Design and Implementation of Trusted Connection in Software-Defined Networking: Postprint

Authors: Li Zhaobin, Liu Mengtian, Wei Zhanzhen, Wang Shourong

Date: 2018-05-18T00:00:00+00:00

Abstract

Software Defined Networking (SDN) separates the control plane and data forwarding plane, enabling unified management of the data forwarding plane by the control plane. Currently, the integrity authentication mechanism between devices in the control and data forwarding planes remains inadequate; should devices with compromised platform integrity access the network, they would pose severe security threats to the entire SDN infrastructure. To ensure that connections are established only when both communicating devices are in a complete and trustworthy state, thereby guaranteeing device security and network trustworthiness from the source, this paper proposes a novel SDN trusted connection scheme. This scheme builds upon trusted network remote device authentication technology, leverages the Trusted Platform Module as a trust anchor, and incorporates an integrity authentication process during connection establishment between SDN data forwarding devices and controllers. Testing and analysis demonstrate that the proposed scheme is effective, feasible, and suitable for practical deployment.

Full Text

Design and Realization of SDN Trusted Connection

Li Zhaobin, Liu Mengtian, Wei Zhanzhen, Wang Shourong
(Beijing Electronic Science and Technology Institute, Beijing 100070, China)

Abstract: Software-Defined Networking (SDN) separates the control layer from the data forwarding layer, enabling unified management of the data forwarding layer by the control layer. Currently, the integrity authentication mechanism between control layer and data forwarding layer devices is not yet perfect. If devices with compromised platform integrity connect to the network, they pose serious security threats to the entire SDN network. To ensure that connections are established only after both parties' devices are verified as complete and

trustworthy, thereby guaranteeing device security and network trustworthiness from the source, this paper proposes a novel SDN trusted connection scheme. Based on trusted network remote device authentication technology and utilizing the Trusted Platform Module (TPM) as a trust anchor, this scheme adds an integrity authentication step during the connection process between SDN data forwarding devices and controllers. Test analysis demonstrates that the scheme is effective, feasible, and suitable for practical network environments.

Key Words: software defined networking; trusted connection; integrity authentication; network security

0 Introduction

Software-Defined Networking (SDN) separates the control layer from the data forwarding layer, enabling the control layer to uniformly manage and configure the data forwarding layer through interfaces, thereby simplifying network management, improving network flexibility, and reducing network adjustment costs. However, this enhanced variability also introduces security issues that require urgent resolution. One critical challenge is ensuring that devices in both the control and data forwarding layers can establish connections normally and securely. When a data forwarding device (Open vSwitch, OVS) newly joins the network, it can establish an underlying communication connection by configuring the controller' s IP address in the control layer, and then negotiate the highest mutually supported OpenFlow protocol version. If the negotiation succeeds, the connection is established. However, neither the controller nor OVS performs platform integrity authentication on the peer device during connection establishment, meaning both types of devices could potentially connect to compromised equipment. As the core of SDN, once a controller suffers an attack and fails to function properly, it may lead to network paralysis. Meanwhile, OVS, as the specific executor of tasks, if exploited by malicious devices, can similarly impact network packet forwarding and even the entire network [1,2]. Therefore, adding device integrity verification before connection establishment is extremely necessary to address the authentication problem between controllers and OVS.

Regarding authentication for connection establishment, the OpenFlow specification allows users to independently choose whether to adopt TLS security mechanisms when establishing channels. However, the complexity of TLS deployment in practice and its inherent limitations make secure channel implementation difficult, leading most users to prefer establishing connection channels using TCP plaintext. Moreover, TLS protects data but cannot assist in verifying the integrity of device software and hardware [3]. Additionally, while introducing a third-party certification authority to issue certificates for devices is another authentication solution, its complex processing has prevented widespread adoption. Physical isolation is also considered an effective method for protecting controllers and other devices, but this approach imposes high requirements on

deployment environments and administrator management, making it difficult to implement [3]. Zhang et al. [4] proposed recovery mechanisms such as path protection and link protection after controller-switch failures, which can effectively help attacked devices restore functionality in a short time. However, if already-connected counterfeit devices repeatedly launch attacks, they will still impact the network. Zhou [5] and Pan [6] respectively introduced trusted computing technology to complete inter-controller authentication or improve switch protocols, but these approaches only addressed specific aspects.

