Exploring an Identity Binding Attack in SDN

William Kovacs

1 INTRODUCTION

Software defined networks (SDNs) offer a greatly different approach to networking than traditional networks. In SDNs, the control plane is separated from the data plane; moving from the switches to a central controller. This can result in the movement of what was once distributed state into this central location. A particular class of state, referred to as identifier bindings, and how it can be exploited in this nascent technology is explored in the paper "Identifier Binding Attacks and Defenses in Software-Defined Networks" by Jero et al. [10].

'Identifier bindings' are simply how different identifiers that refer to the same entity, such as MAC and IP addresses, are associated with each other. It is through these bindings that allow for the layering of protocols, and gives users the assurance that messages reach their intended target. For instance, in a network, when a host pings a specific IP address, it will be routed to a machine with that IP. In reality, the host is pinging the IP that goes to the MAC address that the network believes is bound to the IP (perhaps via ARP), and that MAC address actually corresponds to the device the sender actually wanted to reach. The aforementioned paper lists the different bindings as network location to device, MAC to location, IP to MAC, host name to IP, and username to hostname.

Many attacks exist focus on disrupting these bindings to accomplish things as drastic as impersonations, such as ARP spoofing (altering a MAC-IP binding), or as simple as eavesdropping, such as MAC flooding (removing MAC-IP binding). The paper that I will be examining makes the claim that the attacks are more severe in SDNs than in traditional networks due to three key differences: the presence of security mechanisms, rule consistency, and the location of the control plane. The first two are not nearly as important as the third one.

The first one simply notes how in traditional networks, the standard siwtches and routers that one can buy already have protective measures in place, while for SDNs, the onus of protection has been moved to the controller, and since current open source implementations of these controllers do not offer comprehensive defense, are easily susceptible to this class of attack. This affects the frequency of success that these attacks can, and as the technology develops, will become less of a problem.

Regarding flow consistency, traditional networks are able to update their routes quicker when a change in network is detected, while in SDNs, routes are updated more slowly, as they tend to be based on timeouts. This issue affects the ease at which certain attacks can be implemented, as incorrect rules to redirect traffic may be present longer in SDNs than in traditional networks.

The most serious of differences is what makes SDNs unique: their central controller. In traditional networks, because the control plane and relevant state is distributed throughout the network, attacks need to be more localized. However, in an SDN, having a central location for the control state makes it easier to both implement an attack, as the attacker can be anywhere in the network, and to propagate its effects, as the entire network is affected by its results. This can become especially troublesome when the controller uses the state for proactive routing, as once it has an incorrect binding, it does not give the victim a chance to remedy this mistake. Indeed, if the controller contains a table of IP-MAC bindings and uses Proxy Arp, then if an attacker manages to replace a victim's IP-MAC binding with his own, the victim never gets a chance to reply to an ARP request as he would in a traditional network.

Due to these additional vulnerabilities in SDNs, the authors explored how these identifier bindings can be exploited in this type of environemnt and how to defend against them. In particular, they introduce a new type of attack that they dub "Persona Hijacking" that they claim allows an attacker to takeover all network identifiers of a victim at once (MAC, IP, and hostname), a more detailed discussion of which appears below.

They then present a robust defense to all the different types of identifier binding attacks that they call "SecureBinder". This defense verifies that any changes to these bindings are legitimate, uses source-address port filtering, and utilizing 802.1X to validate devices' MAC addresses.

My goal is to be able to replicate the Persona Hijacking attack to verify whether it can be as potent as they claim. Furthermore, these experiments will be done using the POX controller, which was untested in the paper but suggested to be vulnerable, in order to demonstrate if it is as widespread of an issue as they claim. Due to time constraints I will not perform a formal evaluation of the weaknesses and security of the controllers and their system via a model checker. While I would have liked to implement the defenses as well, the timing of writing what took them over 2000 lines of code in addition to the attacks seemed slightly infeasible.

2 RELEATED WORK

As with anything, as SDNs get more popular, they can become higher priority targets for attacks, especially due to its

nascent nature and centralized control. To reduce the risks of any such attacks before SDNs become widely deployed, a fair amount of research has already been performed. Much of these have been thoroughly explored in several research surveys [2, 3, 11, 14]. Essentially, while the centralized controller does provide greater flexibility, it opens up many exploitation routes that have been well-studied, such as denial-of-service [4].

Furthermore, the design of SDNs can lead to the introduction of exploitable race conditions. Xu et. al. were able to demonstrate how the asynchronous nature of SDN allowed for attacks similar to Time of Check to Time of Use ones that could result in controller crash or service disruption [15]. For instance, they discovered that with appropriate timing of the switch-join and switch-leave events in the floodlight controller, they were able to trigger a Null-Poitner Exception.

