Programmable System Security in a Software-Defined World – Research Challenges and Opportunities*

Guofei Gu (TAMU), David Ott (VMware), Vyas Sekar (CMU), Kun Sun (GMU)

Ehab Al-Shaer, Alvaro Cardenas, Yan Chen, William Enck, Hongxin Hu, Dennis Moreau, Cristina Nita-Rotaru, Don Porter, Mike Reiter, Sandra Scott-Hayward, Srinivasan Seshan, Mike Swift, Xiaofeng Wang, Vinod Yegneswaran and Other Workshop Attendees†

December 2018

*Supported by the National Science Foundation (NSF), USA. The workshop was held August 13-14, 2018, Fairfax, VA. More details available at https://sites.google.com/view/nsf-sps-workshop/.

†A full list of attendees is available in the appendix.

Table of Contents

Executive Summary

Recent years have seen a dramatic and rapid paradigm shift in computing from static control systems (often implemented in hardware), to dynamic, easily-reconfigurable, software-defined systems. The researchers and practitioners have just begun to scratch the surface of how the ever-increasing software-defined everything (SD-X) changes the landscape of cybersecurity. On one hand, a software-defined world adds potentially new attack surfaces that deserve new research investigation. On the other hand, considering the fact that everything can be defined by the software, now we can have a new playground to redesign the security mechanisms and services. With programmable security, we can also better embrace the new advances in big data and AI to provide more intelligent and adaptive security for the software-defined world. We believe the opportunity is ripe for academics to make foundational contributions, collaboratively with industry, to shape the next 5 years of research in this new space, **SPS (Software-defined Programmable Security)**. Emerging data centers, cloud networks, IoT and edge computing also provide a fertile playground to consider disruptive software-defined programmable security designs.

While many research directions are interesting, this report outlines a few high-priority areas:

- New abstractions for data/control planes aimed specifically at security, and new architectures that integrate diverse SD-X domains (networking, processing, storage, etc.) for a more powerful and comprehensive security framework.
- Trust across multiple stakeholders, and integration of diverse SD-X domains.
- A better understanding of attack surfaces and adversarial methods within modern software-defined infrastructures.
- New programming and language paradigms for programmable security.
- New Applications paradigms that exploit programmable paradigms in innovative ways.
- New formal and experimental methodologies for reasoning about software-defined security,
- The Integration of emerging AI/ML and data-driven capabilities into programmable system security.
- The Application of programmable security approaches to emerging platforms and infrastructure domains such as edge computing, IoT, cyber-physical infrastructures.

Overall, this report helps define the vision of a new generation of security technologies in the rapidly expanding world of software-defined infrastructures and devices.

1 Introduction and Overview

We increasingly live in a software-defined world where systems that were once implemented as rigid control systems or fixed function hardware systems are now highly programmable. This trend has opened up an exciting new space for research on novel approaches to system control, management, applications, and services. Today's early examples include multi-tenant clouds, software-defined networking (SDN) [2], network function virtualization (NFV), software-defined infrastructure (SDI) [5], and software-defined radios (SDR). Individually, these SD-X systems present large research challenges, and these problems are compounded when they are interconnected into a software-defined world. Our interest is in identifying research challenges and opportunities in the area of programmable security within this context, or SPS (Software-defined Programmable Security).

In essence, "software-defined" realizes programmability through an architectural approach in which hardware resources are virtualized; that is, abstractions of physical capabilities are made available to applications or higher-level services in a way that is decoupled from the underlying physical device or infrastructure. To date, software-defined approaches have been realized mostly in the context of datacenters which may simultaneously deploy software-defined network, storage, and compute stacks (a.k.a., virtualization). But it can broadly be viewed as a programmable framework for any device or compute context (e.g., IoT, edge computing).

While SD-X technologies have rapidly proliferated within industry and received considerable systems research attention, the paradigm has not been fully exploited in approaching a wide array of important security challenges. The objective of this workshop is to identify those research challenges and opportunities to exploit SD-X approaches in making system security more programmable, agile, orchestrated, and intelligent. This workshop has created a much-needed opportunity for a cross-cutting group of researchers to fill out the vision of what programmable security based on SD-X could be, including research challenges, long-term visions, and key issues. In the process, this workshop will promote a more focused community and vision where traditionally disparate communities previously worked in isolation and without a more ambitious system security vision within the context of complex software-defined infrastructures.

In the workshop, researchers discussed the following new research directions in software-defined programmable security (SPS): (1) new architectures that integrate diverse SD-X domains (networking, processing, storage, etc.) and new abstractions for data/control planes aimed specifically at security, (2) trust across multiple stakeholders in SPS, (3) a better understanding of attack surfaces and adversarial methods within modern software-defined infrastructures, (4) new programming and language paradigms for programmable security, (5) new applications paradigms that exploit programmable paradigms in innovative ways, (6) new formal and experimental methodologies for reasoning about software-defined security, (7) the integration of emerging AI/ML and data-driven capabilities into programmable system security, and (8) the application of programmable security approaches to emerging platforms and infrastructure domains such as edge computing, IoT, and cyber-physical infrastructures.

Below is a summary of group ideation and discussion.

2 New Research Challenges and Opportunities

2.1 Architectures and Abstractions

2.1.1 Architectural Challenges

A key question discussed initially by the group is what notion of programmable security we are considering. Understanding and defining "software-defined programmable security" (SPS) and "programmable security" would seem to be a basic challenge for the research community.

It was suggested that our initial vision of SPS simply borrow notions from the architectural framework of software-defined networking (SDN) [2]: a centralized control point and view of the infrastructure to be secured, a northbound API offering security programmability across the infrastructure, and a southbound API that maps security control plane actions (e.g., monitoring, policy configuration and enforcement) to individual platforms.

While this provides a much-needed starting point, a central challenge in the SPS architectural space is to explore other architectures and/or more specific architectures that enable the goals of SPS (programmability, agility, orchestration, intelligence) in different ways. Different approaches may offer different tradeoffs in what is enabled versus system costs, implementation complexity, integration requirements, and so on.

With this starting point model in mind, group attendees discussed research issues as follows:

- Complexity management. How might SPS offer a layer that maps high-level security policy and algorithms to low-level configuration and run-time systems on constituent platforms? In this sense, the SPS could behave like a translation layer: high-level abstractions are translated in an automated way to the underlying platforms, thus managing layered complexity and platform heterogeneity.

