Advances in Computer Science Research (ACRS), volume 54
International Conference on Computer Networks and Communication Technology (CNCT2016)

HAuth : A Novel Approach for Network Visibility Protection

Xin WANG[1,2,*], Neng GAO[1] and Ling-chen ZHANG[1]

[1]State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Acoademy of Sciences, China

[2]University of Chinese Academy of Sciences, China

*Corresponding author

Keywords: SDN, Host Usurp Attack, Host Authentication

Abstract. *Software-Defined Networking (SDN) is a new paradigm that offers services and applications great power to manage network. Based on the consideration that the entire network visibility is the foundation of SDN, many attacks emerge in poisoning the network visibility, leading to severe host hijacking. Meanwhile, many defence approaches are proposed to patch the controller. We notice that existing patches missed considering the situation that original IP address of the host server would be hijacked when the host server goes offline temporarily, such as regular maintenance or host migration. In this paper we present Host Usurp Attack, which exploits the vulnerability above to pretend the victim server. Furthermore, we propose a security extension HAuth, which can automatically confirm the legitimate hosts through the authentication server and provides hosts authentication log to network providers. Our evaluation shows that HAuth effectively guarantees the trustworthiness of the network visibility. In particular, HAuth introduces a minor overhead on SDN controllers..*

Introduction

Software-Defined Networking (SDN) has emerged as a new network paradigm to innovate the ossified network infrastructure by separating the control plane from the data plane, as well as providing holistic network visibility and flexible programmability. As the brain of the network, an SDN controller provides users a great tool to design and control the network using their own applications atop the controller's core services. SDN, particularly its popular realization OpenFlow, has been increasingly employed in academic environment and commercial networks. With the widespread deployment, more and more applications have been studied to increase the efficiency of the network and improve the network security.

The logically centralized control function maintains the state of network and provides instructions for data plane. It takes advantage of the complete network view to analyze and correlate the feedback of packets from the network. Benefited from this mechanism, network can be run efficiently. Once the controller provides upper applications and services with wrong network state, these applications will not be trusted. Thus correct network state information is the prerequisite of SDN. Recently, many researchers [2,4] have noticed the attacks on topology management. In these attacks, host hijacking attacks are easy to implement and has great harm. Although these schemes proposed solutions, they are ineffectual for high capacity attacker.

In this paper, we study network topology network of the mainstream OpenFlow controllers and identify new vulnerabilities. Upon the exploitation of the Host Tracking Service, an attacker can take over the location of a network server to phish its service subscribers and even usurp the network server. The existing schemes mainly consider the situation that the network visibility will not be attacked when the host is online. But when an important server need be offline temporarily, the attacker will detect this situation and disguise himself as the offline server. Then OpenFlow controller will mistake attacker as the offline server because the attacker can pass all test of existing scheme. When origin server reboots, it may be mistaken as attacker by the protection scheme. Our new attacks share some similarities in spirit to traditional spoofing attacks in legacy networks (e.g., ARP Poisoning Attack), and we need effective solution to solve them. Besides, the existing schemes[2,4] are not a perfect system as they stands but they still catch duplicate host attachment events to detect the proposed attack. In addition, they have no way to share those trusted hosts among multiple controllers.

In order to mitigate such attack and improve the authentication efficiency in multiple controllers, we investigate possible defense strategies. We note that it is difficult to simply use static configuration to solve the problem, because it's tedious and error-prone manual effort and is not suitable for handling network dynamics, which is a valuable innovation of SDN. To better balance the security and usability, we propose HAuth, which uses digital certificate to authenticate network equipment automatically and shares authentication information among multiple controllers. By using HAuth, network administrators can easily manage the entire network access equipment and can prevent host hijacking attack.

Background

In this section, we provide an introduction to SDN/OpenFlow and its Host Tracking Service implemented in the existing OpenFlow controllers.

SDN/OpenFlow Background

Software Defined Networking is a new paradigm that decouples control plane of network device from its data plane and implement all control plane as a unified software platform which we call SDN controller. OpenFlow is a popular realization. In SDN architecture, the firmware, where data plane stands, forwards data traffic according to a set of rules specified by SDN controller. Different from legacy networks, SDN controller can hold the whole network topology visibility and make the best routing decision according to the topology information. Besides, we can easily configure security policies in centralized controller.

