“Bolt-on” Network Security for Advanced Manufacturing Deployments

Submitted in partial fulfillment of the requirements for
the degree of
Doctor of Philosophy
in
Electrical and Computer Engineering

Matthew M. McCormack

BS, Electrical and Computer Engineering, United States Air Force Academy
SM, Aeronautical and Astronautical Engineering, Massachusetts Institute of Technology
MS, Flight Test Engineering, United States Air Force Test Pilot School

Carnegie Mellon University
Pittsburgh, PA

December 2020

Acknowledgments

I am deeply grateful to many outstanding people who helped make this dissertation possible.

I am immensely grateful to my advisor, Vyas Sekar, for being a tremendous counselor and mentor. Vyas has been instrumental in teaching me how to approach a problem from first principles and articulate my thoughts. His sharp questions and comments have always pushed me to dig deep into details. I am thankful for his honest feedback that improved me as a researcher. I am extremely lucky to have had his insights and encouragements along this journey.

I would also like to thank my committee for their support and encouragement: Grace Lewis, Anthony Rowe, and Sandra DeVincent Wolf. Thanks to Grace for her leadership of the KalKi project and an introduction to the Software Engineering Institute. Thanks also to Sandra for the introduction to the metal additive manufacturing world and sharing your enthusiasm for better understanding how to secure it. While I have not had the pleasure to collaborate closely with Anthony, I have deeply appreciated his feedback and questions.

I am also fortunate to have worked with many great collaborators: Guyue Liu, Tianlong Yu, Sanjay Chandrasekaran, Brian Singer, Amit Vasudevan, and Sebastián Echeverría. I also benefited from many fruitful discussions with other CyLab graduate students: Rahul Sharma, Sekar Kulandaivel, Soojin Moon, Jarrett Booz. Further, I am greatly indebted to Todd Baer for his willingness to share his experiences in the manufacturing domain and provide unfettered access to multiple metal additive machines.

As a Ph.D. student, I spent a significant amount of my time at CyLab. I would like to thank the members of CyLab for providing a welcoming and collaborative environment. I am grateful to the remaining CyLab faculty for their advice, our research group members for many discussions, and the faculty and students of the Tuesday systems seminar for their valuable feedback on my talks. The administrative aspects of the Ph.D. programs were easy thanks to the staff members at CyLab and the Department of Electrical and Computer Engineering: Brigette Bernagozzi, Toni Fox, Karen Lindenfelser, Chelsea Mendenhall, Jamie Scanlon, and Nathan Snizaski.

Thank you to my church family at Redeeming Grace Church. I was extremely blessed by your fellowship and encouragement. Thank you for your prayers and the many ways you blessed my family during our time in Pittsburgh.

I saved the most important for the last. I am indebted to my family for their unconditional support, prayers, and encouragements. To my wife, Meghan, this would not have been possible without you and your never ending love and encouragement. You are a truly incredible woman, and I thank God that He brought you into my life. Thank you for the unending love you show our four children. Matthew Charles, Margaret, Christopher, and Louise, I also could not have done this without your love and support. Thank you for your encouragement and hugs.

The work presented in this thesis was supported in part by the CONIX Research Center, one of six centers in JUMP, a Semiconductor Research Corporation (SRC) program sponsored by DARPA. The research was also supported by the Carnegie Mellon University (CMU) Manufacturing Futures Initiative, made possible by the Richard King Mellon Foundation, with additional support from the CMU NextManufacturing Center. Additionally, the Software Engineering Institute, a federally funded research and development center sponsored by the United States Department of Defense also supported this work. Further support was provided by NSF award CNS-1564009. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the United States Air Force or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on.

Thesis Committee Members:
Vyas Sekar (Chair)
Grace Lewis
Anthony Rowe
Sandra DeVincent Wolf

Abstract

Industry 4.0 is driving manufacturing centers to utilize networked devices, many of which are potentially deployed with security vulnerabilities. Unfortunately, these devices often lack effective host-level protections and may have service lives beyond the vendor's support. At the same time, traditional network security solutions, such as firewalls, often leave coverage gaps and lack the necessary trust to ensure they do not become launchpads for future attacks. Therefore, adopting Industry 4.0 potentially amplifies the manufacturing domain's attack surface, creating new ways for attackers to steal proprietary data, sabotage manufacturing operations by making defective parts, and deny users access to critical machines.

This dissertation aims to design a practical system for defending manufacturing deployments from network attacks. We leverage advances in software-defined networking to provide device-specific network protections that can be "bolted-on" to existing manufacturing networks in the form of a security gateway. Such a bolt-on approach allows for protecting existing machines without requiring modifications to the machines or their software. For a security gateway to be effective it must (1) be able to identify and mitigate vulnerabilities present in manufacturing devices, and (2) be trusted to enforce these protections even when the gateway itself is under attack.

The key contributions of this thesis are the following. We build a vulnerability assessment tool, C3PO, for analyzing networked 3D printers and their deployments, which we then use to evaluate 13 networked 3D printers and 5 manufacturing center deployments. Our evaluation identified common vulnerabilities such as susceptibility to denial of service attacks, not encrypting sensitive data in transit, and a lack of network isolation. These identified vulnerabilities inform the device-specific network protections the security gateway must provide. Next, we design a low-cost, trusted security gateway system, Jetfire, by building on top of a micro-hypervisor root of trust. We use formal modeling to guide the application of micro-hypervisor provided capabilities to provide an end-to-end guarantee that all packets are processed by the correct network protection (e.g., those identified by C3PO). We then demonstrate how this trusted architecture can be used to secure networked 3D printers by mitigating identified vulnerabilities as well as providing more elaborate protections such as behavior-based anomaly detection.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The manufacturing domain is undergoing a revolution that is creating new cyber security risks. These security risks arise from a heterogeneity of newly networked devices which potentially allow an attacker to steal data or alter operations. As the industry is dependent upon its currently deployed, high-cost infrastructure, a practical solution is needed that allows for the continued usage of legacy machines while mitigating their security vulnerabilities. Such a solution must provide a trusted means of mitigating the vulnerabilities of heterogeneous machines without altering their operations. This thesis seeks to answer how to build a practical system for defending manufacturing deployments from network attacks.

1.1 Industry 4.0 Magnifies the Need for Network Security

The new revolution in manufacturing, often referred to as "Industry 4.0" [75], is increasing the attack surface of a manufacturing center by connecting more machines to the network. Consider the small manufacturing shop depicted in Figure 1.1, where during its transition to Industry 4.0 it incorporates new technologies such as Industrial Internet of Things (IIoT) and 3D printing while simultaneously connecting everything to a common network. Legacy manufacturing machines (e.g., the milling machine in Figure 1.1) though not designed with network operations in mind,

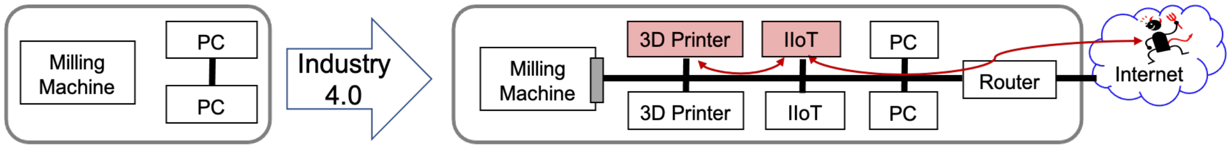


Figure 1.1: Example manufacturing center transitioning from traditional operations to Industry 4.0 operations. Where Industry 4.0 introduces new technologies (3D printing), potentially weak devices (IIoT), and connects these and legacy devices to a network. These changes increase the potential for network attacks.

are connected to the network to increase their productivity. Furthermore, new devices such as networked 3D printers and IIoT devices are deployed to provide new capabilities (e.g., on-demand production environment details [29]) and collect additional data that informs future actions (e.g., predictive maintenance[166]). While these changes promise many beneficial impacts, they also bring new security concerns as also observed in prior work [21, 114, 223].

Unfortunately, attackers are increasingly taking advantage of these vulnerabilities [44, 90, 98, 119, 159, 163]. Verizon's 2020 Data Breach and Incident report noted that the number of reported attacks in the manufacturing domain have tripled [204]. With the cyber security company Rapid 7 noting that small organizations are especially experiencing this and seeing a significantly a higher rate of attacks (accounting for over 80% of reported attacks) [163]. These attacks are coming from the network, as the Industrial Security Company Dragos noted their 2020 Threat Report where 70% of the attacks they evaluated relied upon an attacker having network access [60]. According to a 2020 NSA advisory, the goal of these attacker's is often to: disrupt the manufacturing process's operations (such as modifying machine actions or making them unavailable) and to steal proprietary information [137]. The adoption of Industry 4.0 increases a manufacturing center's attack surface and magnifies network vulnerabilities that allow an attacker to steal proprietary data [57, 243] and cause incorrect operations [153, 235] (see Section 2.1 for a detailed taxonomy and discussion about potential attacks). The manufacturing domain needs a practical security solution to protect it from these and other network attacks.

1.2 Manufacturing Domain Network Security Challenges

A practical security solution must protect both legacy and new machines from both remote and insider network attacks that aim to steal data, sabotage machine operations, or deny network access to the machine. This is challenging due to: (1) protecting a heterogeneous set of already deployed devices and (2) the providing a trust guarantee that network protections are applied.

- **Protecting a heterogeneous set of already deployed devices (C1):** Manufacturing deployments are likely to contain a mix of heterogeneous devices from multiple vendors in order to support diverse operations. For example, consider 3D printing, where a single manufacturing center might have multiple devices to support different materials (e.g., metal vs polymer) and manufacturing techniques (e.g., electron beam vs binder jetting). These heterogeneous devices are often produced by different vendors and often lack a unifying protocol (e.g., PJL, PostScript on paper printers [131]), resulting in a plurality of proprietary protocols. This diversity of protocols makes it challenging to identify and mitigate each device's network vulnerabilities. Further, these devices often have limited support for installing or modifying software, which limits solutions such as installing antivirus or other specialized hardware/software on the machine (such as requiring each device to have a trusted execution environment [167]), as they potentially leaving devices unprotected. Similarly, we want to be able to continue using existing tools and not require developing new workflows for deployed machines. This makes approaches such as deploying Azure Sphere's Guardian impractical as they require developing new cloud-based operations [126]. Thus, we need a solution for defending deployed devices.

- **Providing a trust guarantee that network protections are applied (C2):** Manufacturing machines are often specialized for a single purpose (e.g., 3D printing metal parts) and often have planned lifetimes that are >10 years [61, 110], which can exceed the vendor's existence or software support, potentially resulting in network-connected machines with software vulnerabilities. Alternatively, keeping the software up-to-date requires a software vigilance not traditionally required in the manufacturing domain, especially for security updates that do not directly impact the machine's operational performance. These factors increase the potential of manufacturing machines being vulnerable to network attacks that can result in high financial consequences (e.g., proprietary designs being stolen [243], replacing defective parts [21], machine down-time [196]). Thus, any security solution must guarantee the necessary network protections are enforced, and that any protections do not become launchpads for future attacks. This guarantee needs to be provided at a low-cost to support protecting small manufacturing centers which are currently experiencing an increased number of attacks [163]. Further, such a low-cost solutions should support commonly available hardware and run existing software.

Unfortunately, existing solutions are insufficient for protecting the increased attack surface of Industry 4.0 manufacturing centers. Simply removing vulnerable devices or creating an air gap between the network with manufacturing machines and all other networks is rarely realized in practice [110]. Similarly, traditional security solutions are either not applied (e.g., reluctance to patch software due to concerns it will negatively impact operations [25]) or provide limited coverage (e.g., firewallls deployed at the network perimeter that only enforce coarse-grained protections [227]). While others require devices have specialized hardware (e.g., secure enclaves [167]) or developing new workflows to interact with deployed devices (e.g., developing new cloud based interfaces [126]) that require significant modifications to integrate with existing deployed manufacturing machines. Existing solutions are limited in their ability to address Industry 4.0's security concerns and leave many deployments vulnerable to network attacks such as a malicious insider exploiting an unpatched vulnerability to sabotage a machine's operation.

1.3 Thesis Overview

This thesis aims to address the above challenges by designing a trustworthy network security gateway. A gateway architecture provides a means of "bolting on" network security protections without installing or modify the manufacturing machines' software [95, 227, 229]. The application of software-defined networking techniques allows for transparently applying agile, device-specific protections to all of a machine's network traffic (both local and external). Further background on these security gateways is provided in Section 2.2.2.

Threat Model: The security gateway must provide these defenses in the presence of a strong network attacker with a goal of exploiting a manufacturing device. Our attacker has knowledge of the security architecture as well as network access to all devices, allowing the attacker to inject/modify network messages as well as compromise a device's software stack (including the gateway itself). We limit our attacker to not have physical access. Thus, we need trust that the gateway's protections are applied in the presence of such an attacker.

Requirements: Deploying a trusted security gateway in a manufacturing deployment requires:

- **Understanding the vulnerabilities** of the machines in a given manufacturing domain. This knowledge allows for identifying the necessary device-specific network protections the security gateway must implement.
- **Trusting the security gateway** to perform the specified protections, even when under attack. Such an end-to-end guarantee needs to be available on low-cost hardware.

Towards this, we look to address the aforementioned challenges with respect to networked 3D printers. We select networked 3D printers as being representative of the Industry 4.0 changes to the manufacturing domain. We start by presenting our thesis statement and our key technical contributions. Next, we provide an outline of how this dissertation is organized.

1.3.1 Thesis Statement

This dissertation shows that it is possible to provide small to medium sized 3D printer deployments (e.g., less than 30 networked 3D printers) with trusted, low-cost (e.g., less than $100) device-specific network security. To this end, we (a) build analysis tools to measure network vulnerabilities in networked 3D printers and their deployments, and (b) design and implement a low-cost, trusted system for securing networked 3D printer deployments from network attacks.

1.3.1.1 Research Contributions

In the context of networked 3D printer security, this dissertation makes two contributions:

- **Security Analysis of Networked 3D Printers and their Deployments (Chapter 3):** We design and implement an open-source tool, C3PO [28, 120], for performing systematic security evaluations of networked 3D printers and their deployments. We use C3PO to analyze 13 networked 3D printers (representing 9 vendors, across the spectrum of costs and printing processes) and 5 real-world manufacturing deployments. We identify vulnerabilities that allow an attacker to perform attacks such as driving the printer into a part, modifying the printing instructions "on-the-wire", and Denial of Service (DoS). Specifically, we noted the following. All 13 networked 3D printers were vulnerable to simple DoS attacks (e.g., SYN flood); some requiring a power-cycle to recover. Most (12 of 13) did not encrypt data in transit; all sent plaintext metadata. 6 of 13 were vulnerable to either a published exploit (such as WannaCry [125]) or network inputs that crashed the machine. The deployments often unnecessarily placed networked 3D printers on publicly accessible networks, allowing networked 3D printers to be remotely accessed via IP. Additionally, deployments contained a significant proportion (>41%) of embedded devices (e.g., cameras) that could be used as potential launchpads for future attacks.

- **Low-cost, Trusted Security Gateway (Chapter 4):** We design and implement a low-cost, trusted security gateway [47, 121]. The design leverages formal modeling to systematically identify key loci for applying relevant protections to a software-defined security gateway architecture to enable trusting the gateway's packet processing. We adopt a *micro-hypervisor-based system architecture* as it provides a root-of-trust that supports integrating protections in commodity software across a broad hardware base (to be cost-effective) and allows for rapidly adding new protections (providing extensibility) [9, 198, 200]. As determined by our formal model, we use key micro-hypervisor provided capabilities such as *attestation* to verify software instances, *mediation* to enforce correct packet routing, and *isolation* to prohibit tampering. We integrate these protections into the software processing packets on the gateway and controller to provide an end-to-end guarantee that all output packets are processed by the correct middlebox in a known good state. Finally, we enable fine-grained per-device security policies on low-cost platforms by reducing the footprint of canonical security middleboxes such as intrusion prevention systems (IPS).

1.3.2 Outline

The remainder of this dissertation is organized as follows (with an overview in Figure 1.2). Chapter 2 surveys related work identifying security vulnerabilities in 3D printers and broader networked devices. Additionally, it discusses network security and approaches for integrating trust into the underlying security architecture. We build upon this background by proposing a tool for measuring 3D printer's network vulnerabilities in Chapter 3. We derive insights from the identified network vulnerabilities to inform the design of potential network defenses for mitigating these known vulnerabilities. In Chapter 4, we design and implement a low-cost, trusted security gateway, Jetfire, to ensure that all output packets are processed by the correct network function. This gateway provides a platform for bolting-on defenses to deployed networked 3D printers. We then combine these in Chapter 5 and demonstrate how diverse defenses can be run on top of our trusted gateway to defend manufacturing deployments. First, by showing how known vulnerabilities (e.g., those identified by Jetfire in Chapter 3) can be mitigated using existing network functions (e.g., firewalls to block unused open ports). Second, by showing how unknown vulnerabilities can potentially be mitigated using state-of-the-art tools to apply data-derived limits to each device's network traffic (e.g., learning and enforcing Manufacturer's Usage Description specifications to provide access control). We then conclude in Chapter 6 with limitations and directions for future work.

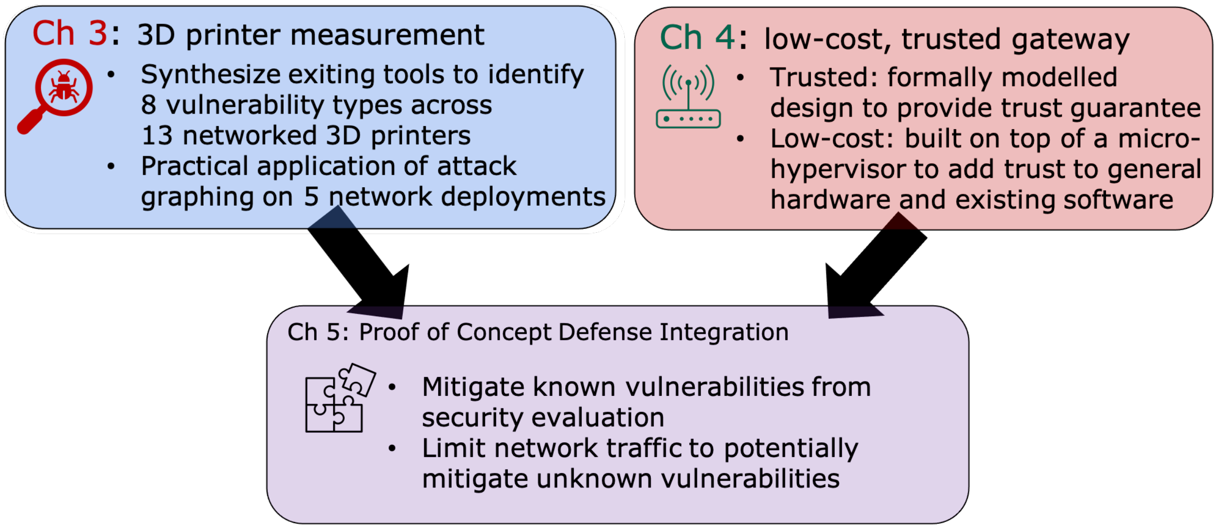

Ch 3: 3D printer measurement
• Synthesize exiting tools to identify 8 vulnerability types across 13 networked 3D printers
• Practical application of attack graphing on 5 network deployments
Ch 4: low-cost, trusted gateway
• Trusted: formally modelled design to provide trust guarantee
• Low-cost: built on top of a micro-hypervisor to add trust to general hardware and existing software
Ch 5: Proof of Concept Defense Integration
• Mitigate known vulnerabilities from security evaluation
• Limit network traffic to potentially mitigate unknown vulnerabilities

Figure 1.2: Thesis Outline: Chapter 3 discusses our measurement of network vulnerabilities in 3D printer deployments, Chapter 4 discusses our low-cost, trusted gateway system, and Chapter 5 discusses our integration of defenses onto the gateway.

Chapter 2

Related Work

In this chapter, we provide background on potential threats to networked 3D printers and security solutions which motivate using software-defined security gateways as a pragmatic approach for defending networked 3D printers. We build this background by presenting two taxonomies, first a taxonomy of security threats for 3D printers and second a taxonomy of prior security solutions. Currently, networked 3D printers lack a systematic measurement of network vulnerabilities exposed by deployed 3D printers. To define a baseline set of network vulnerabilities, we create a taxonomy of threats that allow an attacker to steal data, print defective parts, deny network access, or cause physical damage. This baseline aids in determining the types of protections that a security solution must provide. Our taxonomy of security solutions presents how recent work in software-defined networking allows for integrating classical security solutions into a local security gateway to mitigate potential threats to a networked 3D printer. Finally, we look at approaches for trusting these software-defined architectures.

2.1 3D Printer Threat Landscape

Multiple works have taxonomized the attack surface of 3D printers [142, 215, 222, 223, 224]. We briefly describe a 3D printing workflow to aid understanding prior attacks and defenses. This

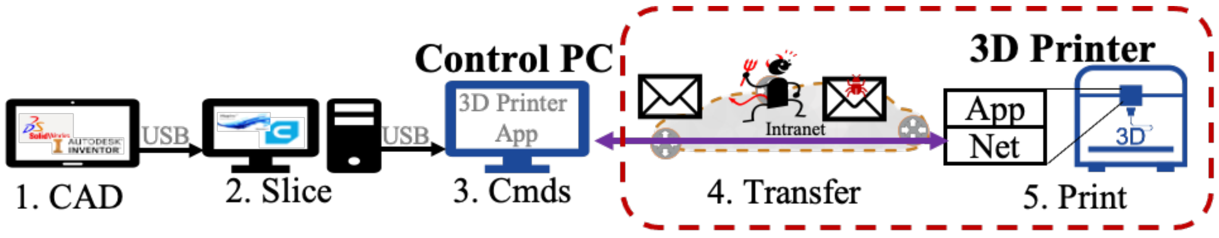


Figure 2.1: General 3D printing workflow, we focus on security risks inside the red box.

thesis focuses on threats from a network attacker. To understand how these attacks from prior surveys could potentially be realized by a network attacker, we analyze network vulnerabilities identified in other networked devices (e.g., IoT, office printers, etc.) to identify the types of potential threats a security solution for networked 3D printers must defend against.

2.1.1 3D Printing Workflow

Additive manufacturing, often referred to as 3D printing, creates a physical object by sequentially joining layers of deposited material [16]. This process enables fabricating structures that are not possible with traditional manufacturing methods [223]. The future of manufacturing relies on 3D printing as it reduces the cost of building complex parts, allows rapid design iteration, and enables on-demand production [29].

A networked 3D printer's operating model mirrors that of a peripheral device (e.g., an office printer), and differs from many IoT devices (e.g., [8]) in two ways: (1) 3D printers lack mobile apps,[1] (2) networked 3D printers are primarily accessed by PCs on the local network whereas IoT devices often interface with vendor cloud endpoints. Across printing processes (e.g., fused deposition modeling, electron beam, etc.) a common workflow is used for interacting with a networked 3D printer. We specifically detail sending a print job, as most other interactions are a subset of this workflow.

Workflow: A typical 3D printing workflow (shown in Figure 2.1) consists of the following five steps (where the first three steps can either be performed on the same host or multiple hosts).

[1]Some vendors are beginning to release mobile apps for remote monitoring [116, 147].

1. *Generate digital representation (CAD).* First, a digital representation of the object to be printed must be created, often as a stereolithography file (*.STL).

2. *Convert to vertical layers (Slice).* The digital representation is then divided (or sliced) into vertical layers, defining the sequential layers that will be deposited during the printing operation.

3. *Convert to printing commands.* Each layer is compiled into a set of machine-specific commands (e.g., G-code[2] [101]) that define the printing actions for each layer. Additional metadata such as printing speeds and layer height is included with these commands.

4. *Transfer commands over the network.* Printing commands are generated for every layer and placed into a file (with other metadata such as the number of layers) that is sent over the network to the 3D printer. Most networked 3D printers have an open network socket waiting to receive commands from a control PC.

5. *3D Print.* Once the networked 3D printer receives the file, it is placed into non-volatile storage and added to a local print queue. The commands in the file are then either executed immediately or after a user action (e.g., pushing a button).

From this baseline on 3D printer operations, we next look at published attacks and defenses.

2.1.2 3D Printer Security Evaluations

Prior works taxonomizing the attack surface of 3D printers have broadly grouped theoretical and demonstrated attacks based upon their impact into two categories: physical impacts (e.g., sabotage machine operations, altering the part's structure making it defective) and cyber impacts (e.g., theft of intellectual property, denial of service). We discuss demonstrated attacks within each of these categories below.

[2] G-code was used by 3 of the 13 networked 3D printers we analyzed (Section 3.3.2).

Attacker's Goal	Design Files	3D Printer	**Network**
Sabotage Operations		Modify firmware [71] Firmware static analysis [128]	Out-of-limit cmds [153] Hazardous cmd
Print Defects	Add void [21, 186] Modify orientation [234]	Modify firmware [130, 179] Side-channel detection [18, 20, 185]	Modify cmds on-the-wire
Steal Data	Malware steals CAD [243] Watermarking [68, 79]	Side-channels [4, 181] Min acoustic side-channel [40]	Spoof printer [57]
DoS			Make printer unavailable

Attack Vector

Table 2.1: Prior work on security in advanced manufacturing, characterized by attacker goal and the attack vector. Previously shown attacks are red and defenses are blue. The shaded cells denote our contributions, demonstrating attacks in violet.

2.1.2.1 3D Printer Attacks

Attacks with physical impacts result in malicious changes to either the machine's operations or the physical parts being printed. Attacks with cyber impacts do not alter the physical part created but look to provide the attacker an advantage such as proprietary information or making the networked 3D printer unavailable.

Sabotage Operations: Altering a 3D printer's firmware can result in catastrophic failures (e.g., the machine causing physical damage [71]). Similarly, more subtle changes to the firmware can ensure printed parts do not meet their prescribed specification (e.g., extruding too much material [130], reducing metal fusion by altering measured temperature [179]). To gain an understanding of the potential for these and other attacks on the 3D printer's firmware, Moore *et al.* [128] analyzed an open source 3D printer firmware (for polymer fused deposition modeling machines) where they noted the potential for buffer overflows due to the use of `strcat` function and fixed sized global buffers. 3D printers currently lack input filtering to detect these types of malicious inputs. Further, an attacker can use a 3D printer's interface maliciously, such as Do *et al.* [57] noted, where a network attacker could maliciously send commands to stop the current printing operation. This is possible due to networked 3D printers having poor authentication of entities they accept commands from.

Print Defects: 3D printers are also vulnerable to attackers altering the design files (i.e., STL files) such that they produce an object with similar external dimensions but altered physical properties. A particular concern, is adding internal defects such as voids that reduce the final part's physical properties but cannot be easily detected [186, 234]. Prior work has looked at modifying these files while they are on a control PC prior to being sent to a 3D printer, with one end-to-end example that resulted in a latent defect [21].

Steal Data: Prior work looking at purely cyber impacts on additive manufacturing has predominantly focused on stealing a part's digital design data (e.g., the CAD file for a 3D printed part) using malware or side channels (e.g., acoustic data to recreate a 3D printer's operations). The ACAD/Medre.A worm [243] propagated through PCs used for generating part's digital representations and sent copies of CAD files to an attacker's server. Multiple works have studied the use of side channels (e.g., acoustic measurements from a cell phone) to allow recreating a part's digital design. The side channels analyzed include: acoustic [3], acoustic and magnetic [77, 181], acoustic and gyroscopic [18], and thermal [4]. In terms of attacks leveraging the network to steal data directly from a networked 3D printer, Do *et al.* [57] analyzed a single type of networked 3D printer and identified vulnerabilities in its authentication mechanism that allow an attacker to retrieve data stored locally on the 3D printer (i.e., the current and previously printed part files). Unfortunately, this analysis was specific to a single vendor's network protocol and not easily generalizable to other 3D printers.

Denial of Service (DoS): While many of the taxonomies have indicated the potential for DoS attacks [215, 222], these specific types of attacks have not been demonstrated by prior work analyzing the security of networked 3D printers.

2.1.2.2 3D Printer Defenses

Unfortunately most defenses for networked 3D printers proposed in the literature have targeted protecting intellectual property using techniques such as watermarking [68, 79, 80] or by minimizing side channels [40]. Others have relied upon side channels as a means of detection using: acoustics [20, 39], impedance [6, 205], and power consumption [129]. As noted by Belikovetsky *et al.* [21] manufacturing environments have historically relied upon an *air gap*, where manufacturing machines are completely isolated from other networks (e.g., business networks). Unfortunately, this is often not realized in practice as highlighted by McGurk, who after conducting hundreds of vulnerability assessments noted an average of 11 direct connections between production and enterprise networks [110]. Thus a pragmatic security solution is needed to protect networked devices in the manufacturing domain [19, 27].

2.1.3 Internet of Things Security Vulnerabilities

Industry 4.0 is adding networked devices to the manufacturing domain. To analyze network-specific security threats, we start by reviewing existing standards [81] and best practices [52, 141] to gain an understanding of potential network vulnerabilities. Due to limited data on security evaluations of networked 3D printers, we broaden our search space and look at vulnerabilities identified on other networked devices (e.g., IoT, office printers, etc.). We use this background to provide an understanding of potential network security threats (summarized in Table 2.2).

2.1.3.1 Standards and Best Practices

Industry standards have been created for networked devices to promote secure design and operation. Additionally, advocacy groups have developed best practices to aid vendors who are developing new networked IoT devices.

Standards: In the manufacturing domain, the International Society of Automation (ISA) and International Electrotechnical Commission (IEC) developed ISA/IEC 62443, which provides an overarching series of standards regarding security vulnerabilities in industrial automation and control systems [81]. Within this set of standards, ISA-62443-4-2 discusses the technical security requirements for individual components. Specific to network security, it details requirements about: authentication, data confidentiality, system integrity, and availability. Additionally, ISA-95 specifies a network architecture that provides isolation between operational machines (e.g., networked 3D printers) and traditional IT devices (e.g., office PCs).

Outside the manufacturing domain, the European Telecommunications Standards Institute (ETSI) developed ETSI TS 103 645 which provides baseline requirements for the cyber security of consumer IoT devices [62]. This standard specifies similar requirements to IEC-62443, including: communicating securely, minimizing attack surface (not having open, unused ports), and validating input data among others. Similar recommendations have been made by the US National Institute of Standards and Technology (NIST) [24, 63].

Best Practices: Best practices have been generated for IoT devices; these provide general groupings of security deficiencies identified in deployed devices. The Open Web Application Security Project® (OWASP) identified the ten most common vulnerabilities in deployed IoT devices. Their list includes vulnerabilities such as insecure interfaces (where sending a malformed input causes the devices to crash) and insecure data transfer (i.e., not using encryption when sending data) [141]. Similarly, the IoT Security Foundation (IoTSF) identified 15 design items that impact a device's security, such as using encryption and signing software updates [52].

These standards and best practices highlight many foundational computer security concepts such as data confidentiality, access authentication, and least privilege. However, these concepts are not always implemented. We use the key security concepts identified in the standards and best practices to guide a survey of security vulnerabilities identified in networked devices.

Vulnerability Type	**3D Printers**	**IoT**
Insufficient Access Authentication	[57]	[1, 7, 8, 53, 66, 72, 123, 135, 178, 197]
Vulnerable Network Services	[223]	[1, 8, 10, 38, 135, 178]
Vulnerable Applications	[130]	[1, 7, 8, 66, 72, 123, 131, 133, 178]
Vulnerable Update Process		[1, 8, 72, 178]
Data Exposed During Transfer		[1, 7, 8, 13, 38, 53, 66, 72, 133, 135]
Vulnerable Default Settings	[124]	[8, 10, 38, 123, 178]
Vulnerable to DoS		[1, 7, 13, 38, 53, 72, 123, 178, 197]

Table 2.2: Prior network security evaluations of 3D printers and IoT.

2.1.3.2 Findings on Related Devices

We analyze findings from evaluations of manufacturing robots, IoT, and office printers to inform our evaluation and defense of networked 3D printers.

Robots and Drones: Closest to the manufacturing domain are works analyzing the security of robots and drones. Quarta *et al.* [55] performed a security evaluation of deployed industrial robots and noted an increase in network connectivity and a lack of security awareness (with 28 industrial robots directly accessible over the Internet). Specifically, they highlight the ability of an attacker to violate safety constraints because of weak authentication, network services exposing configuration files, applications vulnerable to buffer overflows, and software updates that do not use code signing to verify the code. Dieber *et al.* [55] analyzed the robot operating system (ROS [154]), and identified vulnerabilities due to allowing unauthorized inputs and susceptibility to DoS attacks. Similarly, drones were shown to be susceptible to DoS attacks degrading their ability to track a target [197]. These vulnerabilities and others have prompted others to look for ways to identify potentially malicious inputs [32, 237].

Security Evaluations of IoT Devices: The Internet of Things (IoT) provides a broad collection of devices being added to a network, from cameras to simple sensors. Unfortunately, researchers have identified numerous security vulnerabilities in these devices [1, 7, 8, 13, 26, 38, 53, 66, 72, 123, 133, 135, 178]. Relevant security concerns identified in these works include: data exposed during transfer (e.g., data sent without encryption, creating a man in the middle situation, ability to replay messages) and vulnerable applications (e.g., code injection vulnerabilities, back doors and other open ports, DoS). Alrawi *et al.* [8] analyzed 45 smart home devices using a conglomeration of existing security analysis tools. They found that 27 devices (60%) use partial or no encryption when sending data and 18 devices (40%) had at least one unpatched vulnerability.

Office Printers: Müller *et al.* [131] demonstrate the insecure state of networked office printers by analyzing 20 different office printers (spanning 8 printer vendors), identifying the ability to: create a DoS condition (all 20), manipulate a print job (14 of 20), and disclose information (16 of 20). Their security analysis tool, PRET, leverages a set of common network protocols and printer languages (i.e., PJL, PostScript) to allow repeating analysis across printers from multiple vendors. Unfortunately, networked 3D printers lack a common protocol or language, making this tool unable to be applied to networked 3D printers.

In order to mitigate these vulnerabilities, many vendors have been integrating security features into the printers (e.g., trusted boot, security information event management integration) [99]. A recommended practice for protecting networked office printers is to place them behind a dedicated print server which isolates the office printer on a VLAN [131]. This print server can enforce actions such as user authentication and an IPS to block malicious inputs. We next discuss potential protections that can be applied to networked 3D printers.

2.2 Network Security for Industrial Internet of Things

To address the threat landscape to networked 3D printers, we next look to understand potential bolt-on network security gateway solutions. First, we provide a background on classical network security solutions and software-defined networking. We discuss these to give a foundation for bolt-on network security gateways, which are built on top of software-defined networking constructs and provide a mechanism for realizing device-specific classical security protections to each device's network traffic. Finally, we look at approaches for adding trust to these software-defined security gateways to ensure their protections are enforced.