- Leveraging low-level programmability. How can SPS architectures include low-level and programmable features on constituent platforms? SPS isn't just a high-level abstraction layer for programming northbound applications, it must include low-level run-time mechanisms, cross-layer considerations, and programmable features in the underlying platforms. An SPS architecture should be designed to support the entire stack, high to low.

- Multiple solution providers. It is unrealistic to assume a single solution provider owns the entire infrastructure and provides all needed security solutions (e.g., malware detection, security analytics) in a unified manner. How can SPS architectures be constructed to provide a common framework but, like an open architecture, allow multiple solution providers?

- Comprehending multiple stakeholders. More generally, how can SPS architectures comprehend and serve a realistic cast of compute infrastructure stakeholders – infrastructure providers, security administrators, security solution providers, software developers, compliance auditors, platform designers, various notions of "users", and more?

- Common monitoring and diagnostics interface [20]. How could SPS offer a common interface for monitoring the security of an infrastructure, and for providing detailed forensic information when alerts occur, or compromises are detected?

- Performance challenges. How can an SPS architecture be introduced without compromising the performance of a compute infrastructure? This includes many low-level and end-to-end considerations.

- Centralized vs distributed control. If a common infrastructure provides programmable building blocks for multiple solution providers, how should SPS control points be handled? Are they fully centralized which creates the problem of who controls them? Are they distributed which would seem to undercut some of the basic principles of SPS? Are there hybrid models that handle control differently across stakeholders?

While an SDN-based vision for SPS is useful as a starting point, many people brought up ways in which the vision will need to be broadened and extended over time. Research questions include:

- New hardware platforms. How will SPS architectures comprehend FPGAs and other programmable devices on future platforms? How will SPS architectures comprehend new IoT devices and gateway architectures at the edge of the network? What new abstractions and security building blocks will be needed?

- Evolution toward comprehensive SDI. While software-defined networking is a prominent framework and offers a lens through which we look at SPS, in fact, the trend is for every aspect of a data center or public cloud or edge computing environment to be software-defined, including networking, storage, compute (e.g., virtual machines, containers), data infrastructures, and more. How will SPS architectures comprehend and leverage a more comprehensive notion of converged or integrated "software defined infrastructure" (SDI)?

- Federated infrastructures. How will future SPS architectures address the broader federation of software-defined infrastructures? SDX, or software-defined internet exchange points, provides a networking case study in how SDN can be extended to consider federated infrastructures. How might this look in programmable security?

2.1.2 Abstraction Challenges

The notion of a "security plane" can offer not only orchestrated functionality across an infrastructure, but a convenient set of programming abstractions that separate the details of platform configuration, monitoring, and policy enforcement from a high-level security application. The analogy follows from SDN: a centralized control plane offers a convenient set of abstractions that allow you to reason about and program the network where previously only platform-level abstractions and configuration frameworks existed.

The grand challenge here for the research community is to define the nature of the security plane, and what the right abstractions are to create a powerful yet flexible programming model.

Some key issues discussed by the group include:

- List of guiding principles. What are the guiding principles for developing SPS abstractions? Beyond separating infrastructure operation from security control, what are we looking for in programmable abstractions and how could they revolutionize the way we think about security?

- Avoiding semantic gaps. A key lesson from prior security research (e.g., virtual machine inspection) is the problem of semantic gaps in layered or encapsulated system architectures. How can layered SPS architectures avoid the problem of losing information between layered abstractions and crippling security monitoring and programmability at higher layers?

- Translating high-level intent. Security policy requirements are most often discussed and specified in high-level natural language. How can SPS abstractions facilitate reasoning about and programming infrastructure security in terms that are understandable to humans, and then can be compiled into platform-specific mechanisms and frameworks that implement high-level intentions?

- Need for user studies. There is an important need for user studies that look at what various user and security administrator needs are, and perhaps including all the stakeholders mentioned above. These studies could guide development of the right abstractions for a security plane.

- Configurable media. What are the right security abstractions and medium of logical reasoning for configurable media?

Group discussion pointed out that programmable security is not without its risks. With programmable security comes:

- New attack surfaces. Any new architecture, set of APIs, and programmable mechanisms will create an attack surface for adversaries to explore and exploit. How can SPS approaches provide controlled exposure and protection for programmable architectures?

- Program errors and bugs. With programmability comes program errors and bugs that will create new security vulnerabilities. What validation and verification approaches will help to minimize this, and how can exposures be contained within the SPS architecture?

2.2 Trust Across Multiple Stakeholders

2.2.1 Issues Related to Multitenancy

One of the key concerns in multitenancy is the need for more context information, and for this information to be exposed bidirectionally between the tenants and the software-defined infrastructure. Two open questions in this regard that was raised were: (A) Where is the context coming from? (B) Do you trust the context that is coming from somewhere?

A general consensus was that we envision the infrastructure providing the hooks to the tenants to give some contextual information. However, there was some residual concern with respect to the

granularity of visibility that a tenant may have; e.g., a guest OS cannot see anything below its level of existence.

Related to trust, the discussion surrounded the issue of trusting software vs. trusting hardware, and more importantly the fact that hardware itself is becoming a software artifact that may have bugs (e.g., in light of Spectre/Meltdown and SGX bugs). The general discussion was that while hardware can have errors at times, using a hardware trust anchor is an important piece of the trust exercise. Some of the more open-ended discussion surrounded the idea of reducing the dependence on hardware as much as possible, either by making some of the trust decisions distributed or by breaking down key infrastructure components into tighter elements that may help us to develop tighter boundaries on what you need to trust.

Finally, there was a discussion around how open source may be an option and how testing might help. However, any single actor in the system may not be able to exhaustively failtest everything, and maybe someone else can give us some sort of a guarantee on the capabilities that were learned from outsourced testing.

2.2.2 Issues Related to Trust in the Control Plane

The key challenge that was discussed here was that the bar of trust and the reliability on the control plane is potentially much higher than any other piece of the software-defined infrastructure, especially as all resource management and security decisions directly depend on the correctness of the control plane.

One major concern is that the software-defined infrastructure potentially makes the control plane much more complex than traditional control planes. In some sense, we are writing Turing-complete code to express the control plane logic in contrast to simpler declarative predicates that were used in legacy infrastructures.