Host Tracking Service in OpenFlow Controllers

Inside an OpenFlow controller, Host Profile is maintained to track the location of a host. In an OpenFlow network, the OpenFlow controller reactively listens to Packet-In messages to maintain Host Profile. During this procedure, the OpenFlow controller mainly handles two relevant host events (i.e., JOIN and MOVE). The scenario for the first event is that, when the OpenFlow controller fails to locate an existing Host Profile according to the information from incoming Packet-In messages, it creates a new Host Profile. In such case, the controller assumes a new host joins the network. The second scenario occurs when the OpenFlow controller successfully locates a Host Profile but finds mismatched location information between the Host Profile and Packet-In messages. In the case, it assumes the host has moved to a new location and then updates the location information inside the corresponding Host Profile.

Host Usurp Attack

In this section, we describe the vulnerabilities in Host Tracking Services and detail the known Host Hijacking Attack. Then we propose and measure another attack unmentioned in previous work, i.e., Host Usurp Attack.

Vulnerability in Host Tracking Service

Host Tracking Service (HTS) in the OpenFlow controllers maintains Host Profile for each end host to track network mobility. As long as hosts (or virtual machines) migrate, HTS can quickly react to such event. In particular, HTS recognizes the motion of hosts by monitoring Packet-In messages. Once being aware that a particular host migrates to a new location, i.e., DPID or ingress Port ID is different from the corresponding entry of the Host Profile, HTS updates Host Profile and optionally raises a HOST_MOVE event to its subscriber services. However, such update mechanism is vulnerable due to the ignorance of authentication.

Using this vulnerability, Gu[4] first proposed Host Location Hijacking Attack. In the attack, the adversary will craft packet with the same identifier of the target host. Once receiving the spoofed packet, the controller will be tricked to believe that the target host has moved to a location, which actually is the attacker's location. As a result, future traffic to the target is hijacked by the adversary. In this procedure, the difficulty to implement attack is that the adversary needs to race with the target host, because any traffic initiated from the target host can correct host location information in the controller. But when the target host is a server, this attack can be easy to implement.

Host Usurp Attack

In this part, we introduce Host Usurp Attack which can usurp the host without being prevented by existing schemes .

In SDN when the target host is in the migration, attackers can easily launch Host Hijacking Attack because the origin location of attacked host is empty. When the target host re-connects the same net with original IP address, it is considered host attacker by TopoGuard because the position with this IP is occupied by attackers and the re-access of origin host is considered another host migration procedure. This is what we called Host Usurp Attack. In traditional OpenFlow controller we can launch Host Hijacking Attack directly, obviously we can launch Host Usurp Attack, too. If we continuously detect whether the target host is online and launch Host Usurp Attack when detecting the target host untouched, it is easy to launch Host Usurp Attack at this time.

In a LAN (Local Area Network), network policy is always the combination of DHCP and fixed IP address. IP of important server does not always change. When some servers need to update or upgrade and then leave the network, Host Usurp Attack is always easy to launch. In clouds and data center networks, the migrations of VMs from one host to another is a frequent phenomenon. In the migration, if the IP of migrated host remains the same, these hosts are under the threat of Host Usurp Attack.

In this paper, we launched a Host Usurp Attack in our experimental environment. We chose Floodlight with TopoGuard[4]. We deployed an Apache2 web server with IP address "10.0.0.1" and several hosts in our customized OpenFlow topology. The experiment scene are shown as Fig. 3. Then we launch the Host Location Hijacking Attack as mentioned above that attacker uses Scapy to periodically inject fake packets in the name of "10.0.0.1". When user invited "10.0.0.1", host hijacking attack is not successful. Then we closed our server host in order to simulate server optimization. Upon a compromised host attacker, we run a Web service and send ARP request to probe the corresponding MAC address of "10.0.0.1". We then use Scapy to periodically inject fake packets in the name of "10.0.0.1". Then we start up the real web server with "10.0.0.1" again. At this time when user visits "10.0.0.1", this request will be directed to attacker instead of the server in link 1. The newly started server is stopped by TopoGuard because host prober in TopoGuard detects that the IP address "10.0.0.1" has been there when "10.0.0.1" re-access the net. In traditional SDN network we can launch Host Hijack Attack directly, but even if with existing defense schemes we can still launch Host Usurp Attack.

Countermeasures

To defense the proposed Host Tracking Problem in SDN networks, we use dynamic defense strategies. In order to realize the authentication of network equipment identity, we introduce PKI technology. PKI (Public Key Infrastructure) is a universal security infrastructure that uses public key concepts and technologies to implement and provide security services. In PKI system, CA (Certificate Authority) is a trusted center in a domain. Digital certificate, which tied a public key and identity information, is the data structure obtained after the signature of the CA's private key. In network communication, digital certificate is a series of data that mark the identity information of the communication parties. It provides a way to verify the identity, which is similar to the identity card in the daily life.

