SOUTH ASIAN JOURNAL OF ENGINEERING AND TECHNOLOGY

Full Length Article

PROVIDING SECURITY AGAINST IP CROWDSOURCED SPOOFING ATTACKS ON CLOUD USING TOPOGUARD ALGORITHM

M.KEERTHIVASAN[1],R.KISHORE[2],M.MALATHI[3],G.MONICA[4],V.SENTHILKUMAR[5],K.KUMARESAN[6], K.DINESHKUMAR[7]

[a] *Department of Computer Science and Engineering, K.S.R College of Engineering, Coimbatore- 641020, Tamilnadu, India*
[a] *Department of Computer Science and Engineering, K.S.R College of Engineering, Coimbatore- 641020, Tamilnadu, India*

***Corresponding Author**

DOI:

ABSTRACT: In this paper, we discussed the brief overview of SDN security survey, we specifically investigate the potential lt heats of man-in-the-middle attacks on the Open Flow control channel, we also describe a feasible attack model in the open flow channel, and then we implement attack demonstrations to show the severe consequences of such attacks. Additionally, we propose a lightweight countermeasure using Bloom filters. We implement a prototype for this method to monitor stealthy packet modifications. The successful attacks can effectively poison the Virtual Machine information, a fundamental building block for core SDN components and topology-aware SDN applications. With the poisoned network visibility, the upper-layer Open Flow controller services/apps may be totally misled, leading to serious hijacking, denial of service or man-in-the-middle attacks. The result of our evaluation shows that our Bloom filter monitoring system is efficient and consumes few resources.

Keywords: IP SPOOFING (Man-In-The-Middle) attacks, IOT (Internet of Things), SDN (Software Defined Networks), Secure computing networks.

1 Introduction

Programming characterized organizing (SDN), which brings numerous new highlights, for example, arrange programmability, incorporated control, and so on., enables owners to naturally deal with the whole system in a flexible and dynamic manner. With these advantages, many trust that the eventual fate of the IoT will be founded on SDN. In this manner, several works [2] and [3] are proposed for the future Io T .As both SDN switches and secure hubs are relatively powerful hubs in a regular IOT sending, they are usually consolidated together, which is an ideal method to integrate the usefulness of SDN. Despite the fact that sending IOT– Secure systems utilizing SDN seems promising, security issues are unavoidable here. As secure hubs and SDNswitches are generally consolidated together, vulnerabilities in secure nodes might be utilized by assailants to bargain the SDNswitches they control. In this way, it is important to have security mechanisms to additionally screen and improve the security of the SDN foundation in IoT– Secure situations.

In SDN, the controller controls every one of the switches through "Open Flow" channels. Directions, and solicitations from the controller ,as well as status and insights from the switches, are transmitted through the OpenFlow channels. Accordingly, the security and unwavering quality of Open Flow channels among the controller and switches are basic for appropriate SDN operation,configuration, and the board. On the off chance that an aggressor were to intercept or potentially adjust the messages on these channels, the person could send counterfeit messages to the switches and the controllers, propelling a wide assortment of assaults, for example, denial of administration or man-in-the-center (IP Spoofing) assaults. Open Flow channels, once blocked, may bring disastrous circumstances to both the system suppliers and their customers.

As a rule, we can't just depend on figure systems. There ought to be other can gather customers ‘sensitive data (e.g., sensor information delineating a client's day by day conduct) by directing the changes to send duplicates of packets containing such data to the aggressor. In this way, sensitive client data will be uncovered to aggressors. With network framework under such a danger, SDN has more security worries than a customary system. Taking another example, the assailant can send counterfeit bundles, in the interest of the switches, to the controller, harming the controller's global view of the system topology. With the mistaken topology, the controller may misconfigure other respectful switches, which may cause the system network blackouts. The results a terrible client experience and considerable income lost. With such potential dangers still practical, SDNs will never completely replace traditional systems. Despite the fact that it offers numerous new attractive features, without taking care of these issues, all the flexibility is good for nothing. Along these lines, work ought to be done to shield the Open Flow channels from interception. One may use figure systems to encode the channel after validation. In any case, confirmation and encryption alone can't ensure the wellbeing of the Open Flow channels.TLS, for instance, is a standout amongst the most mainstream cryptographic protocols.

2 OVERVIEW OF SDN SECURITY

There are clear security advantages to be gained from the SDN architecture. For example, information generated from traffic analysis or anomaly-detection in the network can be regularly transferred to the central controller. The central controller can take advantage of the complete network view supported by SDN to analyze and correlate this feedback from the network. Based on this, new security policies to prevent an attack can be propagated across the network. It is expected that the increased performance and programmability of SDN along with the network view can speed up the control and containment of network security threats.