2.2.1 Background on Network Security Solutions

Traditional security solutions include host-based (e.g., antivirus) and network-based approaches (e.g., firewalls). Unfortunately, not all networked 3D printers can run antivirus software, due to either the inability to install additional software or limited device resources. Similarly, software patching cannot be relied upon, as networked 3D printer operators are often reluctant to apply software updates due to concerns that they may alter the machine's operations [25]. While not unique to the manufacturing domain [209], these devices have long operating lives where successful attacks have high financial impacts [196]. Thus we look to network-based approaches for defending deployed networked 3D printers without requiring any modifications to their software.

Classical Network Security Solutions: Early computer network attacks (e.g., the 1998 Morris worm [184]) spurred the development of network-based security solutions. These include firewalls to separate local and external networks [14, 83] and intrusion detection systems/intrusion prevention systems (IDS/IPS) for inspecting network traffic and identifying potential attacks [15, 73, 136]. Other network functions while not exclusively security focused can be used for security purposes. These include virtual private networks (VPN) for creating an encrypted tunnel to a network [59] and proxy servers to authenticating clients [96]. Unfortunately, none of these traditional solutions alone is sufficient to mitigate all the potential network threats to networked 3D printers. For example, an IDS can detect known exploit payloads, but cannot detect data integrity attacks that modify printing commands. As deployments grow, the management of these network protections becomes increasingly complex, leading to the potential for protections to conflict with each other [231]. Additionally, traditional network security solutions are only deployed at the network perimeter which does not protect against insider threats. Recent advances in networking provide a potential solution to these limitations, such as software defined networking and network function virtualization.

Software-Defined Networking: Software-defined networking (SDN) provides increased flexibility to networks by migrating the controlling logic from distributed hardware to a logically centralized entity [31]. This is commonly referred to as separating the data plane (i.e., hardware forwarding packets) from the control plane (i.e., logic determining where the packets should go). Separating these planes enables using software to dynamically change how packets are routed through the network. Building on these capabilities, traditional hardware functionalities (e.g., firewalls) have been implemented in software, often referred to as network function virtualization (NFV) [127]. NFV is continuing to be optimized, allowing them to run on resource-constrained hosts [118]. The combination of SDN and NFV allows for prescribing in software the specific network processing for each packet.

These advances have been used in the context of improving enterprise network security to provide network protections with context, isolation, and agility [88, 111, 173, 218, 227]. Where SDN allows the network to transparently tunnel all packets to a unique network function, specific to that device. Each of these network functions are realized using NFV, which allows them to be dynamically instantiated and reconfigured. Such techniques have proven effective for enterprise networks with excess computing capability (e.g., spare servers within a data center) to provide multifaceted protections against both insider and remote threats. This success has led to applying these software-defined principles to local network security gateways in small networks (e.g., to protect IoT devices in home networks).

2.2.2 Network Security Gateways

A software-defined gateway architecture [17, 23, 51, 95, 158, 176, 227, 229] has been proposed to secure IoT deployments. Farris *et al.* [64] conducted an in-depth study on the applicability of SDN and NFV for providing security in the IoT domain. They note that SDN and NFV complement traditional IoT security solutions (e.g., authentication, data filtering, encryption) and furthermore add robustness in terms of scalability and agility (ability to change protections). With the potential for a high-overlap between the vulnerabilities in networked 3D printers and IoT devices, we look at some of the proposed software-defined gateway architectures.

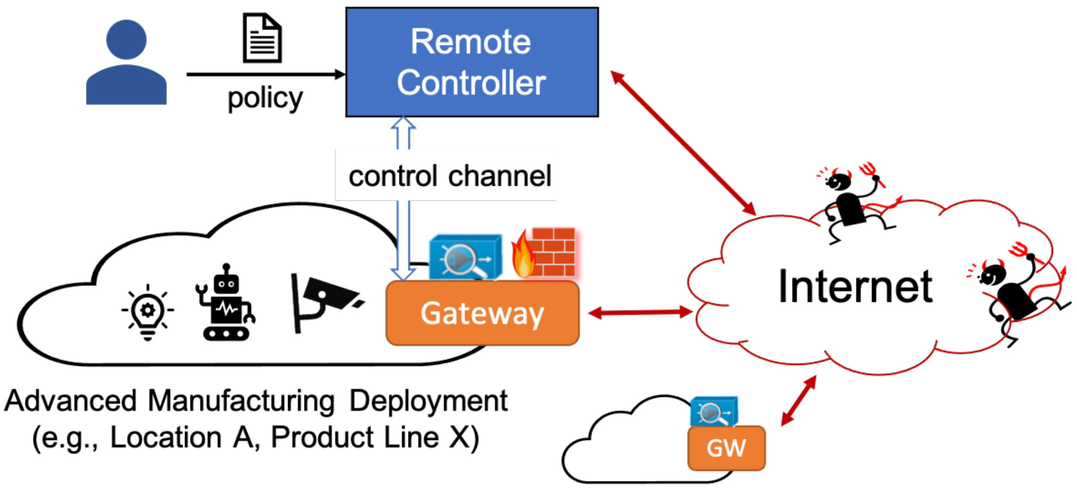

policy
Remote Controller
control channel
Gateway
Internet
GW
Advanced Manufacturing Deployment
(e.g., Location A, Product Line X)

Figure 2.2: Software-defined IoT security gateway architecture, the controller's security policy directs provisioning device-specific middleboxes to process each device's network traffic on a local security gateway.

At a high level, the gateway intercepts all network traffic to and from a network device (e.g., 3D printer) and runs virtualized middleboxes (e.g., firewall) to impose a security policy (e.g., block 3D printers from starting secure shell connections). Compared to traditional static network defenses with baked-in policies, a software-defined architecture uses a *centralized controller* to flexibly define and configure customized policies.

Figure 2.2 shows an example deployment protected by a software-defined security gateway. The controller configures the gateway via the control channel. For example, if a networked 3D printer is found to have an unpatched backdoor [43, 223], the controller could initialize a firewall on the gateway to block access. The number and type of middleboxes depends on the defense strategy. A coarse defense strategy could run a few shared middleboxes for all networked devices [17, 176], while a fine-grained defense strategy may deploy one middlebox *per protected device* [95, 227, 229]. A virtual switch (vSwitch [145]) routes packets to the appropriate middlebox, with the controller dynamically configuring the vSwitch's routing rules.

These bolt-on gateways are promising for securing manufacturing deployments; however, they are currently untrusted. Under attack, these security gateways could become ineffective, or even worse, become a launchpad for new attacks.

2.2.3 Adding Trust to Software-Defined Architectures

Current software-defined gateways [17, 95, 176, 227, 229] lack a root-of-trust, allowing an attacker who has compromised the gateway to reconfigure the gateway such that packets are not processed by the correct middlebox. While there is some prior work on securing individual pieces of the architecture, they lack end-to-end trust. First, recent work on trustworthy middleboxes uses trusted enclaves to run middleboxes inside untrusted cloud environments (e.g., [146, 193]), but this solution requires specific hardware (e.g., SGX [84], TrustZone [11], TPM [12]) which is not widely available. Second, research on securing the controller (e.g., [149, 172]) has been focusing on using permissions to limit the access of multiple applications, but cannot provide runtime protections against an attacker capable of compromising the operating system (OS). Finally, existing secure tunnels (e.g., IPSec, TLS) and work on customized verification protocols (e.g., [94, 107, 108, 132]) can be used to achieve traffic integrity, but they alone are not enough to defend against all attacks. We discuss each below.

Secure Enclaves: Trusted hardware (e.g., SGX [146, 193], MPX [240]) has been investigated for providing increased security guarantees about middleboxes running in untrusted cloud environments. These approaches provide code attestation, confidentiality, and mediation[3] [146], and memory access boundaries checking [240]. Unfortunately, the use of a secure enclave comes with a high performance overhead and lacks generality (limited to a specific CPU and only supporting user space applications with constrained memory allocations). For example, Schwarz and Rossow proposed a secure gateway architecture [167] leveraging trusted enclaves on both the gateway and devices to provide trust in the gateway's operations and messages sent to and from the gateway. However, this architecture cannot provide these trust guarantees to devices without a secure enclave, which many networked 3D printers do not currently have.

Secure SDN Controller Software: Researchers have focused on mediating multiple applications on the controller by adding permissions [87, 168, 172, 212]. Others have looked to ensure

[3]Mediation requires re-implementing the middlebox software.

consistency of routing rules generated by separate applications [86, 149]. Our work looks to support these controllers and provide the ability to provide a foundation for guaranteeing trust in their operations. Others have developed tools to ensure consistency between the control and data planes with respect to packet routing, creating tools for identifying forwarding anomalies [5, 41, 58, 78, 89, 91, 93, 151, 180, 189, 238, 239] and SDN-specific attacks [54, 106, 161, 216]. Unfortunately, none of these provide runtime protections against our threat model.

Trusted Computing: Hypervisors have been used to provide security primitives such as isolation, mediation, and attestation [112, 122, 155, 190, 198, 202]. Micro-hypervisors have been shown to support adding trust to a variety of hardware platforms (x86 [174, 200], ARM [199], microcontroller [9]) while running unmodified software (e.g., Linux) [9, 192, 198]. Additionally, micro-hypervisors have a small Trusted Computing Base (TCB) that is often amenable to formal verification [201]. To provide specific trust guarantees, remote attestation is often realized using a Trusted Platform Module (TPM) [12], leading to multiple software implementations [157, 198]. Similarly, secure routing proposals have used digital signatures to verify packet paths [108, 132].

2.3 Path Forward

We leverage this background to bolt-on network security to deployed 3D printers. Chapter 3 provides a measurement of network vulnerabilities on deployed networked 3D printers and compares them to other networked devices. Additionally, it analyzes how a printer's network deployment can allow an attacker to achieve the attacks in Section 2.1.2.1. These identified vulnerabilities motivate the need to defend these deployments from a network attacker. Chapter 4 discusses the design and implementation of low-cost, trusted security gateway. We add trust to existing network security gateway architectures (those discussed in Section 2.2.2) by adding a low-cost root of trust and applying fine-grained protections to its packet processing operations. Chapter 5 demonstrates the ability to integrate fine-grained, device-specific defenses, such as the classic network security solutions on top of a trusted gateway to mitigate identified vulnerabilities.

Chapter 3

C3PO: A Security Evaluation of Networked 3D Printers and Deployments

In this chapter, we present C3PO, an open-source network security analysis tool for systematically identifying network security threats to networked 3D printers. We use C3PO to gain an understanding of the current state of network security in deployed 3D printers and inform the design of potential network defenses. C3PO's design is guided by industry standards and best practices. It identifies potential vulnerabilities in data transfer, the printing application, availability, and exposed network services. Furthermore, C3PO analyzes the security implications of a 3D printer's network deployment, such as an attacker compromising a camera to modify printing instructions "on-the-wire." We use C3PO to analyze 13 networked 3D printers and 5 real-world manufacturing network deployments. We identified network security trends in networked 3D printers such as a susceptibility to low-rate denial of service attacks (all 13), transmitting unencrypted data (12 out of 13), and being deployed on publicly accessible networks (2 out of 5). We leverage these findings to provide recommendations on securing networked 3D printers and their network deployments.

3.1 Motivation

Attacker Goals: Based on prior work (e.g., [142, 215, 221, 223]), we envision an attacker with one of the following goals:

- *Causing physical hazards* [103]. All networked 3D printers have components that can pose a safety risk (e.g., high-power lasers, high-temperatures heaters, etc.). An attacker could manipulate these to cause a physical hazard, such as starting a fire by commanding the heater to its maximum value while turning off safety features (e.g., the cooling fan) and driving the hot printer nozzle into the part (Figure 3.2).
- *Creating defective parts* [21]. A network attacker could send malicious commands to a 3D printer causing its software to crash midway through printing (Figure 3.3), forcing a multi-hour printing operation to be repeated.
- *Stealing proprietary data* [243]. Large manufacturing centers are composed of multiple networked 3D printers. Often new printing tasks are sent to the first available networked 3D printer. An attacker could have a compromised machine advertise itself as a fake 3D printer to steal designs and create forgeries.
- *Halting printing operations* [100]. An attacker can send thousands of status requests to a networked 3D printer to overwhelm its ability to respond to legitimate requests (Figure 3.4), thereby prohibiting legitimate users from sending new files, resulting in a loss of productivity and potentially costing thousands of dollars [196].

Most demonstrated attacks have ignored the network as an attack vector. Some modified STL files at the control PC before they were sent over the network (e.g., [21, 186]). Others assumed physical access to allow modifying the printer's firmware (e.g., [71, 179]). Network security analysis of 3D printers has been limited to a single vendor and only identified data transfer vulnerabilities–missing availability vulnerabilities [57]. Furthermore, most of the prior work does not identify multiple types of vulnerabilities and does not scale to multiple vendors/protocols. Moreover, the 3D printer's network deployments have been ignored, missing potential multistage attacks (e.g., those leveraging other devices on the network).

Threat model and scope: We limit our attacker to only accessing the 3D printer over the network (i.e., no physical access). As a starting point, we do not consider attackers who are seeking to be stealthy or evade countermeasures. An attacker can start with network access (e.g., insider threat) or gain it by compromising a device on the network. For example, an attacker could gain access to a control PC using an e-mail with a malicious link [21], an IIoT device using default credentials [10], or a networking switch using unpatched vulnerabilities [194]. Thus, the security of a networked 3D printer is impacted by its network deployment.

The combination of a 3D printer's individual vulnerabilities and its network deployment creates an array of possible attack paths for causing a physical hazard, creating defective parts, stealing data, or halting operations. Our C3PO security analysis tool aims to be a generic tool for identifying susceptibility to these types of security risks from a connected device's network API. Additionally, it informs the design of "bolt-on" network defenses.

3.2 Tool Overview

We identify three requirements of C3PO for identifying network vulnerabilities in 3D printers. Unfortunately, existing tools do not meet these requirements. Thus, we design C3PO [28], an open-source security analysis tool for networked 3D printer and their deployments.

3.2.1 Tool Requirements

We identified three requirements for our security analysis tool:

- **R1: Increased coverage of vulnerabilities.** The tool should cover multiple vulnerabilities as often combinations of vulnerabilities are required for an attack to succeed (e.g., a broadcast query and a lack of encryption could be combined to spoof a printer and steal data).

- **R2: Protocol-agnostic.** The tool should not be designed for a specific 3D printer (or protocol, i.e., G-code), but support multiple vendors, including those using a closed-source, proprietary protocol.[1]
- **R3: Addressing complex deployment models.** The tool should be able to analyze complex deployment models and consider the security impacts from other devices which could be leveraged by attackers to achieve their goals.

Existing Tools: We are not aware of 3D printer specific network analysis tools. While many generic network security tools exist, they do not meet all of the above requirements. Existing IoT tools can only detect a small set of vulnerabilities. Additionally, some are device-specific (e.g., IoT Security Checker [49] and IoT Vulnerability Scanner [152] focus on login credentials, PENTOS [206] focuses on wireless security attributes). While others are protocol specific (e.g., PRET [131], OWASP Nettacker [232]) and not compatible with the protocols used by deployed networked 3D printers.

To achieve high coverage, be protocol-agnostic, and address complex deployment models for networked 3D printers, we develop C3PO. It leverages existing tools (i.e., Nessus [191], Mutiny [183], and hping [165]) and adds modules specific to networked 3D printers.

At a high level, C3PO consists of two stages. First, an individual 3D printer analysis for identifying machine-specific vulnerabilities in a standalone 3D printer. Second, a network deployment analysis for identifying potential multistage attack paths through a 3D printer's network deployment using attack graphing. We discuss the first stage and then show how its results are fed into the second stage to aid analyzing the network deployment.

3.2.2 Individual 3D Printer Analysis Stage

To provide coverage of vulnerabilities (R1), we ensure our tool identified network security attributes described in security standards [81] and best practices [52, 141] for networked devices

[1]In our survey, 5 of 9 vendors used distinct proprietary protocols.

Category	Attributes	Reference	In C3PO
1: Data Transfer	Lack Encryption	IEC FR 3 & 4, OWASP #7 IoTSF - G	✓
	Broadcast Advertisement	IEC FR 5	✓
	Command Actuators	IEC FR 2	X
	Lack Authentication	IEC FR 1 & 3, OWASP #1	X
2: Availability	Robust to DoS	IEC FR 3 & 7	✓
3: Printing Application	Insecure applications	IEC FR 2 & 3, OWASP #3 IoTSF - E	✓
	Insecure updates	IEC FR 3, OWASP #4 IoTSF - J	✓
	Management Commands	IEC FR 2	✓
	Insecure default settings	IEC FR 7, OWASP #9	✓
	Lack of device management	IEC FR 1-3 & 6, OWASP #8 IoTSF - F, K & L	X
4: Network Services	Network access	IEC FR 1 & 5, OWASP #2 IoTSF - H	✓
	Outdated libraries	OWASP #5, IoTSF - D	✓
5: Not Applicable	Insufficient privacy	OWASP #6, IoTSF - A	X
	Lack of physical hardening	IEC FR 3 & 7, OWASP #10 IoTSF-B	X

Table 3.1: Security assessment categories for networked devices from industry standards: IEC 62443-4-2 Foundational Requirements (IEC FR) [81], and best practices: 2018 OWASP IoT Top 10 (OWASP) [141] and IoT Security Foundation (IoTSF) [52].

(shown in Table 3.1). After pruning categories that were not applicable to the manufacturing domain (e.g., privacy) or could not be evaluated with only network access (e.g., physical hardening) we grouped the resulting attributes into four categories:

- *Data transfer:* Determining if the data is confidential (encrypted), how devices are found on the network, and if devices limit which network commands are executed (require authentication).
- *Availability:* Checking if the device is robust to traditional denial of service (DoS) attacks.
- *Printing application:* Detecting application vulnerabilities such as crashing inputs and unsigned software updates.
- *Network services:* Identifying exposed network services and their associated software versions.

Some attributes were unable to be analyzed in an automated manner due to an inability to identify a common attribute to test across the various vendor protocols (e.g., testing if a network host is allowed to command the 3D printer's actuators). We manually analyze these and include our findings in Section 3.3.2. These four categories are mapped to four corresponding modules in C3PO's first stage as shown in Figure 3.1.

3.2.3 Network Analysis Stage

Identifying vulnerabilities for a standalone networked 3D printer is the first step, but does not convey the complete security picture because there may be other vulnerable devices (e.g., IIoT cameras, sensors, etc.) in a manufacturing network. For example, a manufacturing deployment might add an internet connected camera on the 3D printer's subnet in order to remotely monitor the printing process. Thus, it is important to identify how these devices could be used by an attacker against networked 3D printers (R3).

The network deployment analysis component of C3PO addresses this problem. Its goal is to create an attack graph which identifies all possible attack paths to the networked 3D printer. Two inputs are required to achieve this goal. First, C3PO needs to automatically identify all of the other devices and their network connections to the networked 3D printer. Second, C3PO needs to identify each device's vulnerabilities in order to find all possible attack paths. However, this is challenging due to the wide array of devices, the complex network deployments, and the lack of models for attacks in the manufacturing domain.

3.2.4 Tool Implementation

We implement C3PO as a Python script [28], that allows for calling specific stages (e.g., network analysis stage) or individual modules within a stage (e.g., only analyzing data transfer for the presence of encrypted traffic). C3PO parses the input network capture using scapy [22] to analyze its contents for the data analysis module and create inputs for the availability module. Other modules are realized by invoking existing network security evaluation tools, such as Cisco's

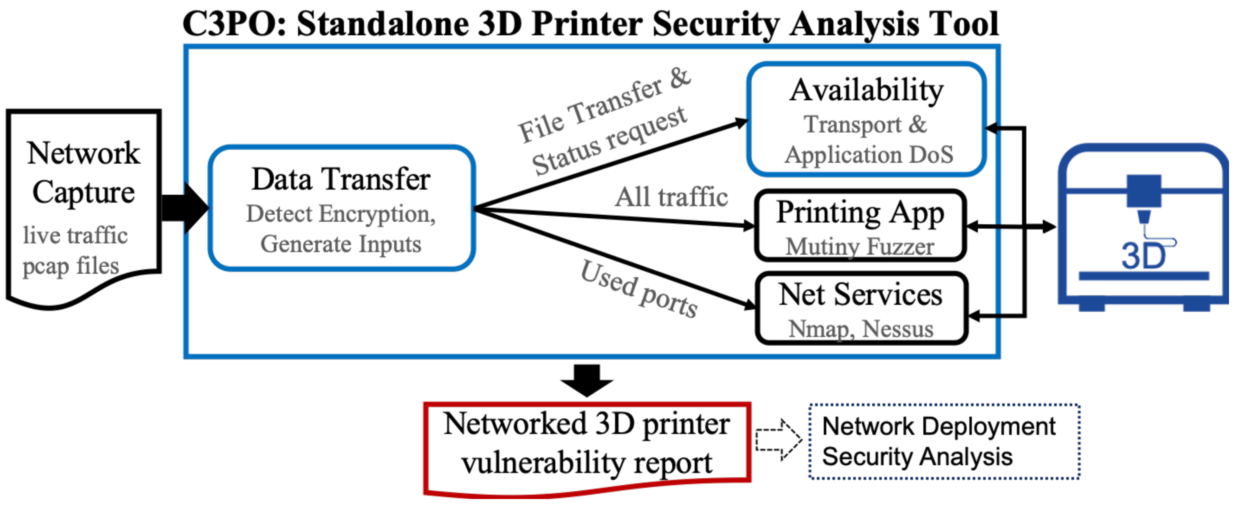

C3PO: Standalone 3D Printer Security Analysis Tool
Network Capture
live traffic
pcap files
Data Transfer
Detect Encryption, Generate Inputs
File Transfer & Status request
All traffic
Used ports
Availability
Transport & Application DoS
Printing App
Mutiny Fuzzer
Net Services
Nmap, Nessus
3D
Networked 3D printer vulnerability report
Network Deployment Security Analysis

Figure 3.1: Overview of C3PO's networked 3D printer vulnerability analysis tool. Blue (Shaded) boxes represent our additions, and black ones are existing tools.

Mutiny Fuzzer [183] for the printing application module, Nmap for the network service module, and MulVAL [140] for the network analysis stage. This code has been publicly released [28], to allow 3D printer vendors and manufacturing center operators analyze their machines. We next discuss the design and implementation of each of these stages and our findings on deployed commercial networked 3D printers.

3.3 Individual Networked 3D Printer Analysis

We begin with our individual 3D printer analysis stage. First we discuss the design and implementation of each module, followed by our findings from analyzing 13 networked 3D printers and 10 home IoT devices.

3.3.1 Individual Stage Tool Design and Implementation

Recall that our individual stage is composed of four modules: data transfer, availability, printing application, and network services (shown in Figure 3.1). We will discuss each of these in turn.

3.3.1.1 Data Transfer Module

As many networked 3D printers use a closed-source, proprietary format to encode their printing commands, it is challenging to differentiate encryption from packed binary data. To overcome this challenge, we leverage prior work (e.g., [207, 210]) to determine if the data is encrypted by using three tests on the data: (1) calculate the entropy per byte, (2) perform a chi-squared test for a uniform distribution, and (3) calculate the serial correlation coefficient. We performed all three test on a per-packet level for the complete network capture. We separated out data with identifiable file headers (e.g., Gzip, JPEG, etc.). We infer that encryption is used if there is high entropy (>6.75 bits),[2] the chi-squared test results in a probability of a uniform distribution (p-value >0.01), and low serial correlation (<0.3).[3] If the data exchanged in both directions passes these tests we consider encryption to be used. If it only passes in a single direction, we consider the commands to possibly be encrypted but not sent over an encrypted channel.

Implementation: We use scapy to parse the TCP payload data from the input network capture. First we divide the packets based upon their direction (i.e., those going to the printer from those sent by the printer). Next, we check these payloads for known identifiers (e.g., a gzip file beginning with 0x1F8B or a JPEG image beginning with 0xFFD8FF). Packets corresponding to such a known file format are ignored from further analysis for use of encryption. The remaining packets are analyzed both individually and as a single group containing the concatenation of all the packet's payloads with the following tests. Entropy per byte is measured by measuring the distribution of each byte and summing the log base two of each. We leverage scipy's chi-sqaured test (from its stats package), and compare the distribution of bytes with a normal distribution. Finally, we use the statsmodels.tsa.stattools module to perform an autocorellation test. The results of each of these tests are reported for the input network capture.

[2]Test files of random string values had a maximum entropy of 6.65 bits.

[3]Files with >128 random bytes had a maximum serial correlation of 0.29.

3.3.1.2 Availability Module

Our availability module identifies DoS at two network layers.

- Transport layer. Analyzes the underlying network layer capabilities of the 3D printer, not sending any data to the printing application. Specifically, we test with a SYN flood (using hping [165]) and TCP connection exhaustion (e.g., multiple TCP sessions).
- Application layer. In order to remain protocol-agnostic, we use the input network capture as the input for generating all test cases. Specifically, we perform a stress test (e.g., sending multiple, concurrent status requests) and partial data exchange (e.g., only send the first 100 Bytes of a printing file, then keep the connection open indefinitely).

We used repeated messages (assumed to be status requests) for the stress test and the stream where the largest amount of data is sent from the control PC to 3D printer for the partial data exchange (assuming this to be the printing command file). Additionally, we run a slowloris [46] variant of the stress test to identify low-rate DoS vulnerabilities.

Implementation: This module takes an input address (port and IP address) and a few of the initial packets to generate inputs to test the networked 3D printer. It provides a series of test to evaluate different potential DoS conditions. First, the industry tool hping [165] is used to launch a SYN flood, and check if the printer remains available. Second, a threaded python program attempts to generate 2,000 simultaneous TCP connections to the networked 3D printer. We re-run this program five times to evaluate different potential DoS situations. Initially, we simply establish TCP connections to determine if there is a per-host limit on open TCP connections and if the printer times out and resets inactive TCP connections. Next, we send junk data (e.g., a single character) to determine if this changes the maximum number of connections or when a timeout is triggered. Subsequently, we replay data from packets in the network capture. Specifically, we replay the first packet on each connection and the first large packet (we assume this is the packet where data is being transferred to the printer). Finally, we replay the initial packet's data one byte at a time to determine if the printer is susceptible to a slowloris-type attack.

3.3.1.3 Printing Application Module

To identify potential vulnerabilities within the networked 3D printer's application software, we used an existing mutational fuzzer (Radamsa [74]) that generates inputs that match the protocol format found in the network capture [183]. This allows C3PO to leverage network fuzzing without having to know the protocol format (e.g., required for [144]) or requiring access to the control PC application (e.g., IoTFuzzer [37]). Fuzzed inputs were transmitted for 30 minutes while a benign status request message was used to identify an input that caused the application to crash. We implement this functionality by integrating a call to Cisco's Mutiny Fuzzer [183].

3.3.1.4 Network Services Module

To identify the network services a 3D printer is exposing, we use an existing network mapping tool, Nmap [115]. Next, we scan the device for susceptibility to known vulnerabilities using the known network security evaluation tool Nessus [191]. C3PO calls functions within these respective programs to collect its evaluation data.

3.3.2 Findings

In this section, we present our findings from running C3PO on 13 networked 3D printers. In total, we identified vulnerabilities within each of the categories identified by industry standards and best practices (Table 3.1): insecure data transfer, lack of robust availability, insecure printing application, and insecure network services.

3.3.2.1 3D Printers Evaluated

The 13 networked 3D printers evaluated ranged from low-cost, desktop polymer machines to \$1M+, industrial metal 3D printers as shown in Table 3.2. The networked 3D printers represent two classes: desktop and industrial. The desktop machines generally print a polymer material and have a lower cost (<\$5,000). While the industrial machines print either polymers or metals,

	3D Printer	Cost (US$)	Release Year	Material	Printing Protocol
Desktop	OctoPi$^{\lhd}$	0	2011	Polymer	G-code
	Machine A	300	2015	Polymer	G-code
	Machine B$^{\oplus}$	1,400	2019	Polymer	G-code
	Machine C	1,500	2014	Polymer	proprietary
	Machine D	2,850	2015	Polymer	proprietary
	Machine E	4,200	2016	Polymer	compressed G-code
Industrial	Machine F*	17,000	2017	Polymer	compressed proprietary
	Machine G*$^{\diamond}$	18,900	2008	Polymer	compressed proprietary
	Machine H*$^{\diamond}$	31,900	2007	Polymer	compressed proprietary
	Machine I$^{\diamond}$	50,000	2007	Polymer	STL
	Machine J$^{\oplus}$	150,000	2016	Metal	proprietary
	Machine K$^{\dagger\diamond}$	600,000	2010	Metal	proprietary
	Machine L*	750,000	2011	Polymer	compressed proprietary
	Machine M†	~1,000,000	2014	Metal	proprietary

$^{\lhd}$: Open-source network front-end that supports 100+ USB controlled machines
$^{\oplus}$: Machines are the first model released by a new vendor
$^{\diamond}$: Machines in operational use but no longer supported by the vendor
*: Machines F, G, H & L are produced by the same vendor
†: Machines K & M are produced by the same vendor

Table 3.2: Individual networked 3D printers evaluated using C3PO, spanning a range of costs and printing processes.

require significant space, and have higher costs (>$15,000). The desktop machines selected were among the top 10 sold on Amazon, and the industrial models were from the top industrial vendors. During our evaluation we transferred printing instructions for the same design file[4] to each networked 3D printer and observed six different protocols (five of which were proprietary). We group our individual networked 3D printer findings based upon the logical network layer where the vulnerability manifests, specifically looking at the transport and application layers.

3.3.2.2 Transport Layer

Our findings in the transport layer revealed an implicit assumption by 3D printers that the network is non-adversarial, placing high trust in the network and those with access to it.

[4]A CAD file for a small boat [50]

Insight 1 (I1): ***12 of 13 networked 3D printers did not use encryption when transferring data over the network (i.e., a local attacker could steal proprietary data).***

Encryption: None of the networked 3D printers encrypted data in both directions (i.e., to and from the 3D printer, reference Table 3.3). Most data transfers exhibit an entropy of <5.48 bits and a serial correlation of >0.38 in one direction.[5] Additionally, only two had a majority of their packets pass a chi-squared test for their data being a uniform distribution, which encrypted data should pass. These two may be encrypting the printing commands file prior to sending it over an unencrypted channel; however, all exposed file meta-data (e.g., filenames, length, etc.). As most did not utilize encryption, known file headers could be identified (e.g., Gzip, JPEG, etc.).

To put this in context of other IoT markets, we also ran C3PO on 11 commodity IoT devices (e.g., Amazon Alexa, D-Link camera, etc.). Among these IoT devices, 6 out of the 11 utilized encryption when transferring data. We manually confirmed that 5 of these IoT devices utilized TLS for data exchange. This suggests that networked 3D printers are behind the state of the art with respect to encrypting data being sent over the network. This is particularly surprising for the industrial networked 3D printers, as it creates a risk that proprietary data could be stolen.

Insight 2 (I2): ***12 of 13 networked 3D printers were vulnerable to simple transport layer denial of service attacks (e.g., SYN flood crashes the 3D printer, requiring a power-cycle).***

We analyzed two transport layer denial of service issues: SYN flood and TCP connection exhaustion (with results shown in Table 3.4). These classic network security vulnerabilities allow many of the DoS attacks we demonstrated.

SYN Flood: During a SYN flood, 9 of the 13 networked 3D printers analyzed were unavailable on the network. Additionally, two (one desktop and one industrial) were still unavailable after the attack and required a power-cycle to regain network connectivity.

[5]Encrypted data has an entropy of >6.75 bits and <0.3 serial correlation.

Device		Encrypted	Entropy/byte		Serial Correlation		Percent Packets
			Sent	Recv	Sent	Recv	$\chi^2 > 0.01$
OctoPi	http	No	4.15	5.53	0.40	0.51	0.20%
	https†	Yes	7.99	7.94	0.0006	0.031	98.8%
Machine A		No	4.26	5.16	0.65	0.47	0.5%
Machine B		No	4.16	5.01	0.39	0.58	0%
Machine C		No	6.64	4.70	0.49	0.48	0%
Machine D		Possible	7.88	5.29	0.10	0.50	0.2%
Machine E		No	6.87	5.14	0.29	0.44	0.3%
Machine F		No	5.54	5.48	0.58	0.62	0%
Machine G		No	3.26	5.36	0.54	0.59	0%
Machine H		No	6.37	5.32	0.47	0.59	0%
Machine I		No	6.75	6.50	0.49	0.48	63.7%
Machine J		No	4.13	2.87	0.35	0.46	41.1%
Machine K		Not analyzed, machine configured as a NAS server*					
Machine L		No	4.61	5.34	0.57	0.42	0%
Machine M		Possible	7.99	5.39	0.02	0.38	93.1%
Paper Printer		No	5.56	N/A	0.22	N/A	0%
IoT A		No	5.92	5.93	0.09	0.16	35.7%
IoT B		No	5.94	6.43	0.08	0.26	0.8%
IoT C		No	5.60	7.31	0.51	0.32	18.5%
IoT D		Yes	7.78	7.78	0.17	0.17	12.6%
IoT E		No	5.04	4.86	0.37	0.43	0%
IoT F		Yes	7.93	7.96	0.08	0.06	97.7%
IoT G†		Yes	7.93	7.99	0.12	0.001	97.7%
IoT H†		Yes	7.99	7.99	0.006	0.02	91.5%
IoT I†		Yes	7.52	7.74	0.33	0.24	74.8%
IoT J†		Yes	7.92	7.76	0.08	0.17	77.3%

†: Manually verified to be using TLS *: Not specific to networked 3D printers

Table 3.3: Entropy, serial correlation and χ^2 measurements use to identify encrypted data transfer from individual devices' network traffic.

TCP connection exhaustion: Most networked 3D printers were designed assuming a small number of simultaneous clients (often less than 20). However, an attacker could create a temporary DoS condition with less than 4,000 connections on 10 of the 13 networked 3D printers. In general, the industrial printers allowed a smaller number of connections (6-65) and were thus easier to DoS; while the desktop printers allowed significantly more connections (960-4,000).

Of the networked 3D printers vulnerable to TCP connection exhaustion, four did not implement a timeout for inactive TCP connections (one desktop and three industrial). Thus an attacker could slowly create a large number of connections and render the 3D printer unavailable without sending any data (e.g., only need to send SYN and ACK packets for each connection).