But the work of security never ends. Indeed, in [2], the authors recognize how the handling of identities is a pressing issue, as there were no methods that could verify and bind identities to the users. This lack of secure bindings has been present in the standard internet architecture and has been a major source of security issues, particularly with regards to spoofing [13], and has been carried over to SDNs. Indeed, it's such an important issue that when designing a secure SDN, the Open Networking Foundation has robust identity as its second principle [1].

Indeed, without any proper protection in place, it is trivial for an attacker to claim to have any IP or MAC address. Hong et al. further demonstrate that it is possible for an attacker to trick an SDN network into believing that a particular MAC address has moved to a different network location, in what they term network topology poisoning attacks [9]. This is similar to ARP poisoning, but on a network scale.

In order to better secure these bindings, 802.1X was developed for traditional networks and has been been translated over to the SDN world [7, 8, 10]. 802.1X refers to the Extensible Authentication Protocol. In this protocol, before a host can join a network, it must communicate to an authentication server via some authernticator and verify its identity via some form of credentials, such as certificates. In these works, the authentication server tends to be a RADIUS one, and the authenticator is an application that communicates with the controller.

To prevent the aformentioned network topology poisoning attacks, the Hong et al. developed their own authentication system, which eschews the 802.1X approach, and instead actively probes and checks incoming topology changes to verify them [9].

3 THE ATTACK

The main idea behind the Persona Hijacking attack is that an attacker is able to trick a network into believing that the IP address of a victim now belongs to the attacker, down to the IP-MAC binding. This attack is only effective if the controller uses DHCP information for routing. Such a scheme has been used before[5, 12], so there are real scenarios where it can have an impact (granted the latter system employs an authentication scheme that may make this particular attack ineffective). Depending on the network setup, there are one or two stages that need to take place.

The first is an 'IP takeover' phase. During this, an attacker tries to abuse the DHCP protocol in an attempt to get the corresponding server, located in either the controller itself or externally, to lease the victim's IP to the attacker. To achieve this, the attacker first forges a DHCP Release message that makes the victim's IP address available for distribution by the server. Then, the attacker tries to obtain this address by flooding the network with DHCP Discover messages, using random MAC addresses, until he is offered the victim's IP address. This flooding is necessary as it's impossible to determine when the requisite IP will be leased, especially since some servers may offer unused addresses over recently released ones. When the victim's IP is offered, the attacker can accept it with a corresponding DHCP Request message, and if the network uses DHCP information, it now thinks that the victim's IP address is bound to the MAC present in the Discover request. Furthermore, any host name associated with that IP address will transitively be associated with that MAC address.

The authors also talk about a 'Flow Poisoning' phase, though it seems to be an addendum to the previous phase. It's used in the case where the DHCP server attempts to verify whether the IP address is still active before offering it, such as by flooding a corresponding ARP request. This action is recommended by the DHCP RFC [6], but not required. In particular, several open-source SDN controllers that offer DHCP do not perform this check by default, such as POX and ONOS. This portion of the attack would only be relevant under these controllers only when an external DHCP server is being used. If this verification was performed by the controller, then it's quite psosible that the attack as a whole would not succeed, as it would require redirecting traffic from the controller as opposed to another host.

The Flow Poisoning phase abuses an interesting race condition, present only in SDNs: the flow rules installed on a switch are not synchronized with the controller's view of the network. For example, in POX, when a rule is added to a switch, it remains there for it's timeout period. Even if the controller's view of the netwrok changes, the flow remains. By forging a message from the DHCP server, the attacker

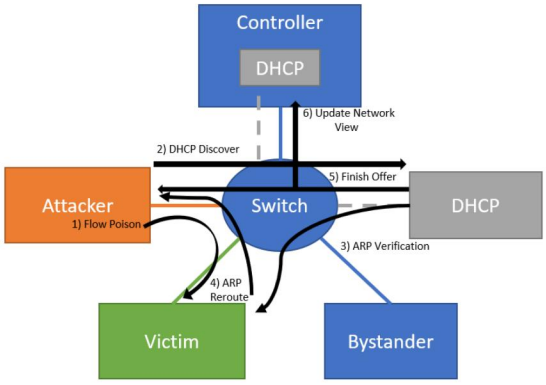

Figure 1: Overview of Persona Hijacking Attack after the forged DHCP Request was sent (omitted to reduce clutter). Labels of steps are attached to tail of arrow (except for step 6). The flow poisoning and DHCP discovery messages are repeated until the server offers the desired IP. The DHCP server be located in either the controller or externally. For the steps of the attack, an external one is used.

can set up a flow that directs output intended for the DHCP server from the victim to the attacker. When the server does send out the verification ARP, the victim's response will be redirected to the attacker, and the server will believe that the victim is indeed no longer active.