Identity Information of Network Equipment

In order to identify a network equipment, the easiest way is to identify IP address and MAC address. If you use IP address of the network device as the identity, there are two problems: first, the IP address of the network device may be changed, on the other hand, the IP address can be fake. Therefore, we choose MAC address of the device as the identity information. We need to fill in the MAC address of the device in the certificate Common Name domain. The Ethernet card address of the computer requesting certificate must be recognized by the network administrator, meanwhile it must be legal and can be safe used, e.g. it must have a certificate issued by the network administrator.

Network Device Authentication Protocol

The design of the device authentication module uses the challenge response mechanism to realize the authentication between the host and the device authentication module, using PKI technology to realize digital signature and certificate management, as shown in Fig. 1.

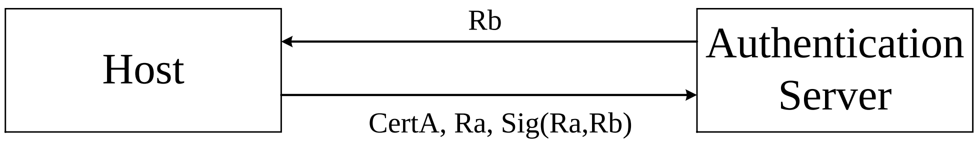

Figure 1. Certification Schematic

Specific steps are as follows:

1. Authentication server generates a random number Rb and sends it to the host;

2. Host generates random number Ra, and use the network equipment certificate to sign Ra, Rb. Then host will send Certificate A, Ra and the signature to the authentication server;

3. The authentication server will verify the result of the signature.

Hauth System

In this section, we detail the design and implementation of a new security extension HAuth, to protect the SDN networks from Host Usurp Attack. And we also provide those trusted hosts to network administrators and other SDN controllers in the network. Our goal is to provide complete protection of host entity in the network and provides network providers with trusted user information.

Overview

The basic idea of HAuth is to introduce a unified authentication server to achieve the identity of the network equipment certification. In HAuth, we use external certification to ensure the identity of host, and to share identity information in different networks.

In HAuth every host that wants to access the network must have a legal certificate recognized by the network provider. The certificate must use MAC address as identity information. In addition, every host that wants to access the net must install a program in advance to complete the certification process mentioned in Fig. 1. In order to defend these attacks, it is worthwhile. If SDN has become the mainstream, these measures can be built when the host are produced.

Fig. 2 illustrates the architecture of our defense system. HAuth system consists of two parts: HAuth in controller and authentication server. The authentication server is used to authenticate the host and return results. HAuth in controller recognizes the results. When the host wants to access this network, it must get the certificate approved by network administrator.

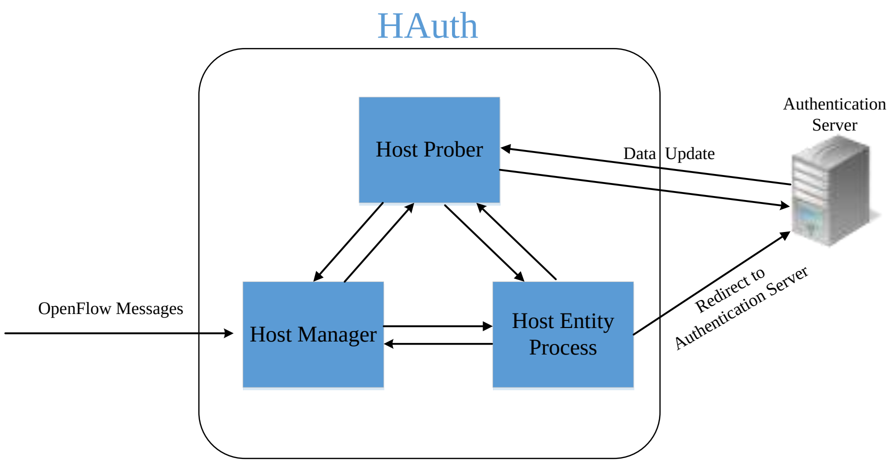

Figure 2. The Architecture of HAuth

Design

Host Management

In order to ensure that the access to the host is credible, Host Manager must check the host profile when new Packet_In arrives at the controller. In Host Tracking Service, the host profile will be directly stored in the memory. Then the controller will make decision according to the memory. In our design, when the packet arrives at the Host Manager module, it first check whether this host profile has been existed in Host Profile Storage module. If the host has been existed in Host Profile Storage and it matches the existing profile and it doesn't exceed the expiration time of the host certification, this packet will be processed according to normal steps. If this packet is from the authentication server, Host Management will delete the marked destination host profile in Host Profile Storage according to the destination of this packet. Otherwise, the packet will be transferred to Host Entity Process. In addition, Host Manager will ignore those packets that are not associated with host (e.g. LLDP).

