Towards Secure and Dependable Software-Defined Networks

Diego Kreutz
kreutz@lasige.di.fc.ul.pt

Fernando M. V. Ramos
fvramos@fc.ul.pt

Paulo Verissimo
pjv@di.fc.ul.pt

LaSIGE/FCUL, University of Lisbon, Portugal

ABSTRACT

Software-defined networking empowers network operators with more flexibility to program their networks. With SDN, network management moves from codifying functionality in terms of low-level device configurations to building software that facilitates network management and debugging. By separating the complexity of state distribution from network specification, SDN provides new ways to solve long-standing problems in networking — routing, for instance — while simultaneously allowing the use of security and dependability techniques, such as access control or multi-path.

However, the security and dependability of the SDN *itself* is still an open issue. In this position paper we argue for the need to build secure and dependable SDNs *by design*. As a first step in this direction we describe several threat vectors that may enable the exploit of SDN vulnerabilities. We then sketch the design of a secure and dependable SDN control platform as a materialization of the concept here advocated. We hope that this paper will trigger discussions in the SDN community around these issues and serve as a catalyser to join efforts from the networking and security & dependability communities in the ultimate goal of building resilient control planes.

Categories and Subject Descriptors

C.2.4 [**Distributed Systems**]: Network Operating Systems

Keywords

Security, Dependability, SDN, Threat Vectors, Controllers

1. INTRODUCTION AND MOTIVATION

Operating and maintaining a computer network is an arduous task. To express the required high-level network policies, network operators need to configure each individual network device separately — from a heterogeneous collection of switches, routers, middleboxes, etc. — using vendor-specific and low-level commands. In addition to configuration complexity, networks are dynamic, and operators have little or no mechanisms to automatically respond to network events. It is therefore difficult to enforce the required policies in such a continually changing environment.

With the separation of the control plane from the data plane that lays the ground to the Software Defined Networking paradigm, network switches [1] become simple forwarding devices and the control logic is implemented in a logically centralized controller — though in principle physically distributed [1]. In SDN, the controller is the entity that dictates the network behavior. The logical centralization of the control logic in a software module that runs in a standard server — the network operating system [2] — offers several benefits. First, it is simpler and less error-prone to modify network policies through software, than via low-level device configurations. Second, a control program can automatically react to spurious changes of the network state and thus maintain the high-level policies in place. Third, the centralization of the control logic in a controller with global knowledge of the network state simplifies the development of more sophisticated network functions. This ability to *program* the network in order to control the underlying data plane is therefore the crucial value proposition of SDN.

SDN provides new ways to solve age-old problems in networking (e.g., routing [3]) while simultaneously enabling the introduction of sophisticated network policies, such as security and dependability. An example is Ethane [4], an SDN architecture that allows managers to enforce fine-grained access control policies. However, the security and dependability of the SDN *itself* has been a neglected topic up to now. Only very recently has the community started to address these issues. Porras *et al.* [5] provided a security enforcement kernel in SDN controllers to allow security based prioritization. By extending the original idea, the same authors proposed FRESCO [6], a framework to ease the development and deployment of security applications in SDN. Since many commercial switches now support the OpenFlow [2] protocol [7], this technology has been deployed in a number of production networks. Security and dependability issues are therefore becoming a serious concern for the industry [8, 9].

Curiously, the main causes of concern lie in SDN's main benefits: network programmability and control logic centralization. These capabilities actually introduce new *fault and*

[1] In this paper, we use the term switch(es) as an example of an SDN data plane device.

[2] The open interface between networking devices and the controller that is currently the enabler for the most common materialization of an SDN.

HotSDN'13, August 16, 2013, Hong Kong, China.

attack planes, which open the doors for new threats that did not exist before or were harder to exploit. Traditional networks have "natural protections" against what would be common vulnerabilities in regular IT systems. Namely, the closed (proprietary) nature of network devices, their fairly static design, the heterogeneity of software, and the decentralized nature of the control plane represent defenses against common threats. For example, an attack exploiting a peculiar vulnerability of a specific set of devices from a single vendor would potentially harm only part of the network. This diversity is comparatively smaller in SDNs. A common standard (e.g., OpenFlow) among vendors and clients can also increase the risk, by the possible introduction of common faults in compliant implementations of the protocols and the control plane software. An attack similar to Stuxnet [10], a well-designed worm targeting very specific networked infrastructures, said to have impaired the operation of hundreds of those devices by modifying their control programs and configurations automatically, could have dramatic consequences in a highly configurable and programmable network. It is more than likely that such targeted attacks, also called *advanced persistent threats* [11], will be developed against SDNs, if the opportunity of success presents itself.