The major difference between this and a similar attack in a conventional network is the timing: in SDNs, the window of opportunity is much larger. In a conventional network that used learning switches, if this was attempted then the attacker would have to set up the route after the DHCP server's verification ARP (otherwise the ARP would override whatever the attacker sent), but before the victim sends back a response. However, in an SDN, after establishing the flow, it remains even as the verification passes through, and in POX, this means at least 10 seconds of rerouting.

The complete attack, as I implemented it, including the Flow Poisoning aspect, is presented in figure 1.

4 EXPERIMENTAL SETUP

I implemented this attack in two networks: one where the DHCP server is located in the controller and one where the DHCP server is an external server. Both of these scenarios were set up using the Mininet simulator version 2.3.0d1 with a POX controller. As mentioned previously, these experiments were performed with POX in order to better verify their applicability to another controller that was not used in the original paper, which tested both ONOS and RYU. The network topology, as seen in figure 1, simulated was similar to the one used in the paper: three hosts, the DHCP server, and the controller, all connected to a single switch. The hosts represent the attacker, the victim, and a bystander that is used to test whether it can contact the victim once the attack is completed. The DHCP server can either be external or it can be handled by the controller, which is the major distinction between the two scenarios. This alternate is represented in figure 1 as dashed lines.

The first scenario that I tested was using the POX provided DHCP server. As with ONOS, as tested in the original, POX's version of DHCP does not verify whether an IP address is still live, even if it was recently released, before offering it. This means that the flow poisoning aspect of the attack is unnecessary for this scenario. To verify that the whole attack finishes, it suffices to see if the attacker is able to get a DHCP offer for the victim's IP address. However, to further explore the type of network that can be affected by this type of attack, I also differentiated how the network interacted with the DHCP requests. In this scenario, I used the l2 learning switches provided by POX and a modified proxy ARP application that would generate IP to MAC bindings based on the DHCP lease events generated by the server.

The second scenario used a host that acted as a DHCP server by running udhcpd from BusyBox v 1.27.2, the same type used in the paper. While it is mostly intended for embedded systems, it does provide a simple, easy to set-up variant. Furthermore, it does try to verify whether an IP address is still alive by sending a relevant ARP before offering the address. This means that the flow poisoning is crucial to the success of the attack. While the verification of the attack is the same as in the other scenario (e.g. receive an offer for the victim's IP), the network uses the information in a different way: it installs flows based on the IP address port mapping learned by snooping on DHCP messages.

My flow poisoning phase in this scenario slightly differed from that of the original's. They had sent an ICMP ping with the DHCP server's MAC address to the victim as soon as the DHCP offer was received by the attacker. I made two modifications. The first is that instead of sending a ping, I had to send an ARP packet. The provided POX l2 learning switch controller installs flows based on the headers of sent packets, so in order to redirect the verification ARP, I needed to send a spoofed ARP. The other change is that I would send the packet before every DHCP discovery message. This is because the DHCP server would verify with the ARP before sending the offer, so while the bandwidth required for this attack inreased dramatically, it's the only way that I could see to accomplish it.

In each scenario, to verify the effects of the attack, pingAll was run to check the connectivity of the victim with the other hosts. This was done in both scenarios, as they used two separate environments. It should be noted that this is

separate from just looking at whether it succeeds, which is determined by receiving the requiste offer.

All of the experiments were tested on a Google Cloud instance that was running Ubuntu 18.04 with 1 virtual CPU and 3.75 GB of memory. Scapy was used to generate the forged DHCP packets.

5 RESULTS/DISCUSSION

For the first scneario, the Persona Hijacking Attack was always a success: the attacker was always able to obtain the victim's IP address via the DHCP server. This itself isn't too surprising, as the DHCP server provided by the POX controller is incredibly bare bones, and without the need to use the flow race condition, there is nothing that would prevent it from taking effect. On the other hand, the attack's effet were fairly inconsistent. The victim was only blackholed 50% of the time over the course of 12 trials.

I'm not entirely sure what causes this inconsistency, as the switch has no flow rules associated with it by the time this test occurs. However, I suspect that it is due to the order that the pings are serviced. For instance, if those from the victim reach the other hosts before any other ping starts, then when the bystander sends their ping, they can store the truth, and don't need to send an ARP and get the poisoned response. If not, the proxy ARP gives the hosts the attacker's MAC.