In our design, we use Host Profile Storage module to manage host profile information. This module is just for easy management. Despite the host profile, Host Profile Storage also record the expiration time of the host.

Host Entity Process

When Host Entity Process module receives the packet, it stands for that these hosts are suspect. In our design, we use Host Entity Process to select paths for these hosts to redirect these requests to fixed authentication server, then issue these rules to switches. Meanwhile, we also mark this host profile and store it in Host Profile Storage. Compared with the direct use of the authentication between the controller and the host, although the use of external authentication server will increase more procedures, it can significantly reduce the amount of packets between the controller and the switch. The direct use of the controller in identifying certificate will waste much bandwidth between the controller and the switch. Host Entity Process module will redirect this packet to the authentication server.

Host Prober

In order to get the authentication results, the controller must communicate with the authentication server. In our design, we use application to communicate with the authentication server. Then we use rest api to update Host Profile Storage. In our Host Profile Storage, we also add the expiration date of the certificate. When the packet arrives in, Host Manager also inspect whether the authenticated host has been expired. Host Prober is used to get the result from the authentication server and update Host Profile Storage immediately when it receives update from authentication server.

Authentication Server

The authentication server is used to implement the authentication of the host and the device through the challenge response mechanism. The authentication process can be seen in Fig. 1. Once the authentication is complete, the server will store the host and its expiration time. Meanwhile the server will also transfer the result to fixed application host, which can be configured in advance. In addition, the authentication server also provides an interface through which trusted host can access the authentication information of hosts. This is a simple authentication protocol based on the challenge response mechanism, and its security has been proved.

Implementation

We have developed a prototype implementation of HAuth on the master version of Floodlight. We implement an authentication server with PHP and MySQL. We modify about 300 lines of JAVA to modify the floodlight controller. In addition, we use shell script in host to complete host response. HAuth is compatible with OpenFlow v1.1.0, and Floodlight v0.90 controllers.

Evaluation

We evaluated a prototype implementation of HAuth to examine the effectiveness and performance.

Our physical testbed can be seen in Fig. 3. We use five servers with Intel I5 CPUs and 12GB of RAM running 64bit Ubuntu Linux V13.11. The controller is HAuth in one of these servers. The application server has two Rtl8139PCI NICs. Our OpenFlow switches are NETGRARWR3800 which run OpenWRT firmware with an OpenFlow extension.

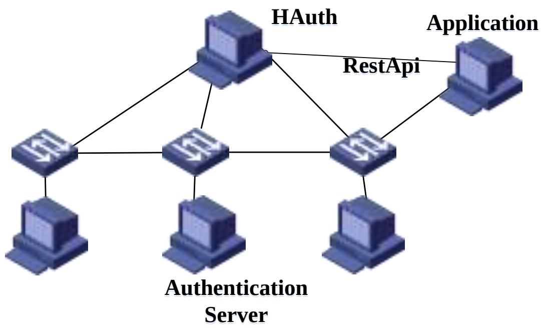

Figure 3. Experiment Environment

Effective

We first measured the effectiveness of our implementation against the attacks. Our experiment is conducted in the OpenFlow network environment including the Floodlight controller with HAuth. We launched aforementioned usurp attacks in the environment and testified the reactions of the fortified Floodlight controller.

Detecting Host Usurp Attack

An adversary can spoof the identity of a target host to hijack its location information inside OpenFlow controllers. Note that we assume the target host is not compromised by the adversary. With only TopoGuard, when the target host left the network, adversary could take over this IP address because this way does not violate the precondition and the post-condition defined in SDN controller. When the target host got back with the origin IP address, it was judged guilty because the target host in the previous location is still there.

With HAuth, we first access to the authentication server to register the target host. Then the host is off the line. At this time an adversary tried to launch Host Hijack Attack with the target host. Then HAuth redirected this packet to the authentication server. Only those who pass the verification can continue to be accessed. If the adversary can not be verified, the visitor's request will be discarded.

Table 1. Overhead Comparison of Host Authentication

Authentication Way	Authentication Time (Average)
Server	2.6ms
HAuth	3.8ms
Controller	33.5ms

Performance

Host authentication carried out in specialized authentication server, thus it puts little pressure to OpenFlow controller performance. OpenFlow controller only handled the result of authentication result. The performance penalty imposed by HAuth mainly comes from data update in Host Entity Process. In this experiment, we leverage Java System.nanoTime API to measure the running time of program snippets. In our experiment we compare the time when Host Prober gets new authenticated host and host authentication time. In addition, we also record the time required to carry out certification in the controller. From Table 1 we say HAuth doesn't take too more time.

