A Dynamic Defense Using Client Puzzle for Identity-Forgery Attack on the South-Bound of Software Defined Networks

Zehui Wu[1], Qiang Wei[1], Kailei Ren[1] and Qingxian Wang[1]
[1] State Key Laboratory of Mathematical Engineering and Advanced Computing
Zhengzhou, Henan 450001 - China
[e-mail: wuzehui@foxmail.com, xdweiqiang@163.com, yoyoh_h@163.com, wangqingxian2015@163.com]
*Corresponding author: Zehui Wu

Received May 9, 2016; revised October 12, 2016; accepted November 29, 2016; published February 28, 2017

Abstract

Software Defined Network (SDN) realizes management and control over the underlying forwarding device, along with acquisition and analysis of network topology and flow characters through south bridge protocol. Data path Identification (*DPID*) is the unique identity for managing the underlying device, so forged *DPID* can be used to attack the link of underlying forwarding devices, as well as carry out DoS over the upper-level controller. This paper proposes a dynamic defense method based on Client-Puzzle model, in which the controller achieves dynamic management over requests from forwarding devices through generating questions with multi-level difficulty. This method can rapidly reduce network load, and at the same time separate attack flow from legal flow, enabling the controller to provide continuous service for legal visit. We conduct experiments on open-source SDN controllers like Fluid and Ryu, the result of which verifies feasibility of this defense method. The experimental result also shows that when cost of controller and forwarding device increases by about 2%-5%, the cost of attacker's CPU increases by near 90%, which greatly raises the attack difficulty for attackers.

Keywords: Cyber Security, Software Defined Networks, Southbound APIs, Dynamic Defense

This research was supported by the National High Technology Research and Development Program of China (Grant NO. 2012AA012902) and National Science Fund for Distinguished Young Scholars (Grant NO. 61402526).

https://doi.org/10.3837/tiis.2017.02.012 **ISSN : 1976-7277**

1. Introduction

Software Defined Networks (SDN) is the most popular emerging technology with rapid development in the field of network communication in recent years. SDN is regarded as one of the core technologies for next generation of the Internet [1] due to the disruptive change it brings to current network architecture. The nature of SDN is changing the current completely closed network architecture with integration of software and hardware, while at the same time opening up the programming ability of network to users which could decouple business from network [2][3]. **Fig. 1** shows the three-layer architecture of current Openflow-based SDN which achieves widespread adoption. It shows that the controller is the core of the SDN network because it fulfills user customization of application layer through the north bridge (REST API, etc.), and interacts with data forwarding layer through the south bridge (OpenFlow, OF-CONFIG, etc.) [4].

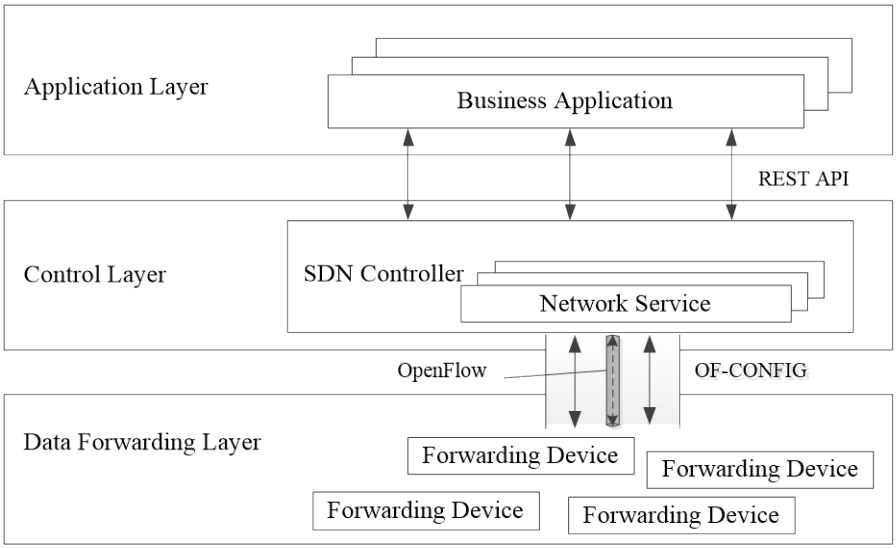


Fig. 1. Three-layer logical architecture of SDN

The south-bridge protocol is divided into two types, which are responsible for network control and network management, respectively. As shown in **Fig. 1**, the OpenFlow protocol is responsible for network control, while OF-CONFIG protocol is used for network management and device configuration. In a typical OpenFlow-based SDN network, OpenFlow is the only way for controller to complete issue of command for data forwarding and update of flow table, as well as pushing the topology information of forwarding facility. It is also the core channel for SDN to realize centralized control. SDN allocates a 64-bit *DPID* (Data Path Identification) for each forwarding device, which is used by the controller to update information and manage device during operations such as querying some information and issuing flow table. Therefore, *DPID* is the unique identifier for a forwarding device, and the change of *DPID*'s value will make controller identify a newly added forwarding facility in the network [5].

In 2013, Shin [6] and Dover [7][8] studied the message mechanism between controller and forwarding device in Floodlight, and found that if the forwarding device sends a FEATURES_REPLY message with a new *DPID* to the controller again after completion of the handshake, the controller will add a new item in the flow table. This finding indicates that

the changing the *DPID* value could make the controller add new record in flow table. With this vulnerability, the attacker could use a host to simulate large amount of forwarding devices to establish connection with the controller by creating new *DPID* to generate a FEATURES_REPLY message and sending it to the controller just after the handshake. With those attack methods shown above, memory space of the controller will be fully filled after a period of time, along with its CPU being almost occupied with full load. Thus the controller could not provide service for legal forwarding device any more, causing DoS (Denial of Service) attacks.

In both BlackHat 2014 and BlackHat 2015, Gregory [9][10] proposed and proved that the controller can not distinguish between two forwarding facility which have same hardware address (datapath_id and MAC address). Therefore, the attacker can utilize any host to simulate a forwarding device which has the same hardware characters with target device, and use it to send messages such as HELLO and FEATURES_REPLY to the controller, thus breaking the connection between the device and the controller.