On the down-side, the SDN platform can bring with it a host of additional security challenges. These include an increased potential for Denial-of-Service (DOS) attacks due to the centralized controller and flow-table limitation in network devices, the issue of trust between network elements due to the open programmability of the network, and the lack of best practices specific to SDN functions and components. For example, how to secure the communication channel between the network element and the controller when operated in the same trust domain, across multiple domains, or when the controller component is deployed in the cloud?

In the past few years, a number of industry working groups have been launched to discuss the security challenges and solutions. Meanwhile, researchers have presented solutions to some SDN security challenges. These range from controller replication schemes through policy conflict resolution to authentication mechanisms. However, when the extent of the issues is compared to the degree of attention placed on them, it is clear that without a significant increase in focus on security, it is possible that SDN will not succeed beyond the private data center or single organization deployments seen today. The main objective of this paper is to survey the literature related to security in SDN to provide a comprehensive reference of the attacks to which a software-defined network is vulnerable, the means by which network security can be enhanced using SDN and the research and industry approaches to security issues in SDN. The paper is structured as follows: Section II provides a context to the work by introducing the characteristics of SDN. In Section recent SDN and Open Flow security analyses are presented followed by a categorization of the potential attacks to which the architecture is vulnerable. Research work presenting solutions to these attack types is then presented in Section IV. The arrows in Fig. 1 indicate the attack categories for which solutions have been proposed and, by extension, those areas which have not yet received research attention. In Section V, the alternative view of SDN security is introduced with a survey of the research work dealing with security enhancements based on the SDN architecture. In Section VI, the two perspectives on SDN security are compared with improved functionality, open challenges, and recommended best practices identified. Section VII highlights open standards and open source industry group work on SDN security. Future research directions are identified in Section VIII. The paper is concluded in Section IX. For clarity, an overview of the Security Survey structure is presented in Fig. 1.

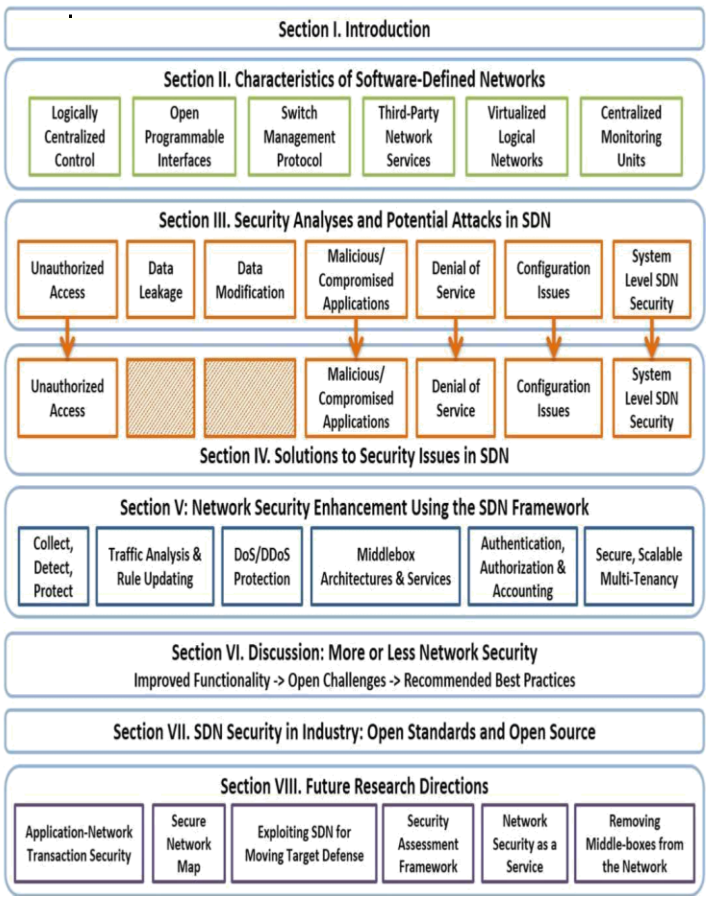

Section I. Introduction
Section II. Characteristics of Software-Defined Networks
Logically Centralized Control
Open Programmable Interfaces
Switch Management Protocol
Third-Party Network Services
Virtualized Logical Networks
Centralized Monitoring Units
Section III. Security Analyses and Potential Attacks in SDN
Unauthorized Access
Data Leakage
Data Modification
Malicious/ Compromised Applications
Denial of Service
Configuration Issues
System Level SDN Security
Unauthorized Access
Malicious/ Compromised Applications
Denial of Service
Configuration Issues
System Level SDN Security
Section IV. Solutions to Security Issues in SDN
Section V: Network Security Enhancement Using the SDN Framework
Collect, Detect, Protect
Traffic Analysis & Rule Updating
DoS/DDoS Protection
Middlebox Architectures & Services
Authentication, Authorization & Accounting
Secure, Scalable Multi-Tenancy
Section VI. Discussion: More or Less Network Security
Improved Functionality -> Open Challenges -> Recommended Best Practices
Section VII. SDN Security in Industry: Open Standards and Open Source
Section VIII. Future Research Directions
Application-Network Transaction Security
Secure Network Map
Exploiting SDN for Moving Target Defense
Security Assessment Framework
Network Security as a Service
Removing Middle-boxes from the Network