In summary, as we point out in this paper, SDNs bring a very fascinating dilemma: an extremely promising evolution of networking architectures, versus a dangerous increase in the threat surface. We should preserve the benefits of the first, by counteracting the dangers of the second. Therefore we argue, in this position paper, for the need to consider security and dependability as first class properties of future SDNs, which should be built into the design from the first hour and not bolted on.

We continue by describing several threat vectors that may enable the exploit of SDN vulnerabilities in Section 2. In Section 3 we investigate the open questions and sketch the design of a secure and dependable SDN control platform as an example materialization of the concept here advocated. In section 4 we conclude the paper.

2. THREAT VECTORS

Software-defined networks have two properties which can be seen as attractive honeypots for malicious users and a source of headaches for less prepared network operators. First, the ability to control the network by means of software (always subject to bugs and a score of other vulnerabilities). Second, the centralization of the "network intelligence" in the controller(s). Anyone with access to the servers that host the control software can potentially control the entire network.

In this section we describe the seven main potential threat vectors we identified in SDNs (Figure 1). Our goal is not to use these potential problems to claim that software-defined networks are inherently less secure than current networks. What we argue is that SDNs pose threats of a *different nature* that need therefore to be dealt with differently. On the contrary, if the SDN is properly designed and deployed, we believe this new network environment will definitely present a quantum leap in network architecting, not only in functionality but also in resilience.

Threat vector 1: forged or faked traffic flows, which can be used to attack switches and controllers. This threat can be triggered by faulty (non-malicious) devices or by a

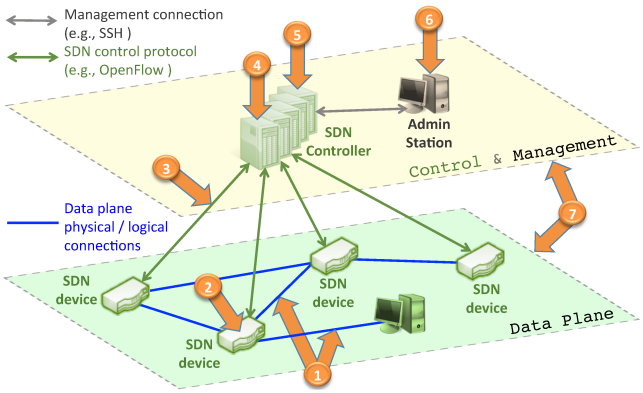

Figure 1: SDN main threat vectors map

malicious user. An attacker can use network elements (e.g., switches, servers, or personal computers) to launch a DoS attack against OpenFlow switches (e.g., targeting to exhaust TCAMs) and controller resources. A simple authentication mechanism could mitigate the problem, but if an attacker assumes the control of an application server that stores the details of many users, it can easily use the same authenticated ports and source MAC addresses to inject authorized, but forged, flows into the network.

Possible solutions: The use of intrusion detection systems with support for runtime root-cause analysis could help identify abnormal flows. This could be coupled with mechanisms for dynamic control of switch behavior (e.g., rate bounds for control plane requests).

Threat vector 2: attacks on vulnerabilities in switches, which can easily wreak havoc with the network. One single switch could be used to drop or slow down packets in the network, clone or deviate network traffic (e.g., for data theft purposes), or even inject traffic or forged requests to overload the controller or neighboring switches.

Possible solutions: The use of mechanisms of software attestation, such as autonomic trust management solutions for software components [12], is a possible mitigating factor. Mechanisms to monitor and detect abnormal behavior of network devices can also be useful to defeat this kind of threats.