Discussion

Nowadays the concept of information security is becoming more and more important. The appearance of SDN brings new security boundary. Solving these security boundary issues is the key to the deployment of SDN. In SDN there exists some basic components (e.g. Host Tracking Service). Once they are compromised, the whole network will become no credible. These basic vulnerabilities are generally caused by the imperfect initial design considerations. These designs are often followed by default for the correct and are taken very little consideration of their vulnerabilities. We hope our work can draw attention from SDN security researchers to consider more on the SDN design idea. In addition, SDN is just a kind of design method, we shouldn't follow the fixed openflow format blindly. We hope that our work will lead to more thought about SDN.

Related Work

In this section, we investigate security research in the SDN domain and network verification in SDN networks.

Security Research for SDN Networks

To date, the security issue of SDN have been being widely discussed in both academia and industry. FRESCO[1] introduces an OpenFlow security application development framework which can provide modular composable security services for application developers. Avant-Guard[3] shows a new attack (which is called data-to-control plane saturation attack) against SDN networks and provide solutions to prevent such attacks. FloodGuard[7] provides a new solution to this attack. Besides, SPHINX[2] proposes a unified approach to use network graphs to detect attacks that violate those learned flow graphs. TopoGuard[4] introduces a new way of port labeling which is similar with DAI (Dynamic ARP Inspection) in normal networks. Different from their work, this paper deeply investigates the vulnerabilities causing Host Spoofing Attacks, as well as a low-overhead real-time defense solution. And our scheme can be easily transplanted to their schemes.

Network Verification

Header Space Analysis[6] (HSA) leverages static analysis to detect forwarding and configuration errors. NetPlumber[5] introduces a real-time network-wide policy checking tool using HSA and can also verify arbitrary header modifications. All these verification solutions verify the logic correctness of the control plane and the data plane, however fail to locate the network topology exploitations discussed in this paper.

Our work mainly focuses on host authentication. FortNox[8] introduces a SDN tunneling attack and presents two mechanisms, role-based authorization and security constraint enforcement, to solve corresponding security challenges. SE-Floodlight[9] introduces attacks caused by SDN applications conflict and provides app-based authentication and access control to solve applications conflict and audit applications behavior. By introducing authentication scheme, our solutions can make it perfect to defect Network Topology Poisoning Attack.

Conclusion

In this paper, we propose new SDN-specific attack vectors, Host Usurp Attack, which can seriously challenge the network-wide visibility of SDN. We demonstrate that even in existing protections, Host Usurp Attack can still effectively poison the network topology information in host migration or host maintenance. Then we present HAuth, a new security extension to OpenFlow controllers, which provides automatic host authentication and host authentication log. Finally, our prototype implementation shows that using authentication server to verify the host is credible. We hope our work can attract more attention to SDN security research.

References

[1] 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

[2] M. Dhawan, R. Poddar, K. Mahajan, and V. Mann. Sphinx: Detecting security attacks in software-defined networks. In NDSS, 2015.

[3] S. Seungwon, Y. Vinod, P. Phillip, and G. Guofei. Avant-guard: Scalable and vigilant switch flow management in software-defined networks. In CCS'13, 2013.

[4] S. Hong, L. Xu, H. Wang, and G. Gu. Poisoning network visibility in software defined networks: New attacks and countermeasures. In NDSS'15, 2015.

[5] P. Kazemian, M. Chang, and H. Zeng. Real time network policy checking using header space analysis. In NSDI'13, 2013.

[6] P. Kazemian, G. Varghese, and N. McKeown. Header space analysis: Static checking for networks. In NSDI'12, 2012.

[7] H. Wang, L. Xu, and G. Gu. Floodguard: A dos attack prevention extension in software-defined networks. In DSN, 2015.

[8] P. Philip, S. Seungwon, Y. Vinod, F. Martin, T. Mabry, and G. Guofei. A security enforcement kernel for openflow networks. In Proceedings of the First Workshop on Hot Topics in Software Defined Networks, HotSDN ’12, pages 121 – 126, New York, NY, USA, 2012. ACM.

[9] P. Phillip, C. Steven, F. Martin, S. Keith, and Y. Vinod. Securing the software-defined network control layer. In Proceedings of the 2015 Network and Distributed System Security Symposium (NDSS), San Diego, California, 2015.