Fig 1: overview of SDN Security

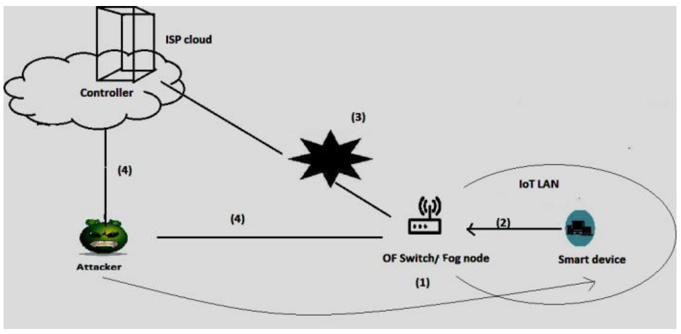
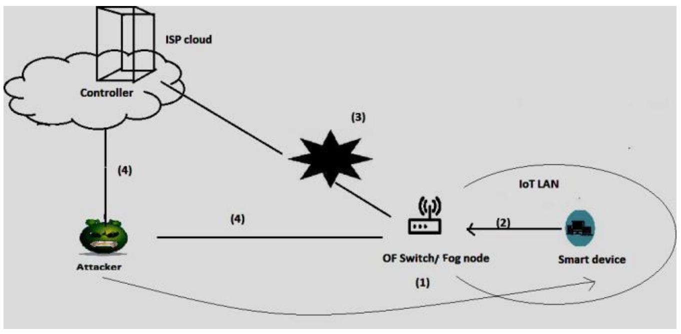


Fig 2: Attacking model



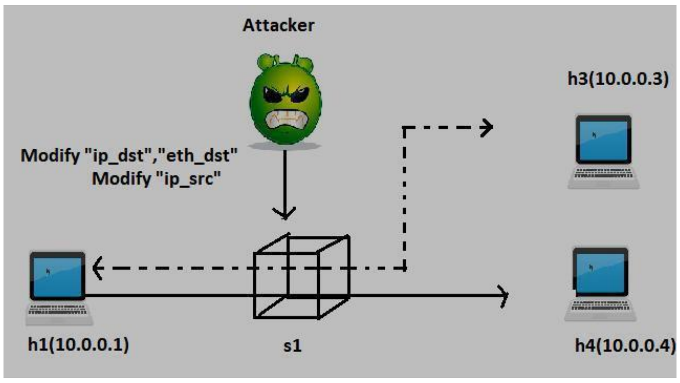


Fig 3: Traffic redirection attack.

Time	Source	Destination	Protocol	Info
1.001096000	10.0.0.1	10.0.0.4	ICMP	Echo (ping) requestid=0x08fc, seq=2/512
1.001121000	10.0.0.3	10.0.0.3	ICMP	Echo (ping) request id=0x08fc, seq=2/512
1.001447000	10.0.0.3	10.0.0.1	ICMP	Echo (ping) reply id=0x08fc, seq=2/512
1.001457000	10.0.0.4	10.0.0.1	ICMP	Echo (ping) reply id=0x08fc, seq=2/512

Fig4:Redirection attack. Packet capture result of h1 ping h4.

3. ATTACK DEMONSTRATION

Here, we present three assault exhibitions. In the principal, the aggressor diverts streams in the information plane. The second epitomizes how the assailant can gather data from the information plane. The last, indicates how the assailant can harm the controller's perspective on the system. We portray just three assault situations out of numerous situations. The total range of conceivable assaults is right now obscure.

A. Condition Set-Up

We use Floodlight, an open source SDN controller, as ourSDN controller, and use Mininet to reproduce a system in our examinations. The controller and switches convey through Open Flow v1.3. To disentangle our demos, we accept that the assailant, the controller, and the Mininet VM are situated on a similar nearby system. This supposition does not influence the consequence of our demos in light of the fact that the assailant can generally block OpenFlow channels with parodying procedures, for example, ARP mocking. This is conceivable as long as the aggressor exists in the way between the switch and the controller. Since Mininet is running on a virtual machine, every single mimicked switch share a similar IP address and remotely interface with the controller. Our assault contents assault just the Minine tvirtual machine, catching every single reproduced switch. Our configuration does not influence the last consequence of the demos because the procedure to assault the switch's interface is indistinguishable to assaulting the Mininet virtual machine. Our assault contents are written in Python v2.7 utilizing the well known scapy library, which is helpful for creating, sending, and sniffing bundles. We utilize this library to fabricate counterfeit Open Flow directions for the switches. In our demos, we use ARP parodying systems to block the Open Flow channel.