Threat vector 3: attacks on control plane communications, which can be used to generate DoS attacks or for data theft. As is well-known in the security community, using TLS/SSL does not per se guarantee secure communication, and that compromises the controller–device link. Various papers report the weaknesses of TLS/SSL communications and its major anchor of trust, the PKI infrastructure [13]. The security of those communications is as strong as its weakest link, which could be a self-signed certificate, a compromised Certificate Authority, or vulnerable applications and libraries. For instance, there are many man-in-the-middle vulnerable implementations of SSL being used in world-wide critical systems [14]. Moreover, the TLS/SSL model is not enough to establish and assure trust between controllers and switches. Once an attacker gains access to the control plane, it may be capable of aggregating enough power force (in terms of the number of switches under its control) to launch DDoS attacks. This lack of trust guarantees could even enable the creation of a virtual black

hole network (e.g., by using OpenFlow-based slicing techniques [15]) allowing data leakage while the normal production traffic flows.

Possible solutions: The use of oligarchic trust models with multiple trust-anchor certification authorities (e.g., one per sub-domain or per controller instance) is a possibility. Another is securing communication with threshold cryptography across controller replicas [16] (where the switch will need at least n shares to get a valid controller message). Additionally, the use of dynamic, automated and assured device association mechanisms may be considered, in order to guarantee trust between the control plane and data plane devices.

Threat vector 4: attacks on and vulnerabilities in controllers, which are probably the most severe threats to SDNs. A faulty or malicious controller could compromise an entire network. The use of a common intrusion detection system may not be enough, as it may be hard to find the exact combination of events that trigger a particular behavior and, more importantly, to label it as malicious. Similarly, a malicious application can potentially do anything it pleases in the network, since controllers only provide abstractions that translate into issuing configuration commands to the underlying infrastructure.

Possible solutions: Several techniques can be used, such as replication (to detect, remove or mask abnormal behavior), employing diversity (of controllers, protocols, programming languages, software images, etc.), and recovery (periodically refreshing the system to a clean and reliable state). It is also important to secure all the sensitive elements inside the controller (e.g., crypto keys/secrets). Furthermore, security policies enforcing correct behavior might be mapped onto those techniques, restricting which interfaces an application can use and what kind of rules it can generate to program the network (along the lines of security-based prioritization [5]).

Threat vector 5: lack of mechanisms to ensure trust between the controller and management applications, whereby similarly to threat number 3, controllers and applications lack the ability to establish trusted relationships. The main difference from the referred threat would lie in the way the certification is made. The techniques used to certify network devices are different from those used for applications.

Possible solutions: Mechanisms for autonomic trust management could be used to guarantee that the application is trusted during its lifetime.

Threat vector 6: attacks on and vulnerabilities in administrative stations which, as it is also common in traditional networks, are used in SDNs to access the network controller. These machines are already an exploitable target in current networks, the difference being that the threat surface as seen from a single compromised machine increases dramatically in SDNs. It becomes easy to reprogram the network from a single location.

Possible solutions: The use of protocols requiring double credential verification (e.g., requiring the credentials of two different users to access a control server). Also, the use of assured recovery mechanisms to guarantee a reliable state after reboot.

Threat vector 7: lack of trusted resources for forensics and remediation, which would allow to understand the cause of a detected problem and proceed to a fast and secure mode recovery. In order to investigate and establish facts about an incident, we need reliable information from all components and domains of the network. Furthermore, this data will only be useful if *its* trustworthiness (integrity, authenticity, etc.) can be assured. Similarly, remediation requires safe and reliable system snapshots to guarantee a fast and correct recovery of network elements to a known working state.

Possible solutions: Logging and tracing are the common mechanisms in use, and are needed both in the data and control planes. However, in order to be effective, they should be indelible (a log that is guaranteed to be immutable and secure). Furthermore, logs should be stored in remote and secure environments.

Table 1: SDN specific vs non-specific threats

Threats	Specific to SDN?	Consequences in SDN
Vector 1	no	can be a door for DoS attacks
Vector 2	no	but now the impact is potentially augmented
Vector 3	yes	communication with logically centralized controllers can be explored
Vector 4	yes	controlling the controller may compromise the entire network
Vector 5	yes	malicious applications can now be easily developed and deployed on controllers
Vector 6	no	but now the impact is potentially augmented
Vector 7	no	it is still critical to assure fast recovery and diagnosis when faults happen

Table 1 summarizes the seven threat vectors and includes information about its specificity to SDN. As can be observed, threats number 3, 4, and 5, are not present in traditional networks. They are specific to SDNs as they arise from the separation of the control and data planes and the consequent introduction of a new entity in these networks — the logically centralized controller. On the other hand, threat vectors 1, 2, 6, and 7 were already present in traditional networks. However, the impact of these threats may be potentially augmented — or at least it may be expressed differently — and as a consequence it may need to be dealt with dissimilarly.