That being said, the breakout also discussed some of the potential opportunities. The first, is that the control plane likely has a tighter envelope of correct operation and semantics that may be amenable to some form of formal verification. There was discussion surrounding the expressivity of existing frameworks and tools and whether we have the right kind of formal tools to express this, and that classic datalogs may not be sufficient. There were two alternatives that were discussed wherein the control plane developer provides annotations to aid formal analysis vs restricting the behavior via higher level language constructs, and there was some consensus that it might be easier to implement language level security features than rely on the programmer annotations. Finally, there was some discussion that some of the recent intent driven efforts underway in the academic and research communities may actually be a step in the right direction as it might make the management plane and composition/optimization tasks cleaner.

2.2.3 Issues of Trust Across Multiple Stakeholders

The discussion around multiple stakeholders focused on two key aspects of the trust: (1) sharing data and (2) enabling cross-domain automation of key programmable security features.

The discussion noted that there were several efforts underway to define standards to enable different stakeholders to share information and incident response. In this respect, the software programmable infrastructures may help implement the protection mechanisms for this data at the

right place. However, there was the oft-raised concern of incentives and whose interest is it in to actually deploy the capabilities, and what information is to be shared and how we can narrow that to enable more participation.

The second perhaps under-explored domain is the issue of automation/actuation of security capabilities across stakeholders. Traditionally, there has been some form of manual testing and approval process, and that the default mode in which such remote tasks run is fail open; i.e., disable protection under failure. There was discussion that SPS may be able to enable a large number and variety of failure models, and automation that goes beyond what was previously possible; e.g., slowing down traffic or honeypotting them, etc. In particular, emerging primitives like SGX can enable such interactions, since these only require integrity properties in some form which are easier to reason about.

Closing remarks: Finally, the discussion briefly touched on the issue of transition to practice and how we can best reach coordination and consensus on the right trust boundaries and interfaces. Some of the discussion surrounded the issue of repetition/reinvention in academic/research endeavors and how we can avoid it going forward.

2.3 New Attack Surfaces

While a software-defined world has created many new opportunities, programmable environments cannot escape one of the major challenges in computer systems today: how to ensure resilience against attacks. The goal of this session was to identify new attack surfaces in software-defined programmable security (SPS) and associated research issues.

Motivating Software-Defined X (SD-X): Group attendees identified the following examples of software-defined environments to guide our discussion on SPS attack surfaces: multi-tenant clouds, software-defined networking (SDN), network function virtualization (NFV), software-defined infrastructure (SDI), software-defined radios (SDR), software-defined internet exchange points (SDX), software-defined storage, programmable hardware including FGPAs, and even programmable power units and batteries.

2.3.1 Existing attack surfaces in SD-X

Before examining new attack surfaces, the group first made the observation that given today's systems are not perfectly secure, there are many existing attack surfaces that will simply persist in SD-X. So SD-X will be adding new surfaces to an existing list of attack surfaces in our modern day platforms and software stacks.

The group identified at least three existing attack surfaces as examples:

- Bugs and vulnerabilities in current operating systems,
- Bugs and vulnerabilities in web browsers, and
- Bugs and vulnerabilities in network protocols stacks. This includes identity binding attacks across different layers of the stack [9]. (Networking examples of this include DNS spoofing and DHCP forgery.)

2.3.2 New attack surfaces in SD-X

SD-X programmable security environments bring new attack surfaces which the group classified as functional vulnerabilities and interface vulnerabilities.

Functional vulnerabilities include attack surfaces created by the SD-X architecture and its core functionality. Key attack surfaces include:

- Centralized control. Controllers within an SD-X infrastructure, when not implemented in a distributed manner, are subject to classical vulnerabilities associated with centralization (e.g., denial of service [16, 19, 25], the potential for adversary control over the entire infrastructure [7, 21, 22]). This is because the controller represents a single point of trust and failure.

- Orchestration. Adversaries can find ways to interfere with or falsify communications, group message-passing, controller directives, synchronization points, and so on. This has the effect of weakening or preventing orchestration in SD-X programmable security architectures.

- Dynamic/reactive response. Adversaries can find ways to attack or exploit security schemes designed to respond dynamically to security-relevant events. For example, an attacker can create bogus events that cause a defender to waste resources on security response, to focus on a decoy event that disguises another action, or to manipulate the state of the system in order to find exploits [10].

Interface vulnerabilities include attack surfaces created by SD-X software and communication interfaces which are needed to support programmability, interoperability, and extensibility. Key attack surfaces include:

- Programmable APIs. It is notoriously difficult to design and implement programmable application interfaces that offer both flexibility/functionality and safety/robustness. (Beyond implementation issues, work in language-theoretic security shows this to be a problem with deep theoretical underpinnings.)

- Interoperability. The need for interoperability between platform components of the programmable security architecture creates software and communication attack surfaces that are inherent to the complexity of format mappings and translation.

- Third party applications. SD-X environments commonly support third party applications and services which create new access vectors into the infrastructure. Programmable security architectures may, themselves, support third party applications (e.g., malware scanning) which creates attack surfaces [15].

2.3.3 Addressing New Attack Surfaces: Research Challenges and Approaches

Group attendees discussed the following approaches to addressing new attacks surfaces in software-defined programmable security:

- Automation. While automation is a key approach in SPS, the group discussed its double-edge character. On the one hand, automation exploits programmability and enables many of the benefits of SD-X security. But on the other hand, it makes the propagation of attacks faster, enables complex attack chains, and creates additional attack surfaces. (e.g., Adversaries can leverage automation to attack multiple switches, propagate compromised software, alter flow tables, compromise cryptographic infrastructure.)

- Machine learning. Similarly, ML also has a double-edge character in that it can amplify the benefits of automation by adding intelligence in various respects, but at the expense of creating attack surfaces that adversaries can manipulate (e.g., adversarial learning). Mechanisms are needed to insure correct function.

- Context information. A major strength of SPS solutions is the opportunity to leverage information aggregated from various parts and layers of SD-X infrastructure. For example, topology information can be combined with network traffic analysis, application deployment information, and global security policy to understand whether observed system events are an emerging threat.