For the second scenario, everything worked smoothly: the attacker would always be able to get the victim's IP address and this would always cause the victim to be blackholed, at least in the runs that I have performed. Because POX uses both an idle and a hard timeout when a flow rule expires no matter what, it is possible that after the flow is installed for the first time and the attack reaches a 30 second interval, an ARP response from the victim can escape and reach the server. While possible, it is unlikely as it requires that the victim's IP is being offered at a 30 second interval. My variant of the attack is a bit easier to get it to work than the 90% success of the paper's, as it tries to ensure that the fake flow is consistently up, though at a higher cost of messages sent.

While a certain amount of DHCP snooping and information utilization is expected to make this attack do anything, it is interesting to see how much of a difference where this info is being used makes. The authors discuss how this attack has a severe impact when DHCP information is being used to manage forwarding rules, as done in the second scenario, so it is reassuring to see that even when this info is not being used to such a deterministic effect it is able to have an impact.

While this attack is not overly complex, it does demonstrate the core aspects for binding attacks that the authors identify in current SDNs, as mentioned in the introduction: the use of a central controller, flow consistency, and software. The central controller is what enables this type of attack: it provides a single point that, if tampered with, can affect the entire network. For instance, in a traditional network ARPs, and thus attacks that use them, are localized to a particular subnet, bounded by the routers (to my understanding, router's should not broadcast ARPs). However, in an SDN this restriction may not exist. Once an attack like Persona Hijacking is able to poison the controller's view of the network, it can affect all routing that it is involved with, e.g. proxy ARP or proactive routing. However, this all depends on how the network is set up, for it is very easy to construct a network where each subnet is governed by its own controller, thereby removing this type of issue.

This attack does demonstrate the weakness of flow inconsistency between the controller's and the switch's view. Without the flow poisoning aspect of the attack, the second scenario would be unable to take place; indeed, to be safe, I ran the simulation without the attacker forging the server's ARP and it would invaraibaly fail. This inconsistency is what really made Tme interested in this type of attack, and while it is better explored in [15], this provides a simple example of how it can be abused. Furthmore, conventional networks don't have to deal with this type of issue nearly as much, as in standard POX, a fake flow can persist for up to 30 seconds, even if the controller was tricked then realized its error, as it's probably not going to delete the flow.

The last major issue is what they refer to the use of 'bare-metal switches', but I feel is more apt to say that SDN is based on software. Like any software, it can be exploited unless defenses are put into place. Software offers more freedom and flexibility, but with that comes the ability to easily miss something. Before, being able to buy a switch was tantamount to having certain security standards in place. However, with SDNs, especially if using open source, then even simple things like ways to protect against ARP poisoning may not be present (or a simple ARP to verify an unused IP before sending a DHCP offer). Saying that, it's unclear as to how much people would rely on the vanilla software provided, especially when something like the POX DHCP server explicitly states how bare-bones it is.

6 LIMITATIONS/WEAKNESSES

One of the major limitations of this work is not having a firm understanding of the architecture of typical SDNs. While there are certainly papers that help understand certain aspects of how things should be done, the sheer flexibility of SDNs offer many choices that seem plausible, but are hardly used. Indeed, it is quite possible that even if a paper described how something ought to be done does not mean that it is actually done in practice. With a better understanding of

how SDNs are used in practice, I would be able to better tailor the experiments to see how well the attack could work in practice.

Another weakness is the lack of experiments performed to test how different network conditions could affect the efficacy of these attacks. This is due to things taking longer than I had expected, particularly with the slight intricacies in putting everything together, such as the difference required between a DHCP discovery for a controller based server and an external one.

Going along with the above weakness, the scale of the experiments is one. While the size of the network topology does mirror that used in the paper, it does not fully explore how this type of attack could functoin in an actual network, nor it does it help to demonstrate the importance of the network-wide coverage of such an attack versus that of the localized variant (e.g. akin to ARP poisoning) as much as it could.

7 FUTURE WORK

Some future work that can be done to further explore the ideas in this paper include varying the conditions of the servers to analyze how much they can impact the attack. In particular, it would be interesting to see the impact of if there is a high turnover of IP addresses (such as in a public place), and thus a lot of messages coming from the DHCP server, on the success rate of the flow poisoning attack. One expect it to drop as flows for the server would be consistently installed, but if aggressive enough, the flow poisoning might be able to prevail. In addition to these types of experiments, it would be really interesting to see if it is possible to analyze analagous attacks in conventional networks to further drive the difference home. While performing a live experiment would be particularly insgihtful, simulations could suffice.