B. Traffic Flow Modification

The most direct assault is to stealthily modify the unfortunate casualty switch's sending table. In our examination, the aggressor obstructs a specific host's traffic stream and diverts the stream to another host. Fig. 2 demonstrates the possibility of this assault. The assailant embeds two Open Flow bundles, which contain stream table adjustment directions, into the Open Flow channel. The first Open Flow bundle educates the change s1 to adjust the goal IP and MAC address of any parcels initially bound for host h4. The new IP address and MAC address are that of host h3. The second Open Flow bundle directions the change to alter the source IP address of any parcels starting from h3, to the IP address of h4. Thus, if h1 attempts to speak with h4, it will really be diverted to h3, leaving h1 unconscious that it is speaking with an alternate host. To test the assault, we let h1 ping h4 and catch the bundles transmitted utilizing Wireshark. Fig. 3 demonstrates the bundle catch results (from every one of the interfaces in s1). In the figure, the principal passage demonstrates that s1 gets the ICMP bundle from h1 (10.0.0.1) with the goal h4 (10.0.0.4). In the wake of being prepared by the switch, the bundle's goal IP address has been changed to h3's (10.0.0.3) (the second passage). In spite of the fact that not appeared in Fig. 3, from the answer of h3 (the third passage), the MAC address of the bundle is likewise changed. Going through s1 once more, the source IP address is changed back to the IP address of h4 (the fourth passage). These diverted ways can't be gathered by h1. On the off chance that h1 is a Web camera that attempts to speak with a cloud server h4 however out of the blue speaks with a pernicious machine h3, all delicate data from h1 will be presented to the assailant.

C. Information Collection

The aggressor may likewise stealthily gather data by changing the switch sending table. Fig. 4 outlines the essential thought of a data accumulation assault. The aggressor first fashions an Open Flow parcel, which contains stream table alteration directions, and sends it to the injured individual switch. The assailant teaches the change to send a duplicate of every bundle focusing on h4 to the "controller," which is really the aggressor. When the unfortunate casualty switch refreshes its sending table, the assailant will get every one of the parcels initially bound for h4. We let h1 ping h4 and again catch all parcels from every one of the interfaces of s1 utilizing Wireshark. Fig. 5 demonstrates the catch result. In this exhibit, we let the assailant essentially sends back the ping parcel only to test. Fig. 6 demonstrates the consummation purpose of h1's ping bundles. We can see that the host gets two copy answers, one from h4 and the other from the assailant. Comparable as the past exhibition, delicate data will be spilled to the aggressor, however both the customer and the server won't know about the busybody.

D. Topology Poisoning Attack

In SDNs, the controller learns the worldwide topology through LLDP bundles. Assume the controller directions change s to yield a LLDP parcel through port eth1. Another switch s' gets this parcel on port eth2. Switch s' Incorporates both this bundle and the port eth2 number in a packet in message and sends it to the controller. From this message, the controller realizes that port eth1 in s interfaces with port eth2 in s'. On the off chance that the aggressor alters the LLDP bundles, the controller will have an off base perspective on the worldwide topology. Fig. 7 demonstrates the fundamental thought of this assault. The assailant stealthily alters both the yield port and the max_lenfield in the packet_out message. The max _lenfield shows the most extreme number of bytes the change can send to the controller. On the off chance that this field is set to 0, and the yield port is set to the controller, s1 basically disregards this message. Along these lines, s2 gets no opportunity to get the LLDP parcel, let alone forward the bundle back to the controller. In the event that the aggressor does likewise to s2, the controller will reason that these two switches are not associated. Fig. 8 demonstrates the topology produced by the controller amid the assault. Fig. 8 demonstrates the DPID of each switch. The DPID of s1 is "00:00:00:00:00:00:00:01" while the DPID of s2 is "00:00:00:00:00:00:00:02." The third switch, which isn't appeared in Fig. 7, isn't associated with this assault. In all actuality, s1 and s2 are associated. Nonetheless, the controller is tricked into imagining that they are most certainly not. On the off chance that there is a bundle investigation center box along the s1– s2 interface, the assailant can utilize this technique to go around review.