Device	SYN Flood	Maximum TCP Connections	TCP Timeout
OctoPi	○	2123	900 seconds
Machine A	●*	1	6 seconds
Machine B	●*	1,194	340 seconds
Machine C	○	295	47 seconds
Machine D	○	998	None
Machine E	○	4,000	30 seconds
Machine F	●	33	150 seconds
Machine G	●	10	60 seconds
Machine H	●	33	150 seconds
Machine I	●	1	None
Machine J	●*	6	None
Machine K	●	4,095	60 seconds
Machine L	○	65	None
Machine M	○	10	10 seconds
Paper Printer	●	7	300 seconds

○: No impact ●: DoS * 3D printer required power cycling after DoS attack

Table 3.4: Characteristics of networked 3D printer's listening TCP socket.

The more robust machines generally allowed each host a limited number of connections (10-135), less than its maximum capacity. An attacker could delay network operations (e.g., increase the time required to send printing commands over the network) but could not render the networked 3D printer unavailable.

3.3.2.3 Application Layer

Moving up to the application layer, we noted an assumed trust between the control PC and the 3D printer. This is evidenced by the lack of authentication between the control PC and the 3D printer. When coupled with other aspects of the 3D printer's network operations (as discussed below), this creates significant vulnerabilities.

Insight 3 (I3): ***12 of 13 networked 3D printers did not authenticate the control PC (e.g., any host on the network could send commands that the 3D printer would execute).***

Device	Receive Files	Management Commands	Actuator Commands
OctoPi	✓†	✓†	✓†
Machine A	✓	✓	✓
Machine B	✓	✓	
Machine C	✓*	✓†	
Machine D	✓*		
Machine E	✓	✓	✓†
Machine F	✓*		
Machine G	✓*	✓	
Machine H	✓		
Machine I	✓*		
Machine J	✓	✓	✓
Machine K	✓*		
Machine L	✓*		
Machine M	✓*		
Paper Printer	✓	✓†	

*: Required UI action †: Required authentication

Table 3.5: Network application programming interfaces (APIs) available on networked 3D printers and their operating requirements.

Authentication: Given proprietary protocols, C3PO can only guess if the connection is authenticated by analyzing the initial data packets sent after a TCP handshake. If similar packets are repeated, it assumes this is a data exchange (e.g., status request) and no authentication occurred. We manually validated each 3D printer's network traffic to determine if authentication was utilized. Only one desktop 3D printer appeared to be using authentication and one additional desktop printer supported authenticating a subset of commands.

Insight 4 (**I4**): ***3 of 13 networked 3D printers execute unauthenticated commands received over the network (e.g., attacker can drive the print head into a part, Figure 3.2).***

Network APIs: Each networked 3D printer exposes a set of network APIs to support printing operations that fall into three categories: (1) receive files, (2) basic management (e.g., pause, abort, etc.), and (3) direct actuator commands (shown in Table 3.5).

All networked 3D printers were able to receive files (e.g., printing commands) over the network, though each vendor utilized a different protocol (e.g., compressed G-code over HTTP to proprietary command format over a Windows communication foundation protocol). Additionally, some networked 3D printers provide an API for issuing management commands (e.g., pause the current printing job). An attacker could maliciously use these exposed management commands (e.g., adding delays by pausing or restarting the current printing task). Finally, a small number of networked 3D printers provided direct access to 3D printer actuators (e.g., moving the print head), which could be used to cause more serious problems. We show one such example.

Execute Actuator Commands: One networked 3D printer blindly executed actuator commands sent over the network, directly executing G-code commands as they arrive. This creates vulnerabilities because of two factors. First, the networked 3D printer does not authenticate the control PC sending the commands; it accepts commands from any network host. Second, the machine does not limit commands that will be executed while it is printing (e.g., moving the print head).

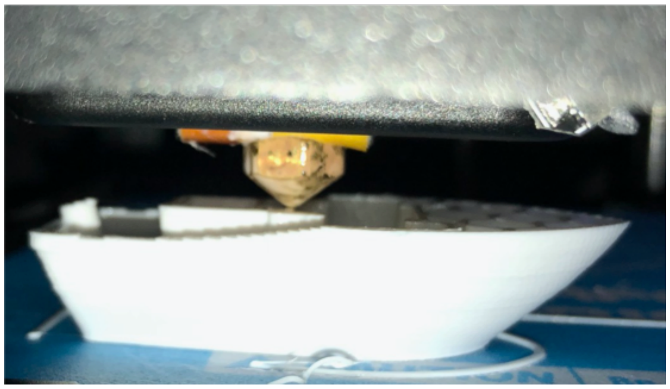
Thus, in the middle of printing a part, an attacker can send malicious actuator commands (e.g., increase heater temperature, drive print nozzle into part, etc.). The printer will perform the malicious command at its current location in the print file. This creates a safety risk that allows an attacker to cause the networked 3D printer to create an object different from what was specified (e.g., a defective part due to damage from the printer nozzle, shown in Figure 3.2).

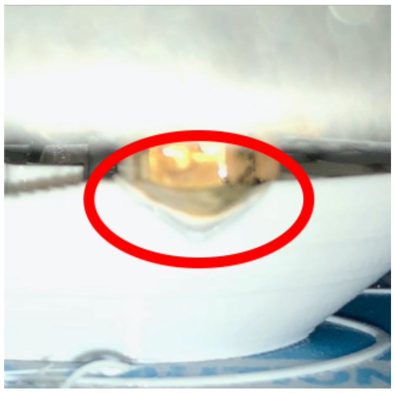
To demonstrate this vulnerability, we emulated a network attacker with a goal of creating a defective part. We connected to the networked 3D printer by simply opening a TCP connection. Next, we sent a single command[6] to the 3D printer, driving the 220°C print head into the part (Figure 3.2b) and creating a defect by melting the plastic (Figure 3.2c).

Insight 5 (I5): ***4 of 13 networked 3D printers had printing applications that lacked input filtering (e.g., a malformed input could crash the networked 3D printer's firmware).***

---

[6]A G-code command to move the print nozzle down (e.g., `G1 Z-10.00`).


(a) Normal printing operation. Note separation between nozzle and boat.


(b) Attack: print nozzle impacts boat.

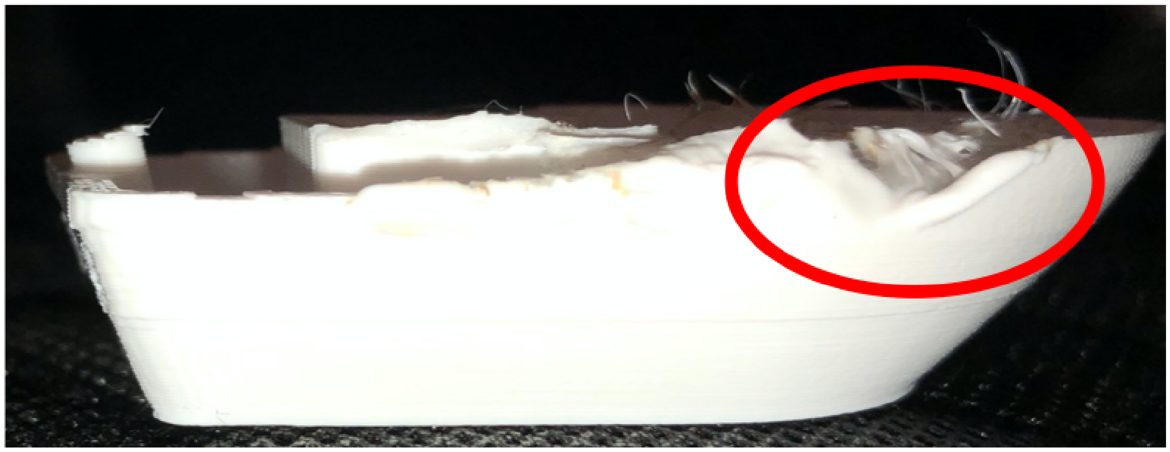
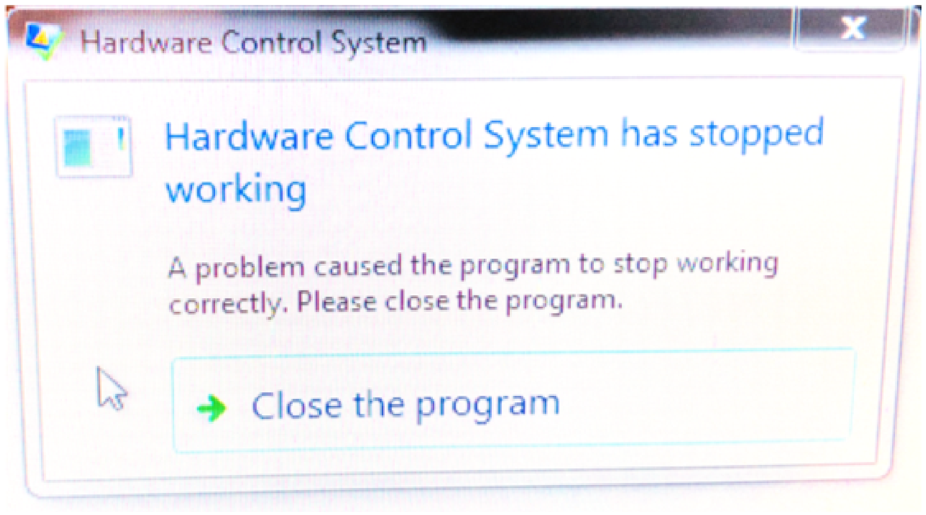

(c) Permanent damage to part after attack.

Figure 3.2: A network API that executes actuator commands while printing allows an attacker to create defects.

Lack of Input Filtering: While most networked 3D printers generate unique filenames at the 3D printer, one of the industrial machines had the client (control PC application) generate unique file names. The networked 3D printer would blindly save received files with their provided filename. If two files with the same filename but differing data were received, the printer firmware would crash (error message shown in Figure 3.3). The crash would persist across reboots, and could only be cleared by starting the machine in a "safe-mode" (where the printing application is not started) and deleting the file.

Malicious Inputs: While some 3D printer applications did not crash when given a fuzzed input, three desktop machines crashed from malformed inputs. For example, one expected an


Hardware Control System

Hardware Control System has stopped working

A problem caused the program to stop working correctly. Please close the program.

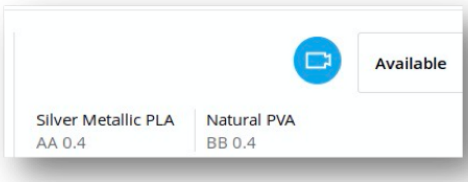
Figure 3.3: Error message after sending two different files with the same filename.

HTTP PUT request of ‘GETPRINTERINFO’, while slightly modifying this request by adding garbage characters to the beginning (e.g., ‘GRINTERINFINFOGETPRINTERINFO’) caused the machine’s firmware to crash. This is similar to well-known injection attacks against web-servers [148]. The crash causes the current printing task to stop, and upon power-cycling, the printing task must be restarted at the beginning. We additionally tested for susceptibility to compression bombs [117], but did not identify any susceptible networked 3D printers.

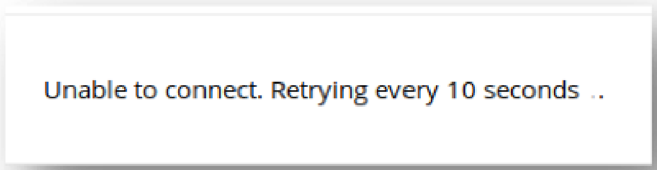
Insight 6 (**I6**): ***11 of 13 networked 3D printers were vulnerable to application layer DoS attacks (e.g., an attacker transmitting 1,000+ status requests simultaneously, renders the 3D printer unable to receive new print files).***

We identified three types of application layer DoS attacks: (1) stress test where a high volume of requests are sent to the networked 3D printer, (2) low-rate DoS attacks (e.g., `Slowloris`), and (3) stopping a data transfer before it completes without closing the underlying connection.

Stress Tests: While the transport layer usually limits the maximum number of connections, three networked 3D printers (two desktop, one industrial) supported fewer simultaneous status requests (e.g., 4,000 TCP connections to 576 status requests). Further, one of these waited for



(a) Available prior to DoS


Unable to connect. Retrying every 10 seconds ..

(b) Unable to connect during attack

Figure 3.4: All 3D printers analyzed exhibited a DoS vulnerability (many at $<$10 kbps).

the client (i.e., attacker) to terminate the connections even when data was no longer being sent, due to the printer disabling TCP timeouts.

Slowloris: Three industrial networked 3D printers exhibited vulnerability to a `Slowloris`-type attack [46]. These machines accept data transferred one byte per packet (allowing up to five seconds between packets), and would not process the data until all the bytes of a protocol message were received. This means a standard status request message can be used to DoS the printer for up to 45 minutes by sending only $\sim$290bps per connection.

Partial Data Transfer: Three networked 3D printers (two desktop, one industrial) had unique vulnerabilities when only part of a file is transferred. These machines disabled TCP timeouts when receiving a file, allowing an attacker to start but not complete multiple file transfers (some vulnerable to as few as 10). This rendered the 3D printer unavailable, which would persist as long as the attacker's TCP connections remained established without sending any data. Furthermore, one of the desktop machines required a power-cycle to recover from this attack, as the DoS continued even after the attacker closed all open TCP connections.

Insight 7 (**I7**): ***10 of 13 networked 3D printers respond to a control PC's broadcast query for 3D printers (e.g., attacker can spoof a networked 3D printer to create a MitM situation where printing files could be modified in transit).***

Device	Minimum Bandwidth for DoS	Susceptible to `Slowloris`	Susceptible to Partial Transfer
OctoPi	108 kbps		✓
Machine A	0.96 kbps		
Machine B	834.7 kbps		
Machine C	No DoS		
Machine D	679.6 kbps		✓
Machine E	368.1 kbps		
Machine F	2.2 kbps	✓	
Machine G	0.67 kbps	✓	
Machine H	2.2 kbps	✓	
Machine I	6.4 kbps		✓
Machine J	8.2 kbps		✓
Machine K	Not analyzed*		
Machine L	4.4 kbps	✓	✓
Machine M	No DoS		✓
Paper Printer	8.7 kbps		

* 3D printer configured as a shared network directory

Table 3.6: Application-layer DoS vulnerabilities characteristics for networked 3D printers.

3D Printer Discovery: In order to send printing instructions to a networked 3D printer, a control PC must first find the 3D printer on the network. Most networked 3D printers utilize an existing UDP-based, broadcast protocol (e.g., mDNS, LLMNR, SSDP, etc.) to enable zero-configuration networking. These protocols begin with the control PC sending out a broadcast query. At a minimum, these protocols provide the control PC with the hostname and IP address for each networked 3D printer. Some also include additional details in their reply (e.g., firmware version). In the event of multiple replies for the same networked 3D printer, the control PC only utilizes the first reply and drops subsequent ones.

3D Printer Spoofing: This becomes a security vulnerability as the control PC does not authenticate the 3D printer identified by its broadcast query before sending printing commands. An attacker attempting to impersonate a networked 3D printer only needs to reply to the control PC's broadcast query before the networked 3D printer. Subsequently, the attacker must imitate the 3D printer's network API, which can be as simple as a listening TCP socket. As this point,

Device	**Network Services**		**Hosts Contacted**		**Known attacks**		**Old Libraries**
	Exposed	Used	Local	Remote	Metasploit	CVE	
OctoPi	3	2	1	0	0	0	-
Machine A	1	1	1	0	0	0	-
Machine B	4	1	1	0	0	3	FTP server
Machine C	3	3	1	1	0	1*	SSL v2
Machine D	1	1	1	0	0	0	-
Machine E	1	1	1	0	0	0	-
Machine F	5	1	1	0	Unable to run		-
Machine G	1	1	1	0	Unable to run		-
Machine H	1	1	1	0	Unable to run		-
Machine I	2	1	1	0	0	0	-
Machine J	10	1	1	0	0	2	Apache
Machine K	5	3	1	0	1	0	Remote Desktop
Machine L	1	1	1	0	Unable to run		-
Machine M	14	4	1	0	1	3	SMB
Paper Printer	6	3	1	0	0	0	-
IoT A	0	0	0	7	0	0	-
IoT B	3	0	0	6	0	0	-
IoT C	2	1	1	0	1	2*	Default Credentials
IoT D	2	2	1	51	0	1*	-
IoT E	0	0	0	2	0	0	-
IoT F	0	0	0	2	0	0	-
IoT G	2	0	0	15	0	0	-
IoT H	5	1	0	23	0	0	-
IoT I	0	0	0	11	0	0	-
IoT J	0	0	0	19	0	0	-

*CVE for a weak cipher suite.

Table 3.7: Network services and known vulnerabilities identified for individual devices.

the control PC will send printing commands to the attacker thinking they are destined for the networked 3D printer, allowing the attacker to steal data or worse modifying design files (e.g., adding defects) before forwarding the file to the real 3D printer.

Insight 8 (**I8**): ***6 out of 13 networked 3D printers had unnecessary network services exposed (e.g., an attacker could control a networked 3D printer using an exposed service à la [10]).***

Open Ports: Exposed network services were identified using the Nmap tool [115].[7] In order to identify if the network services were used, we analyzed the networked 3D printer's network traffic for use of these ports. The majority of 3D printers had unused, exposed TCP services, with some exposing up to 10 unused services (reference Table 3.7). Interestingly, we noted that in general higher cost networked 3D printers had more unused, open ports. This is likely due to the increased complexity of printing operations performed by the higher cost 3D printers. When comparing with commodity IoT devices, we noted fewer unused ports and that networked 3D printers primarily communicated with hosts on the local network while the IoT devices often contacted multiple remote hosts.

Insight 9 (**I9**): ***4 out of 13 networked 3D printers had network services vulnerable to known exploits, often from out of date libraries (e.g., an attacker could utilize a published attack to gain root access on the 3D printer [119]).***

Known Vulnerabilities: Multiple existing tools were used to perform vulnerability scans of the 3D printer's network services. These tools checked for susceptibility to Metasploit attack modules[113], Common Vulnerabilities and Exposures (CVEs) [67], and web server vulnerabilities [187]. Note, some scans were unable to be completed.[8] Four networked 3D printers were vulnerable to published exploits due to outdated libraries (reference Table 3.7).

This becomes a security concern, as we observed a disconnect between software updates for the printing application and the supporting libraries. On some networked 3D printers, supporting libraries were not updated when the firmware was updated (e.g., no OS patches were applied). This left the networked 3D printer vulnerable to known/released exploits (e.g., WannaCry [125]).

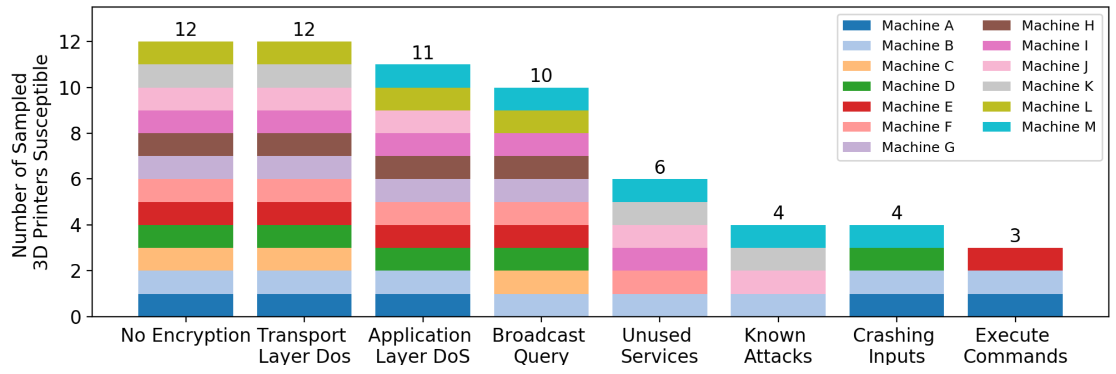


Figure 3.5: Summary of individual 3D printer findings, ordered by pervasiveness.

3.3.2.4 Summary of key findings

In summary, C3PO identified 33 vulnerabilities across 13 networked 3D printers evaluated (our consolidated findings are given in Figure 3.5 ordered by pervasiveness of the security issue). Twelve did not use an encrypted channel (though two may send already encrypted files). Additionally, all networked 3D printers were vulnerable to DoS attacks, some resulting in the machines being unavailable until it was power cycled. Ten utilized broadcast protocols (e.g., mDNS, SSDP, LLMNR) which an attacker could spoof to create a MitM situation between the control PC and the networked 3D printer. Four had applications that were susceptible to malformed inputs, requiring a power-cycle to recover. Combinations of these vulnerabilities allowed us to perform the four attacks in Table 3.8 on multiple networked 3D printers.

In analyzing our findings, we noted a couple of trends. As the cost of a networked 3D printer increased, there was not a significant reduction in the number of identified vulnerabilities. A part of this is likely due to issues such as lack of encryption and susceptibility to DoS being pervasive across all networked 3D printers analyzed. We did note that the higher-cost industrial machines were more likely to be running additional services and therefore more likely to be vulnerable to published exploits (especially as the machines aged). In contrast, the desktop networked 3D printers were more likely to expose network APIs that allowed for directly manipulating the

[7]Machines D, F, G, H, I, & L did not use any of the top 1,000 TCP ports.
[8]The 3D printer would establish connections then not send replies, causing the scanner to wait indefinitely.

		Attack			
	3D Printer	Hazard	Modify print	Crash app	DoS
Desktop	OctoPi				✓
	Machine A	✓		✓	✓
	Machine B		✓	✓	✓
	Machine C		✓	✓	✓
	Machine D		✓		✓
	Machine E		✓		✓
Industrial	Machine F		✓		✓
	Machine G		✓		✓
	Machine H		✓		✓
	Machine I		✓		✓
	Machine J				✓
	Machine K				✓
	Machine L		✓		✓
	Machine M		✓	✓	✓

Table 3.8: Attacks we demonstrated on networked 3D printers, illustrating susceptibility to a range of attacker goals.

printing actuators. Additionally, desktop machines were more likely to have their firmware crash from malformed inputs.

A networked 3D printer's release year did not directly impact the number of vulnerabilities identified, with machines of different ages having a similar number of vulnerabilities. However, we noted that known best practices were least likely to be incorporated on a vendor's initial product, as these machines generally had the most vulnerabilities, regardless of its release year (e.g., a 2019 model had the most vulnerabilities). We assume this is likely due to the pressure to bring a product to market; however, with these machines likely having lifespans of 10+ years (potentially never being patched) it creates significant security risks. Next, we identify how the network deployment allows an attacker to use these vulnerabilities.

3.4 3D Printer Network Deployment Analysis

As a 3D printer's network security is not limited to the device itself, but also its network deployment we build a tool for analyzing their network deployments. We use attack graphing to

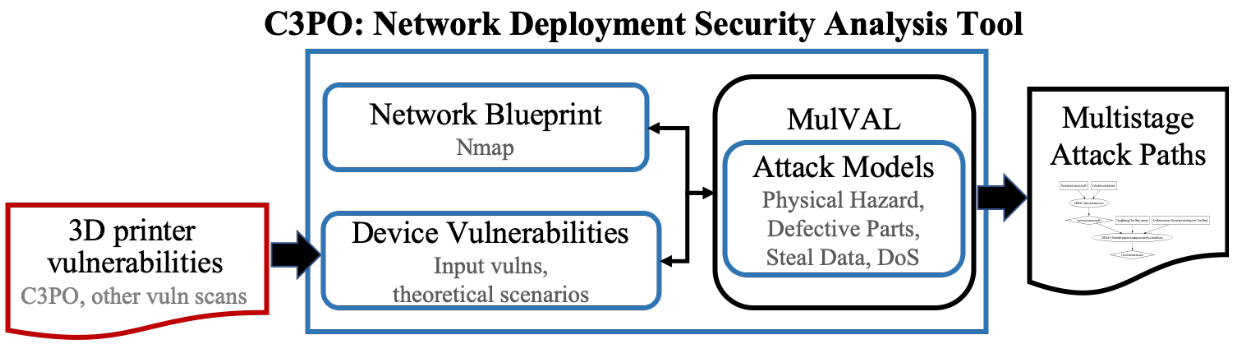

C3PO: Network Deployment Security Analysis Tool
3D printer vulnerabilities
C3PO, other vuln scans
Network Blueprint
Nmap
Device Vulnerabilities
Input vulns, theoretical scenarios
MulVAL
Attack Models
Physical Hazard, Defective Parts, Steal Data, DoS
Multistage Attack Paths

Figure 3.6: Overview of C3PO's network deployment analysis tool, extending prior attack graphing tools, blue (Shaded) boxes represent our additions.

identify how other devices on the network could be part of an attack path targeting a networked 3D printer. We discuss the design and implementation of this tool below.

3.4.1 Network Analysis Tool Design and Implementation

Our network deployment component includes three modules as shown in Figure 3.6.

Network Blueprint Module: Creates a network topology, listing all of the devices a single hop from our key assets (i.e., networked 3D printers and control PCs).

Device Vulnerabilities Module: Includes vulnerabilities for both networked 3D printers and other devices. The networked 3D printer's vulnerabilities can be provided by C3PO's individual 3D printer analysis. For other devices, we can either apply known vulnerabilities (e.g., from a vulnerability scan) or incorporate theoretical scenarios (for example, scenarios could be based upon common vulnerabilities for devices, such as IIoT cameras having default credentials [10]).

Attack Models Module: Takes the outputs from the previous two modules to create an attack graph. It consists of the set of networked 3D printers to evaluate, the attacker goals, the attacker's starting location (i.e., same local network or a remote network), and a mapping of vulnerabilities to attacks. With these inputs and models, we extends the attack graphing tool MulVAL [140] to perform a "what-if" analysis. It uses vulnerabilities from our theoretical scenarios (e.g., what attack paths exist if all IIoT cameras on the network have default credentials) to generate an attack graph, showing how an attacker can cause a physical hazard, create defective parts, steal data, or halt operations on the networked 3D printer(s) being evaluated.

Implementation: In C3PO, we approximate a network blueprint by using Nmap [115] to detect all machines on the same subnet. The network blueprint output along with the device vulnerabilities are parsed into a JSON file that specifies the scenario being evaluated. C3PO then parses this JSON into a MulVAL compatible input file to generate an attack graph.

MulVAL [140] is written in Prolog to evaluate a series of predicates in order to identify attack paths. We extend this functionality by defining predicates for the four attacker goals presented in Section 2.1.2.1: sabotage operations, print defects, steal data, and create DoS. These attacker goal predicates depend upon identifying situations that allow an attacker to create situations such as compromising a network device and executing malicious code, creating a man-in-the-middle situation, and others. MulVAL outputs identified attack graphs as both a .CSV listing and graphically as a PDF.

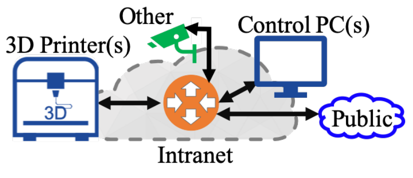
3.4.2 Findings

We evaluated 5 real-world 3D printer network deployments in order to gain an understanding of how networked 3D printers are currently deployed. This allowed us to demonstrate the benefits of C3PO as we analyzed large and complex networks.

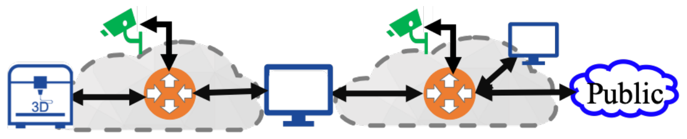
3.4.2.1 3D Printer Deployments Evaluated

The 5 network deployments evaluated ranged from small, single 3D printer deployments (e.g., small, research-focused additive manufacturing labs) to an active makerspace with four types of networked 3D printers on multiple subnets. The five network deployments can be grouped into three deployment categories based upon their network blueprint (depicted in Figure 3.7):

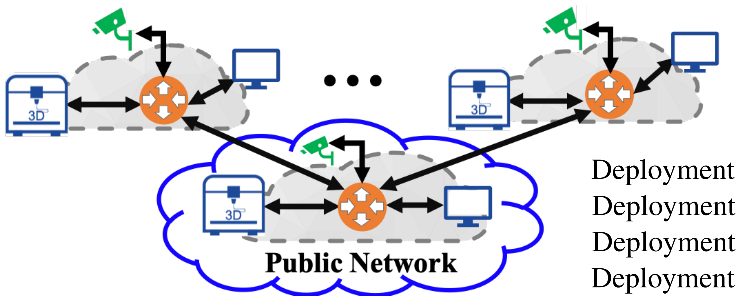
- *Flat network.* All devices are on the same subnet.



(a) Flat Network



(b) Purdue Architecture [213]



(c) Complex

	Net Type	3D Printers	PCs	Other
Deployment A	Purdue	1	1	5
Deployment B	Flat	1	1	9
Deployment C	Flat	3	39	27
Deployment D	Complex	2	8	14
Deployment E	Complex	18	80	96

Figure 3.7: Types of real-world 3D printer network deployments, and description of those surveyed.

- *Purdue Enterprise Reference Architecture [213].*[9] Networked 3D printers are on an isolated subnet, which is bridged by PCs with multiple NICs.

- *Complex.* A publicly accessible subnet, often connected to multiple subnets.

For each network deployment, the network devices identified during the network scan were placed into 4 categories: (1) networked 3D printers, (2) PCs, (3) other devices (e.g., IIoT, paper printers, building automation, etc.), and (4) network hardware. Each deployment had a large number of other devices, accounting for >41% of all the devices on each network. These devices often have weak security properties, increasing the security risks to 3D printers on the same network.

We analyzed 19 scenarios, where each scenario had a different set of assumed vulnerabilities (e.g., network hardware having code execution vulnerability [194], or other devices, such as IIoT cameras, having default credentials [10]). These scenarios were generated from a combination of prior attacks (e.g., Stuxnet using malicious USBs in order to compromise PCs connected to manufacturing networks) and discussions with operators (e.g., need for legacy systems that were

[9]This is the architecture specified in ISA-95 [2].

Device Category	**Scenario**	**Assumed vulnerability**
Baseline	0	No assumed vulnerabilities
PCs	1	Malicious USBs (e.g., [102])
	2	Malicious links (e.g., Phishing)
	3	Old OS (e.g., Windows 95)
Network hardware	4	Exploitable firmware (e.g., [194])
Other devices (e.g., IIoT)	5	Default credentials (e.g., [10])
	6	Exploitable firmware (e.g., [139])
Network hardware & PCs	7	Scenario 1 & 4
	8	Scenario 2 & 4
	9	Scenario 3 & 4
Network hardware & other devices	10	Scenario 4 & 5
	11	Scenario 4 & 6
Other devices & PCs	12	Scenario 1 & 5
	13	Scenario 1 & 6
	14	Scenario 2 & 5
	15	Scenario 2 & 6
	16	Scenario 3 & 5
	17	Scenario 3 & 6
All	18	Scenario 1, 2, 3, 4, 5 & 6

Table 3.9: Theoretical vulnerability scenarios (based upon prior and related attacks) used to evaluate each network deployment for allowing network attacker to achieve their goal.

added to the network). The complete list of scenarios is given in Table 3.9.

Each scenario was analyzed once with an attacker on the local network (e.g., an insider threat) and again with an attacker on a remote network (i.e., starting on a public network). The total number of attack paths for all attacker goals (i.e., cause a physical hazard, create defective parts, steal data, or halt operations) were normalized based upon the number of networked 3D printers and the number of devices with assumed vulnerabilities on the network, the results are shown in Figure 3.8. On average, C3PO identified 5 multistage attack paths to each 3D printer per insecure device on the network.

Across the network deployments analyzed we observed two trends. First, we noted a lack of network isolation. Many networked 3D printers were deployed with a large number of unnecessary and unrelated devices. Attack graphing aided our identification of how these can directly impact the security of a networked 3D printer.

Insight 10 (I10): ***2 of 5 surveyed 3D printer network deployments made 3D printers easily accessible to an attacker (e.g., placing networked 3D printers on the public internet).***

Lack of Network Isolation: Most networked 3D printers were configured to be on a private network and only accessible by other devices on the same subnet. However, one network deployment placed networked 3D printers on public IP addresses. This was not required for the 3D printer's operation. Using the Censys [34] and Shodan search engines [175], 49 additional networked 3D printers from the same manufacturer were found similarly configured with publicly accessible IP addresses. This configuration allows anyone on the internet to view the networked 3D printers' camera output, as well as potentially being able to remotely stop 3D printing jobs.

Similarly, other researchers found >3,700 publicly accessible hosts running a popular web interface for 3D printers in 2018 [124]. Despite the documentation suggesting access control be enabled, many of these instances were found to not require any authentication for sending commands to the 3D printer's actuators and viewing their attached web cameras.

After discussing our findings with the manufacturing center operators, they have since modified their network deployment and removed these 3D printers from the public internet.

Insight 11 (I11): ***2 of 5 surveyed 3D printer network deployments had a non-network hardware device that bridged subnets. A vulnerability in this device amplifies the number of attack paths (e.g., 54% of attack paths in Deployment A rely on a specific PC being compromised).***

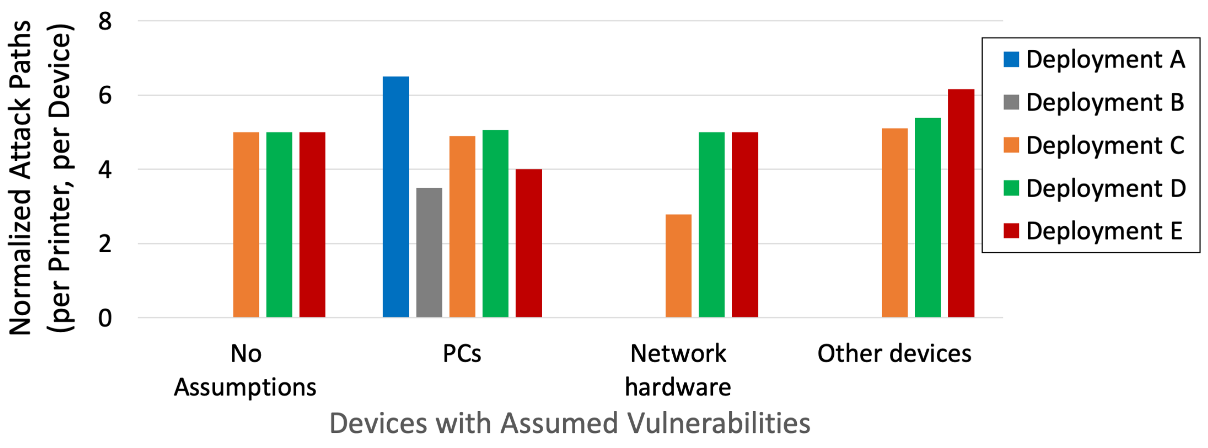
Devices bridging networks: In Deployment A, the 3D printer appeared to be on an isolated network. However, the control PC bridged multiple networks, some of which eventually access the internet. Thus if a remote attacker could compromise this host (e.g., using a malicious link in an e-mail [21]), it would enable them to access the 3D printer as if they were on the local network. This can be observed in our simulation data where attack paths only exist when the category with the bridging devices (generally control PCs) is assumed vulnerable. This analysis aids in identifying which devices are most critical to secure, as they are a part of the largest number of attack paths. Defenders can use this type of analysis to prioritize security efforts and resources to minimize the threats to a networked 3D printer.