It is evident that existing device authentication schemes rarely mention authentication between controllers and switches. However, establishing secure connections as the starting point for all functions is crucial for overall network operation, making it self-evident that improving the connection process is necessary. After analysis, this paper combines the trusted network remote attestation protocol, utilizes the Trusted Platform Module as a trust anchor, selects Ryu as the controller, and adds a device integrity authentication process without changing the original workflow. This approach modifies and extends the OpenFlow protocol, OVS system, and Ryu system to achieve trusted connections between Ryu and OVS, preventing compromised devices from threatening network security.

1.1 SDN and OpenFlow

The SDN architecture [7] is shown in [Figure 1: see original paper], consisting of the application layer, control layer, and data forwarding layer. The control layer interacts with the other two layers through northbound and southbound interfaces (such as OpenFlow). The control layer calls the data forwarding layer according to application layer functional requirements, while the data forwarding layer performs data forwarding, storage, and uploading based on instructions from the control layer.

OpenFlow, as the standard specification for building SDN networks, is a centralized control model whose elements include controllers, OVS, connections established through OpenFlow channels between them, and OVS flow tables [8]. In SDN, channel establishment, device configuration, and periodic interaction are all implemented through OpenFlow messages. OpenFlow supports three types of messages: Controller-to-Switch, Asynchronous, and Symmetric [8]. Symmetric messages can be initiated by either controllers or data forwarding devices, typically used for device initialization without requiring prior requests or confirmations. The Hello messages sent before controllers and data forwarding devices establish connections and the periodically sent Echo messages both belong to the Symmetric category. Although these are the simplest message types, they are indispensable for channel establishment and maintaining stable connections.

1.2 Trusted Platform and Remote Device Trusted Authentication

In trusted platforms, the Attestation Identity Key (AIK) is used to provide platform identity proof [9]. Platform Configuration Registers (PCR) record system operational status by storing dynamic trust measurement digest values [10]. Since PCR values are critical for device integrity authentication, to prevent malicious tampering or forgery, registers cannot be read or written arbitrarily through ports but can only be modified through reset and extend operations.

Trusted computing authentication technology utilizes embedded hardware TPM encryption chips to execute an irreversible trust transfer mechanism during device startup. Following the principle of measuring each layer by the previous one, it completes measurement from system firmware to the device platform and stores each stage' s trust measurement digest values in PCR [11,12]. By comparing PCR values, it identifies whether device software and hardware have been tampered with, rather than relying solely on user identity authentication after system startup completion. However, when remote device identity authentication is involved, trusted authentication technology needs to cooperate with remote attestation technology.

Currently, Direct Anonymous Attestation (DAA) is the most basic remote attestation scheme widely applied in remote identity device authentication. This scheme determines the challenger and prover from different trust domains in the interaction, then completes authentication through a challenge/response protocol that exchanges and verifies authentication values [13]. With remote attestation technology, trusted platforms can also generate integrity reports for remote devices. The remote challenger first generates a random number and sends it to the prover. The prover signs the current device' s PCR values along with the random number using the attestation identity key and reports it to the challenger. The challenger can use the random number to prevent replay attacks and verify the prover' s integrity by comparing its PCR values with expected values ([Figure 2: see original paper]).

---

2 SDN Trusted Connection Design

2.1 Trusted Connection Model Design

The SDN trusted connection model spans the control layer and data forwarding layer, primarily composed of the Trusted Platform Module, controller, OVS, and a trusted connection channel responsible for integrity authentication. In this model, OVS and controllers verify each other' s platform integrity check values when establishing connections to determine whether the device is trustworthy, thereby deciding whether to complete the connection establishment. Only when both parties pass authentication simultaneously can the connection be successfully established and continue with subsequent configuration query interactions.

If either party fails authentication, the connection is terminated. The trusted connection model is shown in [Figure 3: see original paper].

2.2 Integrity Authentication Process Overview and Security Proof

2.2.1 Specific Integrity Authentication Process The integrity authentication process is a crucial component of the SDN trusted connection model, with specific procedures shown in [Figure 4: see original paper]. Symbol explanations for the integrity authentication process are provided in .