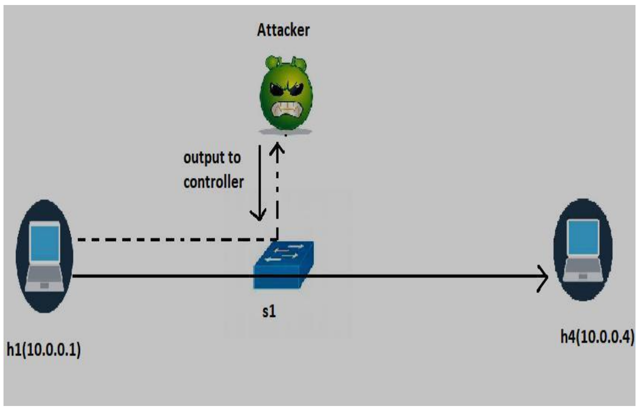
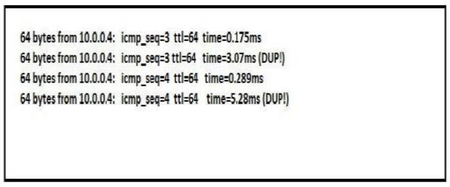


Fig: 5 Information collection attack.

Time	Source	Destination	Protocol	Length	Info
19.270245000	10.0.0.1	10.0.0.4	OF 1.3	206	Of_packet_in
19.274617000	10.0.0.1	10.0.0.4	OF 1.3	204	Of_packet_out
20.271880000	10.0.0.1	10.0.0.4	OF 1.3	206	Of_packet_in
20.277751000	10.0.0.1	10.0.0.4	OF 1.3	204	Of_packet_out

Fig: 6 Information collection attack. Packet capture of h1 ping h4



```
64 bytes from 10.0.0.4:  icmp_seq=3  ttl=64  time=0.175ms
64 bytes from 10.0.0.4:  icmp_seq=3 ttl=64   time=3.07ms (DUP!)
64 bytes from 10.0.0.4:  icmp_seq=4  ttl=64   time=0.289ms
64 bytes from 10.0.0.4:  icmp_seq=4  ttl=64    time=5.28ms (DUP!)
```

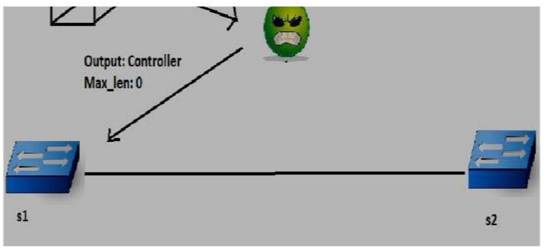


Fig: 7 Information collection attack: h1 ping h4 in terminal.

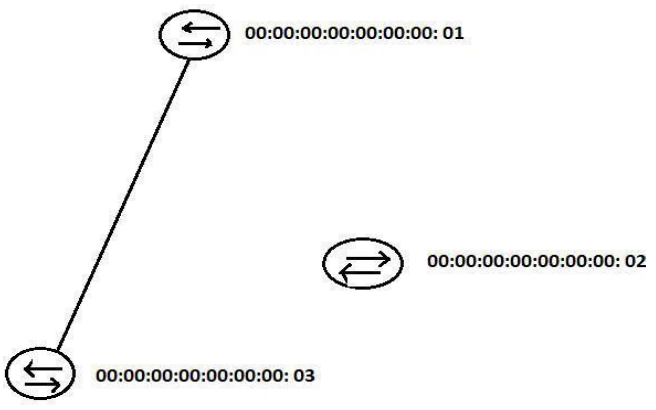


Fig: 9 Topology poisoning attack, Controller view.

4. COUNTERMEASURE

In this segment, a countermeasure and its Open Flow augmentation to identify IP Spoofing assaults on Open Flow channel will be proposed. As referenced in the past segment, the assailant can stealthily adjust bundles in the information plane by transforming at least one switches' sending table. To identify such a danger, one clear thought is to give the controller a chance to inquiry every one of the bundles that the switches sent, and after that look at them one by one. Be that as it may, this guileless technique will significantly expand the weight of both the controller and the system, and furthermore it isn't effective. To facilitate the weight, we propose a technique to recognize bundle alterations utilizing a Bloom channel. Sprout channel is a space-productive information structure, which is utilized for testing the presence of a component in a set.