These seven threats we identified show that the attack surface for SDNs is augmented, when compared to traditional networks, and that different mitigation techniques need to be put in place. We believe this offers strong arguments for the need to consider security and dependability since the first steps of the design phase. With that motivation, in the next section we sketch the design of a secure and dependable control platform that tries to address several of these threats.

3. SECURITY & DEPENDABILITY IN SDN

In this section we present some background on security and dependability and discuss mechanisms and techniques to be considered on the design of a secure and dependable control platform.

3.1 Background

To the best of our knowledge, none of the SDN controllers proposed thus far address security and dependability beyond using simple authenticated communication channels and control data replication among controller instances. For example, no mechanisms are used to assure trusted switch-controller association (to avoid malicious devices in the network) or to detect, correct or mask faults of system components. Moreover, no techniques are used to assure data integrity and confidentiality in or between controllers.

In a security and dependability perspective, one of the key ingredients to guarantee a highly robust system is fault and intrusion tolerance. The two main fault models are crash and Byzantine (a.k.a., arbitrary faults). Crash fault tolerant services support only benign failures such as a crashed process, operating system or machine, being a narrow subset of the arbitrary model. Byzantine fault tolerant (BFT) systems are capable of tolerating any abnormal behavior, i.e., intentional or non-intentional faults, while the service keeps its correct operation. Faults (e.g., bugs, misconfigurations, attacks) and errors can be masked automatically as they happen, by using state machine replication [17]. Furthermore, in order to ensure the perpetual and unattended operation of the system, errors can be removed with self-healing techniques [18], so that there is never an excessive number of compromised devices. Both automatic recovery and perpetual and unattended operation seem to be relevant objectives in the context of SDNs.

The literature on Byzantine fault tolerance is broad, ranging from large-scale systems [19, 20] to resource-efficient solutions [19, 21, 22]. Nevertheless, BFT alone is not enough to guarantee a highly available dependable system, needing self-healing mechanisms as a complement. Techniques such as proactive-reactive recovery [18], for example, can be used to assure the system liveness. These techniques rely on the idea of rejuvenating compromised components (be it by accidental or malicious faults).

Intrusion-tolerant architectures [23] are a step in the direction of this *automatic security* paradigm. Intrusion-tolerant systems remain working correctly and are capable of assuring properties such as integrity, confidentiality and availability, despite the presence of faulty or compromised components due to successful attacks.

A secure and dependable control plane helps improve the overall network resilience [24], which is our final goal. A resilient system is one that self-adapts to the dynamics of environment conditions, e.g., one that performs self-healing in the presence of persistent threats and where protection parameters, such as number of replicas, length of keys, etc., can automatically increase in case of a severe attack.

3.2 Secure and Dependable Control Platform

In this section we present the general design of the secure and dependable SDN control platform we propose. Figure 2 illustrates a simplified view of the architecture. In the remainder of this section we briefly introduce and discuss the several mechanisms which we consider using to address the threat vectors identified in SDNs.

Replication. One of the most important techniques to improve the dependability of the system is replication. As can be seen in figure 2, our controller is replicated, with three instances in the example. Applications should be replicated as well. Besides replicated instances of the controller, in the figure we can observe application B also replicated in all controller instances. This mixed approach ensures tolerance of both hardware and software faults, accidental or malicious. Replication makes it possible to mask failures and to isolate malicious or faulty applications and/or controllers. Moreover, in case of a network partition, application B, with the proper consistency algorithms, will still be able to program all network switches, contrary to application A.