- Verification. An important area of work in SD-X programmable architectures is verification: how to prevent attack surfaces created by configuration or design bugs, and how to verify the correctness of complex system state like network traffic, execution threads across the infrastructure, device IO integrity, information/data flow, and so on.

- Exploring new approaches. There are a wide variety of strategies to be explored in approaching SPS, from assuming untrusted orchestration, to the use of Byzantine resilience techniques, to secure outsourcing frameworks, and more.

The group discussed some associated issues that should be included in this research space as follows:

- Performance. SPS solutions must be performant to be practical and deployable.

- Reliability. Insuring reliability within the context of complex and scaled SD-X infrastructures is difficult.

- Usability. How to make potentially complex SPS solutions usable, both by security application programmers and by SD-X infrastructure administrators? There is often a trade-off between usability and security.

- Reconstructing attack chains. As the SD-X environment becomes more complex, analyzing state and reconstructing attack chains can become more complex. It becomes hard to monitor inter-connected components, and there are many

- Predicting user behavior. User behavior is often unpredictable and difficult to model which makes designing security difficult.

- Dependencies. Because SD-X environments often include many devices, many layers of the system stack, and many interoperating elements, there is big challenge in how to construct SPS solutions when there are so many dependencies.

- Legacy code. Related to the issue of dependencies is the need to address legacy code within SD-X environments, and within SPS solutions. Approaches may leverage decomposition, isolation mechanisms, and containment schemes.

- Stateless execution. How to monitor and secure stateless execution environments is an interesting recent question since conventional information for security analysis may not be available, and state is highly ephemeral.

- Multiple stakeholders. Researcher in SD-X security must take into account a complex matrix of stakeholders – infrastructure providers, security administrators, system software providers, application developers, software/hardware solution vendors, system users, and so on.

- Auditing. While often overlooked in academic research, SPS solutions need to consider the real-world need for third party auditing. Solutions need to generate data demonstrating policy enforcement and efficacy.

2.3.4 Additional Issues

One major challenge for research in software-defined programmable security is access to realistic testbeds. Testbeds are needed, for example, to study attack surfaces associated with interoperability. Ideally, testbeds should be scaled to real-world levels. The group debated whether GENI (Global Environment for Network Innovations) is adequate and discussed briefly other alternatives like OpenCloud and Project Silver (UNC).

What makes the problem of testbeds even more difficult is that researchers often cannot anticipate all the ways in which SD-X will be used, creating a chicken-and-egg problem: how to model SD-X infrastructure in realistic ways when new research and applications are continually redefining testbed parameters. Broadly, what should a testbed look like for research?

Another major challenge is programmable security solutions for interconnected SD-X environments. Problems are compounded when individual systems are interconnected into a software-defined world, when larger SD-X environments are federated, and when many different parties are contributing code and services.

There is a large space of research issues here: how to understand the compounded attack surfaces in a systematic way, new types of attacks on interconnected SD-X environments, how to contain the risks associated with third party code, how to define and enforce security policies in a dynamic and federated environment, the role of the user in securing an increasingly complex software-defined world, and more.

2.4 New policy Programming Paradigms

This session focused on flexible policy enforcement as enabled by software-defined platforms of the future. The basic tenet of the session was that future software-defined platforms should permit greater adaptation, flexibility, and programmability in security policy enforcement, and the session's goal was to illustrate some of the research challenges in achieving that outcome.

The discussion ranged over the following topics:

2.4.1 Generality vs. support

Existing frameworks for enforcing security policies can be quite general, while at the same time providing only modest help to the developer. Consider Linux Security Modules (LSM), for example; in a nutshell, this framework provides policy-enforcement "hooks" into the Linux kernel to enable a loadable module to mediate access to select kernel objects caused by user-level system calls. The loadable module can then implement the policy with which it is programmed, which can be any policy permitted by the information available to the module from its vantage point. While general, LSM has been argued both in person at this meeting, and in articles to provide inadequate support for a range of desirable policies (some mentioned below). There was considerable debate regarding whether LSM should serve as an exemplar for enabling policy enforcement in future software-defined infrastructures, or if something richer would be warranted.

2.4.2 Types of supported security policies

There are numerous types of security policies, and so a natural question when considering programmable policy frameworks is the range of policy types that should be supported. Information-flow policies (e.g., "do not allow data from X to flow to Y"), quantitative information-flow policies, policies with obligations (e.g., "permit action X provided that it will be logged", or "permit use of data X by subsystem Y, provided that X will be deleted after Y is done with it"), provenance (e.g., "permit action X if induced using only trusted data" for some notion of "trusted"), history-based policies (e.g., "allow action X if action Y was not previously taken", or "allow action X at most Y times"), DRM-like policies (e.g., "allow data X to be used for purpose Y, but not for Z") are just some examples of the rich types of policies that can be useful in various scenarios. To support the full range of conceivable policies, there would seem to be little choice but to opt for the most general and programmable solution possible. Limiting attention to subclasses of useful policies might enable frameworks that provide better programmer support, however.

2.4.3 Multi-staged, quantitative policy engines

With the growth of data analytics in support of security policies (e.g., account or network reputation), security policy enforcement will increasingly consist of multiple stages of analysis and be based on quantitative (and error-prone) measures. It thus may be suitable for future security-policy frameworks to embrace workflows as an organizing principle for multiple stages of analysis. The use of quantitative scores in policy enforcement raises immediate questions regarding how thresholds should be set to reduce quantitative scores to qualitative classifications (e.g., "benign"), and what backup plans can be put in place to counteract classification errors.

2.4.4 Policy conflicts among stakeholders

Multi-tenancy, which is a common ingredient in many visions of future software-defined frameworks, is one factor that can give rise to policy conflicts. While it is unlikely that a system could reliably and seamlessly resolve all conflicts without human intervention, general and safe approaches to resolving subclasses of conflicts might still be possible and warrant additional attention [13].

2.5 New Security Applications

2.5.1 What applications are possible using SPS?

a) Can SPS enable most traditional security applications, such as Firewall, IDS/IPS, DDoS Detector, and Scan Detector?