Printers attacking printers: A networked 3D printer can also be part of an attacker's multistage attack. Larger deployments often have multiple 3D printers (e.g., Deployment C). If one of these networked 3D printers is compromised by an attacker (e.g., using a published exploit) it can be used for attacking other 3D printers on the same network. This is similar to how an IIoT device could be used by an attacker to launch attacks on 3D printers. Thus, adding a new type of networked 3D printer could alter the threats posed to existing 3D printers in a manufacturing center.

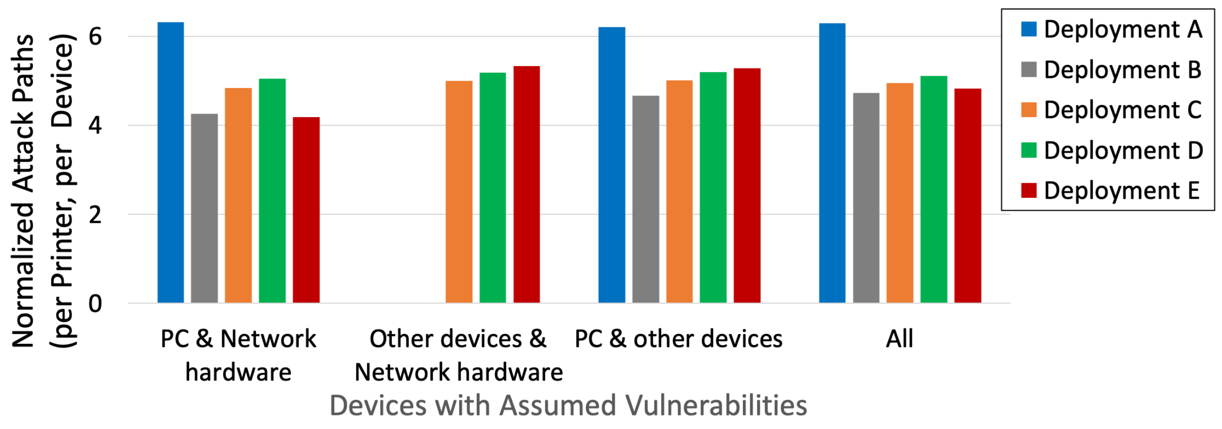
Summary: C3PO demonstrated its ability to analyze real-world 3D printer deployments. All of the surveyed 3D printer network deployments were found to contain a majority of non-traditional IT devices (e.g., IIoT). C3PO was able to use theoretical attack scenarios to identify devices, which if compromised, result in the greatest increase in the number of possible attacks paths. We grouped the attack paths a remote attacker (e.g., on a public network) could perform based upon the vulnerabilities assumed for each category of device (depicted in Figure 3.8). We plot the data normalized for the number of networked 3D printers in the deployment as well as the number of devices with assumed vulnerabilities to allow for comparison between networks of different sizes. For example, Deployment A has a maximum of 44 attack paths while Deployment E has 15,773 attack paths (both having approximately five attack paths per networked 3D printer and assumed vulnerable devices).

3.5 Implications from Network Security Findings

As our study shows, today's networked 3D printers and deployments contain a number of security vulnerabilities. We focus on four high-level vulnerability categories shown in Table 3.10



(a) Normalized number of attack paths when assuming vulnerabilities in a single device category.



(b) Normalized number of attack paths when assuming multiple vulnerabilities across combinations of device categories.

Figure 3.8: Normalized number of potential multistage attack paths for each 3D printer network deployment analyzed given different assumed vulnerabilities.

and summarize potential classical security techniques for mitigating each (see Chapter 5 for a discussion about realizing these defenses).

Our goal here is to suggest *pragmatic* defenses rather than wholesale changes to the entire ecosystem or suggest draconian measures that will impact operations (e.g., avoid other IoT products or lockdown systems). Specifically, our discussions with the operators of these manufacturing centers suggest that they cannot adopt a single, fixed security solution, as they often need 3D printers from different vendors to perform different operations (e.g., printing polymers vs. metals). Thus, manufacturing centers need a flexible defense that can be tailored to the spe-

Vulnerability	**Noted In**	**Implication**	**Recommendation**
Vulnerable to traditional DoS	13 of 13	Block access to 3D printer	Limit concurrent sessions
No encryption	12 of 13	Steal data	Place 3D printer on a VPN
Unused network service exposed	6 of 13	Increases attack surface	Drop traffic to unused services
Bad inputs crash application	4 of 13	Stop printing operations	Drop improperly formatted inputs
Vulnerable to known exploits	4 of 13	Modify firmware	Implement patch in the network

Table 3.10: Vulnerabilities in surveyed 3D printers, their security implications, and classical security solutions to mitigate the vulnerability.

cific needs of their networked 3D printers. Additionally, as manufacturing centers increasingly incorporate new connected devices (e.g., IIoT) there is a high potential that they will be used in multistage attacks against critical assets (e.g., their 3D printers). Indeed, the increased productivity and efficiency of Industry 4.0 is predicated on incorporating these networked devices [75].

To counter the security vulnerabilities identified by C3PO, we look to apply classical network security solutions such as:

- *Rate Limiting*: Traditional DoS attacks are mitigated by limiting the number of simultaneous connections each host may have with a 3D printer.
- *Encryption and Authentication*: Setting up a VPN tunnel between the control PC app and the networked 3D printer can provide data encryption and host authentication.
- *Signature Detection*: Known exploits can be detected in the network by running an Intrusion Prevention System (IPS) which drops traffic matching the exploit's signature.
- *Input Filtering*: Malformed inputs can be dropped by running an IPS to filter inputs such that it only allows data payloads matching the networked 3D printer's expected protocol.

What is needed is a platform for applying these defenses, specifically configured for each device, to defend 3D printer network deployments. Such as the low-cost, software-defined security gateways proposed for defending IoT deployments [17, 95, 176, 229]. These route each device's traffic through a security gateway which leverages software-defined networking and network function virtualization to apply device-specific protections to each device's network traffic.

3.6 Summary

C3PO allows for systematic security evaluations of networked devices and their network deployments. We presented an example use case where we analyzed the security of 13 networked 3D printers and 5 active manufacturing network deployments. We identified 33 vulnerabilities related to lack of encryption, unpatched known vulnerabilities, crashing inputs, and multiple types of DoS. Next, we demonstrated a practical application of attack graphing for identifying potential multistage attack paths in 3D printer network deployments. Analyzing 19 simulated scenarios, we identified 3D printer on public networks, the preponderance of embedded devices in these network deployments, and the potential for 3D printers to be both targets and launch points for attacks.

With the diversity and scale of networked devices in manufacturing networks, we envision that the ideal way to secure these devices is to push security into the network. A promising pragmatic solution is to utilize a security gateway. However, any network protection must be trusted to ensure it does not become ineffective, or even worse, become a launchpad for new attacks. Furthermore, this trust needs to come at a low-cost to ensure it is an accessible option for small manufacturing centers which are more likely to have limited security budgets. Towards this Chapter 4 describes our design of a low-cost, trusted security gateway system that can be deployed locally to defend deployed networked 3D printers.

Chapter 4

JETFIRE: A Low-cost, Trusted Security Gateway

The numerous network security vulnerabilities identified in deployed 3D printers (Chapter 3) demonstrate the need for a security solution that can be applied to already deployed devices. To be practical in the manufacturing domain, a protection system need to defend a diverse set of already deployed 3D printers, while minimize any changes to how they are operated and not adding or modifying the printer's software. Additionally, as a successful attack can have a large financial impact, the security system needs to provide a guarantee that its protections are enforced. Finally, this trust is needed at a low cost, where the system doesn't require any specialized hardware and can run existing software. Thereby allowing small manufacturing centers, which are experiencing an increasing number of attacks [163] while likely having limited security budgets, to deploy these defenses.

Recent efforts have recommended pragmatic "bolt on" security gateways at the network layer to secure deployments of insecure networked devices using software-defined principles [95, 229]. While such gateways are an attractive option, they raise two natural concerns: (1) *Can the gateway architecture be trusted?* and (2) *Can we deliver these benefits to low-cost deployments?*

This chapter presents JETFIRE, a practical, low-cost system with built-in trust for software-defined security gateways [47, 121]. In designing and implementing JETFIRE, we make three key contributions: (1) A practical and deployable basis for trust using a micro-hypervisor root-of-trust; (2) A scalable low-cost system design and implementation to support fine-grained per-device policies; and (3) A formal analysis of the protection JETFIRE offers against infrastructure threats by construction. We demonstrate that JETFIRE provides intrinsic security against a broad spectrum of known attacks against such software-defined architectures. We also show that JETFIRE offers security at low cost (e.g., a $35 Raspberry Pi can effectively support custom per-device IPS instances for small deployments of 50+ devices).

4.1 Security Gateways to Defend 3D Printers

Security gateways have been proposed as a possible non-intrusive defense solution that can be deployed in the network without modifying deployed device operations [17, 95, 176, 227, 229]. We expand upon the background provided in Section 2.2.2 and then discuss our envisioned gateway deployment.

4.1.1 Security Gateway Background

As discussed in Section 2.2.2, industry and academia have proposed securing (potentially vulnerable) IoT devices on edge networks with on-site security gateways [17, 23, 51, 95, 158, 176, 229]. These security gateways are designed to intercept all traffic to and from a device and apply security protections via *middleboxes* at the *network level* (e.g., a firewall). A simple gateway might employ a single network function such as a firewall. However, this is often not fine-grained enough to protect diverse devices. For example, a firewall might mitigate one 3D printer's unused service, while leaving another 3D printer exposed to crashing inputs (when the vulnerability is on the same network service used by the 3D printer for receiving print jobs). To overcome this, recent work [95, 227, 229] has advocated for using a software-defined approach to apply device-specific middleboxes tailored to each device. For example, using a firewall for a 3D printer with unused network services and an IPS for the another 3D printer to detect malicious inputs. This allows for protecting devices that might not support patching (e.g., a device vendor goes out of business) or are unable to run host-based defenses (e.g., antivirus software). The specific protections are determined by a policy agent running on a centralized controller which manages the gateway.

Such a security gateway architecture results in two main operating stages for each device it is defending. First, there is a setup stage, which begins when the security gateway receives a packet from a new device. The gateway queries the controller to determine what action (if any) should be taken. The controller checks for a policy entry for the device's MAC address, if an entry exists, the controller sends the corresponding device-specific middlebox configuration and routing rules to the gateway, otherwise the controller configures the gateway to drop packets from that MAC address. The gateway runs the configuration commands that it receives from the controller, spinning up the device-specific middleboxes and associated routing rules. Once configured, the gateway enters the runtime stage. During this stage, packets are processed by the gateway following the routing rules and middlebox configuration specified during the setup stage without requiring the controller's involvement. Middleboxes are configured to send messages to the controller, for example reporting packets violating a firewall rules. The controller upon receiving these messages queries the security policy to determine if the gateway needs to be reconfigured, for example transitioning from a firewall to an IPS to analyze packet payloads for a potential exploit signature. Thereby allowing the controller to update or reconfigure the device-specific middleobes on the gateway based upon the security policy.

These "bolt-on" security gateways are promising for tackling the challenges of 3D printer network security as they are: practical, deployable, and agile. They are practical as they can be used immediately for defending existing deployments, at a low-cost (e.g., less than $100 [17, 95]). They are deployable as they can run on existing hardware and support existing software. Additionally, their defenses are applied transparently, thereby allowing devices to use their existing

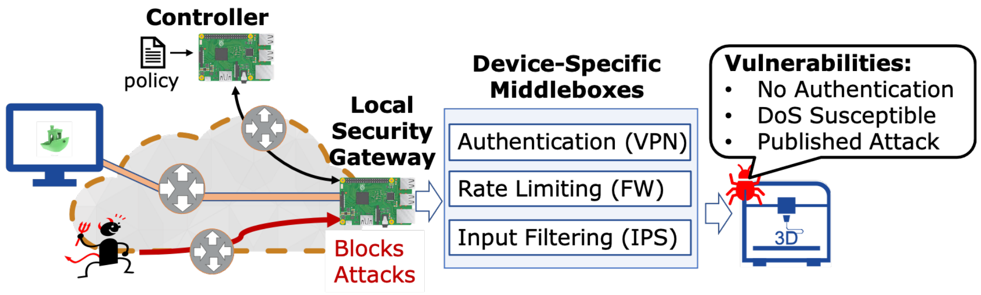


Figure 4.1: Conceptual overview of a local security gateway system for defending deployed networked 3D printers.

workflows. Finally, they are agile as they can provide network protections which are tailored to each device. Next, we discuss a proposed deployment of such a security gateway system to defend networked 3D printers.

4.1.2 Proposed 3D Printer Security Gateway Deployment

We envision the security gateway to be deployed locally, in close proximity to the devices it is protecting (as shown in Figure 4.1). For example, at the switch next to a local cluster of networked 3D printers connected to a wired local area network (LAN), or where the gateway serves as a WiFi access point for wireless devices. Where multiple gateways could be deployed across a manufacturing floor, each defending a set of nearby devices. For example, one gateway defending a group of polymer networked 3D printers, and another gateway defending metal additive machines in a separate location. For the controller, it could be deployed locally (e.g., with other computing resources) or remotely (e.g., hosted in a cloud environment). While a remote controller would incur additional latency and potentially have times when it is out of contact with the gateway(s), we view this as having minimal impact on the gateways operations as after the initial setup stage the gateway can process packets without a connection to the controller. Any latency of availability issues with the controller would only impact new devices being connected to the gateway and policy transitions, which occur infrequently. For the remainder of this chapter, we will focus on a locally deployed controller and a single gateway.

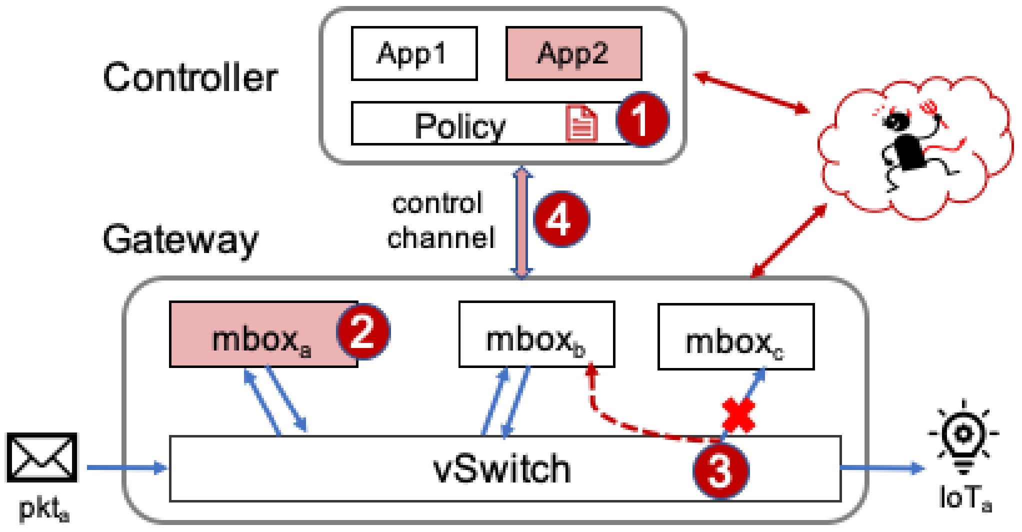


Figure 4.2: Example attacks against current software-defined security gateway architectures altering a gateway's packet processing.

4.2 Need for Trustworthy Gateways

While software-defined gateways are promising for securing manufacturing deployments, it becomes *ineffective* when the architecture itself is *under attack*. We present four concrete, real-world attacks against such gateways, as identified by prior work [106, 121, 146, 193, 208, 225] and limitations of existing piecemeal solutions to mitigate all of these attacks.

As shown in Fig. 4.2, the controller (1), components on the gateway (2,3), and interaction between the controller and the gateway (4) can be exploited. While we discuss each attack separately, it is possible for a single attack to consist of a combination of these example attacks.

Attack 1 - Tamper with security policy(A1): The first attack (1 in Figure 4.2) targets the controller's security policy (derived from [208]). An attacker who gains access to the controller can modify the security policy. For example, specifying a more lenient middlebox (e.g., changing an IPS to a firewall), allowing the attacker's exploit to transit the gateway without detection.

Attack 2 - Alter middlebox operations (A2): The second example attack (2 in Figure 4.2) targets modifying the middlebox (similar to threats noted in [146, 193]). An attacker can gain access to the gateway, using credentials (e.g., from a data breach) or an unpatched vulnerability

Project	Root of trust	Mitigates Attacks			
		A1	A2	A3	A4
Prior Gateways [17, 95, 176, 227, 229]	OS	N	N	N	N
Trustworthy Middleboxes [76, 146, 167, 193]	SGX, TrustZone	N	Y	N	N
Secure Controllers [86, 149, 172]	OS	N	N	N	N
Secure Protocols [59, 94, 107, 108, 132]	OS + Crypto	N	N	Y	Y
JETFIRE (Our system)	Micro-hypervisor + Crypto	Y	Y	Y	Y

Table 4.1: Prior gateways and piecemeal security solutions.

(e.g., [138, 220]). Once on the gateway, the attacker can modify the middlebox's configuration and remove rules that block a device's known vulnerability. Now the attacker's exploit can bypass the security gateway's middlebox protections and compromise the device.

Attack 3 - Alter packet path (A3): The third example attack (③ in Figure 4.2) targets a packet's path on the gateway (similar to [121]). An attacker that has compromised the OS could modify the packet's header to cause it to be routed to the incorrect middlebox, which fails to block the exploit payload.

Attack 4 - Inject malicious control channel messages (A4): The fourth example attack (④ in Figure 4.2) targets the control channel (noted in [106, 121, 208, 225]). In practice, a secure control channel (e.g., TLS) is not often used [170, 217]. An insecure channel allows an attacker to inject malicious messages. For example, sending openflow commands to reconfigure the vSwitch such that packets bypass a middlebox, thereby allowing the attacker's exploit to pass through undetected.

Limitations of existing solutions: As shown in Table 4.1, current gateways [17, 95, 176, 227, 229] do not secure any of the above attacks. While there is some prior work securing individual pieces of the architecture, they still lack end-to-end trust (see details in Section 2.2.3). First, recent work on trustworthy middleboxes uses trusted enclaves to run middleboxes inside untrusted cloud environments (e.g., [146, 193]), but this solution requires specific hardware (e.g., SGX [84], TrustZone [11], TPM [12]) which is not widely available. Second, research on securing the controller (e.g., [149, 172]) has been focusing on using permissions to limit the access of multiple applications, but cannot provide runtime protections against an attacker capable of compromising the OS. Finally, existing secure tunnels (e.g., IPSec, TLS) and work on customized verification protocols (e.g., [94, 107, 108, 132]) can be used to achieve traffic integrity, but they alone are not enough to defend against all attacks.

4.3 Overview

In the previous section, we have made a case for a trustworthy gateway. For such a gateway to be deployed in practice across a wide range of manufacturing deployments, it must also be low-cost. In this section, we first define what we mean by trustworthy and low-cost, then present JETFIRE's overall architecture, assumptions, and challenges.

4.3.1 System Goals

Overarching Trust Property (G1): a trustworthy software-defined security gateway should provide a guarantee that *all output packets are processed by the correct middlebox, even when under attack.* We formulate this guarantee in Section 4.4 and examine how it mitigates existing attacks in Section 4.8.

Achieving the overarching trust property (G1), requires a root-of-trust that current software-defined security gateway approaches lack (Table 4.1). This root-of-trust must provide foundational capabilities (e.g., memory isolation) to enable building a *holistic defense* on both the data and control planes.

Low-Cost (G2): A security gateway that meets the above trust requirements alone is not very useful if it cannot be readily deployed within today's manufacturing ecosystems. We view small, localized deployments in manufacturing floors as those that are most likely to experimentally deploy a security gateway (often having less than 20 localized devices [33, 82, 156, 195]) and experience more frequent attacks [163]. However, they often have limited budgets for network security. A high-end solution would not be practical for these customers. We therefore focus on providing low-cost gateways for small to medium enterprises supporting upto 50 devices with a single hardware gateway costing less than $100.

4.3.2 Key Components

To meet these goals, we envision JETFIRE, a *trustworthy* and *low-cost* software-defined security system. While we focus on a local deployment, we envision our system enabling different deployment models such as a new trustworthy "security-as-a-service" offering that providers (e.g., ISPs, CDNs) can offer to consumers. Where a security provider could run the controller in a cloud environment, and provide security gateway(s) for their customers to deploy locally in their manufacturing center.

In contrast to existing gateway architectures [95, 227, 229], JETFIRE runs both the controller and gateway software on top of a carefully chosen *root-of-trust* (4.5.1).The root-of-trust provides capabilities for isolating sensitive data (e.g., control policy, secure keys) and attesting the integrity of running software (e.g., middleboxes), while supporting many hardware platforms and commodity software (e.g., Linux, Docker). Building upon the root-of-trust, we add four extensions to achieve our overarching trust property (G1) while mitigating the attacks previously mentioned in Section 4.2.

1. We migrate the control policy into the isolated memory protected by the root-of-trust (mitigating attack class ①).
2. We use a performant attestation approach to verify the integrity of running software (mitigating attack class ②).
3. In each gateway, we design a trusted signing mechanism between the vSwitch and each middlebox to protect a packet's path and data (mitigating attack class ③).
4. We create a secure channel between the controller and the gateway to protect control messages (mitigating attack class ④).

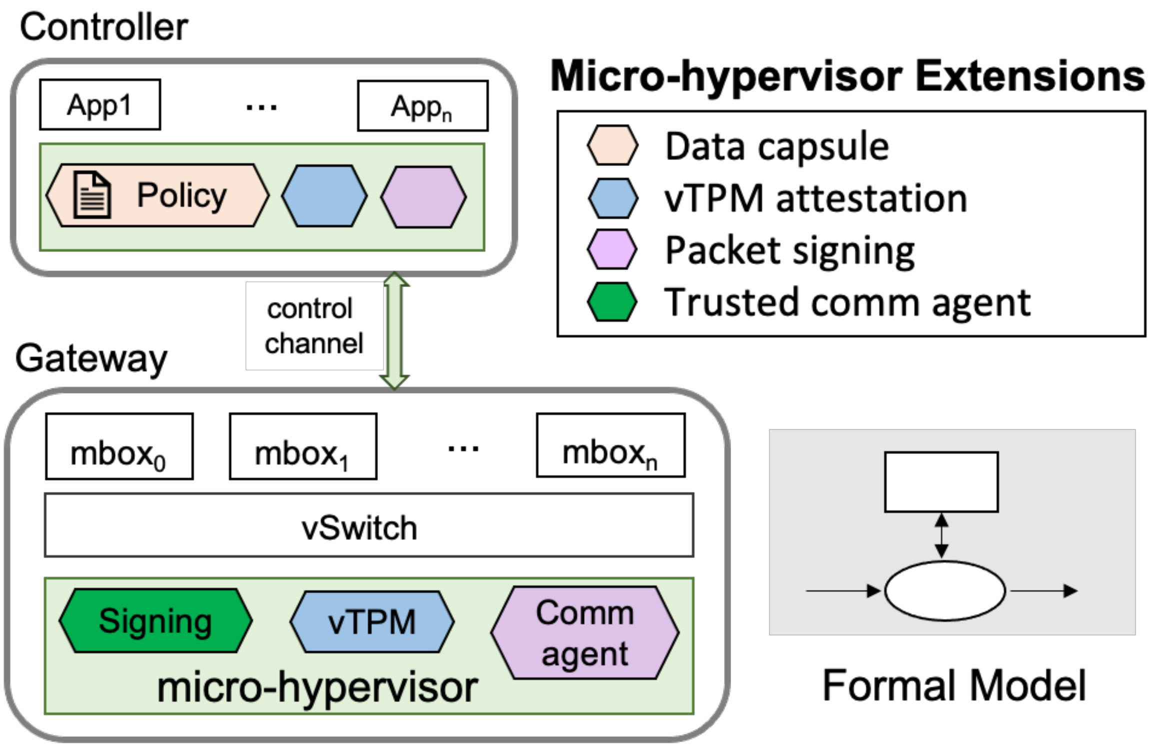


Figure 4.3: JETFIRE's trusted security gateway adds fine-grained protections to provide low-cost, end-to-end packet processing guarantees.

Our formal system model (Section 4.8.1.1) and comprehensive system evaluation (Section 4.8.2) show that JETFIRE can achieve our trust property and mitigate several prior attacks.

4.3.3 Assumptions

System Assumptions: We assume all packets to and from an device must go through the gateway as their *first-hop*. And we scope our system to only providing network protections to devices using an IP-based network. While some devices use other protocols (e.g., BLE, ZigBee), many use IP directly or connect to a hub on an IP network. We also assume middleboxes are correctly implemented and are able to block all network exploits targeting the device they are protecting.

Threat Model: We consider a powerful network adversary capable of compromising the gateway and controller's operating system (OS) via the network. The adversary's aim is to render the gateway ineffective, and then access unprotected devices. The adversary can flexibly choose combinations of attacks from the literature (e.g., attacks in Section 4.8.2), but cannot directly access a device or use an evil twin attack (e.g., [162]) to bypass the gateway.

We do not aim to protect against an attacker generating DoS conditions[1] on the gateway itself (e.g., maliciously dropping packets that arrive on the gateway so that they are never forwarded to the appropriate end-host [225]), nor provide confidentiality to packets and middleboxes (e.g., [146, 193]). Attacks modifying the controller's global network view by impersonating a device or advertising false paths (e.g., [54, 208, 225]) are out-of-scope of this work.

4.3.4 Challenges

We highlight two challenges towards achieving our trustworthy and low-cost goals.

- **Good performance with a small TCB (Section 4.5)** For a trustworthy security gateway to be used in practice, it must provide good performance. Additionally, the added trust should not come at the expense of a large TCB. Our key idea is to use a combination of isolation and attestation techniques so that we can isolate small pieces of critical software while attesting bigger, legacy software components.

- **Scalable middleboxes on low-cost platforms (Section 4.6)** A low-cost gateway needs to support continuing growth of networked devices ([33, 82, 156, 195]). However, existing low-cost platforms (e.g., Raspberry Pi) only support 3 IPS middleboxes (see details in Section 4.9.1), which is insufficient for many manufacturing deployments. We identify that memory consumption is the main bottleneck for scalability and propose two optimization techniques for scaling support to approximately 50 devices.

[1]A Jetfire gateway can be used to mitigate the DoS attacks identified by C3PO in Chapter 3; however, the gateway architecture creates a new DoS vector. An attacker who compromises the gateway could maliciously drop packets on the gateway so that they do not reach their end destination (e.g., the networked 3D printer being protected by the gateway).

4.4 System Trust Properties

We create a formal model of today's software-defined security gateways (Section 2.2.2) to inform the design of JETFIRE (see discussion and evaluation of this model in Section 4.8). This model helps us define our overarching trust property and derive four required sub-properties protecting critical components and interfaces. These properties then define JETFIRE's trust requirements for achieving our overarching trust property.

4.4.1 Overarching Trust Property

Given a network where all of a device's inbound and outbound traffic goes through our trusted security gateway, GW, our goal is to ensure that any packet, pkt, output by the gateway was processed by the correct middlebox while operating in a known state, so that benign packets are allowed and malicious packets are dropped. Our trust goal can be denoted as:

$$\begin{aligned} \forall pkt \in BenignPkt,\ processPkt(GW, pkt) &= \text{Allow} \\ \forall pkt \in MaliciousPkt,\ processPkt(GW, pkt) &= \text{Drop} \end{aligned} \tag{4.1}$$

To achieve this goal in the presence of an attacker, the entire gateway architecture must be trustworthy, expressed formally as:

$$\begin{aligned} &TrustedGateway(GW, Controller) \iff \\ tamperProof(policy)\ \wedge\ &correctInstance(vSwitch, mbox_{0,\ldots,n}, apps_{ctl}) \wedge \\ secureChannel(channel_{gw}, channel_{ctl})\ \wedge\ &\forall mbox_i,\ authenticateRoute(vSwitch, mbox_i) \end{aligned} \tag{4.2}$$

Where:

$$\begin{aligned} GW &= \{vSwitch, \{mbox_0, \ldots, mbox_n\}, channel_{gw}\} \\ Controller &= \{policy, apps_{ctl}, channel_{ctl}\} \end{aligned}$$

4.4.2 Required Sub-Properties

Our overarching security property is composed of four sub-properties, where each maps to a trustworthy requirement: *tamperProof* (TR_{sw1}), *correctInstance* (TR_{sw2}), *secureChannel* (TR_{comm1}), and *authenticateRoute* (TR_{comm2}).

Security Policy Isolation and Mediation (TR_{sw1}): The first sub-property is to protect the security policy stored in the controller. Controller applications are subject to attacks [208, 225] which make the security policy vulnerable. As the correctness of the rest of the system is based upon this policy, we need to ensure it is tamper proof. To achieve this, the security policy needs to be isolated in protected memory with all access mediated by a trusted entity (e.g., blocking the OS and other untrusted applications from accessing the security policy). Our security policy isolation and mediation sub-property (TR_{sw1}) can be denoted as:

$$\begin{aligned} &\mathit{tamperProof}(\mathit{policy}) \iff \\ \mathit{isolatedMemory}(\mathit{policy}) \;\wedge\; &\mathit{mediatedAccess}(\mathit{policy}) \end{aligned} \tag{4.3}$$

Component Instance Validation (TR_{sw2}): Besides the security policy, the software of key components must not be altered by an attacker (e.g., Attack 2 where the middlebox was altered [146, 193]). Such alterations can be detected by validating key software components are running the correct instance. Software components that must be validated includes the controller application, vSwitch and all middleboxes. Our component instance validation sub-property (TR_{sw2}) can be denoted as:

$$\begin{aligned} &\mathit{correctInstance}(\mathit{vSwitch}, \mathit{mbox}_{0,\dots,n}, \mathit{app}_{ctl}) \iff \\ \mathit{remoteAttest}(\mathit{apps}_{ctl}) \;\wedge\; \mathit{remoteAttest}(\mathit{vSwitch}) \;\wedge\; &\forall \mathit{mbox}_i, \mathit{remoteAttest}(\mathit{mbox}_i) \end{aligned} \tag{4.4}$$

Packet Path and Data Validation (TR_{comm1}): Each packet must be routed to the correct middlebox as specified by the security policy. Prior work on Internet routing has advocated for per-hop path authentication to validate that packets followed the specified path [108, 132]. We aim to provide similar guarantees in order to detect packets maliciously routed to the wrong middlebox (e.g., Attack 3). In particular, we need to verify whether the *intended path* of a packet has been enforced, and whether packet data has been modified in-between. We denote our packet path and data validation sub-property (TR_{comm1}) as:

$$\begin{aligned} &\forall mbox_i, authenticateRoute(vswitch, mbox_i) \iff \\ &\forall pkt, \big(intendedPath(pkt, policy) = mbox_i\big) \implies \\ &\big(actualPath(pkt) = vSwitch \leadsto mbox_i \leadsto vSwitch\big) \ \wedge \ unmodifiedData(pkt, vSwitch, mbox_i) \end{aligned} \tag{4.5}$$

Control Message Integrity and Authentication (TR_{comm2}): To protect against control channel attacks (e.g., Attack 4)[54, 106, 208, 217, 225], we aim to ensure that the control channel is secure. To achieve this, the control channel needs to be authenticated and encrypted so that data transmitted over the channel has not been modified or spoofed (e.g., only the controller can send middlebox configuration commands to the gateway). Meanwhile, the *secret keys* used by the channel need to be isolated and any access is mediated by a trusted entity, denoted:

$$\begin{aligned} &secureChannel(channel_{gw}, channel_{ctl}) \iff \\ &authenticatedEncrypted(channel_{gw}, channel_{ctl}) \ \wedge \\ &isolatedMemory(keys) \ \wedge \ mediatedAccess(keys) \end{aligned} \tag{4.6}$$

A system that provides these properties will be secure *by construction* and mitigate the example attacks in Section 4.2. Next, we design a system to provide these trust requirements.

4.5 Design for Trust

Based on our formal model of software-defined security gateways (Section 4.8.1.1), we first identify a low-cost, low-TCB root-of-trust that provides the foundational security capabilities (e.g., isolation) required to achieve our overarching trust property (Section 4.4.1). Then, we discuss our approach for building fine-grained security protections that realize each of our trust requirements (Section 4.4.2) on top of this root-of-trust.

4.5.1 Micro-hypervisor as a root-of-trust

As prior software-defined security gateways lack a root-of-trust (Table 4.1), we begin by selecting an appropriate root-of-trust. The root-of-trust must be available for low-cost hardware (e.g., ARM) and support legacy software without requiring reimplementation to be practical.

Design Alternatives for Root-of-Trust: The design space for root-of-trust can be categorized along two axes (summarized in Table 4.2). First, hardware dependent approaches (e.g., SGX enclaves [146, 193]) have been used to secure middleboxes in cloud environments. Unfortunately, these hardware features are not common on low-cost platforms and only support limited applications (e.g., no system calls [84]). On the other hand, pure software approaches such as formal verification and secure programming languages too have limitations. However, these approaches are hard to directly deploy today as they require significant reimplementation and verification effort to add protections to existing software. As many commonly used software applications can span over 100,000 lines of C/Java, these approaches quickly become intractable.

Why a Micro-hypervisor: Rather than using a pure hardware or software approach, we advocate using a hybrid approach in the form of a *micro-hypervisor*. A micro-hypervisor [122, 174, 198, 200] is in essence a software reference monitor [164], that acts as a guardian of system resources (e.g., files, sockets). Hypervisors have been used to integrate fine-grained security protections into commodity software; e.g., identifying covert malware, providing trusted system calls, attestation, debugging, tracing, application-level integrity and confidentiality, trustworthy resource

Root-of-Trust	**Security Capabilities**	**Low-Cost**	
		Legacy software	Hardware
Trusted enclave (SGX, TrustZone)	isolation, attestation	No system calls, Limited memory	Limited processors
Secure languages (Rust, OCaml)	isolation, mediation	Requires reimplementation	x86, ARM
Micro-hypervisor	isolation, mediation, attestation	Supports	x86, ARM, microcontroller

Table 4.2: Comparison of root-of-trust design space options.

accounting, on-demand I/O isolation, trusted path, and authorization [36, 56, 65, 112, 122, 155, 169, 171, 177, 190, 190, 202, 203, 211, 219, 226, 236, 241, 242]. A *micro-hypervisor* retains the foundational capabilities (isolation, mediation, attestation and extensibility) of traditional feature-rich hypervisors, but with a small software base that is amenable to formal verification to ensure it is implemented without vulnerabilities [9, 198, 200]. Further, they only require hardware support for virtualization, allowing it to run on most existing low-cost hardware platforms (e.g., ARM [199], x86 [174, 200], microcontroller [9]). It directly supports commodity software without any limitations or modifications. Thus, a micro-hypervisor is a low-cost root-of-trust that is well suited to the manufacturing domain.