In order to defend against the identity forgery attack described above, an optional security mechanism based on TLS protocol is proposed for the communication channel between controller and forwarding device in the subsequent version of OpenFlow v1.3 specification. This mechanism supports two-way authentication and encryption for communication, which, however, still has some shortages, because users are allowed to close the function of authentication and encryption, and no corresponding measure is designed to deal with the situation where the secure authentication fails [2][11]. Moreover, as most vendors worry about the effect of authentication and encryption on the system performance, the TLS functionality is disabled on default. Instead, a simple TCP handshake connection is used [12]. For those controllers with TLS encryption (e.g. Ryu), the TLS encryption is realized with OpenSSL, which has exposed many security vulnerabilities that attacker can use to bypass the encryption [13].

Based upon the above problems and the research foundation on them, we proposes a dynamic defensive mechanism to defend against identity forgery attack which utilizing the DPID of forwarding device to cheat SDN controller. There are two main differences between our method and existing methods: (1)Methods employing TLS or any other authentication mechanism cannot address DoS attack, because attackers can launch identity forgery or DoS attack against the authentication service because authentication does not eliminate the possibility of identity forgery or DoS attack, but just migrates the attack to other objects. (2) Current methods are static, while our approach is dynamic which improve the difficulty of attacks greatly.

This mechanism is based on Client-Puzzle, and has the following three advantages:

(1) For *DPID* forgery attack on forwarding device, which is proposed by Shin [6] and Dover [7], we can precisely identify forged *DPID* with the mechanism proposed in this paper to maintain the stability of current link.

(2) For DoS attack on controller caused by *DPID* identity forgery proposed by Dover [8] and Gregory [9], this mechanism is able to quickly reduce the network load, and at the same time separate the attack stream from the legal stream in all network flows, which enables the controller to serve the legal requests without interrupt.

(3) The defense for the above attacks would only increase a little in controller's load, but augment greatly in attacker's price, thus making the attacks more difficult.

This paper is organized as follows: the first part introduces the related work of this paper; the second part conducts an abstract analysis of the threat model for *DPID* attack; the third part describes the principle for dynamic defense based on Client-Puzzle model, along with its

execution flow; the fourth part gives a detailed exposition on the algorithm and its corresponding analysis of the Client-Puzzle model; the fifth part tests effect of the model; the last part concludes this paper, and discusses the future work.

2. Related Work

For security of the channel between SDN forwarding device and SDN controller, Pedigree [14] proposed a method of flow label in 2009. This method firstly labels all data flow forwarded to the controller, which would check legitimacy of the flow according to the labels. It would check whether data resource is under unauthorized visit, and whether matching with white list is successful. Whenever illegal data flow is detected, the corresponding flow rules are deployed in the forwarding device to filter such a flow. Although the method of flow label is able to solve problems such as unauthorized visit, escape and propagation of malicious code, it can not protect the resource under the condition that SDN has not classified different authority for different resource yet. Furthermore, the flow labels would increase the load of controller, which is then turned into bottleneck of the network.

In 2011, Yao [15] proposed a method based on original address determination to prevent the denial of service attack on SDN controller. This method uses VAVE (Virtual source Address Validation Edge) to identify IP Spoof. It takes advantage of traffic analysis and dynamic strategy update of SDN to redirect flows that do not conform to the predefined strategy of forwarding equipment to the controller, and let the controller to conduct legitimacy judgment of their original addresses. At the same time, the controller will dynamically update the strategy of forwarding equipment to limit reception of data flow with illegal original address. This method can reduce attack flow on the controller to some extent, but there are two shortcomings: (1) It relies on the illusion that DoS attack would not come from forwarding device, while in fact forwarding device in SDN is usually emulated by virtualization software (NFV), which means that attack flow might come from forwarding device controlled or simulated by attacker; (2) It greatly increases the load of the controller which on one hand needs to conduct legitimacy judgment of original IP addresses, while on the other hand needs to update strategies of all switches in the domain, leading to large amounts of load and bandwidth overhead.

With respect to the problem of VAVE controller as a bottleneck, ident++ [16] protocol can be used to reduce the load of controller, which would allocate tasks such as security authentication and policy updates to forwarding devices and hosts. Ident++ sets up a third party channel among hosts, forwarding devices, the controller and servers. When a host initiates visit request to a server through forwarding devices, the request is redirected to the controller, which would then use ident++ to require additional authentication information from the host and the server. If the authentication is successful, the controller would then send flow rules for forwarding packets to the forwarding devices which is responsible for forwarding requests from the host. There are also two disadvantages: (1) While it solves the problem of the controller being a bottleneck to some extent, it requires communication channel between the controller and the server, which is not included in the original SDN architecture; (2) It expands the attack surface with additional use of third party channels between hosts, forwarding devices, the controller and servers, and ident++ may introduce new security risks.

In 2014, Liyanage [17] took a detailed analysis on the threat model of security channel between the controller and the forwarding equipment, and pointed out that the security channel

is under the threats such as DoS attacks, replay attacks and IP Spoofing. Upon the analysis, He put forward the SDMN (software defined mobile networks, sdmn) oriented defense mechanism, which employs defensive strategies with very high practicability. However, this method cannot be directly applied to SDN, because SDN is not equipped with some of SDMN's characters like high mobility and cellular message encryption, and it would make controller become a performance bottleneck when applied in SDN.

For the disadvantages of the above posterior defense methods based on flow check, load migration or attack detection, the researchers studied the authentication mechanism among forwarding devices, users and the controller. In 2014, Dangovas [18] realized a Floodlight-based system called AAA to conduct authentication, authorization and measurement in SDN, thus enhancing the security of SDN network. AAA can carry out authentication and traffic binding for OpenFlow switches and users. Authentication is completed by RADIUS server, and access control is achieved with LDAP (Lightweight Directory Access Protocol) protocol. Nonetheless, this method imposes a large impact on network performance. Besides, it is not clearly defined how the access is implemented using LDAP.

In regards to high cost of AAA, Toseef [19] proposed C-BAS, a method for authentication and authorization management in cross-layer model. C-BAS is developed on basis of AAA, and it supports functions like identity authentication with user certificates and management for role authorization. Therefore, it can counter attacks against AAA, and improve scalability and performance in distributed management.

With the above analysis, we can know that the research path of defense for the security of controller and the secure channel between controller and forwarding device goes as 'attack detection→load migration→identity authentication'. The attack detection has the problem of uncontrollable false positive rate and false negative rate, the load migration is hindered by the performance bottleneck, and the identity authentication just migrates the attack load, and did not confront such an attack. The CP-based SDN dynamic defense for SDN south bridge presented in this paper can reduce network attack traffic, which helps filter out legitimate access flow. On the other hand, it can increase cost for the attacker, which greatly increased difficulty of the attack.