Work has been done to enable traditional security applications, such firewall and DDoS defense, in SDN. Some recent work has also demonstrated the possibility of virtualizing firewall [8], IDS [11], DDoS detector based on NFV [1, 4, 26], network monitoring and measurement [17, 23, 24], network security access control [6, 12], and security services composition [14]. However, there are issues which need to be explored in the future, including how to maintain detection states and how to make quick decisions in a programmable security environment. It is also clear that there is not a one-to-one correspondence between existing security applications and programmable alternatives implemented using SPS; that is to say, there is no one-to-one replacement for all traditional security applications using SPS.

b) Can SPS enable new application paradigms, such as Software-Defined Agility (SDA), and Moving Target Defense (MTD), which exploit programmable paradigms in innovative ways?

Using SPS, we can support both software-defined agility and moving target defense. Some existing work has demonstrated how to enhance agile defenses using anomaly or threat detection to dynamically adapt security policies. In fact, multiple forms of moving target defense are possible given the unique features provided by programmable security, such as network-wide visibility. Thus, there are possibilities of enabling new applications paradigms based on SPS. Of course, some issues, such as how can we reason about the level of agility, need exploration.

c) Can SPS enable new application paradigms integrating AI/ML?

SPS creates an opportunity for new application paradigms integrating AI/ML because, among other reasons, programmable security is rich with information about what the security topology looks like. The context provided by the programmable security could be used to help enable training data for AI/ML models, and to use these models to understand events that just happened based on anomaly reports. New kinds of security applications featuring novel uses of AI/ML algorithms are possible.

2.5.2 How do you build a security application on top of SPS building blocks?

a) Do we need high-level composition languages like Click? Does the language need to provide more features, such as orchestration and interaction?

The group discussed issues associated with what happens when we have to go across domains/sites with programmable security. Programmable security means that we can instrument a domain to provide information, and we can leverage that information as context in understanding

events in additional domains. Although we didn't discuss Click (which provide modular functionality) in detail, the group considered cross-site implementation of analytics engines based on programmable security. As an SPS building block, asset isolation would seem to be key. The control and communication channels of such building blocks provides many programmable security capabilities.

b) Could we generate security applications from natural language-based security descriptions?

Group consensus was that from natural language, we could drive at least one level lower in automatically generating security requirements for objects within the agent's environment. For example, it could be a decision about the restrictions on a service. But natural language doesn't provide enough specificity for automating control topology at lower levels, configuring strong isolation, and or generating specific policies in firewalls and IDSes. We discussed briefly that reference monitors can help to bridge the gap.

2.5.3 How to implement security applications?

a) Can we implement security applications in virtualized environments? Can we implement security applications using low level programmability, such as programmable switches?

There are clear performance and efficiency arguments for making emerging technologies like P4, containers, FPGAs, and GPUs more available. But we can also use them to support advanced security features (e.g., crypto acceleration). As new security applications emerge, there is good reason to consider how different kinds of customizable processing capabilities can be matched to different kinds of security requirements.

b) Can we implement security applications in mobile devices based on SPS?

It is clear that we can implement security applications in other software-defined environments including mobile devices. There seems to be a natural affinity between security policies on mobile devices and policies in datacenter-hosted services. This may be a good reason to consider what that allows in terms of aligning policies across both of these platforms with respect to the programmable security, and how to make the bridge seamless. Implementing security applications/policies on mobile devices allows some malleability and flexibility, which allows us to decide if they match the policies in datacenter-hosted services. If they don't, we should decide how we can adjust the security posture on mobile devices to be consistent with those in datacenter-hosted services.

2.6 Formal Reasoning and Experimental Verification

Formal verification and reasoning may be able to play a substantial role in deploying software-programmable security. Specific targets for formal verification include:

- Ensuring low-level implementations of a policy satisfy high-level intents for the policy, or that distributed implementations of a policy match a centralized implementation.
- Ensuring correct composition across layers, across components, and across administrative domains.
- Synthesizing low-level rules to implement high-level intents, such as firewall/reachability settings, ACLs, and co-location of services.

Experimental verification complements formal methods by providing counter-examples: identifying where policies may not match intents. Testing and verification are not substitutes for each other, as testing only shows that a system fails, not that it succeeds. Thus, formal methods providing similar properties, such as model checking approaches [18], could be useful.

2.6.1 What can be verified?

A central issue with applying formal methods to security is generation of policy: what is the intent of operators/users/administrators? For some policies, such as isolation, this is easy: no sharing outside of a protection domain. Similarly, firewall policies that block classes of traffic may also be simple to specify: block all traffic that doesn't conform to known good data against known ports, from known locations.

For other policies, this may be much more complicated: which services should be connected, and with which shared data? This information may only be available as a set of high-level intents that need to be translated to specific policies, such as "teams should have shared access to common data" and "personnel data is only available to human resources". These intents may be at times contradictory, for example when employees want to access their own personal data, or prospective managers want data on a potential transfer. Thus, a difficult first step is to specify what properties must be verified. Often there is a mismatch between the properties that you actually check for and the high-level properties that we care about because defining these high-level properties is complex and getting such analysis to scale is hard.

Another challenge arising when deploying verification and reasoning is how to express policy. While there has been great work on programming language design, and many languages have large user communities across multiple platforms, this has not occurred with policy languages. Rather, policy languages tend to be specific to a single platform and a narrowly tailored set of concerns. More research on general-purpose policy languages may be needed. For example, how can one express taint-tracking, data exfiltration, and discretionary access control concisely in a single language?

2.6.2 Determining specifications

Verification often requires tedious manual effort to either write code in a format that can be verified (such as seL4, originally written in Haskell, or FSCQ, a formally certified file system). As a result, it is difficult to apply to existing code. There have been some efforts in this direction, notably Bryan Parno's work on TLS verification [3]. Another possibility is to apply Specification Mining tools that, given programs, they will find specifications of behavior. This allows the specifications themselves to be checked against intents, and modifications to a program to be verified against the behavior of previous version.

One approach to specifying behavior is to instead define important generic properties, much as safety and liveness are used for proving protocols correct. Some common properties that could be useful for security are non-interference (two entities cannot affect each other's behavior), equivalence (two policies, perhaps at different levels of implementation, yield the same outcomes), and correctness. But what is meant by correct can vary depending on the context. For example, whether it is correct to accept packets from unknown senders depends on the application. Furthermore, one

must consider what the trusted computing base (TCB) is for verification, and exactly what properties can be trusted. Likewise, specifying security properties for transient states in a distributed system may be challenging, as there may be no single global state.

