* real-time constraints), as well as security. Cyber attacks against CPS may have different intentions and access vectors then threats to general-purpose systems. For example, threats to availability and physical safety are higher-priority threats than against general-purpose systems. Such threats necessitate more secure OS designs for CPS. We have summarized several recommendations to address these types of threats, including memory-safety, isolation, recovery, and trust/privilege minimization. Importantly, incorporating these recommendations into a CPS-focused OS will have different performance and design considerations (*e.g.* real-time correctness), then when realized in a general-purpose systems.

References

[1] A. Burns and R. Davis. Mixed criticality systems – a review, 2018.

[2] A. Clements, N. Almakhdhub, S. Bagchi, and M. Payer. ACES: Automatic compartments for embedded systems. In *Proceedings of USENIX Security*, 2018.

[3] Adrian Dabrowski, Johanna Ullrich, and Edgar R. Weippl. Grid shock: Coordinated load-changing attacks on power grids: The non-smart power grid is vulnerable to cyber attacks as well. In *Proceedings of ACSAC*, 2017.

[4] Gerwin Klein, Kevin Elphinstone, Gernot Heiser, June Andronick, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, et al. seL4: Formal verification of an OS kernel. In *Proceedings of SOSP*, 2009.

[5] Amit Levy, Bradford Campbell, Branden Ghena, Daniel B. Giffin, Pat Pannuto, Prabal Dutta, and Philip Levis. Multiprogramming a 64kB computer safely and efficiently. In *Proceedings of SOSP*, 2017.

[6] Charlie Miller and Chris Valasek. Remote exploitation of an unaltered passenger vehicle. *Black Hat USA*, 2015.