3. Attack Models and Hypotheses

3.1 Attack Models

In order to describe our method more clearly, we give the list of symbols used for this work, as shown in **Table 1**.

Table 1. Symbols for this work

Name	Description
DPID	the unique identifier of forwarding device, usually composed of the 48-bit MAC address of the device and a 12-bit prefix
$C = \{c_{i=1,2,\ldots}\}$	the set of controllers

$A=\{a_{i=1,2,...}\}$	the set of attackers
$S=\{s_{i=1,2,...}\}$	the set of forwarding devices
V	the number of connection request accepted by controllers from attackers and forwarding devices in a limited time
R	the resource of controllers
L	the length of buffer queue for flow table provided by controller C
$T^i_{lifecycle}$	the life cycle of the i-th item in flow table
τ_{RTT}	the time for heartbeat packets which refers to the round-trip time for connection maintaining packets to travel around between the controller and its connected forwarding devices.
$\tau_{average}$	the average time for legal devices to solve questions
P_i	$puzzle$ is represented by P_i, which is composed of several $sub-puzzle$s. $P_i[j]$ stands for the j-th $sub-puzzle$ of P_i. $P_i=\{P_i[1],P_i[2],...,P_i[m]\}$
M_i	i-th execution of model M, which means that i-th forwarding device (A_i or S_i) requests for connection with C
M_i^d	d-th message in i-th execution. For example, the first step of models for forwarding device (A_i or S_i) to request for connection with C, so M_i^1 is $A_i/S_i \rightarrow C$, M_i^2 is $C \rightarrow A_i/S_i$, etc.
$Z<i>$	i-th bit of bit sequence Z
$Z<i,j>$	sub-sequence of bit sequence Z between i-th bit and j-th bit
h	hash function such as MD5, with its length as l
g	times of hash operations per second conducted by the attacker

The attacker can forge *DPID* to carry out the above two types of attack on forwarding device and controller, respectively. A 5-tuple is defined to represent *DPID* attack model: $<C,A,S,V,R>$, in which $V: A \cup S \rightarrow C$ shows a mapping, indicating the number of connection request accepted by controllers from attackers and forwarding devices in a limited time, V_{max} is the maximum number of connection request that controllers can accept, $R=\{L,T^i_{lifecycle}\}$ is the resource of controllers.

According to the above two types of attack using forged *DPID* which are proposed by Shin [6], Dover [7][8] and Gregory [9], the *DPID* identity forgery attack can be abstracted into the following two models.

(1) Model1: Attack Model for Forwarding Device

For a legal forwarding device S_i with *DPID* value as S_i^{dpid}, S_i has set up connection with controller C_i, and $V_{C_i} \le V_{max}$, $L_{C_i} \le R.L_{max}$. At this time, attacker A_i virtualizes a forwarding device S_k which satisfies $S_k^{dpid} = S_i^{dpid}$, and uses S_k to request for connection with C_i. The completion of handshake between S_k and C_i would force S_i to go offline, thus accomplishing the attack against legal forwarding device, which is shown in **Fig. 2**.

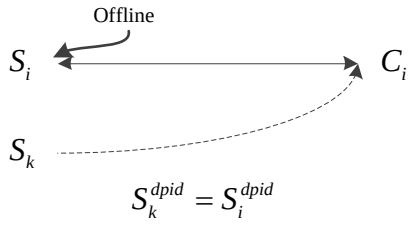


Fig. 2. Attack model for forwarding device

(2) Model2: Attack Model for Controller

Legal forwarding device S_i has set up connection with controller C_i. For C_i, the attacker creates scripts to forge large amounts of fake forwarding devices $S_{k=1,2,...}$, and $S_{k=1,2,...}^{dpid}$ is a randomly generated 64-bit sequence. Then the attacker uses $S_{k=1,2,...}$ to create connection with C_i one by one. When the value of k is large enough to satisfy $L_{C_i} > R.L_{max}$, the flow table of C_i would overflow to make S_i offline, as shown in **Fig. 3**.

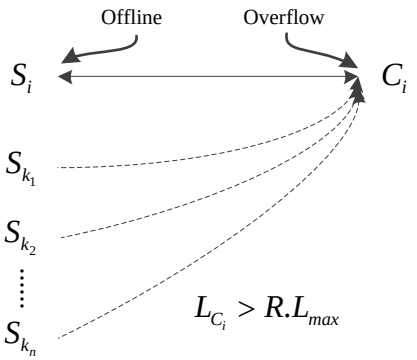


Fig. 3. Attack model for controller

When applying these two attack models described above to different controllers (e.g., Floodlight, OpenDaylight, Ryu, etc.), details of the attack differs. For example, when applying the models to Floodlight, it is necessary to set up connection first, at the end of which value of *DPID* should be modified to make flow table of the controller overflow.

3.2 Hypotheses

Client-Puzzle dynamic defense requires forwarding devices and attackers to solve the problems generated by the controller, so we need put forward hypotheses on their transaction processing logic and their performance.

Hypothesis 1: Forwarding devices and attackers would process data packets in the same order as their reception.

It means that in a certain period of time, forwarding devices may receive multiple problems from controllers, then they solve problems in the order of that 'what comes first would be handled and relied first'.

Hypothesis 2: Time for forwarding devices to solve the PUZZLE is stable.

It shows that if a forward device solves two problems of equivalent level of difficulty in twice, the time involved would not differ obviously. As in SDN, forwarding devices are only responsible for looking up the table and forwarding the data, so the CPU usage is relatively stable, which means this hypothesis is easy to meet.

Hypothesis 3: The attackers do not have unlimited computing ability or storage capacity.

The attackers cannot solve any difficult problems in a very short period of time. Because when the attacker has infinite computing ability, all of the security defenses would fail.

4. Client-Puzzle Defense Model

The model of Client-Puzzle originates from a paper that Juels [20] published in 1999 about using encryption algorithm to solve the problem of semi-connection attack, and [21-24] improved this method. The idea proposed can be summarized as follows: when a client requests for a connection, the server would generate a question which would be sent to the client for solution; the client would send the answer back to the server for verification after solving that problem. If the answer is successfully verified, the following interactions would continue; otherwise there would not be any subsequent interactions. The purpose of this method is to make solving the client-side question more difficult than server-side validation, raising the level of difficulty for the attackers.