We let each switch along one stream privately put parcels of that stream into a Bloom channel. In the event that they put similar parcels into the Bloom channel, individually, these Bloom channels ought to be the equivalent. Hence, the controller can distinguish any bundle adjustments of this stream by gathering all these Bloom channels and checking the distinction between these channels. In the event that there are any contrasts between these channels, it is certain that the parcels are altered amid its conveying. Other than every one of the switches' Bloom channel, we additionally need the inception bundle sending from the sensor in the event that the information parcels are changed at the primary switch. We put a screen procedure in the protected hub. These procedures do likewise as what the switches do, putting bundles from a particular stream into Bloom channels and sending Bloom channels to the controller when asked. The main distinction is that these screen forms cooperate with another example in the cloud instead of the controller. At that point the case advances the Bloom channel to the controller. The reason of utilizing another occasion is to shroud the cooperation between the screen procedure and the controller. As secure hubs much of the time speak with the cloud and these screen possibly associate with the cloud when asked for, the aggressor experiences issues finding To apply this thought, we broaden OpenFlow by including three new message types:

1) BF_INITIAL;
2) BF_SUBMIT; and
3) BF_REPLY. The implications of these messages are presented later. Figs. 9 and 10 show the convention of instating and concluding our Bloom channel strategy, separately. To begin discovery, the controller initially sends all switches an initialization command (BF_INITIAL), which contains the accompanying data:

1) the inspected stream f , spoken to by matching fields utilized in Open Flow; 2) a tag τ , which will be used later;
3) a set S of fields that ought to be discarded when figuring the hash estimations of parcels (fundamental for embeddings into a Bloom channel); and 4) the greatest number of bundles embedded into the channel n. On the off chance that n is set to 0, there is no restriction for embeddings parcels into the Bloom channel. Subsequent to accepting BF_INITIAL, each switch introduces itself as indicated by the parameters and answers with an affirmation (BF_REPLYwith no substance) to the controller. At the point when the controller receives answer from each switch, it triggers the discovery arrange by changing the stream table of the primary change to tag flowf with τ. When the controller needs to gather the

Bloom channels from the switches, it initially alters the stream passage of the labeled stream f in the keep going switch on the way by including a packet inaction. Along these lines, the controller can follow the last parcel of the system. From that point onward, the controller directions the primary change to quit labeling stream f. At the point when there is no bundle from the last switch for a specific time, it conveys BF_SUBIP Spoofingessages to every one of the changes to present their Bloom channels byBF_REPLY messages. The controller thinks about every one of the channels to discover whether there is any distinction among them.If any distinction is discovered, the controller will caution the manager about the getting into mischief switches

A. Constraint of the Countermeasure

This methodology works much of the time by and by. In any case, in some extraordinary cases, for example, all the Open Flow channels between the controller and switches in a single stream way has been blocked, our strategy won't work. In addition, if the assailant adjusts fields that are not in set S, this paper won't work either

Single crystal XRD for the grown crystals was taken using ENRAF NONIUS CAD-4 X-Ray diffaractometer with Mo Kα (λ=0.7107 Å). The obtained unit cell parameters are a=3.93 Å, b=24.94 Å c=28.67 Å, From the single crystal XRD it is concluded that grown crystal belongs to Orthorhombic system with the noncentro symmetric space group of *Fddd [17].*

5. IMPLEMENTATION

In this area, we will expand on the execution of our Bloom channel screen framework, which can recognize bundle alterations in SDNs. In particular; we will show the review of the framework and portray all segments of the framework.

A. Framework Overview

The screen framework, which we allude to as the "Sprout channel screen framework," comprises of two sections. One is executed in Floodlight controller, and the other is actualized in Open v Switch (OVS). Fig. 11 demonstrates the design of our framework. The controller side has one module named "Blossom channel screen," which is in charge of conveying BF_INITIAL and BF_SUBMIT messages to OVS, gathering answers from OVS, and looking at the switches' channels. This module offers two REST APIs for chairmen or different applications to lead the Bloom channel recognition stage. The switch parcel comprises of two segments. As a rule, the switch has two assignments for every bundle: 1) extricate analyzed fields (or information) and 2) embed separated substance into the Bloom channel. In OVS, every one of the bundles are gotten and sent in the data path, a module that is running in piece space where extraction begins. In any case, any deferral inside the data path can influence the sending speed. In this manner, we put the hash capacity and

Bloom channel inclusion code into the client space. Along these lines, the switch can embed the removed substance while sending parcels in the data path. The switch likewise has one part to speak with the controller, getting Open Flow messages from the controller, setting off the Bloom channel location stage, and answering with the filled Bloom channel to the controller.