TABLE:1 Symbol Explanations for Integrity Authentication Process

Symbol	Meaning
R	Controller, one of the authentication subjects
O	Switch, one of the authentication subjects
nonce_ryu	Random number generated by Ryu during authentication request, hashed with corresponding switch measurement values
nonce_ovs	Random number generated by OVS during authentication request, hashed with corresponding controller measurement values
PCR_IDR	PCR index specified by Ryu during authentication request, OVS must provide specific values for authentication based on this index
PCR_IDO	PCR index specified by OVS during authentication request, Ryu must provide specific values for authentication based on this index
OPUB	OVS pre-configured public key
RPUB	Ryu pre-configured public key
O-1	OVS pre-configured private key
R-1	Ryu pre-configured private key
R_PCR	PCR value of Ryu at specified index
O_PCR	PCR value of OVS at specified index
AIK_OVS-1	OVS Attestation Identity Key private key for verifying data signatures
AIK_Ryu-1	Ryu Attestation Identity Key private key for verifying data signatures

Symbol	Meaning
Hash( )	Hash function based on SHA-1 engine in TPM

2.2.2 Formal Description and Security Proof of Integrity Authentication Process BAN logic [5] is the most influential tool among various logic systems for proving protocol security, consisting of 10 basic syntax and semantics. Additionally, BAN logic inference includes 5 main rules composed of 7 theorems. Below is the security proof of the integrity authentication process using BAN logic.

1) **Formal Description of Integrity Authentication Process**

```
R → O: {{nonce_ryu||PCR_IDR}R-1}OPUB
O → R: {{Hash(O_PCRid||nonce_ryu) ||nonce_ryu||IDO} AIK_OVS-1}RPUB
O → R: {{nonce_ovs||PCR_IDO}O-1}RPUB
R → O: {{Hash(R_PCRid||nonce_ovs) ||nonce_ovs||IDR} AIK_Ryu-1}OPUB
```

2) **BAN Logic Security Objectives**

Based on the security requirements of the integrity authentication process, the security objectives are set as shown in .

TABLE:2 Security Objectives for Integrity Authentication Process

Objective	Meaning

3) **BAN Logic Initialization Assumptions**

To verify the above BAN logic security objectives, the following initialization assumptions conforming to BAN logic are made:

```
|   _   →                    |   _   →
| #(   _  )              | #(   _  )
| #( _  )                  | #( _  )
| #( )                          | #( )
| #( _  id)                  | #( _  id)
```

4) **BAN Logic Security Proof**

Using the security initialization assumptions from 3), we reason and verify the security objectives from 2):

Inference 1: From R → O: {{nonce_ryu||PCR_IDR}R-1}OPUB, we know O {nonce_ryu||PCR_IDR}R-1. Since | , according to the message meaning rule [5], we can infer: | |~nonce_ryu||PCR_IDR. According to the message reception rule and belief rule [5], we can infer: | |~nonce_ryu and | |~PCR_IDR. Since | #(nonce_ryu) and | #(R_PCRid), according to the nonce verification rule [5], we obtain | | nonce_ryu and | | PCR_IDR.

Inference 2: From O → R: {{Hash(O_PCRid||nonce_ryu) ||nonce_ryu||IDO}AIK_OVS-1}RPUB, and since | _ → , we have {Hash(O_PCRid||nonce_ryu)||nonce_ryu||IDO}AIK_OVS-1. According to the message meaning rule [5], we can infer: | |~Hash(O_PCRid||nonce_ryu)||nonce_ryu||IDO According to the message reception rule and belief rule, we can infer: | |~O_PCRid and | |~IDO. Since | #(O_PCRid) and | #(IDO), according to the nonce verification rule, we obtain | | O_PCRid and | | IDO.

Inference 3: From O → R: {{nonce_ovs||PCR_IDO}O-1}RPUB, we can apply similar reasoning as Inference 1 to obtain | | nonce_ovs and | | PCR_IDO.

Inference 4: From R → O: {{Hash(R_PCRid||nonce_ovs) ||nonce_ovs||IDR}AIK_Ryu-1}OPUB, we can apply similar reasoning as Inference 2 to obtain: | | R_PCRid and | | IDR.