For the above *DPID* attack model which is made up of 5-tuple, the process for controllers under Client-Puzzle model to deal with the connection form forwarding devices is shown in Algorithm. 1 which follows the presentation style of Kaiwartya [25-26].

(1) When controller C receives connection request from forwarding device S_2, it would extract *DPID* value of S_2 from the message msg, and look up the table. If a forwarding device S_1 with the same *DPID* is found to have connected to C, C would generate a question $puzzle$, and send it to S_2. Then S_2 would solve the question and send the $solution$ back to C for verification. After verifying the correctness of $solution$, C would calculate the time interval between the sending of $puzzle$ and the receiving of $solution$. If the time interval is greater than τ_{RTT} which is the time for heartbeat packets, C would determine that *DPID* value is illegal. If the time interval is no large than τ_{RTT}, C would believe that S_1 has dropped offline, and try to establish connection with S_2 ($S_1 = S_2$).

(2) When C has received a large number of requests from different forwarding devices

$S_{k=1,2,...}$ (their *DPID* value is shown in *DPID.recv*[]) in a short period of time, and the number of requests goes beyond the preset threshold, then C is considered to be under attack. At this time, C would generate a series of questions expressed as *puzzles*[], and send them to $S_{k=1,2,...}$ respectively. $S_{k=1,2,...}$ should solve *puzzles*[] one by one in a timely order, and each $S_{k=i}$ sends *solutions[i]* back to C. After verifying the correctness of *solutions[i]*, C would record the time interval from sending *puzzles*[i] to receiving *solutions[i]*. If the time interval is greater than $\tau_{average}$ which is the average time for legal devices to solve questions, $S_{k=i}$ is considered as illegal forwarding device.

Algorithm. 1: CP-HANDLER

Notations

DPID.recv: the value of *DPID* which received from switch
DPID.table: the list of *DPID*s
puzzle: the data structure of challenges constructed by the controller
solutions: the data structure of solutions for challenges sent by the controller
Threshold: the value of σ set by users
T: the data structure of time recorder

Input *DPID.recv*, *Threshold*

Process

1. initiallization
DPID.recv = recvFromSwitch(*msg*).getDPID()
OFInitialization() //OpenFlow Protocol initialization
2. if (*DPID.recv* == lookup(*DPID.table*)) **then**
puzzle = generate(*param1*,*param2*,…)
T.set()
send(*puzzle*)
…
recv(*solutions*[])
T.stop()
if (verify(*solutions*[]) && $T.stop - T.set \le \tau_{RTT}$ && $\tau_{RTT} < T_{lifecycle}$) **then**
Drop *DPID.recv* and use the previous
Else
return *DPID.recv* //the *DPID.recv* is illegal
end if
end if
3. if (num(*DPID.recv*[])$>$*Threshold*) **then** //under attack
puzzle[] = generate(*param1*,*param2*,…)
for each *puzzle* $\in$ *puzzles*[]
T[i].*set*()
send(*puzzle*)
…
recv(*solutions*[])
T[*i*].*stop*()
if (($T.stop - T.set$))$>\tau_{average}$ && verify(*solutions*[*i*])
Drop *DPID.recv*[*i*]
Else

OFExec()//execute the openflow protocol
end if
end for
end if
Output: *DPID.recv* //illegal value of *DPID*

If *puzzle* in the above stage '(2)' is difficult enough for attacker to solve it in time more than τ_{RTT}, the attack threatening forwarding devices is defended. Because the time for controller and forwarding device to exchange the heartbeat packets is τ_{RTT}, even if an attacker forges *DPID*, he/she cannot respond to controller in τ_{RTT} due to the lack of ability to solve *puzzle* within τ_{RTT}, while the legal forwarding devices can respond in τ_{RTT}, thus enforcing an accurate judgment on the legitimacy of *DPID*.

The above stage '(3)' alleviates the attack threatening the controller. When an attacker's computing power is the same as that of a legal forwarding device, the time required by the attacker to respond to the controller is larger than $\tau_{average}$ because there are a large number of *puzzles*[] to solve in a short time. Under this condition, the defense can not only reduce the attack flow on the controller, but also accurately determine the legal forwarding devices in *DPID.recv*[], and then only forwarding devices forged by the attacker are needed to be dealt with. When the attacker has far more computing power than that of the forwarding device, we can increase the difficulty of *puzzles*[] to meet $|T_i.stop()\text{-}T_i.set() - \tau_{average}| > \sigma$. At this time, time used by the attacker could not stay as stable as time used by the forwarding device (because the attacker's computing power is far more than that of a legal forwarding device, which means that when the number of *puzzles*[] changes, time used by the attacker would change too, while the time used by a forwarding device is unchanged as it only need to solve single *puzzle*), leading to the difference between the time used by the attacker and the time used by the forwarding exceeding preset threshold σ. Under this condition, $DPID.recv[i]$ is considered as forged by the attacker. Utilization of this method is also able to defend against denial of service attacks caused by overflow of the controller's flow table.

5. Design of Client-Puzzle Defense Model

5.1 SUB-PUZZLE Generation

Upon definition of 5-tuple and description of symbols in table 1, we give the process of SUB-PUZZLE Generation.

① Controller C uses parameters like time stamp t and hash function h to work out $x_i[j]$ with length of l, that is, $x_i[j] = h(t, M_i^1, j)$;

② Controller C uses hash function h again to compute $x_i[j]$ to get $y_i[j]$, that is, $y_i[j] = h(x_i[j])$;

③ Controller C selects a sub-sequence $x_i[j] < k+1, l >$ from $x_i[j]$, and combines it with $y_i[j]$. Thus, a $sub-puzzle$ is finally generated.

The concrete execution process is shown in **Fig. 4**.

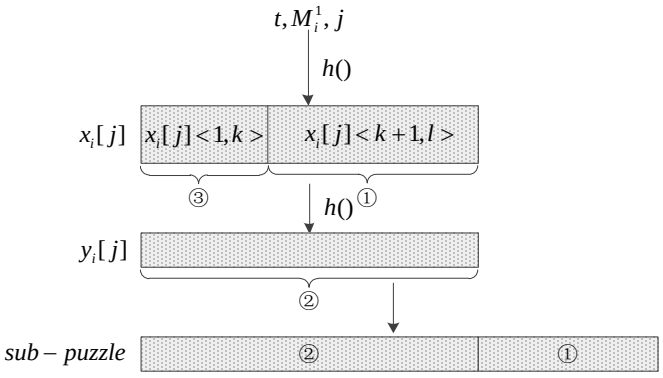


Fig. 4. Generation process for SUB-PUZZLE

5.2 SUB-PUZZLE Solving

The purpose of generating $sub-puzzle$s is to request A_i or S_i to work out the rest bits of $x_i[j]$, that is, $x_i[j]<1,k>$ based on the known $y_i[j]$ and $x_i[j]<k+1,l>$, while the controller is easy to verify using the complete $x_i[j]$. Of course the sub-sequences of $x_i[j]$ can be either continuous or discrete. For $sub-puzzle: \{y_i[j], x_i[j]<k+1,l>\}$, its solution is $x_i[j]<1,k>$, as shown in '③' of **Fig. 4**.