B. Controller Side Design

1) **Bloom Filter Monitor Module:** The principle part of the Bloom channel screen, as we referenced already, is a module in the Floodlight controller, which is consequently stacked amid the instatement of Floodlight. The module has two primary capacities: 1) instating and 2) settling the Bloom filte screen technique. Both of these capacities can be summoned from REST APIs. The work process of these two capacities is equivalent to appeared in Figs. 9 and 10.
2) **Open Flow Library:** To stretch out Open Flow and a few execution classes (actualized under various Open Flow renditions) are embedded into the source code. We additionally change the serialization and OFT ypeenum to help the serialization of these messages with the goal that they can be transmitted through the system.
3) Floodlight Core: To empower Floodlight to deal with our new messages as simply one more standard Open Flow message, we change some center codes of Floodlight. Class OF Switch Hand shake Hand lerise in charge of getting diverse kinds of messages and dispatching them to various parts. We embedded code here to give it a chance to dispatch BF_REPLY messages to a message audience. Along these lines, the Bloom channel screen can get and parse BF_REPLY messages from switches through a message audience.

C. Switch Side Design

1) **Open Flow Extension:** To broaden Open Flow in OVS, we first embed the head structure of our three new Open Flow messages, in the Open Flow head documents, into OVS. At that point, we add new sections in enum OPTRAW and OFTYPE for our new message type. We likewise execute a message developer for BF_REPLY and parsers for BF_INITIAL and BF_SUBMIT, with the goal that the OVS can comprehend these new messages. At last, we add our new message handlers to the Open Flow handler in OVS. The handler parses the message with the parser and proceeds according to the message substance. A few moves might be made, for example, designing the data path through net link, adjusting the stream table to label streams, and answering to the channels produced. With these adjustments, OVS is capable to communicate with Floodlight, which additionally has the Open Flow extension.

2) **Fields Extraction and Element Insertion:** OVS is mainly divided into two sections: 1) v switched and 2) data path. V switched runs in the client space and is in charge of speaking with the controller and dealing with the stream table alongside some different highlights. Data path keeps running in portion space and is in charge of sending parcels. As this part keeps running in piece space, the parcels can be immediately sent. Every one of the parcels gotten by OVS previously go to the data path component where highlight extraction is actualized. Once the switch gets one labeled parcel, it removes fields according to the setup from switched. After extraction, it sends the outcome to v switched utilizing up call, which is an instrument utilized for data path to send messages to v switched. In ourimplementation, we influence this to send the separated header fields to user space. When client space gets the separated field data, it figures the hashes and embeds them into the Bloom channel.
3) **Filter Placement and Initialization:** It is nontrivial to decide where to put the Bloom channel. For the most part, there are a few scaffolds inside one OVS element. Each extension might be associated with a few diverse VMs. On the off chance that we put the channel in the worldwide area, (i.e., all scaffolds share one channel), at that point the traffic streaming between VMs won't be secured. In this manner, each scaffold ought to be treated as a switch element and given their own Bloom channel. In our usage, we put the Bloom channel inside the structure ofproto, which is for Open Flow convention in OVS, since each extension has just a single such information structure, and this structure can be gotten to amid the handling of the up call, where messages of removed substance are gotten. At the point when a scaffold associates with the controller, it will introduce its own ofproto structure. The channel spaces are distributed at the same time. When the channel has been submitted to the controller, the bridge will reset the channel for the following accumulation.
4) **Hash Function:** The hash calculation is executed withMurmur3 32-bit [12]. It is autonomous and consistently distributed, which is able for use in a Bloom channel. Further more ,it is basic and productive. For every parcel, we register the Murmur3 hashes with various seeds (to produce the k vital hashes utilized in the Bloom channel) and the hash yield is truncated by the channel estimate. The choice of k will be talked about in the following area.

6. CONCLUSION

In this review, the proof for the opposite sides of the SDN security coin has been exhibited; that it is conceivable to enhance arrange security utilizing the qualities of the SDN architecture, and that the SDN engineering presents security issues. The end is that the work on upgrades to organize security through SDN is progressively develop. This is confirm by the industrially accessible applications. Be that as it may, inquire about arrangements have been displayed to address a portion of the security issues presented by SDN e.g., how to constrain the potential harm from a malignant/traded off application. Work on these issues is creating empowered by the expanding security focal point of industry-supported institutionalization and research gatherings. We center around the potential risk of IP Spoofing assaults focusing on Open Flow diverts in IoT– Secure situation. We acquaint an assault demonstrate with tell the best way to perform such assault on our proposed SDN design. We likewise actualize three assault demos to uncover how the assault functions in detail. To identify such assaults, we additionally propose a countermeasure utilizing Bloom channel to distinguish IP Spoofing assault. A model of this Bloom channel screen is actualized by expanding the Open Flow protocol. The assessment result demonstrates that the Bloom channel strategy is both lightweight and effective.

REFERENCES

[1] *11th Annual Visual Networking Index: Global IP Traffic Forecast Update*, Cisco, San Jose, CA, USA, 2015.

[2] A. Wang, Y. Guo, F. Hao, T. V. Lakshman, and S.