From the above BAN logic reasoning, the results are consistent with the security objectives set in 2). This process can guarantee the security of information such as nonces and PCR values, demonstrating that the integrity authentication process satisfies the security requirements of the trusted connection model.

2.3 Trusted Connection Establishment Process

The trusted connection establishment process is shown in [Figure 5: see original paper]:

1. OVS and Ryu perform connection initialization and begin establishing a connection;
2. Send OFPT_HELLO messages to negotiate the highest mutually supported OpenFlow protocol version;
3. Ryu initiates an authentication request, sending an OFPT_AUTH_REQUEST message (containing {{nonce_ryu||PCR_IDR}R-1}OPUB);
4. OVS responds to Ryu' s authentication request, replying with an OFPT_AUTH_COMPARE message (containing {{Hash(O_PCRid||nonce_ryu) ||nonce_ryu||IDO} AIK_OVS-1}RPUB);
5. OVS initiates an authentication request, sending an OFPT_AUTH_REQUEST message (containing {{nonce_ovs||PCR_IDO}O-1}RPUB);

6. Ryu responds to OVS' s authentication request, replying with an OFPT_AUTH_COMPARE message (containing {{Hash(R_PCRid||nonce_ovs) ||nonce_ovs||IDR} AIK_Ryu-1}OPUB);
7. Ryu authenticates OVS. If authentication fails, connection establishment fails and an OFPT_ERROR message indicating authentication failure is sent; if successful, the connection is established and subsequent OFPT_FEATURES_REQUEST messages are sent to query OVS configuration;
8. OVS authenticates Ryu. If authentication fails, connection establishment fails and the connection is directly dropped; if successful, the connection state is set to success and OFPT_FEATURES_REPLY messages continue to be received and replied;
9. Periodically exchange FPT_ECHO_REQUEST/OFPT_ECHO_REPLY messages to confirm connection status.

The OpenFlow channel establishment is completed. The specific program execution process in OVS is shown in [Figure 6: see original paper].

2.4 OVS and Ryu Connection Establishment Process

OVS can configure the Ryu address to establish an underlying connection, at which point there is no OpenFlow protocol message exchange. After the OVS system starts running, it uses connection management entities to identify whether the underlying connection has been established. If established, OVS sends an OFPT_HELLO message to Ryu for negotiating the protocol version. Successful negotiation indicates that the connection establishment is complete. The OFPT_HELLO message is used to negotiate the protocol version, and successful negotiation means the connection is established.

In OVS, three entities—ofconn, rconn, and vconn—manage connections hierarchically. vconn, as the most basic connection management entity, is primarily responsible for sending and receiving OFPT_HELLO messages and can quickly report the current connection status upward (as marked in [Figure 6: see original paper]). rconn and ofconn will execute subsequent actions based on the results passed by vconn. Ryu is not as complex as OVS. After startup, Ryu begins receiving OVS messages and responding. After version negotiation is consistent, Ryu confirms connection establishment and proceeds with configuration. Therefore, SDN trusted connection implementation can start from OVS' s three connection management entities and Ryu' s connection response components.

2.5 SDN Trusted Connection Implementation

The SDN trusted connection implementation flowchart is shown in [Figure 5: see original paper].

2.5.1 Adding OpenFlow Trusted Connection Message Types 1) Integrity Authentication Request Message (OFPT_AUTH_REQUEST)

As shown in [Figure 7: see original paper], the message source and data content each occupy 8 bits, respectively indicating the message sender and the carried data content; the data field is 160 bytes, used to send the signed and encrypted random number nonce_ryu or nonce_ovs and the specified PCR index.

TABLE:3 Trusted Connection Authentication Request Message Description

source	content	Meaning
0000	0000	Source is OVS, data content is random number
0000	0001	Source is OVS, data content is other (for extension)
0001	0000	Source is Ryu, data content is random number
0001	0001	Source is Ryu, data content is other (for extension)

2) Integrity Check Value Reply Message (OFPT_AUTH_COMPARE)

As shown in [Figure 8: see original paper], the message source and data placement method each occupy 8 bits, respectively indicating the message sender and how the carried data content is placed into the system; the data field is 160 bytes of encrypted and signed integrity check values.