The steps for forwarding devices to work out $x_i[j]<1,k>$ with received $\{x_i[j]<k+1,l>+y_i[j]\}$ are as follows:

Known: length of $x_i[j]$, hash function h, $y_i[j]$ and $x_i[j]<k+1,l>$ Target: $x_i[j]<1,k>$
1. Set all bits of $x_i[j]<1,k>$ as 0; 2. Compute $h(x_i[j]<1,k>)$ to $y'_i[j]$; 3. If $y'_i[j]=y_i[j]$, get the solution of this SUB-PUZZLE; 4. If $y'_i[j]!=y_i[j]$, modify the last bit $x_i[j]<k>$ of $x_i[j]<1,k>$ from 0 to 1, and turn to step 2; 5. Repeat the steps until $y'_i[j]=y_i[j]$, then we will get the solution of SUB-PUZZLE.

According to the above steps, the worst case is that S could work out $x_i[j]<1,k>$ after 2^k hash operations (the average time is $2^k/2=2^{k-1}$). Therefore, this problem is equivalent to a lookup in a space of 2^k, and it only differs in that each lookup is replaced with hash operation.

5.3 PUZZLE Generation

Controller C would combine the above m $sub-puzzle$s so as to get P_i, and then send P_i to A_i or S_i. The solution of A_i or S_i is also the combination of the above m solutions.

So for P_i, the computational complexity of the controller C can be divided into two stages: the complexity of the construction phase is $2 \cdot m$ hash operations, and the complexity of validation phase is m hash operations. The computational complexity for A_i or S_i is: it needs $m \times 2^k$ hash operations in the worst case, and $m \times 2^k / 2 = m \times 2^{k-1}$ hash operations on average.

5.4 Security Analysis

(1) Difficulty of PUZZLE

For the attacker model for switches, when the attacker A_i is requesting for connection with C, he/she needs to receive and solve the *puzzle* sent by C. On average, the attacker needs $m2^{k-1}$ hash operations. So when the difficulty of *puzzle* meets the following relationship, the attacker would fail under Client-Puzzle defense model: $\frac{m2^{k-1}}{g} > \tau_{RTT}$, that is, $m > \frac{g\tau_{RTT}}{2^{k-1}}$.

For the attack model for controllers, when A_i or S_i requests connection with C, C would allocate a small amount of storage Mem^i_{slot} from Mem_C for interaction of *puzzle*. Assume that C is able to allocate memory for n A_i or S_i, then $n = \frac{Mem_C}{Mem^i_{slot}}$.

① When attacker A_i with the similar computing power with legal forwarding devices generates n different *DPID* to attack controller C, the hash operations needed by the attacker is $n \times m2^{k-1}$. When $\frac{Mem_C \cdot m \cdot 2^{k-1}}{Mem^i_{slot} \cdot g} > \tau_{average}$, that is, $m \cdot Mem_C > \frac{g \cdot \tau_{average} \cdot Mem^i_{slot}}{2^{k-1}}$, the DoS attack against controller C would fail;

② When attacker A_i with far more computing power than legal forwarding devices is conducting the attack, any $DPID_r$ which satisfies $|T_r.stop()\text{-}T_r.set() - \tau_{average}| > \sigma$ should be fake *DPID* forged by the attacker. Filtering n *DPID* would also make DoS attack against controller C fail.

(2) Storage Space of PUZZLE

According to characteristics of one-way hash function, when the length of sub-sequence chosen from $x_i[j]$ is 64-*bit*, a dictionary-based attack can be defeated. When the length of $y_i[j]$ is 64-bit, the hash function has only single solution, which would not conflict. Therefore, the length of *puzzle* should be no more than $16 \cdot m$ bytes.

In addition, as controller C stores nothing about the *puzzle* except timestamp, C needs to execute m times of $x_i[j] = h(s, t, M_i^1, j)$ to verify whether received solution is the same as calculated by C after receiving the solution of *puzzle*. Therefore, the order of verification is not required, and randomly choosing the solutions of $sub-puzzle$s to verify can reduce the number of verification.

6. Test and Evaluation

This section tests and evaluates the feasibility and overhead of Client-Puzzle dynamic defense method. For feasibility test, we choose open-source Fluid as platform for controllers and forwarding devices. For performance test, we use Ryu controller and Mininet simulation platform. The choice of lightweight controller Fluid as the feasibility test platform is due to that Fluid is a lightweight open-source controller without complex business processing logic of commercial controllers (e.g., Floodlight) such as QoS and load balance, and can therefore focus the test on target. On the other hand, choosing Fluid can reduce the complexity of prototype system. Selecting Ryu (v3.6) and Mininet (v2.2.1) as the performance test platform is due to that Ryu is an open-source commercial controller developed in Python, and not developed based on third-party framework (e.g., OpenDaylight is developed based on OSGi framework), which can reduce the impact of third-party framework, and improve practicability of the test. On the other hand, Mininet can simulate SDN networks with different topologies, and can realize seamless migration from simulation environment to real network, which facilitates the practical application of Client-Puzzle dynamic defense method. The parameters for test environment is listed in **Table 2**.

Table 2. Parameters for test environment

Target	**Configuration**
Fluid vm	Intel Core i7-2600 @3.40GHz, 1GB memory, 32-*bit* Ubuntu12.04
Open vSwitch 2.3.1 vm	
Ryu 3.6 vm	Intel Core i7-2600 @3.40GHz, 1GB memory, 32-*bit* Ubuntu14.04
Mininet 2.2.1 vm	
VMware11.1.0	Intel Core i7-2600 @3.40GHz, 8GB memory, 64-*bit* Windows7 SP1 Ultimate 64-*bit*

6.1 Test for Feasibility

We use C++ to implement the MD5 (140 lines) algorithm as the hash operation function. We then add the Client-Puzzle dynamic defense method into source file tls.cc under directory \libfluid_base\fluid of controller Fluid, and recompile the controller. After that, we modify the response module of Open vSwitch to add a Client-Puzzle calculation module. In such execution environment, the time of hash operations tested is about 10^5 times per second.