4.5.2 Micro-hypervisor Extensions for Achieving Trust Sub-properties

Unfortunately, a root-of-trust alone is insufficient for achieving our overarching trust property (Section 4.4.1) as it does not innately integrate with the gateway's software to enforce specific protections. Next, we show how we build on top of the micro-hypervisor to achieve each of our trust requirements. To this end, we build four micro-hypervisor extensions (shown in Figure 4.3): *data capsule* and *vTPM* for protecting the software (TR_{sw1}, TR_{sw2}), and *packet signing* and *trusted agent* for protecting the communications (TR_{comm1}, TR_{comm2}). We discuss our design for realizing each of these sub-properties in the sections below.

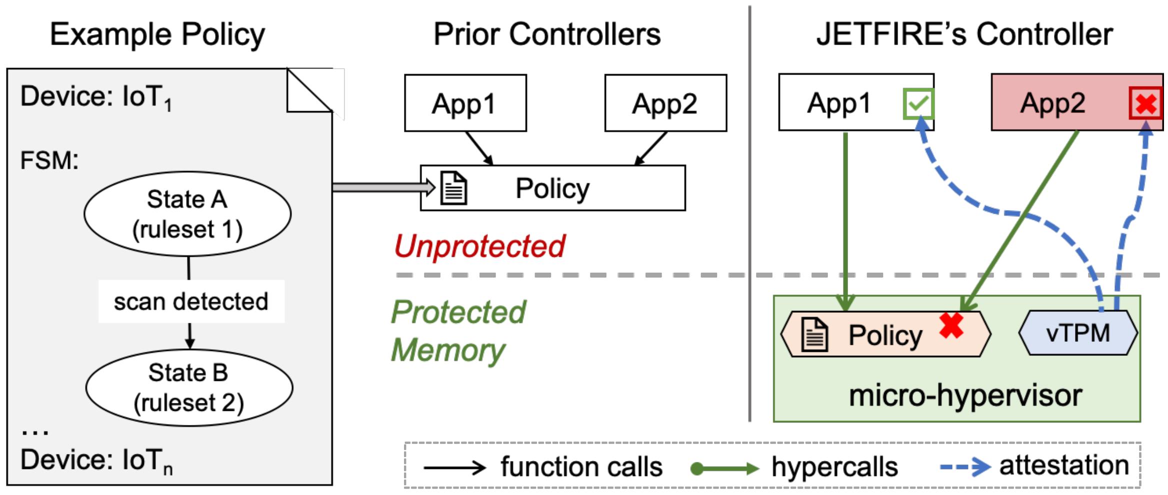


Figure 4.4: JETFIRE migrates the policy file and processing logic into a data capsule, protected by the micro-hypervisor. An application (App2) that is not in the whitelist or fails at attestation cannot access it.

4.5.2.1 Data Capsule for Security Policy Isolation and Mediation

Recall that our trust guarantee requires that the security policy be in isolated memory with mediated access. A naïve approach is to place the entire controller into memory protected by the micro-hypervisor. However, this approach creates two unwanted impacts: (1) it significantly *increases the TCB*, potentially exposing the micro-hypervisor to new attacks.[2] (2) it incurs *performance penalties* for all operations as every system call or external functionality needs to be mediated by the micro-hypervisor.

Our Approach: Instead of placing everything into the TCB, we carve out small critical pieces (e.g., control policy, secret keys) from the software, and migrate them into the micro-hypervisor as a *data capsule*. Each data capsule is isolated from unprotected memory and all accesses are meditated, providing *fine-grained* protection.

Figure 4.4 shows an example data capsule protecting the controller's security policy. The security policy specifies a finite state machine (FSM) for each device. Each state of the FSM describes a specific middlebox configuration (e.g., Snort with ruleset 1). Transitions between

[2]A typical SDN controller (e.g., NOX, ONOS, OpenDayLight) has a code base from 20-300k lines of code, which is an order of magnitude larger than many micro-hypervisors.

states are based upon alert messages sent by the middlebox. For example, an alert message (or series) could indicate a network “scan detected”, triggering the controller to reconfigure the middlebox with ruleset 2 which performs deep packet inspection.

In a traditional security gateway, the security policy along with other controller applications run in unprotected memory which is exposed to attackers (e.g., ① in Figure 4.2). Rather than placing all of them (20k lines of code for OpenDaylight) into the root-of-trust, we only migrate the control policy and its state transition logic into a data capsule, which is a small portion of the controller’s code (195 lines of code). Access to the data capsule which is placed in isolated memory protected by the micro-hypervisor, is then mediated by the micro-hypervisor via code white-listing [150, 202]. This scheme ensures that the data capsule can only be accessed by the middlebox management application. The original code base is then refactored to access these data capsules.

4.5.2.2 vTPM for Component Instance Validation

The code of critical software components (e.g., middleboxes) must be protected from malicious modifications. Instead of carving out individual pieces of these components, we relax our protection to only identifying changes to the code using remote attestation [12, 48] via a *vTPM*. This limits the TCB increase to only the attestation operations and measurements. This combination of isolation and attestation allows us to maintain a small TCB with good per-packet performance.

Our Approach: Attestation is often provided by a Trusted Platform Module (TPM). As a physical TPM is not available on all hardware, we leverage a software implementation, a *virtual trusted platform module (vTPM)* in the micro-hypervisor. We utilize a subset of its capabilities to securely store a chain of measurements, by extending a platform configuration register (PCR), and its ability to securely provide the stored values (i.e., a PCR quote). These vTPM measurements can be trusted as access to the vTPM is mediated by the micro-hypervisor and the PCR values are placed in a data capsule, precluding an attacker from maliciously altering the PCR val-

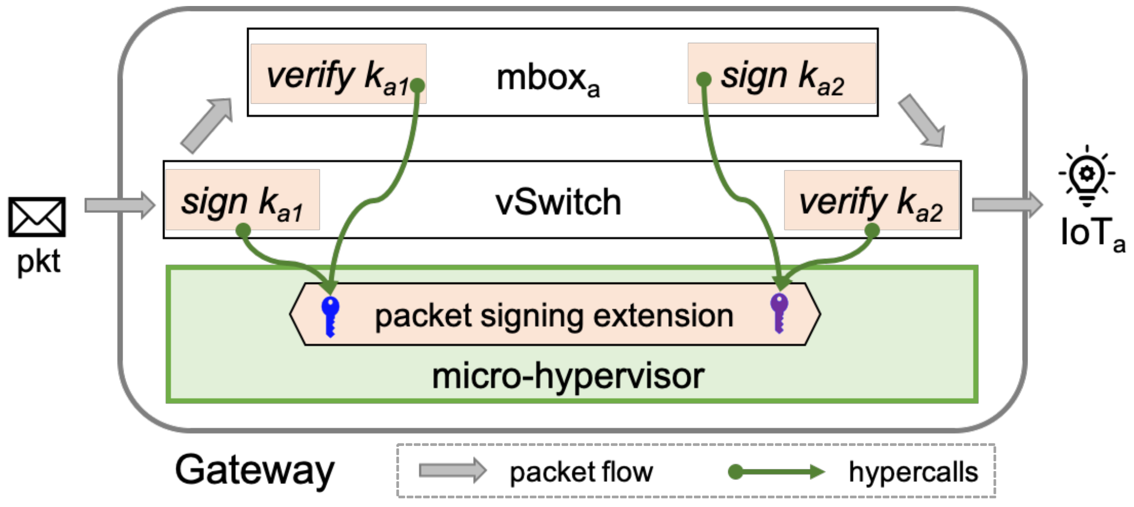


Figure 4.5: Packet signing operations to verify packet path and data between the vSwitch and middleboxes.

ues (e.g., resetting the PCR and/or injecting malicious PCR extensions). While the vTPM does not provide persistent secure storage and attestation keys by itself, it has performance advantages compared to a physical TPM (e.g., not limited by physical memory or data bus).[3] This allows applying vTPM measurements at a fine granularity, providing increased PCRs (100+) and lower access latency (0.9 msecs).

4.5.2.3 Packet Signing for Packet Path and Data Validation

To provide packet path and data validation on the gateway, a strawman solution is to use existing secure tunneling protocols (e.g., IPsec, TLS). Unfortunately, this has two limitations: (1) Most tunneling implementations rely on the OS to protect secret key files (e.g., .ssh directory), allowing an attacker who has compromised the OS to craft secure messages, (2) Existing tunneling approaches are too heavyweight for traffic on the gateway (data path traffic). As we show in Section 4.9.2, when enforcing tunneling between the vSwitch and middleboxes for every packet, it could reduce packet processing throughput by greater than 67% of the baseline throughput.

[3]Note that a vTPM can be bridged with a platform hardware (physical) TPM, if available, to provide persistent secure storage and attestation keys.

Our Approach: Inspired by prior work on path verification protocols (e.g., [108, 132]), we design a simple *packet signing* mechanism to enforce each packet follows the correct path with the correct data. Figure 4.5 shows our packet signing mechanism. We extend the vSwitch and each middlebox by adding two functions: *sign* and *verify* Both functions use keys stored in the micro-hypervisor to generate a digital signature. The $sign$ is triggered when sending a packet and $verify$ is called when receiving a packet.

For example, a packet pkt arrives at the vSwitch, which looks up its intended path to $mbox_a$ (configured by the controller). Then vSwitch calls $sign$ to create a digital signature over the entire packet (header and payload) using key k_{a1}, a unique key shared between the vSwitch and the $mbox_a$. After the middlebox receives the packet, it uses its key k_{a1} to $verify$ the signature. If the packet has been tampered with or routed to the wrong destination (e.g., Attack 3 in Section 4.2), the verification fails and the packet is dropped. After the middlebox processes the packet, it $signs$ the packet using another key (e.g., k_{i2}). Note that this is necessary as the middlebox might modify the packet data, resulting in the previous signature being *obsolete*. The vSwitch similarly verifies this signature.

Compared to traditional tunneling, our verification approach is more lightweight. It does not require expensive setup (unlike TLS) and uses a simpler verification header (unlike IPSec). Furthermore, the micro-hypervisor provides assurances that the digital signatures can be trusted, as it protects the secret keys and sign/verification functions, thereby stopping an attacker from forging signed packets. Thus, our signing mechanism ensures that packets follow the correct paths at a low performance overhead (see comparison in 4.9.2). Next, we discuss how we secure the control channel.

4.5.2.4 Trusted Agent for Control Message Integrity and Authentication

Existing SDN architectures often support an encrypted message exchange mechanism (e.g., IPsec/TLS) for the control channel. Unfortunately, these do not protect the secret keys.

Attack in Section 4.2	**Requirement**	**Defense**
Attack 1	TR_{sw1}	Limit access via data capsule
Attack 2	TR_{sw2}	Attest component via vTPM
Attack 3	TR_{comm1}	Identify modification via packet signatures
Attack 4	TR_{comm2}	Protect secret keys via trusted agent

Table 4.3: Summary of design components for achieving trust requirement in order to mitigate attacks from Section 4.2.

Our Approach: We add trusted agents on the controller and gateway to secure control channel traffic. These agents intercept all control channel messages and protect the secret keys in a data capsule. Further, micro-hypervisor mediation ensures only trusted applications can access the secret keys. We assume that the secret keys are exchanged out of band and that a unique pair exists for each controller and gateway set. Since the secret keys and agents are memory-protected and isolated by the micro-hypervisor, they are immune to attacks from untrusted components including the OS.

4.5.2.5 Design Summary

Table 4.3 summarizes key components of our system design, the trust requirements they address and the class of attacks they protect against. Our system design relies on a micro-hypervisor root-of-trust to provide the needed trust capabilities at a low-cost, running legacy software on a broad base of existing hardware. Our performant, low-TCB hypervisor extensions (data capsule, vTPM, packet signing, and trusted agent) work in synergy towards achieving our overarching trust property. As shown in Table 4.3, the data capsule (TR_{sw1}) blocks an attacker from modifying the security policy (Attack 1). Attestation via the vTPM (TR_{sw2}) detects an attacker modifying a middlebox (Attack 2). Packet signatures (TR_{comm1}) mitigate local attackers modifying packets (Attack 3). Finally, trusted agents (TR_{comm2}) block an attacker from forging control channel messages (Attack 4).

4.6 Enabling Low-Cost with Scalable Middleboxes

In the previous sections, we described how we can design a trustworthy gateway based on a low-cost root-of-trust. Another key challenge is scalability, where a single gateway can protect all deployed devices in a localized area (e.g., a production line). In this section, we first identify the scalability bottlenecks and then present two of our optimizations.

4.6.1 Identifying Bottlenecks

We start with testing how many middleboxes a low-cost Raspberry Pi 3 can run simultaneously. Since each middlebox is assigned to one device, this test shows the maximum number of devices the current platform can protect. We pick Snort [45] as an example middlebox because an intrusion prevention system (IPS) is likely required by all deployments. For each Snort instance, we run it using the default configuration with the full community rule set [45] to provide broad coverage. Unfortunately, we could only run three snort instances simultaneously, which is insufficient.

When running multiple Snort instances, we noted that the main bottleneck limiting scalability was the memory required by each Snort instance. One Snort instance consumes 452 MB of memory. We used Intel's VTune Amplifier profiling tool to identify the most significant contributors to memory consumption and found that the majority of the memory was allocated on the heap for rules and their processing, followed by socket buffers. Based on these findings, we propose two optimizations to reduce the memory consumption.

4.6.2 Optimizations

Customized Configurations: Our first optimization is to customize Snort configurations based on the protected device. Our analysis shows that the memory required by a Snort instance is directly proportional to the number of rules it is configured with ($\sim$27.5 KB per rule). The full community rule set, composed of 10,918 rules, is designed to protect a wide array of devices.

Our idea is to use custom profiles that only contain the rules that are applicable to that particular device (e.g., a device running Linux does not need Windows rules). Thus, we categorize the community rules (based upon rule descriptions and exploit references) into 116 separate categories to more precisely identify which rules might be applicable for a given device. This customization realized up to a five-fold decrease in Snort rules.

We also optimize the socket buffer, the second largest memory consumer. Snort has a default socket buffer of 166 MB, to support analyzing network traffic from multiple devices at rates of 200-500 Mbps. For our use case, supporting such high throughput is unnecessary, as each device has its own instance of Snort and peak bandwidths of less than 30 Mbps.[4] Using this observation, we reduce the socket buffer and free up 163 MB per Snort instance.

Sharing Rules: After categorizing rules for each device, we noted that many of the rules were still common across multiple devices, with only a small fraction of the rules being device-specific. This results in the same rule being in memory multiple times (i.e., once for each device being protected). Our insight is to place these common rules into a shared memory region, so that we only have one instance of the rules in memory (similar to [143]). Subsequent instances can be instantiated at a significantly reduced memory footprint (e.g., 30 MB per instance). Combining these optimizations, reduces the memory footprint of Snort by more than 12x per instance, allowing a single hardware platform to support more than 50 simultaneous Snort instances (Figure 4.7a).

Our approach and optimizations are general and can apply to other platforms and middleboxes as well. For instance, many other low-cost platforms (e.g., OpenWRT routers) have less than 2 GB of RAM giving them the same bottleneck. Similarly, other middleboxes (e.g., Zeek [233], Suricata [188]) that use a common set of rules across multiple devices, could also benefit from our analysis approach and optimizations.

[4]In sampling 13 commercial networked 3D printers, we noted an average throughput of less than 2 Mbps.

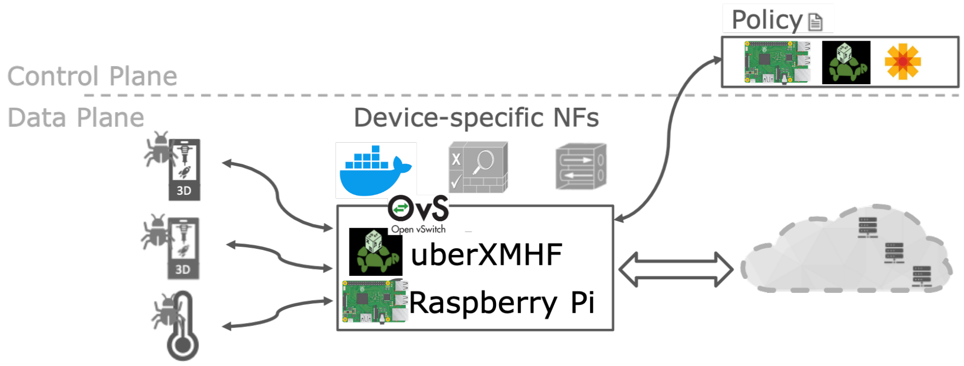


Figure 4.6: Overview of key implementation components of JETFIRE prototype.

4.7 Implementation

We implemented a JETFIRE prototype using two Raspberry Pi 3Bs, one for the controller and the other for the gateway [47]. Both use the uberXMHF micro-hypervisor framework [198] and Raspbian Jessie (Linux 4.4.y). For the controller, we use OpenDayLight (Aluminum), the largest open source SDN controller. For the gateway, we run Dockerized middleboxes (e.g., Snort, Squid, iptables) with OpenvSwitch (OVS 2.12.1) for packet routing. These components are depicted in Figure 4.6. Next, we describe uberXMHF and how we extend it to achieve our trust properties.

uberXMF micro-hypervisor: Our trust architecture is built on top of uberXMHF, an open-source,[5] formally verified, micro-hypervisor framework [198, 199, 201]. We chose uberXMHF because it supports both x86 and ARM platforms and provides a modular framework for compositional verification. This allows for adding hypervisor extensions while preserving the core micro-hypervisor's memory integrity without needing to repeat the verification [198].

We use uberXMHF (v6.0) and realize our protections (Section 4.5.2) as hypervisor extensions. Each hypervisor extension exposes a hypercall interface that is callable from both user and kernel space and allows for transferring up to 4 KB of data. Our *security policy* extension stores and transitions each IoT device's current FSM state, it prohibits adding states by setting

[5] `https://uberxmhf.org`

the maximum number of states for each device's FSM and limiting this to only occur once. Our *packet signing* extension uses the micro-hypervisor's internal cryto library to perform an SHA256-HMAC on an input data buffer using secret keys stored in the micro-hypervisor. Our *trusted agent* uses a secret key in the hypervisor to encrypt a data buffer using AES encryption.

Trusted Data and Code (TR_{sw1}, TR_{sw2}): We integrate our security policy into our modified OpenDayLight controller so that each policy query goes to the micro-hypervisor (to achieve TR_{sw1}). As the hypervisor extensions are in C and the controller in Java, we build a shared library that performs the hypercalls and leverage a Java Native Interface (JNI) to integrate these into the controller's operations. For attestation, we create a Python daemon that measures each middlebox's executables and configuration. Measurements are stored in the the micro-hypervisor's vTPM, using its PCR extend interface. Then measurements are sent to the controller (as a vTPM quote), which the controller checks against the value in its security policy (to achieve TR_{sw2}). These measurements and quotes are then repeated periodically (e.g., 1 min) to detect an attacker adding code or modifying configurations after the middlebox is instantiated.

To demonstrate the performance benefits of using a vTPM, we compared the time required to store a measurement (e.g., extend a PCR) on a physical TPM with a virtual TPM. As shown in Table 4.4, the virtual TPM was 20x faster while providing 8x more measurement storage locations (PCRs).

TPM	**Median Time**	**PCR Registers**
Physical	17.2 milliseconds	24
Virtual	0.86 milliseconds	configurable, up to 120

Table 4.4: TPM PCR extend comparison between virtual and physical TPMs on the Raspberry Pi 3B.

Trusted Communications (TR_{comm1}, TR_{comm2}): We integrate our packet signing into OVS and our Dockerized middleboxes (to achieve TR_{comm1}). We determined that a SHA256-HMAC was optimal on the Raspberry Pi platform by benchmarking a range of potential algorithms, from public-key signatures (e.g., ECDSA) to signed message authentication codes (e.g., HMAC, CMAC, etc.). Our benchmark compared the network throughput when computing a signature in user space for each packet to a baseline throughput when no signatures were calculated (shown in Table 4.5).

Algorithm	Normalized Throughput	Signature Length (bytes)
Baseline (no signing)	1.0	0
HMAC-SHA256	0.221	32
CMAC-AES	0.0313	16
CMAC-RC2	0.0274	8
ECDSA	0.00013	72
UMAC*	0.00067	16
GMAC*	0.27	16
HMAC-SHA1†	0.243	20
HMAC-MD5†	0.236	16

*Requires random nonce/IV † Hash algorithm has been broken

Table 4.5: Signing algorithm comparison on Raspberry Pi 3B for a user space application signing full MTU packets.

Within OVS, we add two *new actions* (sign and verify) to both the user and kernel space virtual switch functionality (where the kernel module realizes a 2x throughput increase (Section 4.9.2). The sign function appends the signature returned by the hypercall to the packet's payload. For this added data to arrive at the middlebox, the packet's headers are modified to account for the increased packet length. The verify function strips the signature from the packet, re-calculates the packet header, and performs a hypercall to verify whether the two signatures match.

For middleboxes, we leverage `NFQUEUE` interfaces to implement our signing in userspace. Since `NFQUEUE` can intercept both received and output packets, we added a userspace callback to perform operations similar to the sign and verify actions added to OVS. This allows for an unmodified packet to be analyzed by the network function (e.g., Snort). We also integrate our trusted agent into the middleboxes (to demonstrate TR_{comm2}). We implement a Python daemon that checks the middlebox's log files for alerts. Upon a modification, the daemon performs a hypercall to encrypt the new data prior to sending it to the controller which decrypts the data using a hypercall to its trusted agent (integrated similar to the security policy).

SDN Controller and Example Middleboxes: Realizing our prototype system required extending the OpenDayLight controller (adding approximately 2k lines of code). This includes adding functionality for remotely configuring the Dockerized middleboxes (leveraging the Docker API), sending flow rules to that included our added actions (as these are not a part of the OpenFlow protocol), and integrating remote attestation of middleboxes. Additionally, we realize example middleboxes running commodity network functions: (1) Snort IPS to block known vulnerabilities, (2) `iptables` as a firewall, and (3) Squid authenticating HTTP proxy to add authentication for devices with default credentials.

4.8 Security Evaluation

We analyze the security of JETFIRE's design along four axes: model-based validation of our design (Section 4.8.1), the architecture's robustness to attacks from the SDN literature (Section 4.8.2), validation of our implementation using synthetic attacks (Section 4.8.3), and measurement of the increase of the micro-hypervisor's TCB (Section 4.8.4).

4.8.1 Validating JETFIRE's Design

We build a formal model of our software-defined security gateway to specify our trust properties (4.4). We describe this model and then evaluate it using bounded model checking to validate our design is secure by construction.

4.8.1.1 Formal Model Description

We specify our software-defined gateway model using the Alloy modeling language[85]. We briefly introduce Alloy before describing our model.

Alloy Modeling Language: Alloy models are defined using first-order, relational logic. At its core, the Alloy language is an easy to use but expressive logic based on the notion of relations, and was inspired by the Z specification language and Tarski's relational calculus [85]. The Alloy model is compiled into a scope-bounded satisfiability problem and analyzed by off-the-shelf SAT solvers. We use this analysis to identify *counter examples to constraints* and verify our trust properties.

Software-defined Gateway Model: JETFIRE's software-defined security gateway model consists of a centralized controller and a set of gateways that process packets to and from devices. For brevity, we discuss an example architecture with a single gateway to explain our abridged Alloy model in Listing 1.

1. Controller and Gateway. We first model two key entities: a *Controller* and a *Gateway* using Alloy's `sig` interface (lines 1-10). A `sig`, or signature, defines a set (i.e., Controller) and its relationship to other sets (i.e., each Controller has one Policy, line 2). The controller maintains the security policy, and the control applications use the control channel to configure each gateway based on the policy. Each gateway runs one vSwitch and a set of middleboxes. The gateway receives commands over the control channel for instantiating middleboxes and installing paths in the vSwitch. Each path specifies which middlebox a specific device's traffic should be routed through.

2. ProcessPkt. We model how the gateway processes packets using Alloy's `function` interface (lines 11-18). A `function` evaluates a series of statements and returns all possible solutions. A packet received by the gateway is sent to the vSwitch for routing. The vSwitch routes the packet to the specific middlebox (line 12). Then the middlebox processes the packet and determines if the packet is benign or malicious (line 13). Benign packets are routed back to the switch and sent to the device while all other packets are dropped.

Listing 1 Abridged formal model of JETFIRE's trusted software-defined security gateway architecture.

```
 1: sig $Controller$ {
 2:     policy : $one\ Policy$,
 3:     apps: $set\ Application$,
 4:     controlchannel : $one\ Channel$
 5:
 6: sig $Gateway$ {
 7:     vswitch : $one\ vSwitch$,
 8:     mbox : $set\ Middlebox$,
 9:     controlchannel : $one\ Channel$
10:
11: function PROCESSPKT($pkt : Packet,\ g : Gateway$)
12:     $g.mbox_i$ = ROUTEPKT($pkt, g.vswitch$)
13:     $pkt.state$ = MIDDLEBOXPROCESS($pkt, g.mbox_i$)
14:     if $pkt.state$ == Benign then
15:         $pkt.action$ = Allow
16:     else
17:         $pkt.action$ = Drop
18:     return $pkt.action$
19:
20: pred TRUSTEDGATEWAY($g : Gateway,\ c : Controller$)
21:     TAMPERPROOF($c.policy$)
22:     SECURECHANNEL($g.controlchannel, c.controlchannel$)
23:     REMOTEATTEST($c.apps$)
24:     REMOTEATTEST($g.vswitch$)
25:     for $g.mbox_i$ in $c.policy$ do
26:         REMOTEATTEST($g.mbox_i$)
27:         AUTHENTICATEROUTE($g.vswitch, g.mbox_i$)
28:
29: assert PROCESSPKTCORRECTLY($g : Gateway,\ pkt : Packet$)
30:     TRUSTEDGATEWAY($g$)
31:     $pkt \in BenignPkts \implies$ PROCESSPKT($pkt, g$) == Allow
32:     $pkt \in MaliciousPkts \implies$ PROCESSPKT($pkt, g$) == Drop
```

3. TrustedGateway. Next, we define a trusted gateway (Eq 4.2) using Alloy's `pred` interface (lines 20-27). A `pred`, or predicate, evaluates a series of constraints. It returns true only if all the constraints are met and false otherwise. Thus, the following conditions must all be met for a gateway to be trusted. First, an attacker must not be able to tamper with the policy on the controller (line 21, Eq 4.3). Second, the control channel between the controller and the gateway must be secure so that it is immune to an attacker injecting malicious messages (line 22, Eq 4.6). Third, the correct software must be running on the controller, vSwitch, and middlebox (lines 23-26, Eq 4.4). Finally, each packet must follow the path specified by the controller. This must be enforced by the vSwitch and each middlebox (line 27, Eq 4.5). If all of these conditions are true then the gateway is trusted.

4. ProcessPktCorrectly. Finally, we define our goal (Eq 4.1) that all output packets were processed correctly using Alloy's `assert` interface (lines 29-32). In Alloy, an `assert` claims that a series of statements must be true based upon the model, and will generate a counter example if any of the claims do not hold to be true. A trusted gateway achieves the goal of allowing all benign packets while dropping all malicious packets.

4.8.1.2 Model Evaluation

We analyzed our system model up to a bound of 100 (i.e., 100 instances of each `sig`) and were unable to identify a counter example resulting in the model outputting a packet processed by an incorrect middlebox. Additionally, we systematically removed constraints related to our trust requirements (e.g., middlebox code does not need to be attested, violating TR_{sw2}) and verified that each resulted in a counter example that violated our overarching trust property. This analysis provides confidence in our our system's design and trust requirements.

Our Alloy model aided in identifying nuances and helped us refine our design. The model highlighted the need to prohibit packets from completely skipping a middlebox. For example, if a middlebox signs input and output packets with the same key it allows a packet to bypass the middlebox without being detected. Similarly, our model highlighted software components that either needed to be trusted or be regularly attested in order to trust the system's operation (e.g., the controller software). Next, we extend this model to evaluate JETFIRE's applicability beyond this use case, and look at securing broader SDN architectures.

Listing 2 Example Alloy analysis of prior attacks, search for ability of attacker to modify a controller application's state.

```
1: pred CANMODAPPSTATE $(c : Controller,\ a : Apps)$
2:     $a$ in $c.apps \ \wedge\ a.state == \text{Exposed}$
3: assert ATTACKERNOTMODTRUSTAPPSTATE$(c : Controller,\ a : Apps)$
4:     TRUSTCONTROLLER$(c)$
5:     $a.state == \text{DataCapsule} \implies$ CANMODAPPSTATE$(c,\ a) == \text{False}$
```

Attack Type	Example Attack	Our Defense	Mitigates
(a) Controller Application	A1: Manipulate controller's state [225]	Data Capsule + vTPM	✓
	A2: Manipulate controller's operations [106, 208, 225]		✓
	A3: Manipulate command or variable [106, 172]	Data Capsule	✓
(b) Control Channel	A4: Sniff messages [106]	Trusted Agent	✓
	A5: Inject messages [54, 106, 208, 217, 225]		✓
	A6: Modify messages [106, 208, 225]		✓
(c) Gateway Application	A7: Subvert middlebox execution [146, 193]	Data Capsule + vTPM	✓
	A8: Manipulate command or variable [106, 172]	Data Capsule	✓
(d) Data Channel	A9: Modify packet path [121]	Packet Signing	✓
	A10: Modify packet data [121]		✓

Table 4.6: JETFIRE's mitigation of known SDN attacks.

4.8.2 Robustness to Prior Attacks on Similar Architectures

We further evaluated JETFIRE's system model against 10 representative attacks from the SDN security literature (summarized in Table 4.6) [54, 106, 121, 208, 217, 225]. To identify if our system could protect against these attacks, we extend our Alloy model. Listing 2 provides an example extension for checking if a controller application's software state can be modified (the full model of all attacks can be found in [47]). This example verifies an attacker cannot modify an application's state if it is protected by a data capsule. The attacks we analyze fall into the following four groupings based upon the attack's target: (a) controller applications, (b) control channel, (c) gateway applications, and (d) data channel. We discuss each type of attack below.

(a) Controller Application Attacks: An attacker compromising the controller or a controller application could alter its state or operations (e.g., controller's global network view [225]). JETFIRE can defend against this type of attack by placing the critical data (e.g., security policy in Section 4.5.2.1) into a data capsule, and use a vTPM to attest other pieces.

(b) Control Channel Attacks: Attacks could tamper with an established control channel between the controller and gateway by injecting malicious flow rules into the vSwitch [225]. Our trusted agent (Section 4.5.2.4) on the controller and each gateway can prevent this type of attack by using an encrypted and authenticated channel. Further, JETFIRE can mitigate an attacker on the gateway/controller from accessing the secret keys and sending malicious messages from a compromised host.

(c) Gateway Application Attacks: Attackers could attack applications running on the gateway including middleboxes and the vSwitch. For example, an attack could change the vSwitch's routing rules or modify a middlebox's binary [146]. These will result in incorrectly processing network traffic (e.g., disabling a firewall's drop action). As discussed in Section 4.5.2.2, a combination of vTPM attestation and data capsule isolation can be used to detect such attacks.

(d) Data Channel Attacks: Attackers that have tampered with the OS can modify a packet's processing path or its data, such as bypassing a middlebox or modifying a packet payload. These could result in incorrect gateway operations (e.g., allow a malicious packet the firewall should have dropped). JETFIRE's per-packet signing mechanism (Section 4.5.2.3) can detect such data channel modifications.

This analysis implies our architecture's applicability beyond software-defined security gateways and could be used for securing a wider array of SDN-based architectures. Future extensions such as confidential storage, controller DoS protections, and topology verification could provide additional guarantees against prior attacks.

4.8.3 Synthetic Attacks on the Prototype

Beyond analyzing our system model, we generated synthetic attacks to validate that our prototype implementation provided each of our trust requirements (Section 4.4.2). We discuss each below.

(a) Rogue security policy modification (testing TR_{sw1})**:** We simulate an attacker with local access to the controller attempting to modify the controller's security policy (i.e., loading new values). We verified that the micro-hypervisor access mediation (via code white listing) denies this process access to the security policy (as only the security policy application has access), and that the security policy remains unchanged.

(b) Booting a modified middlebox image (testing TR_{sw2})**:** We start a modified middlebox to simulate an attacker tampering with a middlebox. The controller detects this misconfiguration (based upon the PCR quote it received) within 10 seconds of the middlebox booting.

(c) Malicious control channel injection (testing TR_{comm1})**:** We inject false middlebox alert messages over the control channel to simulate an attacker attempting to change the middlebox on the gateway. As these messages did not go through the trusted agent, the messages were dropped by the controller for failing authentication.

(d) Send packets on wrong path (testing TR_{comm2})**:** To simulate an attacker sending packets to the wrong middlebox, we sign packets with the wrong key (to generate an invalid signature). These malicious packets were injected both before and after the middlebox processes the packet to demonstrate both OVS and the middlebox drop these invalid packets.

These validation tests gave us confidence that our implementation achieves our trust requirements. A more robust guarantee about our implementation could be achieved using code verification; we leave this to future work.

4.8.4 TCB of Micro-hypervisor and Extensions

Recall that one of our challenges from Section 4.3.4 is to achieve our trust properties while keeping a small TCB. Our baseline is the uberXMHF micro-hypervisor used in our implementation, which itself has a small TCB (5544 source lines [199]) and has been formally verified [200, 201]. As shown in Table 4.7, we add three main hypervisor extensions; each extension was implemented in less than 200 source lines of code. All of them add a total increase of 6.6% of TCB size. This keeps the micro-hypervisor code base amenable to (future) formal verification as demonstrated by uberXMHF's x86 verification [201].

Hypervisor Extension	Lines of code	Percent increase
Data capsule	195	3.5%
Packet signing	70	1.3%
Trusted agent	102	1.8%
All extensions	367	6.6%

Table 4.7: The impact of Jetfire's extensions on TCB size.

