Towards a Secure Controller Platform for OpenFlow Applications

Xitao Wen
Northwestern University
Evanston, IL
xw@u.northwestern.edu

Yan Chen
Northwestern University
Evanston, IL
ychen@northwestern.edu

Chengchen Hu
Xi'an Jiaotong University
Xi'an, China
huc@ieee.org

Chao Shi
Northwestern University
Evanston, IL
chaoshi1989@gmail.com

Yi Wang
Tsinghua University
Beijing, China
wy@ieee.org

ABSTRACT

The OpenFlow (OF) paradigm embraces third-party development efforts, and therefore suffers from potential trust issue on OF applications (apps). The abuse of such trust could lead to various types of attacks impacting the entire network. In this paper, we propose *PermOF*, a fine-grained permission system, as the first line of defense, in order to apply minimum privilege on apps. We summarize a set of 18 permissions to be enforced at the API entry of the controller. To accommodate the isolation requirements, we propose a customized isolation mechanism, which achieves comprehensive resource isolation and access control.

Categories and Subject Descriptors

C.2.6 [**Network Operations**]: Network Management

General Terms

Software-Defined Networking, Security

Keywords

OpenFlow; Security; Policy Enforcement

1. BACKGROUND

Standardized OF creates a great surface for control layer attacks as a result of the open interface and the involvement of multiple parties. OF controllers, such as NOX[3] and Floodlight[1], interface between the OF apps and the OS. The OF apps, which run upon OF controller and implement a majority of the functionalities of the control plane, are typically developed by third parties other than the controller vendor. When joining with the controller, the apps automatically inherit the privileges on manipulating the network behavior. Our survey suggest that all state-of-the-art OF platforms expose the full privilege of OF indiscriminately to every app without protection.

In contrast of the severity of the threats, the defense techniques have been only sparsely explored. Apparently, traditional perimeter protection is ineffective, because the controller is typically located within the perimeter. Network virtualization techniques such as FlowVisor[5] help to prevent the potential cross-slice attacks, but do not deliver protection against the attacks within a network slice. Recent OF-specific proposals, including FortNOX[4] and FRESCO[6], care primarily about the rule conflicts that violate existing security policies, which only covers a small subset of potential attack space.

The threat of third-party apps stems from the fundamental challenge to verify the trustworthiness of a program module[2]. Because apps directly interact with critical resources such as flow table and device configuration, people expect in apps the same level of trust as in the controller. However, although the controller platforms are well tested and verified, the quality and goodwill of a third-party component is generally difficult to guarantee on an open platform. We envision two threat models. First, a benign-but-buggy app could become the victim of various of network-based or host-based exploits, which enables the control-plane attacks ranging from information leakage to arbitrary code execution. Second, a malicious app could possibly pass the censorship (if there is any) and be deployed directly into a OF controller. As long as the attackers control an app through any of the above ways, they will have effectively full control of the OF network, leading to a series of control-plane attacks that are both lucrative and stealthy.

2. DESIGN

In this paper, we tackle the problem via a combination of permission system and runtime isolation to enforce least privilege on the level of OF app. We argue that enforcing security policies provides a *deterministic* and *low-cost* remedy for the app over-privilege problem, and meanwhile facilitate the other approaches by narrowing the scope of suspects. We propose *PermOF*, a fine-grained permission system containing a set of OF-specific permissions and the isolation mechanism to enforce the permissions. The main design concern is two-fold: 1) what is the most effective set of permissions? 2) what isolation mechanism should be deployed to enforce the permission control?

Permission Set Design. One critical design of a permission system is the permission set, i.e, the set of access control tokens that can be either granted or denied for an app. We design the permission set according to our understanding of four aspects: 1) the

HotSDN'13, August 16, 2013, Hong Kong, China.
ACM 978-1-4503-2178-5/13/08.

Category	Permission
Read	read_topology
	read_all_flow
	read_statistics
	read_pkt_in_payload
Notification	pkt_in_event
	flow_removed_event
	error_event
	topology_event
Write	flow_mod_route
	flow_mod_drop
	flow_mod_modify_hdr
	modify_all_flows
	send_pkt_out
	set_device_config
	set_flow_priority
System	network_access
	file_system_access
	process_runtime_access

Table 1: Permission Set Summary

threat models, 2) the control messages in OpenFlow protocol, 3) the general API set of the controller implementations, and 4) the functional requirements of the apps.

We summarize the resulting permission set in Table 1. Among the permission categories, *read permissions* manage the availability of sensitive OF information to an app; *notification permissions* manage whether an app should be notified of certain events in real-time; *write permissions* manage the ability to modify certain states of the controller or the switches; and finally, *system permissions* manage the app's access to the local resources provided by the OS, including network, storage, etc.