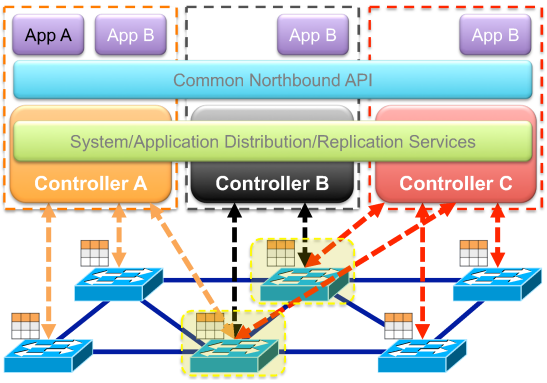

Figure 2: Secure & Dependable SDN

Diversity. Another relevant technique to improve the robustness of secure and dependable systems is diversity [25, 26]. Replication with diverse controllers is a good starting case. The basic principle behind this mechanism is to avoid common-mode faults (e.g., software bugs or vulnerabilities). For example, it is known that off-the-shelf operating systems, from different families, have few intersecting vulnerabilities [26], which means that OS diversity constrains the overall effect of attacks on common vulnerabilities. In SDNs the same management application could run on different controllers. This can be simplified by defining a common abstraction for applications (a northbound API).

Self-healing mechanisms. Under persistent adversary circumstances, proactive and reactive recovery can bring the system back to a healthy state, replacing compromised components, and keep it working virtually forever. When replacing components, it is important that the replacement be done with new and diverse versions of the components, whenever possible. In other words, we should explore diversity in the recovery process, strengthening the defense against attacks targeting specific vulnerabilities in a system.

Dynamic device association. If a switch is associated with a single controller, its control plane does not tolerate faults. Once the controller fails, the control operation of the switch fails and the switch will need to associate with another controller. For this reason, a switch should be able to dynamically associate with several controllers in a secure way (e.g., by using threshold cryptography to detect malicious controllers and authentication, which would hinder man-in-the-middle attacks, for instance). A switch associated with different controllers would be able to automatically tolerate faults (crash or Byzantine, depending on the configuration). Other advantages include increasing control plane throughput (several controllers could be used for load balancing) and reducing control delay [27] by choosing the quickest-responding controller.

Increasing the data plane programmability (near or in the network switches) would be helpful in this respect. Two

approaches could be used for this purpose. One option would be to use general purpose CPUs inside the switch to replace some of the traditional functionality of custom ASIC, as in [28]. Another could be to have a proxy element acting on behalf of the switch. This element could be easily deployed in a small black box attached to the switch, with a general purpose micro-computer.

Trust between devices and controllers. Establishing trust between devices and controllers is an important requirement for overall control plane trustworthiness. Network devices should be allowed to associate with controllers dynamically but without incurring in less reliable relationships. A simple approach would be to have authenticated white lists of known trusted devices, kept at controllers. However, this lacks the flexibility desired in an SDN. Another option is therefore to trust all switches until its trustworthiness is questioned. Malicious or abnormal behavior could be reported by other switches or controllers, based on anomaly or failure detection algorithms. Once the trustworthiness of a switch or a controller would go below an accepted threshold, the switch would be automatically quarantined by all devices and controllers.

Trust between applications and controllers software. As software components present changing behavior due to aging, exhaustion, bugs, or attacks, a dynamic trust model as the one proposed in [12] is required. In this paper the authors propose and demonstrate the feasibility of a model to support autonomic trust management in component-based software systems. They use a holistic notion of trust to allow a trustor to assess the trustworthiness of the trustee by observing its behavior and measuring it based on quality attributes, such as availability, reliability, integrity, safety, maintainability, and confidentiality. The proposed model can also be applied to define, monitor, and ensure the trustworthiness of relationships among system entities.

Security domains. Isolated security domains are a very common technique used in different types of systems. For instance, in operating systems user level applications are not allowed to access kernel level sub-systems. A recent example of effective security domains by design is Chromium [29]. Similarly to operating systems sandboxes, Chromium uses sandboxes to isolate the rendering engine from the browser kernel. Thus, most of the attacks will affect only the rendering engine and not the system kernel. Security domains in SDN control platforms can be explored using techniques such as sandboxing and virtualization. These techniques enable the design of strong isolation modes, through well-defined interfaces that allow minimal (only restricted and strictly necessary) set of operations and communication between different domains.

Secure components. These components are one of the essential building blocks of a secure and dependable system. Secure elements can be used, for example, to provide trusted computing bases (TCB) to assure security properties such as confidentiality. A TCB is a tamper-proof device that can be used to store sensitive security data (e.g., crypto private keys) and execute basic operations on it. Thus, even if the system is compromised, sensitive data will have its confidentiality assured.