(1) Feasibility test for Model 1

According to the constraint $m > \frac{g\tau_{RTT}}{2^{k-1}}$, we set $m = 8$. Fluid is used as controller C, host installed with Open vSwitch is used as forwarding device S. The attacker A is also simulated using host with Open vSwitch installed. Network topology is shown in **Fig. 5(A)**.

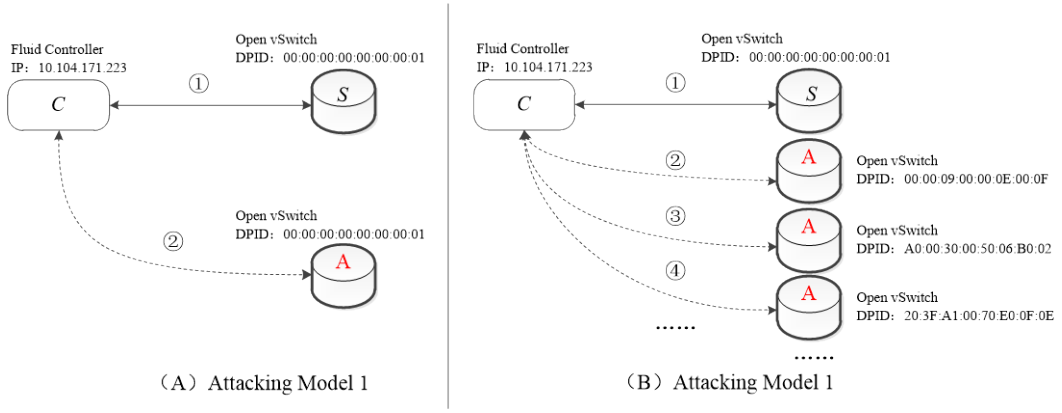


Fig. 5. Network topology for feasibility test

Before the attack, C has normal communication with S. The attacker uses Python script (sdn-evilswitc.py) to generate HELLO request packet with the same *DPID* as targeted host. The results in **Fig. 6** show the impact of CP model on attack model 1. It can be clearly observed that C's communication with S would break up without deployment of CP (Client-Puzzle) dynamic defense (in Non-CP model), and when enforcing the protection (CP model), normal communication between C and S would stay unaffected with information interaction after the attack.

Fig. 6. Prompting messages during feasibility test for attack model 1

(2) Feasibility test for Model 2

For attack model for the controller, there are two cases regarding relationship between the computing power of the attacker and the forwarding device, so there are also two cases about the difficulty $puzzle$:

$$\begin{cases} m \cdot Mem_C > \dfrac{g \cdot \tau_{average} \cdot Mem_{slot}^i}{2^{k-1}} & (Capacity_A \approx Capacity_S) \\ |T_r.stop()\text{-}T_r.set() - \tau_{average}| > \sigma & (Capacity_A \gg Capacity_S) \end{cases}$$

Based on the above constraints, we set: $g = 5 \times 10^5$, $k = 16$, $m = 8$, $\tau_{average}$ =3s, Mem_{slot}^i =256Byte, σ =10ms, Mem_C =14KBbyte.

We still use Fluid and Open vSwitch to set up the testing network, whose topology is shown in **Fig. 5(B)**. To simulate the different cases regarding the difference of the computing power between the attacker and the forwarding device, we simulate the attacker by installing Open vSwitch on hosts with Intel Core i7 equipped with single core, 3.4 GHz frequency and 1GB of memory and hosts with Intel Core i7 equipped with dual cores, 3.4 GHz frequency and 8 GB of memory, respectively. Before the attack, C communicates normally with S. The attacker uses Python script (sdn-controllerflood.py) to generate large amounts of OpenFlow HELLO request packets with the random *DPID*s. The prompting messages shown during the test is consistent with in **Fig. 7**. **Fig. 8** shows the difference of CPU load before and after use of method proposed in this paper, and the attack begins at 20th second.

The results depicted in **Fig. 7** show that when CP model is enabled, not only the normal communication between C and S is not affected, but also the controller's CPU and memory load would decrease after a period of increase. As it can be observed that in Non-CP model, the controller's CPU load increases gradually after a period of continuous attack, and finally gets close to 100%, at which time the legitimate requests would not get timely responses, so the normal communication between controller C and S would break up. With the knowledge of this paper, it is easy to discover that the increase of CPU and memory load is due to generating large amounts of *puzzle* s in a short period of time, and the ability to distinguish legal *DPID*s from fake *DPID*s makes the subsequent communication unaffected by forged *DPID*s, so the load would gradually go down. It should be pointed out that the number of *DPID*s is not enough for occupying all of controller Fluid's memory space in this paper, so CPU utilization could not reach close to full load, and we can only observe the changing curve of memory usage.

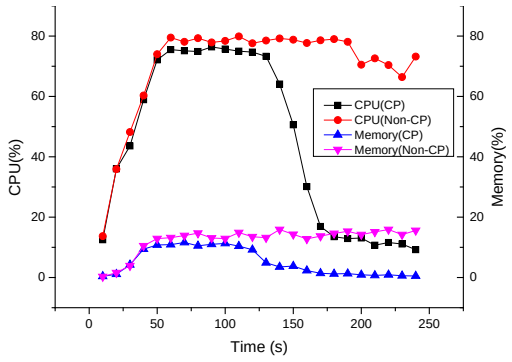


Fig. 7. Changing curve of controller's CPU and memory load

6.2 Test for Performance

We use Python to re-implement the MD5 algorithm and Client-Puzzle dynamic defense module in feasibility test, and we choose SDN controller Ryu implemented in Python as the controller for performance test. We modify the function OpenFlowController::server_loop()

in file Controller.py under directory Ryu/controller, and add dynamic defense module in it. The topology for performance test is shown in **Fig. 8**.