4.9 Performance Evaluation

4.9.1 Scalability

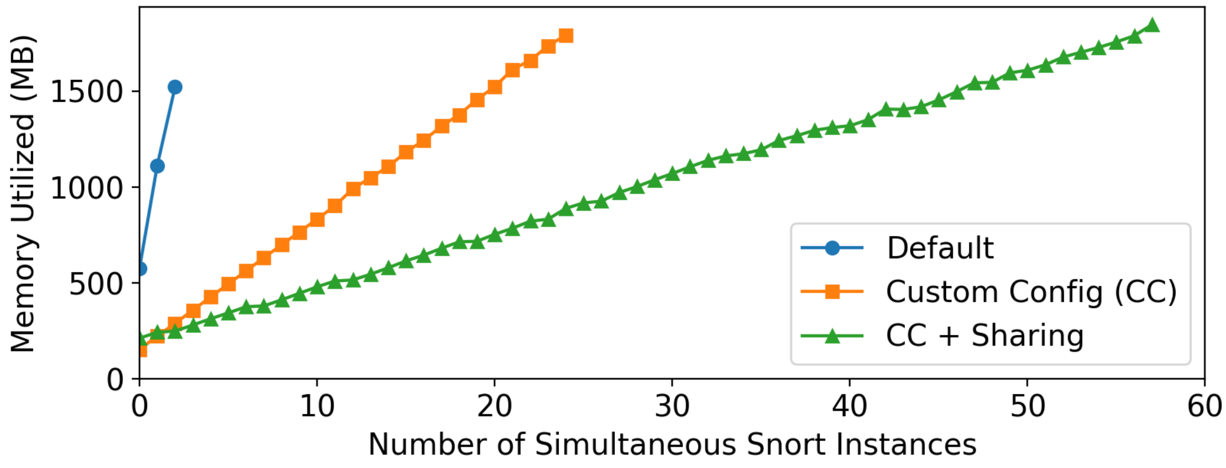
For our architecture to be deployable in many settings (e.g., smart homes, factories), a single hardware gateway needs to support small deployments (<20 devices, Section 4.3.1). This scalability is highly dependent upon the middlebox being used. We utilized the most widely deployed IPS, Snort [45], as we anticipate each device requiring this security functionality.

As shown in Figure 4.7a, applying the optimizations discussed in Section 4.9.1, we achieved a 19x increase in the number of simultaneous Snort instances. In particular, custom configurations (CC) enabled an 8x increase (24 instances, average 69.8 MB/instance), and utilizing both custom configurations and shared memory enabled an additional 2.4x increase (57 instances, average 36.1 MB/instance).

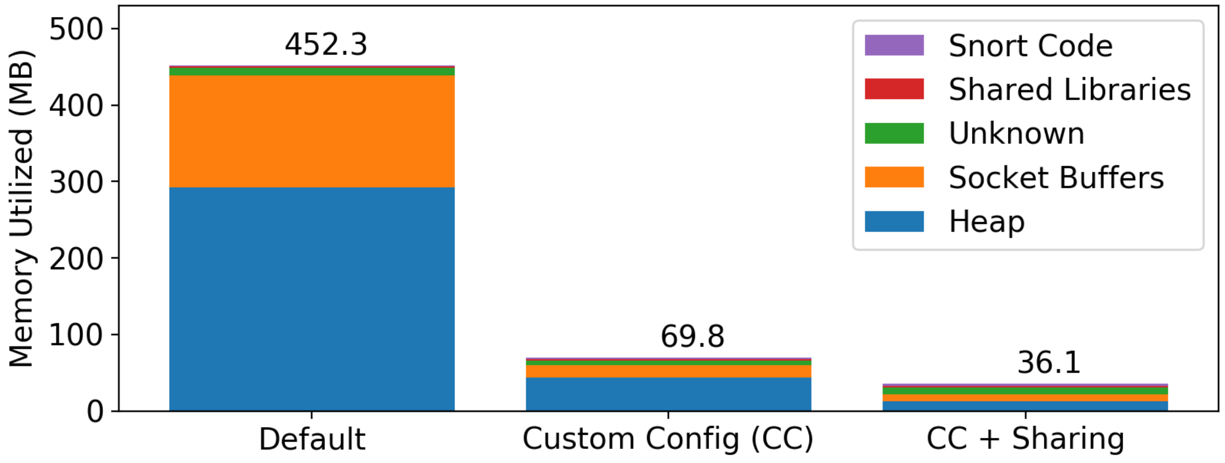
We noted a minimal impact on per-packet latency (with sharing reducing latency) that these scalability optimizations had on HTTP GET requests. As anticipated, the reduced configurations had similar latency. Moving the signature rules into shared memory reduced the median latency by 4.3 milliseconds (52% reduction). We hypothesize that this latency reduction is from the shared memory not being evicted from the cache during context switches.

4.9.2 Packet Processing Throughput

To protect data packets, Jetfire uses a lightweight signing mechanism to add signatures for each packet (Section 4.5.2.3) in vSwitch. In this experiment, we evaluate its throughput impact. As



(a) Simultaneous Snort instances realized for each optimization.



(b) Virtual memory profiling of single Snort instance.

Figure 4.7: Scalability evaluation of the number of simultaneous Snort instances on a Raspberry Pi 3B after optimizations.

shown Figure 4.8, our baseline is ‘OVS kern’, which runs the original OVS kernel module for routing without involving any extra overhead. First, we compare our kernel signing (‘Sign-kern’) with IPsec tunnels, both without protection. We noted our packet signing provided an additional 18% throughput for full MTU, and 19% for 256 Byte packets. Second, we compare our hypervisor *protected* packet signing (‘Hyp-Sign’) with an alternative approach that performs signing in an enclave. As noted earlier, trusted enclaves can only support user space applications, thus attempting to enable signing for vSwitch in an enclave would require operation in user space. We use OVS packet signing in user space (‘OVS user’) to emulate this approach. While

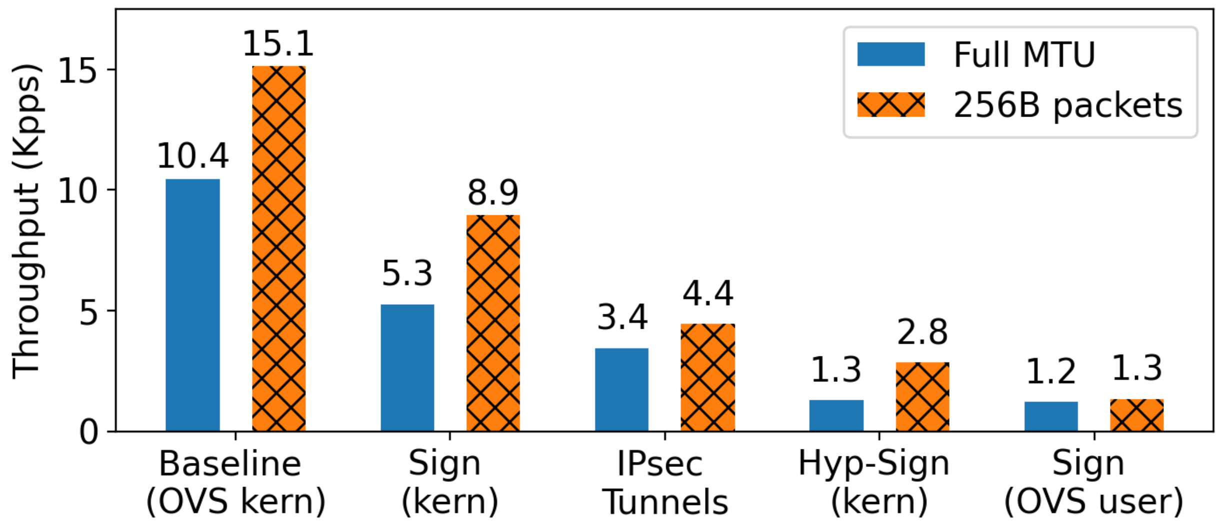


Figure 4.8: Packet signing impact on median packets per second (pps) throughput for user and kernel vSwitch.

this comparison favors our approximation of a secure enclave as the real enclave implementation would add extra overhead (e.g., memory copying), our approach still outperforms this situation, particularly for smaller sized packets.

To further understand the impacts from the micro-hypervisor protected signing, we microbenchmarked this operation. We noted that the packet signing overhead is ∼1 millisecond (544 μseconds for HMAC and 519 μseconds for hypervisor call).

Impact on Real Deployment: We measured the median time for sending a 1 MB file to 3D printer B (which utilized an HTTP interface for sending and receiving files) and noted an increase of 765 milliseconds, which is less than the processing time required to generate the file (1.54 seconds). This deployment shows that JETFIRE can provide strong security protections without impacting the normal use of the machine.

4.10 Summary

JETFIRE addresses a fundamental question for future manufacturing deployments: *How can we create a foundation for trustworthy gateway architectures to retrofit security onto manufacturing* *deployments with potentially insecure devices?* In designing JETFIRE, we tackled key challenges in providing practical foundations for trust and ensuring scalable yet low-cost capabilities for fine-grained security postures. JETFIRE is trustworthy by construction and is backed by a formal validation of its design and interfaces. Our evaluation shows that JETFIRE can serve as the basis for a low-cost, deployable, and trustworthy foundation for future software-defined security gateways. Using JETFIRE, we can support deployments with 50+ devices where each has a customized IPS (Snort) module running in a single Raspberry Pi 3B gateway.

We next discuss deploying this trusted platform to defend networked 3D printer deployments. Specifically looking at deployment considerations and integrating proof-of-concept defenses for mitigating both known vulnerabilities (such as those identified in Chapter 3) and unknown vulnerabilities.

Chapter 5

Demonstration of Fine-Grained Protections for Networked 3D Printers

In this chapter, we discuss an example walk-through of our trusted security gateway, JETFIRE, being deployed in a manufacturing space to mitigate security vulnerabilities in networked 3D printers (such as those identified in Chapter 3). Initially, we discuss hardware requirements to support a JETFIRE deployment, followed by a discussion of the baseline data that need to be collected from each machine to inform the gateway's security policy. Next, we discuss types of defense that a JETFIRE gateway can support. We realize device-specific classical network security tools (e.g., firewalls) to mitigate the known vulnerabilities, such as those identified in Section 3.5. We then show initial steps to automatically identify and patch some of these known vulnerabilities. Finally, we demonstrate the architecture's ability to mitigate potentially unknown vulnerabilities by using data-driven mechanisms that learn and then enforce network protections. Specifically we apply state-of-the-art defenses that implement access control and limit network behaviors.

Platform Architecture	**Cost**	**CPU**	**Memory**	**NIC**	**Controller Startup**	**Max Click Containers**	**Hypercall Latency**
ARM	\$35	4x 1.4GHz	1 GB	100 Mbps	261 seconds	88	0.22 msec
x86	\$200+	4x 3.2GHz	4 GB	1000 Mbps	11.4 seconds	101	162.2 msec

Table 5.1: Security gateway high-level comparison between ARM and x86 based platforms.

5.1 Hardware Requirements for a Deployment

A local deployment of JETFIRE requires both a controller and at least one gateway. Chapter 4 discussed a prototype running both components on local, low-cost ARM hardware. While such low-cost platforms are ideal for deployments needing multiple gateways, migrating the controller to an x86 platform poses benefits from increased capabilities and the potential for cloud hosting the controller. We envision migrating the controller to be cloud hosted, such that deployments might only require adding local hardware gateways further reducing the cost of deployment. While such a configuration does create the potential for increased latency and availability impacts, there is a low probability that they will significantly impact gateway operations as the gateway is designed to enforce its current protections even without a connection to the controller (as discussed in Section 4.1.2). The design of JETFIRE is amenable to this migration because all of the software components are architecture agnostic.

We compare two example hardware platforms in Table 5.1, an ARM platform (Raspberry Pi 3B) which comes at a lower cost, and an x86 platform that provides increased performance. The total number of gateway platforms required for a deployment is dependent upon the number of IP-based networked manufacturing machines being protected. To provide an upper bound on the maximum number of manufacturing machines a single gateway can protect, we set the maximum number of Click[1] containers that can run simultaneously. However, some network functions (e.g., Snort IDS discussed in Section 4.6) may require additional memory and create different scalability limitations.

We note these different hardware platforms provide different constraints. For processor and

[1]Click [97] is an open-source software framework that can be used to realize many network functions.

memory intensive operations, such as starting up the controller, the x86 platform provides a significant performance benefit. The higher cost of a hypercall is acceptable on the controller where hypercalls are not occurring for each packet, whereas the ARM platform provides more performant hypercalls needed to minimize increases to per packet latency while providing scalability in the number of middleboxes it can support simultaneously.

5.2 Device Network Operations Data Collection

With the gateway hardware identified, next a security policy needs to be generated to enforce the needed device-specific network protections. We begin by discussing how a manual audit of the machines can be used to generate this policy and then look at how these protections might be specified in a more automated manner.

In order to identify the specifics for each device's required security policy, we first collect some baseline data about the device's benign network operations. Additionally, we can use the results of any security evaluations to further inform these device-specific policy items. Chapter 3 discussed analyzing a set of deployed networked 3D printers to collect such baseline data. The needed baseline data consists of:

- *Network Capture:* A network capture of the machine under benign operating conditions, which can provide insights about items such as if encryption is used, network services, and hosts it interacts with. In Section 5.4 we discuss how these data can also be used to generate behavioral protections.

- *Targeted Scans:* Specific vulnerabilities can be identified using targeted scans, such as those in C3PO for detecting susceptibility to known exploits and DoS conditions. Similarly, sending fuzzed inputs can potentially identify additional malicious inputs.

We will use these baseline data, collected for 13 commercial networked 3D printers (in Section 3.3.2), as an example to guide the identification of needed network security protections that the security policy should specify.

Vulnerability	Network Function	Memory per Instance	Per-packet Latency*
Send plaintext	VPN server	2.74 MB	3.3 msec
Unused services	Firewall (FW)	1.89 MB	1.9 msec
DoS Susceptible	Connection limiting FW		
Malicious inputs	IPS	17.2 MB	4.3 msec
Combination	FW + VPN + IPS	25.7 MB	5.8 msec

*The baseline per-packet latency (with no NFs) is 1.0 msec.

Table 5.2: Networked 3D printer vulnerabilities and their associated network function mitigation with performance parameters for the Raspberry Pi 3B.

To deploy JETFIRE, a security policy must be generated for a given network deployment, which specifies the device-specific protections needed to mitigate any identified vulnerabilities. We begin by detailing how the manually identified protections discussed in Section 3.5 can be realized on JETFIRE, followed by a discussion about initial steps to automate this process.

5.3 Mitigating Known Vulnerabilities

Recall the networked 3D printer-specific vulnerabilities identified in Section 3.3.2: lack of encryption, susceptibility to DoS, unused network services, and malicious inputs. We discuss each of these categories and identify existing network functions that can be applied to preclude an attacker from leveraging these vulnerabilities. Furthermore, we discuss deployment considerations such as memory requirements and the increased network latency from each protection (summarized in Table 5.2). We begin by discussing mitigations for each vulnerability individually (Section 5.3.1) then how these can be chained together to protect against a set of vulnerabilities on a given machines (Section 5.3.2).

5.3.1 Single Vulnerability Mitigation

Considering the vulnerabilities identified by C3PO in Chapter 3, we highlight three classic network security solutions that are able to mitigate the majority of security vulnerabilities identified. These network functions are a virtual private network (VPN) for encryption and authentication, a firewall (FW) for blocking access to unnecessarily exposed services and to limit each host's number of simultaneous connections, and an intrusion prevention system (IPS) to block known exploits and perform limited input filtering. We discuss each of these in further detail in the following sections.

Implementation: We implement each of these classic network security solutions as a Docker container [47]. Each docker container was configured to have two virtual network interfaces and run an existing open source network security tool (e.g., `iptables` for a firewall). The network security tool was configured to bridge the two virtual interfaces and analyze all packets on the bridge. To promote generality for a given docker image, the security tool was setup to configure itself based upon a provided configuration file. Each container required a baseline set of features to support the JETFIRE functionality of signing/verifying packets and sending messages to the controller. To sign and verify packets, each container needed to have `NFQUEUE` installed along with our sign and verify callbacks. Finally, each container also had a python daemon to take outputs (e.g., log entries for IPS rule violations) and send them to the controller using the trusted agent. This setup allowed for new network functions to be quickly realized by simply installing the corresponding tool in a baseline Docker image with the required JETFIRE components. Each of these docker images can used to process a device's network traffic by specifying it in the security policy.

5.3.1.1 VPN to Encrypt Data

All networked 3D printers surveyed did not encrypt all data sent over the network (with two possibly encrypting the design files, but not metadata sent along with these files). A lack of encryption coupled with limited authentication allows an attacker to steal data using a man-in-the-middle attack. Furthermore, limited integrity checks allow for the data to be stealthily modified such that the printed part is defective. The ideal way to fix such vulnerabilities is by using end-to-end encryption; however, this can only be done by modifying the networked 3D printer's software.

VPN: To encrypt the data between the control PC and the gateway, we create a VPN tunnel. A VPN has the added benefit of also providing host authentication. JETFIRE uses `openVPN` to serve as a VPN server. In our system, control PCs authenticate themselves to the gateway and send encrypted data using pre-shared secret keys. The VPN server requires little memory per instance, on average just 2.74 MB. A Raspberry Pi 3B can handle a maximum of 457 simultaneous instances. However, the encryption operations come at an average latency cost of 3.3 milliseconds per packet on the Raspberry Pi 3B. Further, these encryption operations reduce the maximum throughput to 6 Mbps.

5.3.1.2 Firewall to Block Unused Services and DoS

Nearly half of the networked 3D printers surveyed (6 of 13) exposed unused network services. Several of these unused services had known vulnerabilities or were leveraged in prior attacks. As these services are unused by the networked 3D printer during benign operations, all network traffic to these services can simply be dropped without effecting the 3D printer's operations. Therefore, a simple firewall can reduce these networked 3D printers' attack surface.

Additionally, all analyzed networked 3D printers were vulnerable to DoS attacks. While there were multiple attack strategies that could result in the 3D printer being unavailable, most had a common requirement of the attacker starting multiple TCP connections. Standard operations require at most two simultaneous TCP connections from any host; however, many networked 3D printers allowed a single host to create over 1,000 simultaneous connections. To increase a networked 3D printer's robustness to DoS attacks, the gateway can limit each host's TCP connections thereby requiring an attacker to compromise multiple devices to launch a successful DoS attack.

Firewall (FW): We use the `iptables` firewall to limit both the hosts that can access the networked 3D printer (i.e., only control PCs can send network packets to the 3D printer), and additionally limit each host's number of simultaneous TCP connections. A connection-limiting firewall mitigates an attacker who has compromised a single host on the network from being able to successfully launch a DoS attack on all but two of the networked 3D printers we analyzed. These two networked 3D printers only allow a single TCP connection, which if acquired by an attacker allows them to deny access to legitimate users. Such a FW requires little memory per instance, on average just 1.89 MB. On a Raspberry Pi 3B this allows for a maximum of 138 simultaneous instances, where the Raspberry Pi 3B can process packets through such a firewall with an average latency of 1.9 milliseconds per packet, with a maximum throughput of 15.2 Mbps.

5.3.1.3 Intrusion Prevention System to Block Malicious Inputs

We identified multiple types of malicious inputs, some through fuzzing and others through known exploit signatures. Some of these vulnerabilities can be mitigated by checking for a signature within the packets without impacting benign traffic. However, this is a limited solution, as slight changes to the malicious input may not be detected, allowing attacks to succeed.

Intrusion Prevention System (IPS): We used snort [45] to serve as an IPS and block packets matching known malicious signatures. Signatures for known attacks can be gathered from public repositories, and some malicious inputs can be identified using regular expression matching. The IPS can be configured to require an average of 17.2 MB of memory per instance. On a Raspberry Pi 3B this allows for a maximum of 76 simultaneous instances. Processing packets with an average latency of 4.3 milliseconds per packet, with a maximum throughput of 2.47 Mbps.

5.3.2 Defending Deployed 3D Printers from Known Vulnerabilities

As most networked 3D printers analyzed had multiple vulnerabilities, they require multiple network functions to mitigate all of the vulnerabilities identified. Protection can be achieved by having a service chain or a series of network functions that process each packet. For example,

Device	Vulnerabilities	Service Chain
OctoPi	replay DoS	FW(conn. limiting)
Machine A	plaintext data, SYN flood, malicious inputs	VPN + FW + IPS
Machine B	plaintext data, SYN flood & replay DoS, unused services, known exploits, malicious inputs	VPN + FW + IPS
Machine C	plaintext data, malicious inputs	VPN + IPS
Machine D	plaintext metadata, TCP connection DoS	VPN + FW(conn. limiting)
Machine E	plaintext data, replay DoS	VPN + FW(conn. limiting)
Machine F	plaintext data, SYN flood & slowloris DoS, unused services	VPN + FW
Machine G	plaintext data, SYN flood & slowloris DoS	VPN + FW(conn. limiting)
Machine H	plaintext data, SYN flood & slowloris DoS	VPN + FW(conn. limiting)
Machine I	plaintext data, SYN flood & TCP connection DoS, unused services	VPN + FW
Machine J	plaintext data, SYN flood & TCP connection DoS, unused services, known exploits	VPN + FW + IPS
Machine K	SYN flood, unused services, known exploits	FW + IPS
Machine L	plaintext data, slowloris DoS	VPN + FW(conn. limiting)
Machine M	plaintext metadata, partial data transfer DoS, unused services, known exploits, malicious inputs	VPN + FW + IPS + unique file name check

Table 5.3: Potential network service chains to protect networked 3D printers analyzed by C3PO.

a networked 3D printer (such as Machine B) exposes plaintext data, is vulnerable to DoS and malicious inputs. A gateway would need to deploy a service chain composed of a FW, VPN, and an IPS to mitigate all of these vulnerabilities. Such a service chain would require an average of 25.7 MB of memory per chain (i.e., per networked 3D printer being protected). On a Raspberry Pi 3B this allows for a maximum of 35 networked 3D printers per hardware gateway, processing packets with an average latency of 5.8 milliseconds per packet, with a maximum throughput of 2.77 Mbps.

Based upon our analysis in Chapter 3, we discuss the specific chains required by each of the networked 3D printers surveyed. In general, we noted that most require a FW and a VPN.

5.3.3 Automatically Mitigating Network Vulnerabilities

It is unrealistic for small manufacturing shops to manually configure a network security gateway to mitigate their networked 3D printer's vulnerabilities. Proper configuration requires multiple manual steps. First, security analysis tools such as C3PO must be used to identify vulnerabilities. Second, these vulnerabilities must be mapped to a network function which mitigates the vulnerability. Correctly mapping identified vulnerabilities to network functions requires an expertise which is not guaranteed to be available at small manufacturing centers. Thus, it is impractical for a manufacturing center to manually create its own security policy.

An ideal solution is to have the gateway automatically scan each device and deploy the appropriate network function automatically. We prototype such functionality by using running pieces of C3PO's individual analysis (e.g., open port scan, known vulnerability scan) and automatically deploying a network function that mitigates any identified vulnerabilities. We implement this as a two-stage policy. First, an active scanning stage which runs a detection tool for identifying potential vulnerabilities. Second, the controller configures and deploys the appropriate network function based upon the vulnerabilities detected. We describe two examples below:

- *Unused Network Services:* Needed network services (ports) are defined as a part of the policy to mitigate potential false positives if example network data do not utilize all necessary network ports (e.g., infrequently used port for software updates). We use `nmap` to identify all running services on the device. Any ports not listed in the needed ports are given to the firewall configuration to block access to these unused ports.
- *Known Exploits:* Similar to unused network services, we use `nmap` to scan a networked 3D printer for known vulnerabilities. If one is identified, we check a local database for signatures of this attack. These signatures are added to an IPS's configuration file, and deployed to block these attacks.

Implementation: Incorporating this functionality required two extensions to the controller. First, the controller had to be extended to account for middleboxes that were actively sending packets to a network device, where previously the controller assumed all middleboxes were passive. These middleboxes only had a single network interface, and expected the controller to provide the target's IP address upon its initialization. Second, the controller's API for handling messages from a middlebox had to be extended to parse the scanning tools message (e.g., nmap ports reported) and convert this to a middlebox configuration (e.g., `iptables` rules). Each of these automated tools required approximately two man-weeks implement the extensions.

Unfortunately, these demonstrations of automated protections are limited. First, they often require some domain/device knowledge (e.g., used ports). Second, they only provide incomplete protections and cannot directly identify and mitigate all of the findings gathered by C3PO. For example, findings from fuzzing often cause the 3D printing application/firmware to crash. These crashes often require manual intervention to recover the networked 3D printer. Furthermore, the cause of the crash is not always clear. Both of these make it hard to automate the mitigation logic. In order to provide similar automated configuration and protection, we look to alternative approaches for protecting networked 3D printers from unknown vulnerabilities by limiting network host's with access and network behaviors.

5.4 Mitigating Potentially Unknown Vulnerabilities

Signature-based network protections are inherently limited, as an attacker can often modify the attack to use a polymorphic variant that is not detected by the signature. Behavioral-based network anomaly detection is robust to these as it detects changes in network behavior as opposed to looking for specific signatures. Networked 3D printers have a tractable set of behavioral profiles because they perform a limited set of operations. We demonstrate how JETFIRE can be used to enforce behavioral network profiles: (1) manufacturer's usage description (MUD) [105] specifications, and (2) inferred finite state machine (FSM) network interaction model. We discuss each below in turn.

5.4.1 Enforcing Access Control Using MUD Specifications

MUD is a standard describing networked devices, including their intended communication patterns [105]. These allow a device manufacturer to specify the network operations a device performs. Unfortunately, few devices currently have published MUD specifications. To emulate this type of network behavior enforcement, we use tools to generate example MUD policies from a device's network traffic [69, 70].

We generate example MUD specifications using sample network traffic from the networked 3D printers we surveyed in Chapter 3 using the MUDgee tool [69]. Most networked 3D printers had similar results of allowing local traffic to the port it received print jobs on, with some also contacting a vendor's public server for software updates.

Implementation: We extend the JETFIRE controller to take these MUD specifications as an input and configure the appropriate firewall rules to enforce the MUD specification. To realize this, we first implement a middlebox that forwards the traffic, while also making a local copy. This local copy is then processed using the MUDgee tool to generate a MUD specification. The controller's API is extended to parse the received MUD specification and convert it into firewall rules. We utilize the FSM policy abstraction proposed by Yu *et al.* [227] to specify this type of dynamic security policy on JETFIRE's controller. Where the initial state is to learn the MUD specification and the second state is to enforce it with a firewall. Where the transition is triggered once the controller receives a MUD specification.

The MUD specification is primarily concerned with specifying access control (i.e., limiting which network hosts that can interact with a device), and does not specify additional aspects such as limiting the number of concurrent connections or modeling benign network behavior. Thus, if a compromised host is allowed to send network messages to a networked 3D printer, these MUD specifications may still allow an attack to succeed.

5.4.2 Enforcing Benign Behaviors using FSM Models

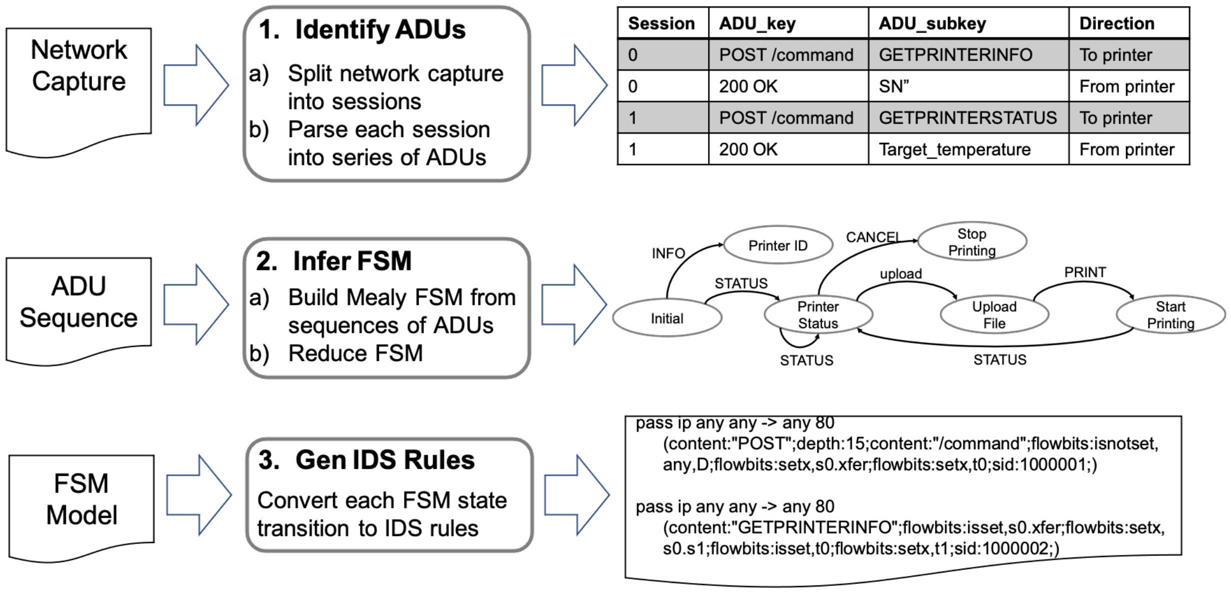
As networked 3D printers only perform a limited set of network operations (e.g., receive a file, send status), they are amenable to having their behaviors completely defined. Yu *et al.* [230] advocate for using FSMs to model their network behaviors. We discuss a prototype application of these FSM-based behavioral protections below.

5.4.2.1 Developing FSM Models

We leverage prior work by Yu *et al.* [230] to build FSM models using a device's plaintext network traffic. Their approach determines behaviors by looking at the series of Application-layer Data Units (ADUs) within each TCP session. They define an ADU as a chunk of application data (potentially spanning multiple packets) in one direction that is specific to the execution of a single task. The collected sequences of ADUs are used to define a Mealy FSM, where the next state is determined based upon the current state and its input (i.e., the previous ADUs). The resulting FSM defines a 3D printer's network behavior. Behavioral based network traffic limitations, in the form of IPS rules, can be derived from the FSM.

We extend this prior work in order to generate FSM behavioral models [47] for nine of the networked 3D printers surveyed in Chapter 3. We selected these nine as they represent 5 distinct network protocols (the remaining protocols were not 3D printer specific, such as sending commands over telnet). To learn these FSMs, we follow the workflow shown in Figure 5.1. First, the Bro/Zeek IDS [233] parses a network capture into sessions and a vendor specific parsing script identifies ADUs. Second, the FSM is inferred using the sequence of ADUs in each session and is reduced to its smallest state representation.

Identifying ADUs: As many of the networked 3D printers surveyed utilized proprietary network protocols, existing parsing tools were insufficient to extract detailed ADUs. Based upon manual analysis of these network captures, we added vendor-specific parsing scripts to extract ADUs from these protocols. These ADUs consisted of an ADU$_{key}$, which can contain multiple



Session	ADU_key	ADU_subkey	Direction
0	POST /command	GETPRINTERINFO	To printer
0	200 OK	SN"	From printer
1	POST /command	GETPRINTERSTATUS	To printer
1	200 OK	Target_temperature	From printer

```
pass ip any any -> any 80
    (content:"POST";depth:15;content:"/command";flowbits:isnotset,
    any,D;flowbits:setx,s0.xfer;flowbits:setx,t0;sid:1000001;)

pass ip any any -> any 80
    (content:"GETPRINTERINFO";flowbits:isset,s0.xfer;flowbits:setx,
    s0.s1;flowbits:isset,t0;flowbits:setx,t1;sid:1000002;)
```

Figure 5.1: Workflow for generating behavioral model enforcement rules from network capture, with example outputs for each stage of processing.

ADU$_{subkey}$ to provide finer resolution on the behavior.[2] For example, a networked 3D printer that sent data over HTTP had an ADU of: `POST /command GETPRINTERINFO`, where "POST /command" is the ADU$_{key}$ and the ADU$_{subkey}$ is "GETPRINTERINFO". The subsequent ADU provided the networked 3D printer's serial number. Without the ADU$_{subkey}$ an ADU of: `POST /command GETPRINTERSTATUS` would be part of the same state, despite it having a different reply message containing twice the amount of data. Beyond simply collecting a string representing the ADU, we also collect metadata about the ADU such as its offset within a packet and the packet's length, as well as the direction (i.e., to or from the networked 3D printer). A portion of an example ADU sequence is shown in Figure 5.1.

In implementing the vendor-specific ADU parsers, the predominant difference between them was identifying a key to signify the start of a potential ADU, where some networked 3D printer's ADUs might be encoded in a JSON format while others could be noted by specific packet sizes (e.g., all commands are in 64 Byte packets). These vendor-specific parsers in combination with

[2]We added the ADU$_{subkey}$ as without them the generated network behavior models had states that encompassed divergent actions.

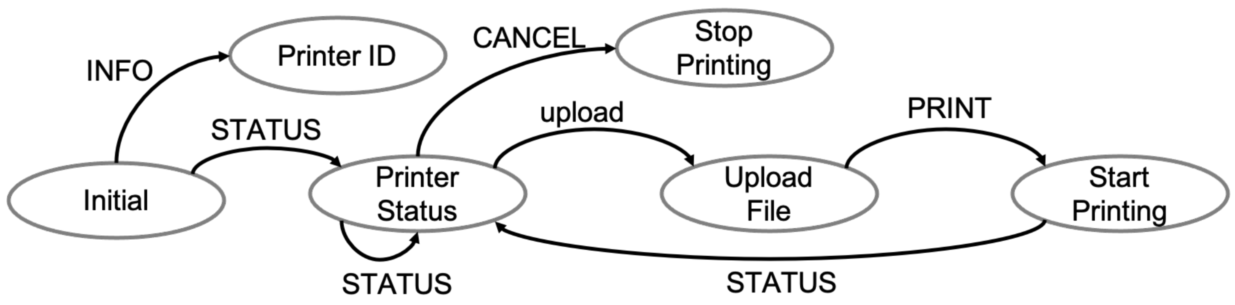


Figure 5.2: Example networked 3D printer behavioral FSM for Machine B. Showing states in ovals with the ADUs as arrows connecting states.

the Bro/Zeek network security monitoring tool generate a log file containing the ADU sequences for each session in a given network capture.

Inferring FSMs: We infer a Mealy FSM model of the network behaviors based upon an input set of TCP sessions, where each session contains a series of ADUs. We assume all ADUs are positive samples (i.e., the network capture did not contain any attack traffic), and build an initial FSM tree. In general this tree consists of repeated sets of requests and responses (e.g., a request for the networked 3D printer's status followed by the 3D printer's response). Next, the initial tree is reduced by creating loops to account for repeated ADU sequences (e.g., periodic status requests). The reduced model is output, with the ADUs specifying the transitions between each state (an example behavioral FSM for Machine B is shown in Figure 5.2). Table 5.4 provides details about the number of states and unique ADUs for the nine networked 3D printers analyzed.