2.6.3 Synthesis

Synthesis techniques can play an important role in securing SPS systems by generating local policies from global policies. This can be applied in both directions: synthesizing lower-level policies (e.g., switches) from global policies (e.g., entire network). However, it can also be applied in the opposite direction: from local policies, determine an appropriate global policy.

Applying synthesis requires a model of the system being synthesized: for top down, this is a model of the lower-level nodes in the system, how they behave and what capabilities they have. For bottom-up policies, this may be a model of what a centralized controller is capable of. Furthermore, synthesis must consider the environments where things are run including the hardware, the operating system, the libraries and protocols. In the past, people have found attacks on systems developed with formal methods because underlying assumptions were wrong. For example, recent architectural attacks (RowHammer, Spectre, Meltdown) break the assumption that a single processor can implement strong isolation. Another important research problem is ensuring isolation across systems and policies that have to be considered in providing these guarantees.

2.6.4 Scalability

Traditional approaches to formal verification start by creating a formal model, and then mapping the model using linear-temporal logic (LTL) to a finite automaton. The various combinations of states reachable by automatons can be verified against the required properties. However, a major problem with such approaches is scalability: cloud-scale systems have 100,000 - 1,000,1000 elements. For example, the Microsoft Azure networking team plans for up to 100 VMs per host and 100 containers per VM. Building formal models for interactions within such large systems is challenging.

One promising approach is compositional verification techniques: proving properties of individual elements, and then proving that the elements can be composed correctly. However, past work has shown that composition may be difficult to achieve.

Another promising approach is using customized models for a particular application domain that greatly restrict the state space of system. For example, past work showed that building a custom symbolic model checker was orders of magnitude faster than using off-the-shelf techniques.

2.7 Integration of AI/ML and Big Data

2.7.1 Data

One of the major elements required for the application of ML techniques is good quality data. With respect to programmable system security and the application of ML within the software-defined infrastructure (SDI), this leads to a number of questions and areas for further research.

- AI techniques work well dependent on the location of data sources. In the distributed SDI, where are the data sources? Much useful data comes from the user side but how much data

can be consumed in the network? This links to the suggestion to adapt ML for SPS, e.g., push the data analysis closer to the user side.

- Due to the range of security challenges, what kind of data do we need to observe, what features do we select etc., in particular for real-time detection and protection? This is a challenge from the data collection (volume) perspective. What effective summarization techniques could we apply without losing fidelity and how do we summarize state data? However, the related opportunity in SPS is the ability to do programmatic data collection. For example, we can dynamically adapt the type/volume/location of data collected based on the state of the network. SPS is a key enabler for more adaptive data collection.

- Closely related to the issue of type of data to collect is the visibility into the data. For example, when all the data is encrypted, what will we use as data on which to base our analysis and decision-making? Even today, when performing threat analysis based on the encrypted data (use of packet headers and metadata), privacy issues arise. However, with SPS we do get new types of metadata that we can leverage.

- The issue of access to appropriate and classified data (rather than outdated data sets) for evaluation was raised. In some areas today, we have a large body of available data. However, in general, it is missing the context required to be able to use the data. We need to invest time and effort in developing platforms/testbeds from which we can gather high quality labelled/classified data with the appropriate contextual information for use in evaluation.

2.7.2 Network and attack dynamics

The application of ML within networking is difficult. Even with a large volume of data, and assuming that the data is labelled, attacks are constantly changing and evolving. With an unpredictable, highly dynamic network architecture, how do you teach an ML model what is right or wrong and minimize the number of false positives?

Existing techniques combining SPS and ML rely on the ability to define malicious traffic patterns in advance. This also minimizes data capture from the data plane during prediction. However, this approach cannot identify real-time, unknown attacks.

2.7.3 Adversarial Attacks

There is a known issue with ML of an attacker being capable of manipulating the ML decision-making, e.g. based on sensor input pollution. Two aspects regarding adversarial ML were discussed in the session. The first is about deep learning techniques and explainability. How do we verify the decision generated by the ML and consequently how to prove that it has not been manipulated by an attacker?

The second is about introducing complexity. There is a question about whether it is actually a good idea to introduce ML into SDI given that it can increase the complexity of an already complicated network and infrastructure, and potentially widen the attack surface which is in direct conflict with the objective of securing the network.

A further discussion related to using ML to find vulnerabilities in the network – potentially a powerful security technique for creating vulnerability assessment tools.

Given the challenges of explainability and vulnerability to manipulation, research is needed to understand what the right ML techniques are for SPS platforms; how to benefit from ML without introducing security issues.

2.7.4 Adaptation of ML for SPS

Due to the distributed location of data sources and differing capabilities of those sources (e.g. low power/computation), we should consider how existing ML techniques could be adapted for SPS. For example, should we separate the data processing or analysis phases to different locations within the SDI in order to increase algorithm efficiency? It might be appropriate to move the ML close to the sensors and decompose ML algorithms themselves to do relevant operations in the appropriate place.

The incorporation of ML into IDS is not necessarily new. However, ML could be used in SPS for taking a set of constraints and high-level user requirements and then using the flexibility of the system to optimize the design (specifically security design). SPS could also be used to support performance and timely ML with fine-grained control. One of the main benefits of SPS is fine-grained control.

2.7.5 Formal methods meet ML

It might be possible to use ML for the synthesis aspect of generating security policies. Currently, the process begins with logical statement of security policies followed by development of lower level implementation. With ML, we could synthesize security policies related to the observed traffic traces, mining the relevant information from the network and freeing the operator from creating security invariants, which can be hard to envision.

2.7.6 Game Theory/Self-Optimizing Networks

Closing discussion centered on protection of the network as a game in which we aim to win the security of the network by tipping the balance of power towards the defender. With respect to strategies in game theory and reasoning about the attacker and defender actions in the network, we identified the link to reinforcement learning. With an RL approach to SPS, we could develop a responsive (self-optimizing) system in which we measure the impact of attack detection and build the learning model based on these results. With the visibility and fine-grained control in SPS, the defender is in the stronger position.

The implementation of this raises some research questions, e.g. How to set up the rules of the game? How to use reinforcement learning to achieve this self-optimized secure network? A final link is made here from game theory to the discussion of adversarial attacks. In order to win the game, a mixed strategy is required to avoid attackers guessing the strategy for attack detection based on knowledge/access to the data and the system.