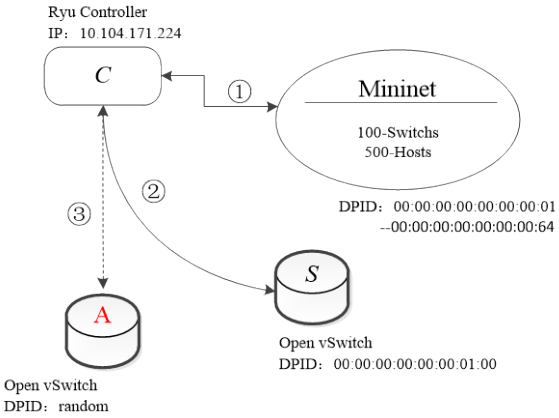


Fig. 8. Topology for performance test

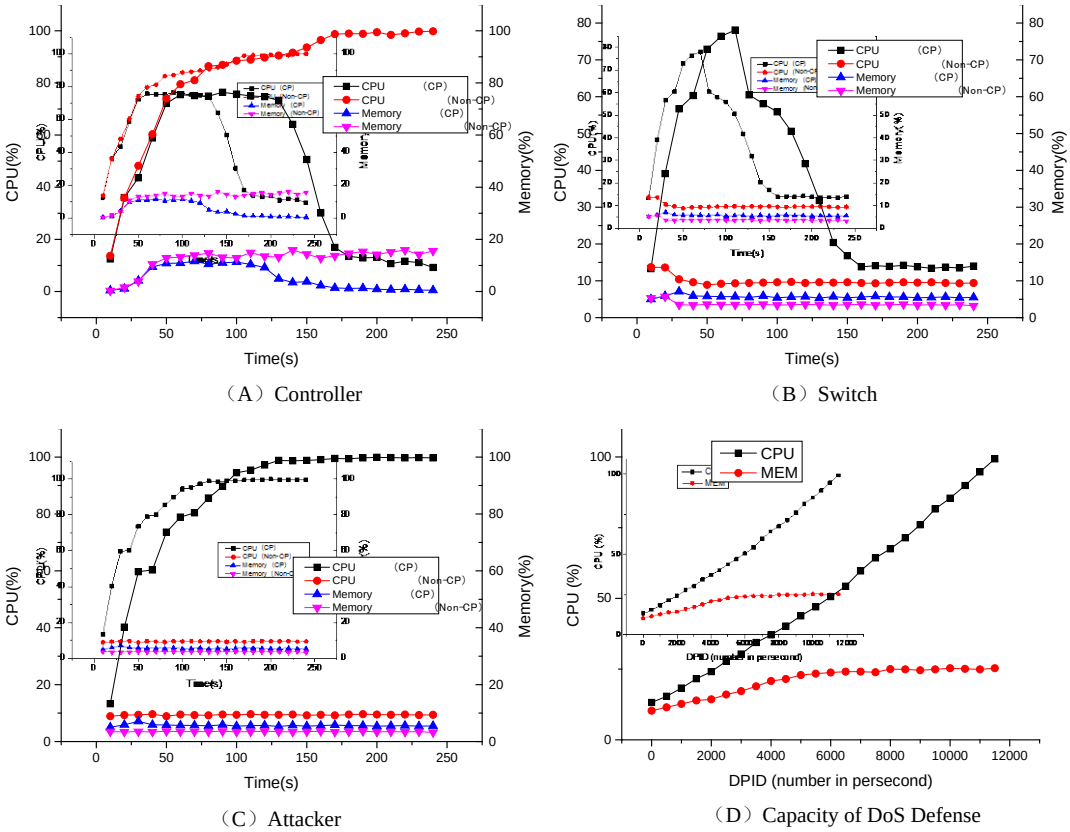
Performance test contains three aspects about the performance change before and after adding CP module in the controller, the forwarding device, and the attacker. It proceeds as follows: ① Using Mininet to simulate SDN network with 100 forwarding devices and 500 hosts, along with Ryu as its controller. Using commad *pingall* in Mininet to connect the whole network; ② Using Open vSwitch equipped with CP module as the object of performance test for forwarding device S, which is remotely connecting to controller C; ③ Using a copy of S as the object of performance test for attacker A, and executing attack script to begin test at 20-th second after the connection of network. The results are shown in **Fig. 9**.

The results in **Fig. 9(A)** indicate that CPU usage differs in Non-CP model and CP model, with the former having a short rise after the attacker initiating the script, while the latter having stabilized high CPU usage due to its busy dealing with too many fake *DPID*s produced by the attacker. Although the memory usage is relatively low, it changes in the similar way with CPU usage. **Fig. 9(A)** has similar changing curve with that in **Fig. 8** with some different details, mainly because **Fig. 8** shows the changing curve for lightweight controller Fluid, while **Fig. 9(A)** shows the changing curve for commercial controller Ryu.

The results in **Fig. 9(B)** clearly convey that our CP model has little effect on the performance of switches . As Ryu which is the controller used in performance test inherently brings memory release mechanism, its memory usage in Non-CP model would gradually go down. After the attacker launches the attack at 20-th second, the CPU usage and memory load of forwarding device in CP model would go up first, followed by a gradual decrease. When CP model is not enabled, as the attacker has generated large amounts of fake *DPID*s in a short period of time, legal forwarding device is forced to lose connection due to processing delay of the controller, so instead, CPU and memory utilization will decrease.

The results in **Fig. 9(C)** indicate that CP-based dynamic defense could significantly increase the attacker's load, raising the cost of the attack. In Non-CP model, the attacker needs simply to generate fake DPIDs, so the CPU and memory usage would not fluctuate. While in CP model, the CPU and memory usage would keep high due to a large number of *puzzle*s the attacker has to solve. **Fig. 9(A)** to **9(C)** indicates that when the performance of controller and legal forwarding device increases by about 2%-5%, the overhead of the attacker's CPU increases by about 90%.

From the results depicted in **Fig. 9(D)**, it can be clearly observed that the load of the controller is approaching 100% under 12000 forged DPIDs, but the memory usage is growing stable gradually due to memory release. The general number of forwarding devices in current data center network is not more than 5000 (with each forwarding device corresponding to one DPID value), so the method proposed in this paper could defend effectively against DoS attack on the controller in current network scale. However, as the size of the network increases, the performance may become a bottleneck. Therefore, in the future, we can study how to apply the CP model in parallel environment to reduce the performance influence of our method on the controller and the switch, such as making the controller generate puzzles and verify solutions in parallel.