Next, we use these models to configure network protections that limit a networked 3D printer's network traffic to the behavior defined in the FSM model.

5.4.2.2 Model-based Network Protections

We used the snort IPS [45] to enforce a networked 3D printer's FSM models on live network traffic. Specifically, we used snort's `flowbits` to implement the FSM logic. However, as snort's inspection is on a per-packet level, we utilized a second set of flowbits to track ADUs that require multiple packets. We compile the FSM model into a set of snort rules that realize the

Device	FSM Details		Network Capture		Snort Config	
	States	ADUs	Duration	Size	Rules	Flowbits
Machine A	21	14	174 seconds	40 MB	56	47
Machine B	7	9	205 seconds	9.7 MB	47	47
Machine D	10	4	20 seconds	10 MB	38	23
Machine E	10	8	61 seconds	3 MB	60	26
Machine F	22	17	1 second	505 KB	44	32
Machine G	9	10	2 seconds	839 KB	24	37
Machine H	16	13	1 second	1.5 MB	31	26
Machine L	12	11	1 second	4.4 MB	28	24
Machine M	2	3	9 seconds	780 KB	47	23

Table 5.4: Details about the FSM behavioral models and their Snort IPS configurations for 9 of the networked 3D printer evaluated in Chapter 3.

behavioral model specified in the generated FSM.

FSM to IPS Rules: Traditionally, an IPS takes an action, such as dropping a packet, upon detecting a specified signature. We inverted this operation to realize our network protections, by configuring the IPS to only allow packets containing the correct ADU keys to be allowed. Packets not matching these constraints are dropped. We extend our controller to take an input model and realize the corresponding network protections. Table 5.4 details the number of rules and flowbits required to realize these network protections with a snort middlebox for the networked 3D printers analyzed.

Online Learning: While most learning is performed off-line, we prototyped having JETFIRE learn a networked 3D printer's behavioral model online and automatically implement the corresponding behavioral protections. We assumed that the initial network traffic from a networked 3D printer is benign (i.e., it does not start out compromised) and that the initial traffic was sufficiently representative of the networked 3D printer's behavior. JETFIRE initially deployed a pass-through middlebox that allowed all network traffic while also collecting a network capture of this traffic. The network capture was then processed to generate an FSM which informed the configuration an IPS that was deployed and replaced the pass-through middlebox.

Implementation: We implemented the behavioral protections as a dynamic policy with two states [47]. The first learns the behavior, by deploying a pass-through middlbox. A copy of all network traffic is collected, and analyzed with Bro/Zeek [233] with our custom networked 3D printer ADU parsers. A Java program parses these logs to infer the networked 3D printer's behavioral FSM model, which is sent to the controller. We decided to send the FSM model due to its small size and to provide future flexibility (e.g., utilize a different IPS). We extended the controller's API to parse the FSM model and convert it into the corresponding snort IPS rules. The controller then replaces the pass-through middlebox with this snort IPS middlebox which only allows network traffic conforming to the learned FSM model. These protections are amenable to the five protocols analyzed, extending it to cover a new protocol would require adding a new ADU parser (as a Bro/Zeek script).

5.5 Vision for Deployment Defenses

The prior sections discussed three complementary defenses: mitigating known vulnerabilities, limiting access control, and limiting network behaviors. While these were shown in isolation, ideally they could be combined to provide a robust network defense. Where upon a device initially connecting a middlebox would be deployed that scanned the device for known vulnerabilities and subsequently collect a network capture of the device's benign behavior to support learning both access control and a behavioral model. After collecting this information, the gateway would deploy middleboxes to mitigate any known defenses and augment these with a firewall enforcing the access control limits and an IPS limiting network behaviors.

5.6 Summary

This chapter demonstrated how multiple, complimentary defenses can be deployed on top of JETFIRE. We demonstrated running the system on multiple hardware platforms (x86 and ARM) as groundwork for enabling the controller to be run by a bare metal cloud-hosting service to reduce the hardware required for a deployment. Next, we demonstrated integrating defenses onto the JETFIRE platform. We realized the network protections to mitigate known vulnerabilities (e.g., those identified by C3PO in Chapter 3), chaining multiple middleboxes to mitigate the vulnerabilities identified by C3PO in deployed networked 3D printers. We the demonstrate initial steps towards automating these defenses by scanning for specific vulnerabilities and deploying appropriately configured network security tools. We further augment these defenses by integrating state-of-the-art tools that learn from network traffic in order to reduce the potential for unknown vulnerabilities to be exploited. Specifically, we demonstrate learning MUD specifications to enforce access control and FSM-based behavioral models to limit network behaviors. These complementary defenses demonstrate the diversity of defenses that JETFIRE can support and can be used together to provide a robust, device-specific network defense.

Chapter 6

Conclusions

In this chapter, we summarize our contributions and discuss their potential impact on the manufacturing domain. We discuss limitations of our proposed solution and conclude by identifying key future research directions.

6.1 Contributions and Impact

We make two overarching contributions: (1) a systematic measurement of network vulnerabilities in networked 3D printers and their deployments and (2) a low-cost, trusted security gateway system for defending small to medium sized deployments.

Security Evaluation of Networked 3D Printers: Our tool, C3PO, analyzed 13 networked 3D printers and 5 network deployments to provide a snapshot of the current state of 3D printers' network security. These security measurements noted that all 13 networked 3D printers analyzed were vulnerable to simple DoS attacks (e.g., SYN flood), most (12 of 13) did not encrypt data in transit, and some (4 of 13) allowed network inputs that crashed the machine. Further, many network deployments did not isolate networked 3D printers, unnecessarily placing them on publicly accessible networks with multiple embedded devices. These findings provide concrete data confirming anecdotal evidence that manufacturing centers have network vulnerabilities.

After disclosing our findings with all of the networked 3D printer vendors, many are implementing our recommendations and some have requested additional analysis of their new 3D printers to improve their product's security. Additionally, our tool has been requested by manufacturing center administrators and used to understand and improve their security posture.

Low-cost, Trusted Security Gateway: Our system, JETFIRE, demonstrates the ability to provide a low-cost, trusted solution to patch real network vulnerabilities in deployed 3D printers. We use model-driven analysis to show that it provides security by construction, guaranteeing that all output packets are processed by the correct middlebox. We achieve this by building on top of a micro-hypervisor and integrating fine-grained protections into the packet processing software. We further optimize the system to provide performant and scalable network-level defenses.

We envision these bolt-on security solutions being used to protect deployed networked 3D printers and other networked devices, mitigating their security vulnerabilities without impacting standard user interactions. These solutions provide a practical solution to an urgent problem, and allowing advanced manufacturing centers to fully utilize the advantages of Industry 4.0 while minimizing their network security risks.

6.2 Limitations

We now examine the limitations of our proposed solutions.

Limitation #1: IP-based Network Communication. Our tools are limited to analyzing and protecting devices that operate on IP-based networks. While all networked 3D printers we analyzed used this network protocol, expanding this work to defend other manufacturing devices is limited as some devices use different wireless networking protocols (e.g., BLE, Zigbee, etc.).

Limitation #2: Operational Impacts. Our analysis tool, C3PO, leverages existing tools to probe for network vulnerabilities. These tools are intrusive and can impact the machine's operations (e.g., crashing the firmware); there are reports of some legacy devices suffering permanent failures when scanned with these tools. Thus our scanning tool may negatively impact some legacy devices ability to operate. Similarly, our bolt-on security gateway increases network latency. While networked 3D printers are robust to this slight increase, some manufacturing machines (e.g., using real-time closed-loop control over the network) might become unusable and need an alternative, low-latency security solution.

Limitation #3: Encrypted Data. Similar to many current network security solutions, JETFIRE does not identify attacks sent over an encrypted channel (e.g., known exploits transmitted over a TLS channel), but can potentially identify anomalous behavior in the meta-data (e.g., sudden large traffic volume in a DoS attack). Complementary works show promise for analyzing encrypted traffic [134, 214]. Such tools require a trusted foundation, as they gain access to the data and could modify it without being detected.

Limitation #4: Middleboxes Mitigating All Vulnerabilities. Our analysis assumed that middleboxes mitigated all of a networked 3D printer's vulnerabilities. It is unlikely such middleboxes will be realized in practice (e.g., zero-day vulnerabilities). Additionally, the mapping of identified vulnerabilities to middlebox protection is currently an imprecise operation. We demonstrated performing this manually and for a subset of vulnerabilities. Additionally, our analysis tool (C3PO) does not guarantee identifying all vulnerabilities, particularly those in the printing application (our fuzzing is limited based upon the input network capture). Complementary tools such as source code analysis could provide more complete coverage of potential vulnerabilities related to the 3D printer's firmware [35, 182]. Finally, JETFIRE's protection of a middlebox's source code from modification is limited to detecting changes to binary executables (using attestation). This does not mitigate all attacks (e.g., return-oriented programming attacks [160]).

Limitation #5: Required Operator Effort. The JETFIRE defenses currently assume a knowledgeable administrator generates the security policy. This complexity limits the ability of a manufacturing center to properly configure a security gateway to ensure the needed defenses are realized. Additionally, it currently lacks a user interface describing the current middleboxes deployed on the gateway. Related efforts have developed prototype interfaces to aid operators understanding the gateway's current configuration [109].

Limitation #6: Deployment Complexity. This thesis demonstrated protecting small to medium sized deployments using a single hardware gateway. The techniques and architecture is designed to be able to scale and support multiple gateways. However, the impacts of realizing such scaling as well as questions such as device mobility pose potential challenges to directly utilizing JETFIRE to defend more complex deployments.

6.3 Future Work

Our ultimate vision is to secure advanced manufacturing deployments from network attacks. Our work in this dissertation has laid some steps towards achieving our goal. To this end, we now identify future research directions:

- A trusted security gateway provides a foundational building block for building **advanced security capabilities** (Section 6.3.1).
- Our security gateway is a pragmatic solution. Additional steps could **enhance its deployability** (Section 6.3.2).

6.3.1 Advanced Security Capabilities

Automated Network Defenses: We demonstrated an initial example of automating network patches in Chapter 5. This could be further generalized to be able to scan for and identify additional types of network vulnerabilities and implement an appropriate mitigation. Identifying some of these vulnerabilities becomes challenging as some tools such as fuzzing often detect vulnerabilities by inducing a crashing condition which might require manual intervention to reconnect to the network. Additionally, identifying the part of the input that caused the crash is not always straightforward. Further, such automated patching needs to address challenges related to updates, identifying new vulnerabilities and removing unneeded network protections after applying software patches to a device.

More Robust Online Learning: We demonstrated applying state-of-the-art defenses that learned from network traffic. These demonstrations utilized a single print job transfer for their learning, additional network samples might be required to reduce false positives and negatives. Additionally, as some network behaviors might not occur naturally during the initial training window expanding these tools to allow for learning updates while rejecting potentially malicious traffic would increase the robustness of such tools.

Active Deception: As network adversaries continue to evolve and advance, it becomes more important to gather information about the attacker. Traditional deception techniques, such as static honeypots, can fail to gain sufficient information about an attacker. Recent studies in cybersecurity and artificial intelligence have shown a great potential of cyber deception strategies in thwarting cyber attacks effectively [30, 92]. Our trusted gateway (Chapter 4) provides a trusted platform for realizing deceptive cyber artifacts (e.g., honeypots, mock sensors, fake services) that defenders can trust to only confuse attackers and thereby reduce an attack's effectiveness without being used by the attacker to deceive the defender.

Federated Machine Learning: As demonstrated by Yu *et al.* [228], data about device operations collected by multiple security gateways can be used to detect malicious behaviors. Such privacy preserving learning is ideal for the manufacturing domain where different manufacturing centers want to maintain their proprietary data while also having a resilience to malicious usage. Our trusted security gateway could be used to collect needed contextual information about device operations, and similar approaches could be used to provide trust in the centralized machine learning agent.

Leverage Physical World Characteristics: This thesis focused on network traffic based defenses. As manufacturing machines are interacting with the physical world, there is the potential for augmenting these defenses by incorporating physical world data (e.g., motor voltage [129], vibrations and acoustics [18]). Such data could be collected by adding sensors [104] which if correlated with the network data might aid in identifying malicious behaviors (e.g., a 3D printer deviating from a part's design).

6.3.2 Enhanced Gateway Deployability

Running Micro-hypervisor on a Bare-Metal Cloud Hosting Service: Our prototype of JETFIRE utilized a local controller. Ideally, the controller could be hosted by a cloud service. Currently, the micro-hypervisor requires knowledge of the hardware platform to specify parameters and modification of the bootloader. Some of this knowledge and access is not always provided by cloud hosting services. Additionally, the micro-hypervisor would likely need to run on top of the cloud service's hypervisor. This integration challenges some of our system's trust assumptions and requires additional integration between the different virtualization layers.

Generalizable Across Different Micro-hypervisors: We utilized uberXMHF [198, 199] for our prototype of JETFIRE because of its amenability to formal verification. Our system design is generalizable and could be realized on alternate micro-hypervisor platforms such as NOVA [192] which also support ARM and x86. Similarly, the small TCB could be traded for a more widespread hypervisor (e.g., XEN [42]); however, such a large code base potentially brings new vulnerabilities.

Evaluation of Operational Impacts: In implementing JETFIRE, we performed benchmark evaluations to characterize the system's impact on a networked 3D printer. These insights could be improved by evaluating the impact of a security gateway on a real-world manufacturing environment. Such a deployment could aid in identifying operational issues.

Increased Scalability: In Chapter 4, we discuss an approach for implementing lightweight middleboxes. This approach could be applied to additional types of middleboxes. Additionally, the gateway's scalability could be further increased by employing checkpoint restore in userspace to save inactive middleboxes and thereby allow for increased scalability. A naive approach for checkpointing middleboxes would be based upon time since last use, but alternate approaches might provide better memory usage.

Support Additional Wireless Network Protocols: Our work focused on IP-based network traffic; however, many manufacturing devices utilize other wireless network protocols (e.g., BLE, Zigbee, etc.). Our approach could likely be applied to these alternate protocols. However, protocol specific challenges must be addressed. Additionally, integrating data from devices using various protocols may provide additional insights and capabilities.

6.4 Closing Remarks

This thesis provides a measurement of network security vulnerabilities in networked 3D printers and demonstrates a low-cost, trusted system to mitigate these vulnerabilities. We hope our findings will aid vendors in developing more secure devices and manufacturing centers in defending their deployed machines. We hope this work inspires other to further security efforts in the manufacturing domain.

Bibliography

[1] ABDUL-GHANI, H. A., KONSTANTAS, D., AND MAHYOUB, M. A comprehensive iot attacks survey based on a building-blocked reference model. *IJACSA) International Journal of Advanced Computer Science and Applications 9*, 3 (2018), 355–373. 16

[2] ACKERMAN, P. *Industrial Cybersecurity*. Packt, 2017. 49

[3] AL FARUQUE, M. A., CHHETRI, S. R., CANEDO, A., AND WAN, J. Acoustic side-channel attacks on additive manufacturing systems. In *2016 ACM/IEEE 7th international conference on Cyber-Physical Systems (ICCPS)* (2016), IEEE, pp. 1–10. 13

[4] AL FARUQUE, M. A., CHHETRI, S. R., CANEDO, A., AND WAN, J. Forensics of thermal side-channel in additive manufacturing systems. *University of California, Irvine* (2016). 12, 13

[5] AL-SHAER, E., ET AL. FlowChecker: Configuration Analysis and Verification of Federated OpenFlow Infrastructures. In *SafeConfig* (2010). 22

[6] ALBAKRI, M., STURM, L., WILLIAMS, C. B., AND TARAZAGA, P. Non-destructive evaluation of additively manufactured parts via impedance-based monitoring. In *Solid Freeform Fabrication Symposium* (2015), vol. 26, pp. 1475–1490. 14

[7] ALI, I., SABIR, S., AND ULLAH, Z. Internet of things security, device authentication and access control: a review. *arXiv preprint arXiv:1901.07309* (2019). 16

[8] ALRAWI, O., LEVER, C., ANTONAKAKIS, M., AND MONROSE, F. Sok: Security eval-

uation of home-based iot deployments. In *2019 IEEE S&P* (2019). 10, 16, 17

[9] AMMAR, M., CRISPO, B., JACOBS, B., HUGHES, D., AND DANIELS, W. Sμv - the security microvisor: A formally-verified software-based security architecture for the internet of things. *IEEE Trans. Dependable Sec. Comput. 16*, 5 (2019). 6, 22, 70

[10] ANTONAKAKIS, M., APRIL, T., BAILEY, M., BERNHARD, M., BURSZTEIN, E., COCHRAN, J., DURUMERIC, Z., HALDERMAN, J. A., INVERNIZZI, L., KALLITSIS, M., ET AL. Understanding the mirai botnet. In *USENIX Security 17* (Vancouver, BC, 2017), USENIX Association. 16, 25, 43, 47, 49, 50

[11] ARM. Trustzone - arm developer. `https://developer.arm.com/ip-products/security-ip/trustzone`, 2020. 21, 61

[12] ARTHUR, W., CHALLENER, D., AND GOLDMAN, K. *A Practical Guide to TPM 2.0: Using the New Trusted Platform Module in the New Age of Security*. Apress, Berkeley, CA, 2015, ch. History of the TPM, pp. 1–5. 21, 22, 61, 72

[13] AUFNER, P. The iot security gap: a look down into the valley between threat models and their implementation. *International Journal of Information Security 19*, 1 (2020). 16

[14] AVOLIO, F. Firewalls and internet security, the second hundred (internet) years. *The Internet Protocol Journal 2*, 2 (1999), 24–32. 18

[15] AXELSSON, S. Research in intrusion-detection systems: A survey. Tech. rep., Citeseer, 1998. 18

[16] BALLETTI, C., BALLARIN, M., AND GUERRA, F. 3d printing: State of the art and future perspectives. *Journal of Cultural Heritage 26* (2017), 172 – 182. 10

[17] BARRERA, D., MOLLOY, I., AND HUANG, H. Standardizing iot network security policy enforcement. In *DISS 2018* (2018). 19, 20, 21, 54, 57, 58, 61

[18] BAYENS, C., LE, T., GARCIA, L., BEYAH, R., JAVANMARD, M., AND ZONOUZ, S. See no evil, hear no evil, feel no evil, print no evil? malicious fill patterns detection

in additive manufacturing. In *26th USENIX Security Symposium (USENIX Security 17)* (2017), pp. 1181–1198. 12, 13, 114

[19] BELDEN. Goodbye air gaps, hello improved ics security. `https://www.belden.com/blog/industrial-security/Goodbye-Air-Gaps-Hello-Improved-ICS-Security`, 2016. Accessed on 2019-02-14. 14

[20] BELIKOVETSKY, S., SOLEWICZ, Y., YAMPOLSKIY, M., TOH, J., AND ELOVICI, Y. Detecting cyber-physical attacks in additive manufacturing using digital audio signing. *arXiv preprint arXiv:1705.06454* (2017). 12, 14

[21] BELIKOVETSKY, S., YAMPOLSKIY, M., TOH, J., GATLIN, J., AND ELOVICI, Y. dr0wned – cyber-physical attack with additive manufacturing. In *11th USENIX Workshop on Offensive Technology (WOOT 17)* (Vancouver, BC, 2017), USENIX Association. 2, 4, 12, 13, 14, 24, 25, 51

[22] BIONDI, P. Scapy. `https://scapy.net`, 2020. 28

[23] Bit defender box 2. `https://www.bitdefender.com/box/`, 2018. 19, 57

[24] BOECKL, K., BOECKL, K., FAGAN, M., FISHER, W., LEFKOVITZ, N., MEGAS, K. N., NADEAU, E., O'ROURKE, D. G., PICCARRETA, B., AND SCARFONE, K. *Considerations for managing Internet of Things (IoT) cybersecurity and privacy risks*. US Department of Commerce, National Institute of Standards and Technology, 2019. 15

[25] BRANDL, D. Why manufacturing software should be tested before updates. `https://www.controleng.com/articles/why-manufacturing-software-should-be-tested-before-updates/`, 2016. Accessed: 2020-10-24. 4, 18

[26] BUTUN, I., ÖSTERBERG, P., AND SONG, H. Security of the internet of things: Vulnerabilities, attacks, and countermeasures. *IEEE Communications Surveys & Tutorials 22*, 1 (2019), 616–644. 16

[27] BYRES, E. The air gap: Scada's enduring security myth. *Communications of the ACM 56*, 8 (2013), 29–31. 14

[28] C3po. `https://github.com/3DPrinter-Security/C3PO`, 2019. 6, 25, 28, 29

[29] CAMPBELL, T., WILLIAMS, C., IVANOVA, O., AND GARRETT, B. Could 3d printing change the world? `https://www.atlanticcouncil.org/images/files/publication_pdfs/403/101711_ACUS_3DPrinting.PDF`, 2011. Accessed: 2019-09-10. 2, 10

[30] CARROLL, T. E., AND GROSU, D. A game theoretic investigation of deception in network security. *Security and Communication Networks 4*, 10 (2011), 1162–1172. 113

[31] CASADO, M., FREEDMAN, M. J., PETTIT, J., LUO, J., MCKEOWN, N., AND SHENKER, S. Ethane: Taking control of the enterprise. In *Proceedings of the 2007 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications* (New York, NY, USA, 2007), SIGCOMM '07, Association for Computing Machinery, p. 1–12. 19

[32] CASELLI, M., ZAMBON, E., AMANN, J., SOMMER, R., AND KARGL, F. Specification mining for intrusion detection in networked control systems. In *25th USENIX Security Symposium (USENIX Security 16)* (2016), pp. 791–806. 16

[33] CENEDESE, A., ZANELLA, A., VANGELISTA, L., AND ZORZI, M. Padova smart city: An urban internet of things experimentation. In *Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014* (2014), pp. 1–6. 62, 65

[34] Censys. `https://censys.io`, 2019. Accessed on 2019-02-25. 51

[35] CHA, S. K., AVGERINOS, T., REBERT, A., AND BRUMLEY, D. Unleashing mayhem on binary code. In *2012 IEEE Symposium on Security and Privacy* (2012), IEEE, pp. 380–394. 111

[36] CHEN, C., MANIATIS, P., PERRIG, A., VASUDEVAN, A., AND SEKAR, V. Towards verifiable resource accounting for outsourced computation. In *ACM VEE* (2013). 70

[37] CHEN, J., DIAO, W., ZHAO, Q., ZUO, C., LIN, Z., WANG, X., LAU, W. C., SUN, M., YANG, R., AND ZHANG, K. Iotfuzzer: Discovering memory corruptions in iot through app-based fuzzing. In *NDSS* (2018). 32

[38] CHEN, K., ZHANG, S., LI, Z., ZHANG, Y., DENG, Q., RAY, S., AND JIN, Y. Internet-of-things security and vulnerabilities: Taxonomy, challenges, and practice. *Journal of Hardware and Systems Security 2*, 2 (2018), 97–110. 16

[39] CHHETRI, S. R., CANEDO, A., AND AL FARUQUE, M. A. Kcad: kinetic cyber-attack detection method for cyber-physical additive manufacturing systems. In *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)* (2016), IEEE, pp. 1–8. 14

[40] CHHETRI, S. R., FAEZI, S., AND FARUQUE, M. A. A. Fix the leak! an information leakage aware secured cyber-physical manufacturing system. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2017* (March 2017), pp. 1408–1413. 12, 14

[41] CHI, P.-W., KUO, C.-T., GUO, J.-W., AND LEI, C.-L. How to Detect a Compromised SDN Switch. In *NetSoft* (2015), IEEE. 22

[42] CHISNALL, D. *The definitive guide to the xen hypervisor*. Pearson Education, 2008. 114

[43] CIMPANU, C. Researcher: Backdoor mechanism still active in many iot products. `https://www.zdnet.com/article/researcher-backdoor-mechanism-still-active-in-many-iot-products/`, 2020. 20

[44] CISCO. Cisco 2018 annual cybersecurity report. `https://www.cisco.com/c/dam/m/hu_hu/campaigns/security-hub/pdf/acr-2018.pdf`, 2018. Accessed: 2019-09-06. 2

[45] CISCO. Snort. `https://www.snort.org`, 2019. 76, 88, 98, 105

[46] CLOUDFLARE. What is a slowloris ddos attack. `https://www.cloudflare.com/`

learning/ddos/ddos-attack-tools/slowloris/, 2019. Accessed on 2019-02-10. 31, 41

[47] Jetfire code. https://github.com/slab14/IoT_Sec_Gateway, 2020. 6, 57, 78, 85, 96, 103, 107

[48] COKER, G., GUTTMAN, J., LOSCOCCO, P., HERZOG, A., MILLEN, J., O'HANLON, B., RAMSDELL, J., SEGALL, A., SHEEHY, J., AND SNIFFEN, B. Principles of remote attestation. *International Journal of Information Security 10*, 2 (2011). 72

[49] COMI, L. Iot-securitychecker. https://github.com/c0mix/IoT-SecurityChecker, 2018. Accessed: 2019-05-15. 26

[50] CREATIVETOOLS. #3dbenchy - the jolly 3d printing torture-test by creativetools.se. https://www.thingiverse.com/thing:763622, 2015. Accessed: 2019-02-01. 33

[51] Cujo. https://www.getcujo.com, 2018. Accessed: 2018-03-23. 19, 57

[52] DAY, J., SHEPHERD, R., KEARNEY, P., AND STORER, R. Secure design best practices guidelines. https://www.iotsecurityfoundation.org/wp-content/uploads/2019/03/Best-Practice-Guides-Release-1.2.1.pdf, 2018. Accessed: 2019-05-02. ix, 14, 15, 26, 27

[53] DEOGIRIKAR, J., AND VIDHATE, A. Security attacks in iot: A survey. In *2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud)(I-SMAC)* (2017), IEEE, pp. 32–37. 16

[54] DHAWAN, M., PODDAR, R., MAHAJAN, K., AND MANN, V. Sphinx: Detecting security attacks in software-defined networks. In *Ndss* (2015), vol. 15, pp. 8–11. 22, 65, 68, 85

[55] DIEBER, B., BREILING, B., TAURER, S., KACIANKA, S., RASS, S., AND SCHARTNER, P. Security for the robot operating system. *Robotics and Autonomous Systems 98* (2017), 192–203. 16

[56] DINABURG, A., ROYAL, P., SHARIF, M., AND LEE, W. Ether: malware analysis via hardware virtualization extensions. In *Proc. of CCS* (2008). 70

[57] DO, Q., MARTINI, B., AND CHOO, K. R. A data exfiltration and remote exploitation attack on consumer 3d printers. *IEEE Transactions on Information Forensics and Security 11*, 10 (Oct 2016), 2174–2186. 2, 12, 13, 16, 24

[58] DOBRESCU, M., AND ARGYRAKI, K. Software dataplane verification. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)* (2014), pp. 101–114. 22

[59] DORASWAMY, N., AND HARKINS, D. *IPSec: the new security standard for the Internet, intranets, and virtual private networks*. Prentice Hall Professional, 2003. 18, 61

[60] DRAGOS. Manufacturing threat perspective. `https://www.dragos.com/resource/manufacturing-threat-perspective/`, 2020. 2

[61] ERUMBAN, A. A. Lifetimes of machinery and equipment: Evidence from dutch manufacturing. *Review of Income and Wealth 54*, 2 (2008), 237–268. 3

[62] ETSI. Etsi ts 103 645: Cyber security for consumer internet of things: Baseline requirements. `https://www.etsi.org/deliver/etsi_ts/103600_103699/103645/02.01.02_60/ts_103645v020102p.pdf`, 2020. Accessed: 2020-10-25. 15

[63] FAGAN, M., MEGAS, K. N., SCARFONE, K., AND SMITH, M. Foundational cybersecurity activities for iot device manufacturers. Tech. rep., National Institute of Standards and Technology, 2020. 15

[64] FARRIS, I., TALEB, T., KHETTAB, Y., AND SONG, J. A survey on emerging sdn and nfv security mechanisms for iot systems. *IEEE Communications Surveys & Tutorials 21*, 1 (2018), 812–837. 19

[65] FATTORI, A., PALEARI, R., MARTIGNONI, L., AND MONGA, M. Dynamic and trans-

parent analysis of commodity production systems. In *Proc. of IEEE/ACM ASE 2010* (2010). 70

[66] FERNANDES, E., JUNG, J., AND PRAKASH, A. Security analysis of emerging smart home applications. In *2016 IEEE symposium on security and privacy (SP)* (2016), IEEE, pp. 636–654. 16

[67] GMBH, G. N. Openvas - open vulnerability assessment system. `http://openvas.org`, 2019? Accessed: 2019-04-03. 44

[68] GUPTA, N., CHEN, F., AND SHAHIN, K. Design features to address security challenges in additive manufacturing. In *Manufacturing Techniques for Materials*. CRC Press, 2018, pp. 23–50. 12, 14

[69] HAMZA, A. Mudgee. `https://github.com/ayyoob/mudgee`, 2019. 102

[70] HAMZA, A., RANATHUNGA, D., GHARAKHEILI, H. H., ROUGHAN, M., AND SIVARAMAN, V. Clear as mud: generating, validating and applying iot behavioral profiles. In *Proceedings of the 2018 Workshop on IoT Security and Privacy* (2018), pp. 8–14. 102

[71] HANG, X. Z. Security attack to 3d printing, 2013. Keynote at XCon2013. 12, 24

[72] HASSIJA, V., CHAMOLA, V., SAXENA, V., JAIN, D., GOYAL, P., AND SIKDAR, B. A survey on iot security: application areas, security threats, and solution architectures. *IEEE Access 7* (2019), 82721–82743. 16

[73] HEBERLEIN, L. T., DIAS, G. V., LEVITT, K. N., MUKHERJEE, B., WOOD, J., AND WOLBER, D. A network security monitor. Tech. rep., Lawrence Livermore National Lab., CA (USA); California Univ., Davis, CA (USA . . . , 1989. 18

[74] HELIN, A. Radamsa-a general purpose fuzzer. *URl: https://gitlab. com/akihe/radamsa* (2018). 32

[75] HERMANN, M., PENTEK, T., AND OTTO, B. Design principles for industrie 4.0 scenarios. In *2016 49th Hawaii International Conference on System Sciences (HICSS)* (Jan

2016), pp. 3928–3937. 1, 54

[76] HERWIG, S., GARMAN, C., AND LEVIN, D. Achieving keyless cdns with conclaves. In *29th USENIX Security Symposium (USENIX Security 20)* (Aug. 2020), USENIX Association, pp. 735–751. 61

[77] HOJJATI, A., ADHIKARI, A., STRUCKMANN, K., CHOU, E., THO NGUYEN, T. N., MADAN, K., WINSLETT, M. S., GUNTER, C. A., AND KING, W. P. Leave your phone at the door: Side channels that reveal factory floor secrets. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (2016). 13

[78] HORN, A., KHERADMAND, A., AND PRASAD, M. Delta-net: Real-time network verification using atoms. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)* (2017), pp. 735–749. 22

[79] HOU, J.-U., KIM, D.-G., CHOI, S., AND LEE, H.-K. 3d print-scan resilient watermarking using a histogram-based circular shift coding structure. In *Proceedings of the 3rd ACM Workshop on Information Hiding and Multimedia Security* (2015), ACM. 12, 14

[80] HOU, J.-U., KIM, D.-G., AND LEE, H.-K. Blind 3d mesh watermarking for 3d printed model by analyzing layering artifact. *IEEE Transactions on Information Forensics and Security 12*, 11 (2017), 2712–2725. 14

[81] IEC. Iec 62443: Network and system security for industrial-process measurement and control. `https://www.isasecure.org/en-US/Documents/Authentication-Required-Specifications/EDSA-3-0-0/CSA-311-Functional-security-assessment-for-compone`, 2018. Accessed: 2018-11-18. ix, 14, 15, 26, 27

[82] IHS. Internet of things (iot) connected devices installed base worldwide from 2015 to 2025 (in billions). `www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/`, 2018. 62, 65

[83] INGHAM, K., AND FORREST, S. A history and survey of network firewalls. *University of New Mexico, Tech. Rep* (2002). 18

[84] INTEL. Intel software guard extensions: Developer guide. `https://download.01.org/intel-sgx/linux-1.7/docs/Intel_SGX_Developer_Guide.pdf`, 2016. 21, 61, 69

[85] JACKSON, D. Alloy: A new technology for software modelling. In *Tools and Algorithms for the Construction and Analysis of Systems* (Berlin, Heidelberg, 2002), J.-P. Katoen and P. Stevens, Eds., Springer Berlin Heidelberg, pp. 20–20. 81

[86] JIN, X., GOSSELS, J., REXFORD, J., AND WALKER, D. Covisor: A compositional hypervisor for software-defined networks. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)* (2015), pp. 87–101. 22, 61

[87] KANG, H., SHIN, S., YEGNESWARAN, V., GHOSH, S., AND PORRAS, P. Aegis: An automated permission generation and verification system for sdns. In *Proceedings of the 2018 Workshop on Security in Softwarized Networks: Prospects and Challenges* (2018), pp. 20–26. 21

[88] KANG, Q., XUE, L., MORRISON, A., TANG, Y., CHEN, A., AND LUO, X. Programmable in-network security for context-aware byod policies. In *Proc. USENIX Security* (2020). 19

[89] KAZEMIAN, P., ET AL. Real Time Network Policy Checking Using Header Space Analysis. In *NSDI* (2013), USENIX. 22

[90] KHAN, A., AND TUROWSKI, K. A survey of current challenges in manufacturing industry and preparation for industry 4.0. In *Proceedings of the First International Scientific Conference Intelligent Information Technologies for Industry (IITI '16)* (01 2016), vol. 1, pp. 15–26. 2

[91] KHURSHID, A., ET AL. Veriflow: Verifying network-wide Invariants in Real Time. In

NSDI (2013), USENIX. 22

[92] KIEKINTVELD, C., LISÝ, V., AND PÍBIL, R. Game-theoretic foundations for the strategic use of honeypots in network security. In *Cyber Warfare*. Springer, 2015. 113

[93] KIM, H., REICH, J., GUPTA, A., SHAHBAZ, M., FEAMSTER, N., AND CLARK, R. Kinetic: Verifiable Dynamic Network Control. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)* (2015), USENIX. 22

[94] KIM, T. H.-J., BASESCU, C., JIA, L., LEE, S. B., HU, Y.-C., AND PERRIG, A. Lightweight source authentication and path validation. In *Proceedings of the 2014 ACM Conference on SIGCOMM* (2014), SIGCOMM '14. 21, 61, 62

[95] KO, R., AND MICKENS, J. Deadbolt: Securing iot deployments. In *Proceedings of the Applied Networking Research Workshop* (2018), pp. 50–57. 4, 19, 20, 21, 54, 56, 57, 58, 61, 63