2.8 Emerging Platforms/Domains: Edge Computing, IoT/CPS, 5G

The Internet of Things (IoT) is comprised of many types of device domains including smart homes, wearables, industrial IoT, vehicular IoT, medical IoT, agricultural IoT, smart cities, etc. It is often useful to target a subset of these verticals when addressing security challenges, as not all challenges apply to all verticals. Broadly, IoT introduces unique challenges for software-defined infrastructures, including (1) edge computing and distributed control, (2) limited API interfaces, (3) heterogeneity of technologies, (4) wireless mesh systems, (5) pervasive and passive sensing of privacy sensitive activities, and (6) interactions with the physical world.

Edge computing gives IoT systems (with limited computational resources but with the need for obtaining real-time computations) the necessary advantages for achieving their goals; for example, streaming K-means can give real-time response for data reduction, and the quality of the results can then be revised by global K-means algorithms. However, edge computing and other distributed systems challenge the conventional expectations in software-defined systems where there is a centralized control system orchestrating the infrastructure. While on-premise computing resources give added resilience (e.g., availability in case there are latency or connection problems to the cloud), it also increases the complexity of managing a software-defined infrastructure. Security policies will increase in complexity because creating and enforcing security policies in a distributed and potentially disconnected system will make it harder to satisfy global policies such as GDPR.

Another challenge for interfacing with IoT devices is the minimal APIs these devices provide to software developers. The diversity of IoT devices also make these general abstractions difficult; for example, the abstractions for a Kinect sensor will be very different than those from wearable devices. The hope is that within a single domain (e.g., a smart home, or an industrial IoT setting) we might be able to have a consensus for these abstractions. Smart home automation platforms are an example of software-defined IoT. Recent work in this domain has demonstrated the need to study credential management, access control, and trigger-action rule combinations with negative consequences.

Expanding on the last point, the heterogeneity of IoT technologies is a challenge for designing security solutions; however, software-defined systems might allow a simplification of this diversity by providing a unified interface for interacting with a variety of IoT systems. One of the challenges for providing these general interfaces is that any abstraction will need to take into account the computational and battery limitations of embedded systems. However, not all devices have the same limitations, which may enable computational- and energy-aware architectures.

Some IoT deployments such as smart cities, industrial wireless networks, and the smart metering infrastructure have large wireless mesh networks. Most of the work for software-defined networks has focused on wired connections, but is there a need to have a software-defined mesh network in wireless? Another important point for security considerations is that several IoT devices have multimodal communications (e.g., they may have a Wi-Fi and a Zigbee radio) and unchecked wireless communications might be leveraged by attackers.

In addition to security, IoT deployments can have significant privacy implications: because IoT devices sense activity passively, their users might not be aware of their privacy exposure. In general, sensors collect more information than needed for a given application, so there is a need to work on

data minimization in a way that allows the data collected to be useful by the developers, while at the same time removing data that is not needed for the functionality. Edge computing might help with data minimization as less data is sent to the cloud.

Finally, IoT systems interact with the physical world and this opens new security opportunities. For example, suspicious activity can be detected by a physics-based intrusion detection system. We can also try to prevent malicious actions by having a policy in an actuator to allow or prevent certain actions. Software-defined infrastructures can also help with incident response in industrial control systems; because attack-detection and response in these settings needs to be done in real-time and (in some cases) without a human in the loop, software-defined infrastructures can help us automatically reconfigure the industrial system during an attack.

3 Conclusion

This workshop brings together networking, systems, and security researchers to discuss and establish a vision for programmable security in modern software-defined infrastructures (e.g., cloud, IoT, or edge computing environments). The output of the workshop is this public report documenting the discussions and recommendations on research directions and frontiers. It is the intention of the organizing committee and sponsors that the workshop and report stimulate research collaboration and the creation of a common research vision between disparate communities.

Overall, workshop participants will help to build community and define the vision for a new generation of security technologies in the rapidly expanding world of software-defined infrastructures and devices.

Acknowledgments

This workshop is supported by the the National Science Foundation (NSF) under Grant no. CNS-1841099. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of NSF.

References

[1] Defenseflow: Sdn applications and ddos attack defense. http://www. radware.com/Products/DefenseFlow/.

[2] Openflow: Innovate your network. http://www.openflow.org.

[3] Karthikeyan Bhargavan, Barry Bond, Antoine Delignat-Lavaud, Cdric Fournet, Chris Hawblitzel, Catalin Hritcu, Samin Ishtiaq, Markulf Kohlweiss, Rustan Leino, Jay Lorch, Kenji Maillard, Jinyang Pang, Bryan Parno, Jonathan Protzenko, Tahina Ramananandro, Ashay Rane, Aseem Rastogi, Nikhil Swamy, Laure Thompson, Peng Wang, Santiago Zanella-Beguelin, and Jean-Karim Zinzindohou. Everest: Towards a verified, drop-in replacement of HTTPS. In *Summit on Advances in Programming Languages (SNAPL)*, May 2017.

[4] S. K. Fayaz, Y. Tobioka, V. Sekar, and M. Bailey. Bohatei: Flexible and elastic ddos defense. In *Proceedings of The 26th USENIX Security Symposium (Usenix Security)*, June 2015.

[5] Guofei Gu, Hongxin Hu, Eric Keller, Zhiqiang Lin, and Donald Porter. Building a security os with software defined infrastructure. In *Proceedings of the Eighth ACM SIGOPS Asia-Pacific Workshop on Systems (APSys'17*, September 2017.

[6] S. Hong, R. Baykov, L. Xu, S. Nadimpalli, and G. Gu. Towards sdn-defined programmable byod (bring your own device) security. In *Proceedings of the 22th Annual Network and Distributed System Security Symposium (NDSS)*, February 2016.

[7] S. Hong, L. Xu, H. Wang, and G. Gu. Poisoning network visibility in software-defined networks: New attacks and countermeasures. In *Proceedings of the 22th Annual Network and Distributed System Security Symposium (NDSS)*, February 2015.

[8] H. Hu, W. Han, G. Ahn, and Z. Zhao. Flowguard: Building robust firewalls for software-defined networks. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking (HotSDN)*, August 2014.