Fast and reliable software update and patching. As no software is free from bugs (or other vulnerabilities), fast and reliable software patching and update is essential to reduce the window of vulnerabilities. Thus, a control platform should be deployed with mechanisms to assure a smooth and secure way of doing updates. Solutions as those proposed in [30] can help in achieving this goal.

To summarize, in Table 2 we identify each of the threat vectors that may be mitigated with the use of the solutions and mechanisms explored in this section.

Table 2: solutions to threat vectors

Solution/mechanism	Threat vectors
Replication	1, 4, 5, 7
Diversity	3, 4, 6
Self-healing	2, 4, 6
Dynamic switch association	3, 4
Trust between controllers & devices	1, 2, 3
Trust between controllers & apps	4, 5
Security domains	4, 5
Secure components	4, 5, 7
Fast and reliable update & patching	2, 4, 6

The solutions just discussed form the core mechanisms of what we are considering in our implementation of a secure and dependable control platform. We nevertheless believe such designs may benefit from the use of traditional techniques, such as firewalls or intrusion detection systems, and novel tools to specify and compose packet-forwarding policies (and updating them in a consistent way) [31] and to check network-wide invariants in real time [32]. These are in fact some avenues we are still exploring.

3.3 Security and Dependability by Design

We now advocate security and dependability by design through an example. Suppose that we have three **replicated** controllers to keep our network in a healthy state. If one controller is buggy or gets compromised we still have two potentially correct controllers. This will be true if controllers are designed in order that they can be easily replicated, are capable of interoperating and providing support to execute applications across controllers. To achieve these characteristics, we need common interfaces for controller integration and interoperation (e.g., three different controllers working together in a same environment), common northbound APIs, and common replication capabilities. In addition to this, the switches will also need to be able to **dynamically associate** with more than one controller. Finally, if we rely on a single controller make for replication, bugs and abnormal behaviors have a high probability of affecting all instances in parallel. In this case, **diversity** helps improve the robustness of the system. In summary, by applying these three techniques — replication, diversity, dynamic switch association — in the design of our system, we are able to increase its security and dependability from the first hour.

3.4 Related work

Security and dependability of SDN still is a field almost unexplored, presenting many challenges and opportunities. There are only a few closely related works, namely [5] and [6]. Their essential idea is to provide a security kernel (e.g., by extending a controller like NOX) capable of ensuring prioritized flow rule installation on switches. Applications are classified in two types, one for security related applications and another for all remaining applications. The first type

represents specialized programs used to ensure security control policies in the network, such as to guarantee or restrict specific accesses to the network or take actions to control malicious data traffic. Flow rules generated by security applications have priority over the others. The security kernel is responsible for ensuring this behavior. FRESCO [6] is an extension of this work that makes it easy to create and deploy security services in software-defined networks. However, none of these works fosters or enforces the security of SDN itself, the goal we are pursuing.

4. CONCLUDING REMARKS

In this paper we argue for the need to consider security and dependability when designing Software Defined Networks. We have presented several threats identified in these networks as strong arguments for this need, together with a brief discussion of the mechanisms we are using in building a secure and dependable SDN control platform.

The novel concepts introduced by SDN are enabling a revolution in networking research. The know-how and good practices from several communities (databases, programming languages, systems) are being put together to help solve long-standing networking problems. We hope that this paper will trigger discussions in the SDN community around issues related to security and dependability, to serve as a catalyser of joint efforts in these critical issues.

5. ACKNOWLEDGMENTS

We would like to thank Vinicius Cogo for comments and the anonymous reviewers for their helpful feedback. This work is partially supported by the EC, through project SecFuNet FP7-ICT-STREP-288349, by FCT, through project TRONE CMU-PT/RNQ/0015/2009, and by CNPq, through grant 202104/2012-5.

6. REFERENCES

[1] T. Koponen et al. “Onix: a distributed control platform for large-scale production networks”. In: *OSDI*. 2010.

[2] N. Gude et al. “NOX: towards an operating system for networks”. In: *Comp. Comm. Rev.* (2008).

[3] M. Caesar et al. “Design and implementation of a routing control platform”. In: *NSDI*. 2005.