Isolation Mechanism. The permission set in Table 1, especially the system permissions, implies several requirements on the isolation mechanism. First, the isolation mechanism should maintain controller's conceptually superior role to apps. Then, the isolation of control flow and data should be established between controller and apps. Finally, controller should be able to mediate all the apps' activity with the outside world.

In order to provide such isolation, we propose an isolation framework as depicted in Figure 1. In our proposed system, controller and apps are isolated in thread containers. On one hand, apps are isolated from controller kernel in a sense that apps cannot call any kernel procedures or directly refer to kernel memory. We achieve it by carefully craft the kernel code, so that apps are not able to obtain object reference from the kernel memory. On the other hand, we introduce an access control layer between the apps and the OS. This shim layer is configured and controlled by the controller kernel, so that undesirable interaction between the apps and the OS will be cut off. This is achieved by modifying the dynamic library of the programming language or the OS.

Apps strictly follow a reactive programming paradigm. Apps are triggered by callback events and then react with a series of controller API calls. We expose controller API through the library attached to the thread container. We extend the thread class to include the controller API functions. Upon receiving an API call from the app, the thread class encapsulates the function call and passes it to kernel via the inter-thread communication facility. Since the API calls are attached with the caller's identity, the controller kernel can easily perform permission control based on the pre-configured policy.

3. FUTURE DIRECTIONS

Performance Trade-off. It is desirable for a security enforcement system to provide flexible and user-friendly policy language

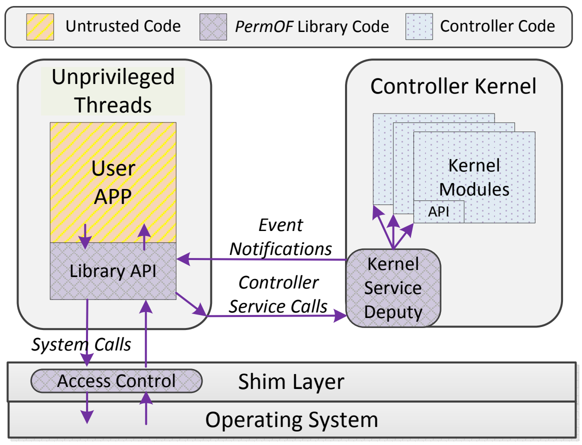

Figure 1: *PermOF* Isolation Framework

for permission composition. However, real world scenarios like data-center network have strict requirements on runtime latency and throughput of OF controllers, which greatly limits the complexity of a policy enforcement system. It is still an open question how to provide the "right" level of abstraction for a permission description language that strikes good trade-off between performance and usability.

Approaches Other Than Permission Control. Multiple levels of defense could also be effective to combat the threats from third-party apps and address the unique challenges in OF environment. First and foremost, the controller may enforce *security check before installation*. A PKI-based authentication should be enforced to ensure the app's authenticity and integrity. In addition, the controller vendor can enforce more advanced censorship with program analysis techniques or manual testing, similar to what is deployed on iOS AppStore. Finally, *anomaly-based behavior monitoring* can also facilitate both online malicious behavior detection and offline forensic analysis. There is a large body of research on behavior-based malware analysis that may be adapted to the OF environment.

4. CONCLUSION

With the involvement of third-party apps, OF controller is subject to the privilege abuse problem, which enables a series of control-plane attacks that could compromise the entire network. To eliminate such threat, we explore the space of potential solutions and then focus on minimizing the privilege of OF apps. We propose *PermOF* a fine-grained permission system that incorporates a customized permission set and a thread-based isolation mechanism.

5. REFERENCES

[1] Floodlight openflow controller. http://bit.ly/UIL73z.
[2] M. Canini, D. Venzano, P. Peresini, D. Kostic, and J. Rexford. A nice way to test openflow applications. In *USENIX NSDI'12*.
[3] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker. Nox: towards an operating system for networks. *ACM SIGCOMM CCR*, 38(3):105–110, 2008.
[4] P. Porras, S. Shin, V. Yegneswaran, M. Fong, M. Tyson, and G. Gu. A security enforcement kernel for openflow networks. In *HotSDN'12*.
[5] R. Sherwood, G. Gibb, K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar. Flowvisor: A network virtualization layer. *OpenFlow Switch Consortium, Tech. Rep*, 2009.
[6] S. Shin, P. Porras, V. Yegneswaran, M. Fong, G. Gu, and M. Tyson. Fresco: Modular composable security services for software-defined networks. In *NDSS'13*.