[96] KOBATA, H., AND GAGNE, R. Securing computer network communication using a proxy server, Jan. 5 2006. US Patent App. 10/766,871. 18

[97] KOHLER, E., MORRIS, R., CHEN, B., JANNOTTI, J., AND KAASHOEK, M. F. The click modular router. *ACM Trans. Comput. Syst. 18*, 3 (Aug. 2000), 263–297. 93

[98] KOLODNY, L. Elon musk emails employees about 'extensive and damaging sabotage' by employee. `https://www.cnbc.com/2018/06/18/elon-musk-email-employee-conducted-extensive-and-damaging-sabotage.html`, 2018. Accessed: 2019-04-12. 2

[99] KOVACS, E. New canon printers bring siem integration, other security features. `https://www.securityweek.com/new-canon-printers-bring-siem-integration-other-security-features`. Accessed:27 Oct 2020. 17

[100] KOVACS, E. Flaw exposes mitsubishi plcs to remote dos attacks. `https://www.securityweek.com/flaw-exposes-mitsubishi-plcs-remote-`

`dos-attacks`, 2019. Accessed: 2019-08-03. 24

[101] KRAMER, T. R., PROCTOR, F. M., AND MESSINA, E. The nist rs274ngc interpreter - version 3. Tech. Rep. NISTIR 6556, National Institute of Standards and Technology, 2000. 11

[102] KUSHNER, D. The real story of stuxnet. `https://spectrum.ieee.org/telecom/security/the-real-story-of-stuxnet`, 2013. Accessed on 2019-02-13. 50

[103] LANGNER, R. Stuxnet: Dissecting a cyberwarfare weapon. *IEEE Security Privacy 9*, 3 (May 2011), 49–51. 24

[104] LAPUT, G., ZHANG, Y., AND HARRISON, C. Synthetic sensors: Towards general-purpose sensing. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems* (2017), pp. 3986–3999. 114

[105] LEAR, E., DROMS, R., AND ROMASCANU, D. Manufacturer usage description specification. *IETF draft* (2017). 101, 102

[106] LEE, S., YOON, C., LEE, C., SHIN, S., YEGNESWARAN, V., AND PORRAS, P. A. Delta: A security assessment framework for software-defined networks. In *NDSS* (2017). 22, 60, 61, 68, 85

[107] LEGNER, M., KLENZE, T., WYSS, M., SPRENGER, C., AND PERRIG, A. Epic: Every packet is checked in the data plane of a path-aware internet. In *29th USENIX Security Symposium* (2020), USENIX Security '20. 21, 61, 62

[108] LEPINSKI, M., AND SRIRAM, K. Bgpsec protocol specification. *Draft-ietf-sidr-bgpsecprotocol* (2013). 21, 22, 61, 62, 68, 74

[109] LEWIS, G., ECHEVERRÍA, S., MAZZOTTA, C., GRABOWSKI, C., O'MEARA, K., VASUDEVAN, A., NOVAKOUSKI, M., MCCORMACK, M., AND SEKAR, V. Kalki: a software-defined iot security platform. In *IEEE Virtual World Forum on Internet of Things*

2020 (2020). 112

[110] LEWIS, J. A. Cybersecurity: Assessing the immediate threat to the united states. *Testimony Before the US House Oversight and Government Reform Committee* (2011). 3, 4, 14

[111] LI, H., HU, H., GU, G., AHN, G.-J., AND ZHANG, F. vnids: Towards elastic security with safe and efficient virtualization of network intrusion detection systems. In *Proc. of the 25th ACM Conference on Computer and Communications Security (CCS'18)* (October 2018). 19

[112] LITTY, L., LAGAR-CAVILLA, H. A., AND LIE, D. Hypervisor support for identifying covertly executing binaries. In *USENIX Security Symposium* (2008). 22, 70

[113] LLC, S. C. Armitrage - cyber attack management for metasploit. `http://www.fastandeasyhacking.com/index.html`, 2019. Accessed: 2019-04-03. 44

[114] LORINCZ, J. Cyber secure manufacturing is smart manufacturing. `https://advancedmanufacturing.org/cyber-secure-smart-manufacturing/`, 2018. Accessed: 2019-09-06. 2

[115] LYON, G. Nmap. `https://nmap.org`, 2018. Accessed: 2018-11-19. 32, 44, 48

[116] MAKERBOT. 6 things you can do with the makerbot mobile app. `https://www.makerbot.com/stories/news/makerbot-mobile-app/`, 2016. 10

[117] MARIE, C. I came to drop bombs: Auditing the compression algorithm weapons cache. `https://bomb.codes`, 2016. Accessed: 2019-05-14. 40

[118] MARTINS, J., AHMED, M., RAICIU, C., OLTEANU, V., HONDA, M., BIFULCO, R., AND HUICI, F. Clickos and the art of network function virtualization. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)* (Seattle, WA, 2014), USENIX Association, pp. 459–473. 19

[119] MATHEWS, L. Boeing is the latest wannacry ransomware victim. `https:`

`//www.forbes.com/sites/leemathews/2018/03/30/boeing-is-the-latest-wannacry-ransomware-victim/#9b1382d66344`, 2018. Accessed: 2018-11-19. 2, 44

[120] MCCORMACK, M., CHANDRASEKARAN, S., LIU, G., YU, T., WOLF, S., AND SEKAR, V. "security analysis of networked 3d printers". In *IEEE Workshop on the Internet of Safe Things* (2020). 6

[121] MCCORMACK, M., VASUDEVAN, A., LIU, G., ECHEVERRÍA, S., O'MEARA, K., LEWIS, G., AND SEKAR, V. Towards an architecture for trusted edge iot security gateways. In *3rd USENIX Workshop on Hot Topics in Edge Computing (HotEdge 20)* (June 2020), USENIX Association. 6, 57, 60, 61, 85

[122] MCCUNE, J. M., ET AL. Trustvisor: Efficient TCB reduction and attestation. In *IEEE S&P* (2010). 22, 69, 70

[123] MENEGHELLO, F., CALORE, M., ZUCCHETTO, D., POLESE, M., AND ZANELLA, A. Iot: Internet of threats? a survey of practical security vulnerabilities in real iot devices. *IEEE Internet of Things Journal 6*, 5 (2019), 8182–8201. 16

[124] MERTENS, X. 3d printers in the wild, what can go wrong? `https://isc.sans.edu/forums/diary/3D+Printers+in+The+Wild+What+Can+Go+Wrong/24044/`, 2018. Accessed: 2019-11-15. 16, 51

[125] MICROSOFT. Microsoft security bulletin ms17-010 - critical. `https://docs.microsoft.com/en-us/security-updates/securitybulletins/2017/ms17-010`, 2017. Accessed: 2018-11-19. 6, 44

[126] MICROSOFT. Guardian modules, 2020. 3, 4

[127] MIJUMBI, R., SERRAT, J., GORRICHO, J.-L., BOUTEN, N., DE TURCK, F., AND BOUTABA, R. Network function virtualization: State-of-the-art and research challenges. *IEEE Communications surveys & tutorials 18*, 1 (2015), 236–262. 19

[128] MOORE, S., ARMSTRONG, P., MCDONALD, T., AND YAMPOLSKIY, M. Vulnerability analysis of desktop 3d printer software. In *2016 Resilience Week (RWS)* (Aug 2016), pp. 46–51. 12

[129] MOORE, S. B., GATLIN, J., BELIKOVETSKY, S., YAMPOLSKIY, M., KING, W. E., AND ELOVICI, Y. Power consumption-based detection of sabotage attacks in additive manufacturing. *arXiv preprint arXiv:1709.01822* (2017). 14, 113

[130] MOORE, S. B., AND GLISSON, W. B. Implications of malicious 3 d printer firmware. In *Hawaii International Conference on System Sciences* (2016). 12, 16

[131] MÜLLER, J., MLADENOV, V., SOMOROVSKY, J., AND SCHWENK, J. Sok: Exploiting network printers. In *2017 IEEE Symposium on Security and Privacy (SP)* (May 2017), pp. 213–230. 3, 16, 17, 26

[132] NAOUS, J., WALFISH, M., NICOLOSI, A., MAZIÈRES, D., MILLER, M., AND SEEHRA, A. Verifying and enforcing network paths with icing. In *Proceedings of the Seventh COnference on Emerging Networking EXperiments and Technologies* (New York, NY, USA, 2011), CoNEXT '11, Association for Computing Machinery. 21, 22, 61, 62, 68, 74

[133] NAWIR, M., AMIR, A., YAAKOB, N., AND LYNN, O. B. Internet of things (iot): Taxonomy of security attacks. In *2016 3rd International Conference on Electronic Design (ICED)* (2016), IEEE, pp. 321–326. 16

[134] NAYLOR, D., ET AL. And then there were more: Secure communication for more than two parties. In *CoNEXT* (2017), ACM. 111

[135] NESHENKO, N., BOU-HARB, E., CRICHIGNO, J., KADDOUM, G., AND GHANI, N. Demystifying iot security: an exhaustive survey on iot vulnerabilities and a first empirical look on internet-scale iot exploitations. *IEEE Communications Surveys & Tutorials 21*, 3 (2019), 2702–2733. 16

[136] NORTHCUTT, S., AND NOVAK, J. *Network intrusion detection.* Sams Publishing, 2002.

18

[137] NSA, AND CISA. Nsa and cisa recommend immediate actions to reduce exposure across operational technologies and control systems. https://media.defense.gov/2020/Jul/23/2002462846/-1/-1/1/OT_ADVISORY-DUAL-OFFICIAL-20200722.PDF, 2020. 2

[138] OESTER, P. Linux kernel memory subsystem copy on write mechanism contains a race condition vulnerability. https://www.kb.cert.org/vuls/id/243144/, 2016. Accessed: 14 February 2020. 61

[139] O'NEILL, P. H. Russian hackers are infiltrating companies via the office printer. https://www.technologyreview.com/f/614062/russian-hackers-fancy-bear-strontium-infiltrate-iot-networks-microsoft-report/, 2019. 50

[140] OU, X., GOVINDAVAJHALA, S., AND APPEL, A. W. Mulval: A logic-based network security analyzer. In *Proceedings of the 14th Conference on USENIX Security Symposium - Volume 14* (Berkeley, CA, USA, 2005), SSYM'05, USENIX Association, pp. 8–8. 29, 47, 48

[141] OWASP. Owasp internet of things project. https://www.owasp.org/index.php/OWASP_Internet_of_Things_Project, 2018. Accessed: 2018-11-18. ix, 14, 15, 26, 27

[142] PAN, Y., WHITE, J., SCHMIDT, D. C., ELHABASHY, A., STURM, L., CAMELIO, J. A., AND WILLIAMS, C. Taxonomies for reasoning about cyber-physical attacks in iot-based manufacturing systems. *IJIMAI 4* (2017), 45–54. 9, 24

[143] PARK, M., BHARDWAJ, K., AND GAVRILOVSKA, A. Toward lighter containers for the edge. In *3rd USENIX Workshop on Hot Topics in Edge Computing (HotEdge 20)* (2020). 77

[144] PEREYDA, J. boofuzz: Network protocol fuzzing for humans. `https://boofuzz.readthedocs.io/en/latest/`, 2017. Accessed: 2019-05-08. 32

[145] PFAFF, B., PETTIT, J., KOPONEN, T., JACKSON, E., ZHOU, A., RAJAHALME, J., GROSS, J., WANG, A., STRINGER, J., SHELAR, P., AMIDON, K., AND CASADO, M. The design and implementation of open vswitch. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)* (2015), USENIX. 20

[146] PODDAR, R., LAN, C., POPA, R. A., AND RATNASAMY, S. Safebricks: Shielding network functions in the cloud. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)* (2018), pp. 201–216. 21, 60, 61, 65, 67, 69, 85, 86

[147] Polar cloud. `https://polar3d.com`, 2019. Accessed: 2019-05-06. 10

[148] POMRAMING, M. J. Injection flaws: Stop validating your input. `https://www.blackhat.com/presentations/bh-usa-05/bh-us-05-pomraning-update.pdf`, 2005. 40

[149] PORRAS, P. A., CHEUNG, S., FONG, M. W., SKINNER, K., AND YEGNESWARAN, V. Securing the software defined network control layer. In *NDSS* (2015). 21, 22, 61

[150] PORTS, D. R., AND GARFINKEL, T. Towards application security on untrusted operating systems. In *HotSec* (2008). 72

[151] PRABHU, S., CHAUDHRY, G. I., GODFREY, B., AND CAESAR, M. High-coverage testing of softwarized networks. In *Proceedings of the 2018 Workshop on Security in Softwarized Networks: Prospects and Challenges* (2018), pp. 46–52. 22

[152] PUNGALIYA, Y. Iotvulnerabilityscanner. `https://github.com/yashpungaliya/IoTVulnerabilityScanner`, 2018. Accessed: 2019-09-06. 26

[153] QUARTA, D., POGLIANI, M., POLINO, M., MAGGI, F., ZANCHETTIN, A. M., AND ZANERO, S. An experimental security analysis of an industrial robot controller. In *2017*

IEEE Symposium on Security and Privacy (SP) (May 2017), pp. 268–286. 2, 12

[154] QUIGLEY, M., CONLEY, K., GERKEY, B., FAUST, J., FOOTE, T., LEIBS, J., WHEELER, R., AND NG, A. Y. Ros: an open-source robot operating system. In *ICRA workshop on open source software* (2009), no. 3.2, Kobe, Japan, p. 5. 16

[155] QUIST, D., LIEBROCK, L., AND NEIL, J. Improving antivirus accuracy with hypervisor assisted analysis. *Journal in computer virology 7*, 2 (2011), 121–131. 22, 70

[156] RAILEANU, S., BORANGIU, T., MORARIU, O., AND IACOB, I. Edge computing in industrial iot framework for cloud-based manufacturing control. In *2018 22nd International Conference on System Theory, Control and Computing (ICSTCC)* (2018). 62, 65

[157] RAJ, H., SAROIU, S., WOLMAN, A., AIGNER, R., COX, J., ENGLAND, P., FENNER, C., KINSHUMANN, K., LOESER, J., MATTOON, D., NYSTROM, M., ROBINSON, D., SPIGER, R., THOM, S., AND WOOTEN, D. ftpm: A software-only implementation of a TPM chip. In *25th USENIX Security Symposium (USENIX Security 16)* (Austin, TX, Aug. 2016), USENIX Association, pp. 841–856. 22

[158] Rattrap. `https://www.myrattrap.com`, 2018. Accessed: 2018-03-23. 19, 57

[159] RESEARCH, I. Ibm x-force threat intelligence index 2018. `https://www.ibm.com/downloads/cas/MKJOL3DG`, 2018. Accessed on 2019-03-21. 2

[160] ROEMER, R., BUCHANAN, E., SHACHAM, H., AND SAVAGE, S. Return-oriented programming: Systems, languages, and applications. *ACM Transactions on Information and System Security (TISSEC) 15*, 1 (2012), 1–34. 111

[161] RÖPKE, C., AND HOLZ, T. Preventing malicious sdn applications from hiding adverse network manipulations. In *Proceedings of the 2018 Workshop on Security in Softwarized Networks: Prospects and Challenges* (2018), pp. 40–45. 22

[162] ROTH, V., POLAK, W., RIEFFEL, E., AND TURNER, T. Simple and effective defense against evil twin access points. In *Proceedings of the First ACM Conference on Wireless*

Network Security (New York, NY, USA, 2008), WiSec '08, Association for Computing Machinery, p. 220–235. 65

[163] RUDIS, B., WOOLWINE, W., AND LIN, K. 2020: Q2 threat report. `https://www.rapid7.com/research/report/2020Q2-threat-report/`, 2020. 2, 4, 56, 62

[164] RUSHBY, J. M., AND RANDELL, B. A distributed secure system. In *1983 IEEE Symposium on Security and Privacy* (April 1983), pp. 127–127. 69

[165] SANFILIPPO, S. hping. http://www.hping.org, 2006. Accessed: 2019-05-10. 26, 31

[166] SCHMID, R. Industrial iot: How connected things are changing manufacturing. `https://www.wired.com/wiredinsider/2018/07/industrial-iot-how-connected-things-are-changing-manufacturing/`. Accessed: 11-02-2020. 2

[167] SCHWARZ, F., AND ROSSOW, C. SENG, the sgx-enforcing network gateway: Authorizing communication from shielded clients. In *29th USENIX Security Symposium (USENIX Security 20)* (2020), USENIX Association. 3, 4, 21, 61

[168] SCOTT-HAYWARD, S. Design and Deployment of Secure, Robust, and Resilient SDN Controllers. In *NetSoft* (2015), IEEE. 21

[169] SESHADRI, A., LUK, M., QU, N., AND PERRIG, A. SecVisor: A tiny hypervisor to provide lifetime kernel code integrity for commodity OSes. In *Proc. of SOSP* (2007). 70

[170] SEZER, S., SCOTT-HAYWARD, S., CHOUHAN, P. K., FRASER, B., LAKE, D., FINNEGAN, J., VILJOEN, N., MILLER, M., AND RAO, N. Are we ready for sdn? implementation challenges for software-defined networks. *IEEE Communications Magazine 51*, 7 (2013), 36–43. 61

[171] SHARIF, M. I., LEE, W., CUI, W., AND LANZI, A. Secure in-vm monitoring using hardware virtualization. In *Proc. of CCS* (2009). 70

[172] SHIN, S., SONG, Y., LEE, T., LEE, S., CHUNG, J., PORRAS, P., YEGNESWARAN, V., NOH, J., AND KANG, B. B. Rosemary: A robust, secure, and high-performance network operating system. In *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security* (2014), pp. 78–89. 21, 61, 85

[173] SHIN, S., XU, L., HONG, S., AND GU, G. Enhancing network security through software defined networking (sdn). In *Proceedings of The 25th International Conference on Computer Communication and Networks (ICCCN'16)* (August 2016). 19

[174] SHINAGAWA, T., EIRAKU, H., TANIMOTO, K., OMOTE, K., HASEGAWA, S., HORIE, T., HIRANO, M., KOURAI, K., OYAMA, Y., KAWAI, E., ET AL. Bitvisor: a thin hypervisor for enforcing i/o device security. In *Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments* (2009). 22, 69, 70

[175] Shodan. `https://www.shodan.io`, 2019. Accessed on 2019-02-13. 51

[176] SIMPSON, A. K., ET AL. Securing vulnerable home iot devices with an in-hub security manager. In *2017 IEEE PerCom Workshops* (2017). 19, 20, 21, 54, 57, 61

[177] SINGARAVELU, L., PU, C., HAERTIG, H., AND HELMUTH, C. Reducing TCB complexity for security-sensitive applications: Three case studies. In *EuroSys* (2006). 70

[178] SISODIA, D. On the state of internet of things security: Vulnerabilities, attacks, and recent countermeasures. *University of Oregon, Tech. Rep* (2020). 16

[179] SLAUGHTER, A., YAMPOLSKIY, M., MATTHEWS, M., KING, W. E., GUSS, G., AND ELOVICI, Y. How to ensure bad quality in metal additive manufacturing: In-situ infrared thermography from the security perspective. In *Proceedings of the 12th International Conference on Availability, Reliability and Security* (New York, NY, USA, 2017), ARES '17, ACM, pp. 78:1–78:10. 12, 24

[180] SON, S., SHIN, S., YEGNESWARAN, V., PORRAS, P., AND GU, G. Model Checking Invariant Security Properties in OpenFlow. In *ICC* (2013), IEEE. 22

[181] Song, C., Lin, F., Ba, Z., Ren, K., Zhou, C., and Xu, W. My smartphone knows what you print: Exploring smartphone-based side-channel attacks against 3d printers. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (New York, NY, USA, 2016), CCS '16, ACM, pp. 895–907. 12, 13

[182] Song, D., Brumley, D., Yin, H., Caballero, J., Jager, I., Kang, M. G., Liang, Z., Newsome, J., Poosankam, P., and Saxena, P. Bitblaze: A new approach to computer security via binary analysis. In *International Conference on Information Systems Security* (2008), Springer, pp. 1–25. 111

[183] Spadaro, J., and Wyatt, L. Mutiny fuzzer. `https://github.com/Cisco-Talos/mutiny-fuzzer`, 2019. Accessed: 2019-05-03. 26, 29, 32

[184] Spafford, E. H. The internet worm program: An analysis. *ACM SIGCOMM Computer Communication Review 19*, 1 (1989), 17–57. 18

[185] Sturm, L., Albakri, M., Williams, C. B., and Tarazaga, P. In-situ detection of build defects in additive manufacturing via impedance-based monitoring. In *27th Annual International Solid Freeform Fabrication Symposium–An Additive Manufacturing Conference* (2016). 12

[186] Sturm, L. D., Williams, C. B., Camelio, J. A., White, J., and Parker, R. Cyber-physical vulnerabilities in additive manufacturing systems: A case study attack on the .stl file with human subjects. *Journal of Manufacturing Systems 44* (2017), 154 – 164. 12, 13, 24

[187] Sullo, C., and Lodge, D. Nikto2. `https://cirt.net/Nikto2`, 2019. Accessed: 2019-04-03. 44

[188] `https://suricata-ids.org`, 2020. 77

[189] Syed, A., Anwer, B., Gopalakrishnan, V., and Van der Merwe, J. Depo: A platform for safe deployment of policy in a software defined infrastructure. In *Proceedings*

of the 2019 ACM Symposium on SDN Research (2019), pp. 98–111. 22

[190] TA-MIN, R., LITTY, L., AND LIE, D. Splitting interfaces: Making trust between applications and operating systems configurable. In *Proceedings of the 7th symposium on Operating systems design and implementation* (2006), pp. 279–292. 22, 70

[191] TENABLE. Nessus. `https://www.tenable.com/downloads/nessus`, 2019. Accessed: 2019-05-03. 26, 32

[192] TEWS, H., WEBER, T., VÖLP, M., POLL, E., VAN EEKELEN, M., AND VAN ROSSUM, P. Nova micro–hypervisor verification. *CTIT technical report series* (2008). 22, 114

[193] TRACH, B., KROHMER, A., GREGOR, F., ARNAUTOV, S., BHATOTIA, P., AND FETZER, C. Shieldbox: Secure middleboxes using shielded execution. In *Proceedings of the Symposium on SDN Research* (2018), pp. 1–14. 21, 60, 61, 65, 67, 69, 85

[194] TUNG, L. Cisco critical flaw: At least 8.5 million switches open to attack, so patch now. `https://www.zdnet.com/article/cisco-critical-flaw-at-least-8-5-million-switches-open-to-attack-so-patch-now/`, 2018. Accessed on: 2019-02-07. 25, 49, 50

[195] UAN PEDRO TOMÁS. Smart city case study: Santander, spain. `https://enterpriseiotinsights.com/20170116/smart-cities/smart-city-case-study-santander-tag23-tag99`, 2017. 62, 65

[196] VADALA, E., AND GRAHAM, C. Downtime costs auto industry $22k/minute - survey. `https://news.thomasnet.com/companystory/downtime-costs-auto-industry-22k-minute-survey-481017`, 2019. Accessed on 2019-02-13. 4, 18, 24

[197] VASCONCELOS, G., CARRIJO, G., MIANI, R., SOUZA, J., AND GUIZILINI, V. The impact of dos attacks on the ar. drone 2.0. In *2016 XIII Latin American Robotics Symposium and IV Brazilian Robotics Symposium (LARS/SBR)* (2016), IEEE, pp. 127–132. 16

[198] VASUDEVAN, A. The uber extensible micro-hypervisor framework (uberxmhf). In *Practical Security Properties on Commodity Computing Platforms*. Springer, 2019, pp. 37–71. 6, 22, 69, 70, 78, 114

[199] VASUDEVAN, A., AND CHAKI, S. Have your pi and eat it too: Practical security on a low-cost ubiquitous computing platform. In *2018 IEEE European Symposium on Security and Privacy (EuroS&P)* (2018), IEEE, pp. 183–198. 22, 70, 78, 87, 114

[200] VASUDEVAN, A., CHAKI, S., JIA, L., MCCUNE, J., NEWSOME, J., AND DATTA, A. Design, implementation and verification of an extensible and modular hypervisor framework. In *2013 IEEE S&P* (2013). 6, 22, 69, 70, 87

[201] VASUDEVAN, A., CHAKI, S., MANIATIS, P., JIA, L., AND DATTA, A. überspark: Enforcing verifiable object abstractions for automated compositional security analysis of a hypervisor. In *25th USENIX Security Symposium (USENIX Security 16)* (Austin, TX, Aug. 2016), USENIX Association, pp. 87–104. 22, 78, 87

[202] VASUDEVAN, A., PARNO, B., QU, N., GLIGOR, V. D., AND PERRIG, A. Lockdown: Towards a safe and practical architecture for security applications on commodity platforms. In *Proc. of TRUST* (2012). 22, 70, 72

[203] VASUDEVAN, A., QU, N., AND PERRIG, A. Xtrec: Secure real-time execution trace recording on commodity platforms. In *Proc. of IEEE HICSS* (2011). 70

[204] VERIZON. 2016 data breach investigations report. `https://enterprise.verizon.com/resources/reports/DBIR_2016_Report.pdf`, 2016. Accessed: 2019-09-09. 2

[205] VINCENT, H., WELLS, L., TARAZAGA, P., AND CAMELIO, J. Trojan detection and side-channel analyses for cyber-security in cyber-physical manufacturing systems. *Procedia Manufacturing 1* (2015), 77–85. 14

[206] VISOOTTIVISETH, V., AKARASIRIWONG, P., CHAIYASART, S., AND CHOTIVATUNYU,

S. Pentos: Penetration testing tool for internet of thing devices. In *TENCON 2017 - 2017 IEEE Region 10 Conference* (Nov 2017), pp. 2279–2284. 26

[207] WALKER, J. Ent: A pseudorandom number sequence test program. `http://www.fourmilab.ch/random/`, 2008. Accessed: 2019-08-03. 30

[208] WANG, H., YANG, G., CHINPRUTTHIWONG, P., XU, L., ZHANG, Y., AND GU, G. Towards fine-grained network security forensics and diagnosis in the sdn era. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security* (New York, NY, USA, 2018), CCS '18, Association for Computing Machinery, p. 3–16. 60, 61, 65, 67, 68, 85

[209] WANG, H. J., GUO, C., SIMON, D. R., AND ZUGENMAIER, A. Shield: Vulnerability-driven network filters for preventing known vulnerability exploits. In *Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications* (2004), pp. 193–204. 18

[210] WANG, R., SHOSHITAISHVILI, Y., KRUEGEL, C., AND VIGNA, G. Steal this movie: Automatically bypassing DRM protection in streaming media services. In *Presented as part of the 22nd USENIX Security Symposium (USENIX Security 13)* (Washington, D.C., 2013), USENIX, pp. 687–702. 30

[211] WANG, Z., WU, C., GRACE, M., AND JIANG, X. Isolating commodity hosted hypervisors with hyperlock. In *Proc. of EuroSys 2012* (2012). 70

[212] WEN, X., ET AL. Towards a Secure Controller Platform for OpenFlow Applications. In *HotSDN* (2013), ACM. 21

[213] WILLIAMS, T. J. The purdue enterprise reference architecture. *Comput. Ind. 24*, 2-3 (Sept. 1994), 141–158. 49

[214] WILSON, J., ET AL. Trust but verify: Auditing the secure internet of things. In *MobiSys* (2017), ACM, pp. 464–474. 111

[215] WU, M., AND MOON, Y. B. Taxonomy of cross-domain attacks on cybermanufacturing system. *Procedia Computer Science 114* (2017), 367 – 374. Complex Adaptive Systems Conference with Theme: Engineering Cyber Physical Systems, CAS October 30 - November 1, 2017, Chicago, Illinois, USA. 9, 13, 24

[216] WU, Y., CHEN, A., HAEBERLEN, A., ZHOU, W., AND LOO, B. T. Automated bug removal for software-defined networks. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)* (2017), pp. 719–733. 22

[217] XIAO, F., ZHANG, J., HUANG, J., GU, G., WU, D., AND LIU, P. Unexpected data dependency creation and chaining: A new attack to sdn. In *2020 IEEE Symposium on Security and Privacy (SP)* (Los Alamitos, CA, USA, may 2020), IEEE Computer Society, pp. 1512–1526. 61, 68, 85

[218] XING, J., WU, W., AND CHEN, A. Architecting programmable data plane defenses into the network with fastflex. In *Proceedings of the 18th ACM Workshop on Hot Topics in Networks* (2019), pp. 161–169. 19

[219] XIONG, X., TIAN, D., AND LIU, P. Practical protection of kernel integrity for commodity os from untrusted extensions. In *Proc. of NDSS* (2011). 70

[220] XU, W., LI, J., SHU, J., YANG, W., XIE, T., ZHANG, Y., AND GU, D. From collision to exploitation: Unleashing use-after-free vulnerabilities in linux kernel. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security* (2015), pp. 414–425. 61

[221] YAMPOLSKIY, M., ANDEL, T. R., MCDONALD, J. T., GLISSON, W. B., AND YASINSAC, A. Towards security of additive layer manufacturing, 2015. 24

[222] YAMPOLSKIY, M., HORVATH, P., KOUTSOUKOS, X. D., XUE, Y., AND SZTIPANOVITS, J. Taxonomy for description of cross-domain attacks on cps. In *Proceedings of the 2nd ACM international conference on High confidence networked systems* (2013), pp. 135–

142. 9, 13

[223] Yampolskiy, M., King, W. E., Gatlin, J., Belikovetsky, S., Brown, A., Skjellum, A., and Elovici, Y. Security of additive manufacturing: Attack taxonomy and survey. In *Additive Manufacturing* (May 2018), vol. 21, pp. 431–457. 2, 9, 10, 16, 20, 24

[224] Yampolskiy, M., Skjellum, A., Kretzschmar, M., Overfelt, R. A., Sloan, K. R., and Yasinsac, A. Using 3d printers as weapons. *International Journal of Critical Infrastructure Protection 14* (2016), 58 – 71. 9

[225] Yoon, C., Lee, S., Kang, H., Park, T., Shin, S., Yegneswaran, V., Porras, P., and Gu, G. Flow wars: Systemizing the attack surface and defenses in software-defined networks. *IEEE/ACM Transactions on Networking 25*, 6 (2017), 3514–3530. 60, 61, 65, 67, 68, 85, 86

[226] Yu, M., Gligor, V. D., and Zhou, Z. Trusted display on untrusted commodity platforms. In *ACM CCS* (2015), pp. 989–1003. 70

[227] Yu, T., Fayaz, S. K., Collins, M. P., Sekar, V., and Seshan, S. Psi: Precise security instrumentation for enterprise networks. In *NDSS* (2017). 4, 19, 20, 21, 57, 61, 63, 102

[228] Yu, T., Li, T., Sun, Y., Nanda, S., Smith, V., Sekar, V., and Seshan, S. Learning context-aware policies from multiple smart homes via federated multi-task learning. In *2020 IEEE/ACM Fifth International Conference on Internet-of-Things Design and Implementation (IoTDI)* (2020). 113

[229] Yu, T., Sekar, V., Seshan, S., Agarwal, Y., and Xu, C. Handling a trillion (unfixable) flaws on a billion devices: Rethinking network security for the internet-of-things. In *Proceedings of the 14th ACM Workshop on Hot Topics in Networks* (2015), pp. 1–7. 4, 19, 20, 21, 54, 56, 57, 61, 63

[230] Yu, T., Sun, Y., Nanda, S., Sekar, V., and Seshan, S. Radar: A robust behavioral

anomaly detection for iot devices in enterprise networks. Tech. rep., CMU CyLab, 2019. 103

[231] YUAN, L., CHEN, H., MAI, J., CHUAH, C.-N., SU, Z., AND MOHAPATRA, P. Fireman: A toolkit for firewall modeling and analysis. In *2006 IEEE Symposium on Security and Privacy (S&P'06)* (2006), IEEE, pp. 15–pp. 18

[232] ZDRESEARCH. Owasp-nettacker. `https://github.com/zdresearch/OWASP-Nettacker`, 2019. Accessed: 2019-05-15. 26

[233] Zeek. `https://zeek.org`, 2020. 77, 103, 107

[234] ZELTMANN, S., GUPTA, N., GEORGIOS TSOUTSOS, N., MANIATAKOS, M., RAJENDRAN, J., AND KARRI, R. Manufacturing and security challenges in 3d printing. *JOM 68* (05 2016). 12, 13

[235] ZETTER, K. Serious security holes found in siemens control systems targeted by stuxnet. `https://arstechnica.com/information-technology/2011/08/serious-security-holes-found-in-siemens-control-systems-targeted-by-stuxnet/`, 2011. Accessed on 2019-02-13. 2

[236] ZHANG, F., CHEN, J., CHEN, H., AND ZANG, B. Cloudvisor: retrofitting protection of virtual machines in multi-tenant cloud with nested virtualization. In *Proc. of SOSP* (2011). 70

[237] ZHANG, M., CHEN, C.-Y., KAO, B.-C., QAMSANE, Y., SHAO, Y., LIN, Y., SHI, E., MOHAN, S., BARTON, K., MOYNE, J., ET AL. Towards automated safety vetting of plc code in real-world plants. In *2019 IEEE Symposium on Security and Privacy (SP)* (2019), IEEE, pp. 522–538. 16

[238] ZHANG, P., LI, H., HU, C., HU, L., XIONG, L., WANG, R., AND ZHANG, Y. Mind the gap: Monitoring the control-data plane consistency in software defined networks. In *Proceedings of the 12th International on Conference on emerging Networking EXperiments*

and Technologies (2016), pp. 19–33. 22

[239] ZHANG, P., XU, S., YANG, Z., LI, H., LI, Q., WANG, H., AND HU, C. Foces: Detecting forwarding anomalies in software defined networks. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)* (2018), IEEE, pp. 830–840. 22

[240] ZHANG, W., SHARMA, A., JOSHI, K., AND WOOD, T. Hardware-assisted isolation in a multi-tenant function-based dataplane. In *Proceedings of the Symposium on SDN Research* (2018), pp. 1–7. 21

[241] ZHOU, Z., GLIGOR, V. D., NEWSOME, J., AND MCCUNE, J. M. Building verifiable trusted path on commodity x86 computers. In *IEEE S&P* (2012). 70

[242] ZHOU, Z., YU, M., AND GLIGOR, V. D. Dancing with Giants: Wimpy Kernels for On-demand Isolated I/O. In *Proc. of IEEE S&P* (2014). 70

[243] ZWIENENBERG, R. Acad/medre.a 10000's of autocad files leaked in suspected industrial espionage. `https://www.welivesecurity.com/2012/06/21/acadmedre-10000s-of-autocad-files-leaked-in-suspected-industrial-espionage/`, 2012. Accessed on 2019-02-13. 2, 4, 12, 13, 24