[4] M. Casado et al. “Rethinking Enterprise Network Control”. In: *ACM Trans. on Networking* 17.4 (2009).

[5] P. Porras et al. “A security enforcement kernel for OpenFlow networks”. In: *HotSDN*. ACM, 2012.

[6] S. Shin et al. “FRESCO: Modular Composable Security Services for Software-Defined Networks”. In: *Internet Society NDSS*. 2013.

[7] N. McKeown et al. “OpenFlow: enabling innovation in campus networks”. In: *Comp. Comm. Rev.* (2008).

[8] S. Sorensen. *Security implications of software-defined networks*. 2012. URL: `http://goo.gl/BiXH2`.

[9] S. M. Kerner. *Is SDN Secure?* 2013. URL: `http://goo.gl/lPn2V`.

[10] D. Kushner. *The Real Story of Stuxnet*. 2013. URL: `http://goo.gl/HIEHQ`.

[11] C. Tankard. “Advanced Persistent threats and how to monitor and deter them”. In: *Network Sec.* (2011).

[12] Z. Yan and C. Prehofer. “Autonomic Trust Management for a Component-Based Software System”. In: *IEEE Trans. on Dep. and Sec. Computing* 8.6 (2011).

[13] R. Holz et al. “X.509 Forensics: Detecting and Localising the SSL/TLS Men-in-the-Middle”. In: *Computer Security*. LNCS. 2012.

[14] M. Georgiev et al. “The most dangerous code in the world: validating SSL certificates in non-browser software”. In: *ACM CCS*. 2012.

[15] R. Sherwood et al. *FlowVisor: A Network Virtualization Layer*. Tech. rep. Deutsche Telekom Inc. R&D Lab, Stanford, Nicira Networks, 2009.

[16] Y. G. Desmedt. “Threshold cryptography”. In: *European Trans. on Telecommunications* 5.4 (1994).

[17] F. B. Schneider. “Implementing fault-tolerant services using the state machine approach: a tutorial”. In: *ACM Comput. Surv.* 22.4 (Dec. 1990).

[18] P. Sousa et al. “Highly Available Intrusion-Tolerant Services with Proactive-Reactive Recovery”. In: *IEEE Trans. Parallel Distrib. Syst.* 21.4 (2010).

[19] G. Veronese et al. “Efficient Byzantine Fault-Tolerance”. In: *IEEE Trans. on Computers* 62.1 (2013).

[20] G. Veronese et al. “EBAWA: Efficient Byzantine Agreement for Wide-Area Networks”. In: *IEEE HASE*. 2010.

[21] R. Kapitza et al. “CheapBFT: resource-efficient byzantine fault tolerance”. In: *ACM EuroSys*. 2012.

[22] J. Hendricks, G. R. Ganger, and M. K. Reiter. “Low-overhead byzantine fault-tolerant storage”. In: *SIGOPS Oper. Syst. Rev.* 41.6 (Oct. 2007).

[23] P. Verissimo et al. “Intrusion-tolerant middleware: the road to automatic security”. In: *IEEE Security & Privacy* 4.4 (2006).

[24] J. Korniak. “The GMPLS Controlled Optical Networks as Industry Communication Platform”. In: *IEEE Trans. on Industrial Informatics* 7.4 (2011).

[25] S. Neti, A. Somayaji, and M. E. Locasto. “Software diversity: Security, Entropy and Game Theory”. In: *7th USENIX HotSec*. 2012.

[26] M. Garcia et al. “Analysis of operating system diversity for intrusion tolerance”. In: *Software: Practice and Experience* (2013).

[27] B. Heller, R. Sherwood, and N. McKeown. “The controller placement problem”. In: *HotSDN*. 2012.

[28] J. C. Mogul and P. Congdon. “Hey, you darned counters!: get off my ASIC!” In: *HotSDN*. 2012.

[29] A. Barth et al. *The Security Architecture of the Chro-mium Browser*. Tech. rep. Stanford University, 2008.

[30] J. H. Perkins et al. “Automatically patching errors in deployed software”. In: *ACM SIGOPS SOSP*. 2009.

[31] N. Foster et al. “Frenetic: a network programming language”. In: *SIGPLAN Not.* (2011).

[32] A. Khurshid et al. “VeriFlow: verifying network-wide invariants in real time”. In: *HotSDN*. 2012.