[9] S. Jero, W. Koch, R. Skowyra, H. Okhravi, C. Nita-Rotaru, and D. Bigelow. Identifier binding attacks and defenses in software-defined networks. In *Proceeding of the 24th USENIX Security Symposium (USENIX Security)*, August 2017.

[10] S. Lee, C. Yoon, C. Lee, S. Shin, V. Yegneswaran, and P. Porras. Delta: A security assessment framework for software-defined networks. In *Proceedings of The 2017 Network and Distributed System Security Symposium (NDSS)*, February 2017.

[11] Hongda Li, Hongxin Hu, Guofei Gu, Gail-Joon Ahn, and Fuqiang Zhang. 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.

[12] A. Nayak, A. Reimers, N. Feamster, and R. Clark. Resonance: Dynamic access control for enterprise networks. In *Proceedings of the 1st ACM Workshop on Research on Enterprise Networking*, August 2009.

[13] P. Porras, S. Shin, V. Yegneswaran, M. Fong, M. Tyson, and G. Gu. A security enforcement kernel for openflow networks. In *Proceedings of ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN)*, August 2012.

[14] S. Shin, P. Porras, V. Yegneswaran, M. Fong, G. Gu, and M. Tyson. Fresco: Modular composable security services for software-defined networks. In *Proceedings of the 20th Annual Network and Distributed System Security Symposium (NDSS)*, February 2013.

[15] S. Shin, Y. Song, T. Lee, S. Lee, J. Chung, P. Porras, V. Yegneswaran, J. Noh, and B. B. Kang. Rosemary: A robust, secure, and high-performance network operating system. In *Proceedings of the 21th ACM Conference on Computer and Communications Security (CCS)*, November 2014.

[16] S. Shin, V. Yegneswaran, P. Porras, and G. Gu. Avant-guard: Scalable and vigilant switch flow management in software-defined networks. In *Proceedings of the 20th ACM Conference on Computer and Communications Security (CCS)*, November 2013.

[17] Seungwon Shin, Haopei Wang, and Guofei Gu. A first step towards network security virtualization: From concept to prototype. *IEEE Transactions on Information Forensics and Security*, 10(10), 2015.

[18] Sooel Son, Seungwon Shin, Vinod Yegneswaran, Phillip Porras, and Guofei Gu. Model checking invariant security properties in openflowproc. In *Proceedings of 2013 IEEE International Conference on Communications (ICC'13)*, June 2013.

[19] Haopei Wang, Lei Xu, and Guofei Gu. Floodguard: A dos attack prevention extension in software-defined networks. In *Proceedings of the 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, June 2015.

[20] Haopei Wang, Guangliang Yang, Phakpoom Chinprutthiwong, Lei Xu, Yangyong Zhang, and Guofei Gu. Towards fine-grained network security forensics and diagnosis in the sdn era. In *Proc. of the 25th ACM Conference on Computer and Communications Security (CCS'18)*, October 2018.

[21] L. Xu, J. Huang, S. Hong, J. Zhang, and G. Gu. Attacking the brain: Races in the sdn control plane. In *Proceedings of The 26th USENIX Security Symposium (Usenix Security)*, August 2017.

[22] Changhoon Yoon, Seungsoo Lee, Heedo Kang, Taejune Park, Seungwon Shin, Vinod Yegneswaran, Phillip Porras, and Guofei Gu. Flow wars: Systemizing the attack surface and defenses in software-dened networks. *EEE/ACM Transactions on Networking (ToN)*, 2017.

[23] C. Yu, C. Lumezanu, V. Singh, Y. Zhang, G. Jiang, and H. V. Madhyastha. Flowsense: Monitoring network utilization with zero measurement cost. In *Proceedings of the 14th International Conference on Passive and Active Measurement (PAM)*, 2013.

[24] M. Yu, L. Jose, and R. Miao. Software defined traffic measurement with opensketch. In *Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, April 2013.

[25] Menghao Zhang, Guanyu Li, Lei Xu, Jun Bi, Guofei Gu, and Jiasong Bai. Control plane reflection attacks in sdns: New attacks and countermeasures. In *Proc. of the 21st International Symposium on Research in Attacks, Intrusions and Defenses (RAID'18)*, September 2018.

[26] Jing Zheng, Qi Li, Guofei Gu, Jiahao Cao, David K.Y. Yau, and Jianping Wu. Realtime ddos defense using cots sdn switches via adaptive correlation analysis. *IEEE Transactions on Information Forensics and Security (TIFS)*, 2018.

4 Appendix

4.1 Workshop Participants

4.1.1 Academia

- Gail-Joon Ahn, Arizona State University
- Ehab Al-Shaer, UNC Charlette
- Alvaro Cardenas, UT Dallas
- Ang Chen, Rice
- Yan Chen, Northwestern
- William Enck, NCSU
- Carol Fung, Virginia Commonwealth University
- Hongxin Hu, Clemson
- Dijiang Huang ,Arizona State
- Kevin Jin, llinois Institute of Technology
- Qi Li, Tsinghua University
- Zhiqiang Lin, Ohio State
- Cristina Nita-Rotaru, Northeastern
- Younghee Park, San Jose State
- Don Porter, UNC
- Mike Reiter, UNC
- Byrav Ramamurthy, Nebraska (Lincoln)
- Sandra Scott-Hayward, Queen's University, UK
- Srini Seshan, CMU
- John Sonchack, Univ. of Pennsylvania
- Chia-Che Tsai, Berkeley/TAMU
- Michael Swift, Univ of Wisconsin, Madison
- Anduo Wang, Temple University, USA
- Xiaofeng Wang, Indiana Univ

4.1.2 Government

- Wenji Wu, Fermilab
- Sukarno Mertoguno, ONR
- Darleen Fisher, NSF/CNS
- Sandip Kundu, NSF/CNS
- Mimi McClure, NSF/CNS
- Matt Mutka, NSF/CNS

4.1.3 Industry

- Johanna Amann, ICSI
- Vinod Yegneswaran, SRI International
- Phillip Porras, SRI International
- Dennis Moreau, VMware
- Chris Shenefiel, Cisco
- Jason Li, Intelligent Automation Inc.
- Lewis Shepherd, VMware
- Robert Ames, VMware

4.1.4 Chairs

- Guofei Gu, TAMU
- David Ott, VMware
- Vyas Sekar, CMU
- Kun Sun, GMU