（A）Controller （B）Switch

（C）Attacker （D）Capacity of DoS Defense

Fig. 9. Results of performance test

6. Conclusion

In this paper, we studied the *DPID* forgery attack over the southbound interface of Openflow-based SDN controller, and proposed a CP-based dynamic defense method with regard to attack on forwarding devices and DoS attack on controllers using *DPID* identity forgery. This method can on one hand reduce the network attack flow and filter out the legal flow, on the other hand increase the overhead of attacker to raise the difficulty for the attack. The dynamic defense in this paper uses the experience of MTD (move target defense). Future work is the research into how to combine MTD with dynamic programmability of SDN, and apply that to defense for north bridge of SDN.

References

[1] D. Kreutz, V. Ramos, and P. Esteves, "Software-defined networking: a comprehensive survey," *Journal of Proceedings of the IEEE*, vol. 103, no. 1, pp. 14-76, 2015. Article (CrossRef Link)

[2] I. Alsmadi and D. Xu, "Security of software defined networks: a survey," *Journal of Computer and Security*, vol. 53, no. 3, pp. 79-108, 2015. Article (CrossRef Link)

[3] S. Scott, S. Natarajan, and S. Sezer, "A survey of security in software defined networks," *Journal of Communications Surveys & Tutorials*, vol. 18, no. 2, pp. 1-21, 2015. Article (CrossRef Link)

[4] M. M. Wang, J. W. Liu, J. Chen J and et al., "Software Defined Networking: security model, threats and mechanism," *Journal of Software*, vol. 27, no. 4, pp. 1-22, 2016. Article (CrossRef Link)

[5] F. Hu, Q. Hao and K. Bao, "A survey on software-defined network and OpenFlow: from concept to implementation," *Journal of Communications Surveys & Tutorials*, vol. 16, no. 4, pp. 2181-2206, 2014. Article (CrossRef Link)

[6] S. Shin, and G. Gu, "Attacking software-defined networks: a first feasibility study," in *Proc. of the ACM SIGCOMM workshop on Hot topics in software defined networking*, pp. 165-176, 2013. Article (CrossRef Link)

[7] M. Dover, "A switch table vulnerability in the Open Floodlight SDN controller," http://dovernetworks.com/wp-content/uploads/2014/03/OpenFloodlight-03052014.pdf, (Access on 2016-03-30).

[8] M. Dover, "A denial of service attack against the Open Floodlight SDN controller," http://dovernetworks.com/wp-content/uploads/2013/12/OpenFloodlight-12302013.pdf, (Access on 2016-03-30).

[9] P. Gregory. "Staying persistent in Software Defined Networks," https://www.blackhat.com/docs/us-15/materials/us-15-Pickett-Staying-Persistent-In-Software-Defined-Networks-wp.pdf, (Access on 2016-03-30).

[10] P. Gregory. "Abusing Software Defined Networks," https://www.blackhat.com/docs/eu-14/materials/eu-14-Pickett-Abusing-Software-Defined-Networks-wp.pdf, (Access on 2016-03-30).

[11] I. Sarmal, S. Singh and A. Singh, "Introducing restricted access protocol to enhance the security and eliminate DDoS attack," *Jounal of Computer Security*, vol. 43, no. 4, pp. 540-553, 2016.

[12] R. Durner and W. Kellerer, "The cost of security in the SDN control plane," http://www.lkn.ei.tum.de/forschung/publikationen/dateien/Durner2015ThecostofSecurity.pdf, (Access on 2016-03-30).

[13] B. Cache, "A timing attack on OpenSSL constant time RSA," https://ssrg.nicta.com.au/projects/TS/cachebleed/cachebleed.pdf, (Access on 2016-03-30).

[14] A. Ramachandran, Y. Mundada and M. Tarig, "Securing enterprise networks using traffic tainting," *Report of Georgia Inst. Technol.*, GTCS-09-15, 2009.

[15] G. Yao, J. Bi and P. Xiao, "Source address validation solution with OpenFlow/NOX architecture," in *Proc. of International Conference on Network Protocols*, pp. 7-12, 2011. Article (CrossRef Link)

[16] A. Akhunzada, "Secure and dependable software defined networks," http://dx.doi.org/10.1016/j.jnca.2015.11.012, (Access on 2016-03-30).

[17] M. Liyanage, M. Ylianttila and A. Gurtov, "Securing the control channel of software-defined mobile networks," in *Proc. of International Conference on Wireless*, pp. 1-6, 2014. Article (CrossRef Link)

[18] V. Dangovas and F. Kuliesius, "SDN-Driven authentication and access control system," in *Proc. of International Conference on Digital Information, Networking, and Wireless Communications*, pp. 20-23, 2014.

[19] U. Toseef, A. Zaalouk, T. Rothe and et al., "CBAS: Certificate-based AAA for SDN experimental facilities," in *Proc. of International Conference on European Workshop of Software Defined Networks*, pp. 91-96, 2014. Article (CrossRef Link)

[20] A. Juels and J. Brainard, "Client puzzles: a cryptographic defense against connection depletion attacks," in *Proc. of International Conference on Network and Distributed System Security Symposium*, pp. 27-39, 1999.
[21] F. Wang and K. Reiter, "A multi-layer framework for puzzle-based denial-of-service defense," *Journal of Information Security*, vol. 7, no. 4, pp. 243-263, 2008. Article (CrossRef Link)
[22] J. Clark and A. Essex, *Commitcoin: Carbon dating commitments with bitcoin*, Springer, Heidelberg, 2012.
[23] J. Becker, D. Breuker and T. Heide, *Can we afford integrity by proof-of-work? Scenarios inspired by the Bitcoin currency*, Springer, Heidelberg, 2013.
[24] R. Bohme, N. Christin and B. Edelman, "Bitcoin: economics, technology, and governance," *Journal of Economic Perspectives*, vol. 29, no. 2, pp. 213-238, 2015. Article (CrossRef Link)
[25] O. Kaiwartya, S. Kumar, K. Lobiyal and et al., "Performance improvement in geographic routing for vehicular Ad Hoc networks," *Sensors*, vol. 14, no. 12, pp. 22342-22371, 2014. Article (CrossRef Link)
[26] O. Kaiwartya and S. Kumar, "Cache agent-based geocasting in VANETs," *International Journal of Information and Communication Technology*, vol. 7, no. 6, pp. 562-584, 2015. Article (CrossRef Link)

Qiang Wei born in 1979, Master's supervisor. His main research interests include ICS security, software vulnerability analysis and SDN network security.

Zehui Wu born in 1988, doctoral candidate. His research interest includes SDN network security.

Kailei Ren born in 1992, Master Degree Candidate. His research interest includes SDN network security.

Qingxian Wang born in 1960, PhD supervisor. His main research interest includes network security.