Chen, "Scotch: Elastically scaling up SDN control-plane using vSwitch based overlay,"in *Proc. 10th ACM Int. Conf. Emerg. Netw. Exp. Technol.*, Sydney, NSW, Australia, 2014, pp.403–414.

[3] D. Wu, D. I. Arkhipov, E. Asmare, Z. Qin, and J. A.

McCann, "Ubiflow: Mobility management in urban-scale software defined IoT," in *Proc.IEEE INFOCOM*, Hong Kong, 2015, pp. 208–216.

[4] Y. Sheffer, R. Holz, and P. Saint-Andre,

"Summarizing known attacks on transport layer security (TLS) and datagram TLS (DTLS)," IETF, Fremont, CA, USA, RFC 7457, 2015.C. Hlauschek, M. Gruber, F. Fankhauser, and C.Schanes, "Prying open Pandora's box: KCI attacks against TLS," in *Proc. 9th USENIX WOOT*, Washington, DC, USA, 2015, p. 2.

[5] SSL Labs. *Survey of the SSL Implementation of the Most Popular Web Sites*.Accessed on Apr.

2016.[Online]. Available: https://www.trustworthyinternet.org/ssl-pulse/

[6] A. Cui, M. Costello, and S. J. Stolfo, "When firmware modifications attack: A case study of embedded exploitation," in

Proc. NDSS, San Diego, CA, USA, 2013.

[7] K. Chen, "Reversing and exploiting an apple firmware update," in *Proc. Black Hat*, Las Vegas, NV, USA, 2009.

[8] S. Hanna *et al.*, "Take two software updates and

see me in the morning: The case for software security evaluations of medical devices," in *Proc. HealthSec*, San Francisco, CA, USA, 2011,

p. 6.

[10] C. Miller, "Battery firmware hacking," in *Proc. Black Hat USA*, Las Vegas, NV, USA, 2011, pp. 3–4.

[11] B. Jack, "Jackpotting automated teller machines redux," in *Proc. Black Hat USA*, Las Vegas, NV, USA, 2010.

[12] *Austin Appleby*. Accessed on Apr. 2016.[Online]. Available: https://sites.google.com/site/murmurhash/

[13] S. Scott-Hayward, S. Natarajan, and S. Sezzer, "A survey of security in software defined networks," *IEEE Commun.Surveys Tuts.*, vol. 18, no. 1, pp. 623–654, 1st Quart., 2016.

[14] S. Shin *et al.*, "Fresco: Modular composable

security services for software-defined networks," in *Proc. NDSS*, San Diego, CA, USA, 2013.

[15] P. Porras, S. Cheung, M. Fong, K. Skinner, and V.

Yegneswaran, "Securing the software-defined network control layer," in *Proc. NDSS*, San Diego, CA, USA, 2015.

[16] S. Matsumoto, S. Hitz, and A. Perrig, "Fleet: Defending SDNs from malicious administrators," in

Proc. ACM WorkshopHot Topics Softw. Defined Netw., Chicago, IL, USA, 2014, pp. 103–108.

[17] S. Son, S. Shin, V. Yegneswaran, P. Porras, and G. Gu, "Model checking invariant security properties in

OpenFlow," in

Proc. IEEE ICC, Budapest, Hungary, 2013, pp. 1974–1979.

[18] S. Hong, L. Xu, H. Wang, and G. Gu, "Poisoning network visibility in software-defined networks: New attacks and countermeasures," in *Proc.NDSS*, San Diego, CA, USA, 2015.

[19] M. Dhawan, R. Poddar, K. Mahajan, and V. Mann, "SPHINX: Detecting security attacks in software-defined networks," in *Proc. NDSS*, San Diego, CA, USA, 2015.

[20] S. Yi, C. Li, and Q. Li, "A survey of secure computing: Concepts, applications and issues," in *Proc. ACM WorkshopMobile Big Data*, Hangzhou, China, 2015, pp. 37–42.

[21] S. Yi, Z. Qin, and Q. Li, "Security and privacy issues of secure computing: A survey," in *Proc. WASA*, Qufu, China, 2015, pp. 685–695.

[22] S. Yi, Z. Hao, Z. Qin, and Q. Li, "Secure computing: Platform and applications," in *Proc. IEEE HotWeb*, Washington, DC, USA, 2015, pp. 73–78.

[23] Z. Hao and Q. Li, "EdgeStore: Integrating edge computing into cloudbased storage systems," in *Proc. IEEE/ACM Symp.Edge Comput.*, Washington, DC, USA, 2016, pp. 115–116.

[24] Z. Hao, E. Novak, S. Yi, and Q. Li, "Challenges and software architecture for secure computing," *IEEE Internet Comput.*, vol. 21, no. 2, pp. 44–53, Mar./Apr. 2017.