Other future work could be getting to the part of the paper that I would have liked to explore if given a lot more time: that is the defense. Having an open source version of this type of defense could be helpful in understanding how authentication can be better performed in SDNs.

8 CODE

The code written for this experiment and instructions to run it can be found at:

https://github.com/kovacswc/pa3.

REFERENCES

[1] 2015. *Principles and Practices for Securing Software-Defined Networks.* Technical Report ONF TR-511. Open Networking Foundation, Palo Alto, CA. https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/Principles_and_Practices_for_Securing_Software-Defined_Networks_applied_to_OFv1.3.4_V1.0.pdf

[2] I. Ahmad, S. Namal, M. Ylianttila, and A. Gurtov. 2015. Security in Software Defined Networks: A Survey. *IEEE Communications Surveys Tutorials* 17, 4 (Fourthquarter 2015), 2317–2346. https://doi.org/10.1109/COMST.2015.2474118

[3] Izzat Alsmadi and Dianxiang Xu. 2015. Security of Software Defined Networks: A survey. *Computers and Security* 53 (2015), 79 – 108. https://doi.org/10.1016/j.cose.2015.05.006

[4] R. Braga, E. Mota, and A. Passito. 2010. Lightweight DDoS flooding attack detection using NOX/OpenFlow. In *IEEE Local Computer Network Conference*. 408–415. https://doi.org/10.1109/LCN.2010.5735752

[5] Martin Casado, Michael J Freedman, Justin Pettit, Jianying Luo, Nick McKeown, and Scott Shenker. 2007. Ethane: Taking control of the enterprise. In *ACM SIGCOMM Computer Communication Review*, Vol. 37. ACM, 1–12.

[6] R Droms. 1997. Dynamic Host Configuration Protocol, RFC 2131. (1997).

[7] Diogo Menezes Ferrazani Mattos and Otto Carlos Muniz Bandeira Duarte. 2016. AuthFlow: authentication and access control mechanism for software defined networking. *Annals of Telecommunications* 71, 11 (01 Dec 2016), 607–615. https://doi.org/10.1007/s12243-016-0505-z

[8] F. Hauser, M. Schmidt, and M. Menth. 2017. Establishing a session database for SDN using 802.1X and multiple authentication resources. In *2017 IEEE International Conference on Communications (ICC)*. 1–7. https://doi.org/10.1109/ICC.2017.7997200

[9] Sungmin Hong, Lei Xu, Haopei Wang, and Guofei Gu. 2015. Poisoning Network Visibility in Software-Defined Networks: New Attacks and Countermeasures. In *Proceedings of the 22th Annual Network and Distributed System Security Symposium (NDSS '15)*.

[10] Samuel Jero, William Koch, Richard Skowyra, Hamed Okhravi, Cristina Nita-Rotaru, and David Bigelow. 2017. Identifier Binding Attacks and Defenses in Software-Defined Networks. In *26th USENIX Security Symposium (USENIX Security 17)*. USENIX Association, Vancouver, BC, 415–432. https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/jero

[11] Benzekki Kamal, El Fergougui Abdeslam, and Elbelrhiti Elalaoui Abdelbaki. [n. d.]. Software defined networking (SDN): a survey. *Security and Communication Networks* 9, 18 ([n. d.]), 5803–5833. https://doi.org/10.1002/sec.1737 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/sec.1737

[12] Haidlir Achmad Naqvi, Sofia Naning Hertiana, and Ridha Muldina Negara. 2015. Enabling multipath routing for unicast traffic in Ethernet network. In *Information and Communication Technology (ICoICT), 2015 3rd International Conference on*. IEEE, 245–250.

[13] J. Rexford and C. Dovrolis. 2010. Future Internet architecture: clean-slate versus evolutionary research. *Commun. ACM* 53, 9 (2010), 36–40.

[14] S. Scott-Hayward, S. Natarajan, and S. Sezer. 2016. A Survey of Security in Software Defined Networks. *IEEE Communications Surveys Tutorials* 18, 1 (Firstquarter 2016), 623–654. https://doi.org/10.1109/COMST.2015.2453114

[15] Lei Xu, Jeff Huang, Sungmin Hong, Jialong Zhang, and Guofei Gu. 2017. Attacking the Brain: Races in the SDN Control Plane. In *26th USENIX Security Symposium (USENIX Security 17)*. USENIX Association, Vancouver, BC, 451–468. https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/xu-lei