TABLE:4 Trusted Connection Authentication Value Reply Message Description

source	means	Meaning
0000	0000	Source is OVS, data is automatically placed
0000	0001	Source is OVS, data is manually placed by administrator
0001	0000	Source is Ryu, data is automatically placed
0001	0001	Source is Ryu, data is manually placed by administrator

Considering various version protocol message situations, the authentication request message identifier is set to 40, and the integrity check value reply message identifier is set to 41.

3) Adding Error Message Type—Authentication Failure

When a switch' s identity fails controller authentication, the controller issues an error message of type AUTH_FAILED, indicating the authentication and connection establishment status of the switch attempting to connect. Instead of directly dropping the connection, the controller first sends an error message explaining the authentication failure reason. When a controller is considered an untrusted device by a switch, the switch disregards the failure reason and directly drops the connection, stopping further message reception.

2.5.2 Ryu System Trusted Connection Implementation 1) Adding Authentication State Scheduler

An authentication state scheduler is added between the handshake and config schedulers in Ryu' s core program handler.py, with corresponding schedulers added in files executing specific functions such as ofp_handler.py to ensure smooth operation of related functions. Ryu originally had four schedulers: handshake, config, dead, and main, where handshake and config respectively handle pre-connection Hello handshaking and post-connection configuration. Before executing corresponding functions for sending/receiving Hello messages or sending configuration query messages, the state must first be adjusted.

2) Defining Newly Added Variables

In various version files such as ofproto_v1_0.py, ofproto_v1_2.py, ofproto_v1_3.py, and ofproto_v1_4.py, newly added OpenFlow message identification codes, message formats, source variable identification codes in messages, and authentication failure error message type identification codes are defined.

3) Constructing, Sending, Receiving Authentication Request Messages and Device Integrity Verification Value Replies

The current version of Ryu supports multiple OpenFlow protocol versions. Taking OpenFlow_1.3 as an example to illustrate this implementation:

In ofp_handler.py and ofproto_v1_3_parser.py, the following are jointly implemented: - Authentication request message construction and parsing - Device integrity verification value reply message construction and parsing - Prover integrity verification: if verification succeeds, continue with subsequent configuration query flow; if it fails, send an authentication failure error message.

2.5.3 OVS System Trusted Connection Implementation 1) Adding "Authentication" State in Underlying Connection Control Entities

In vconn_state, vconn_run, vconn_run_wait, and other components, four states are added: VCS_SEND_RYU_AUTH_REQUEST for sending authentication request messages, VCS_RECV_RYU_AUTH_REQUEST for receiving authentication request messages, VCS_SEND_RYU_AUTH_COMPARE for sending check value reply information, and VCS_RECV_RYU_AUTH_COMPARE for receiving check value reply information. This ensures smooth execution of message construction, transmission, reception, and parsing.

2) Defining New Message Structures

To ensure OpenFlow message processing functions can successfully parse various messages, new message structures are also defined in ofp-util.h:

- In ofp-errors related files, add identification codes, failure reasons, reason descriptions, and identification codes for the authentication failure error type.
- In ofp-msgs related files, add new message type names, supported version ranges, and other basic content.
- Write ofp-auth related files to describe the version support situation for the integrity authentication phase.

3) Writing Message Reception and Extraction Interfaces

Trusted connections are implemented during the initial connection phase, which has not yet entered the OpenFlow protocol message processing phase, so message processing differs slightly. The system' s original message extraction interface cannot accurately extract required information from authentication request messages and integrity check value reply messages at this stage. To avoid conflicts with other interaction information, a message extraction interface ofpbuf_auth_pull is written in ofpbuf.h.

4) Message Construction, Transmission, and Reception

In vconn.c and ofp-util.c files, programs bearing main responsibilities are written: - Implement authentication message construction and transmission - Implement authentication message reception and parsing - Implement device integrity check value reply message construction and transmission - Implement device integrity check value reply message reception and integrity verification

5) Completing Authentication Message Processing Branches in Upper-Level Calling Programs

In related files such as rconn.c and ofproto.c, authentication message processing branches are perfected to prevent the system from treating authentication-related messages as error information during OpenFlow message processing, which would affect integrity authentication and subsequent connection establishment.

---

3 SDN Trusted Connection Testing

3.1 SDN Trusted Connection Test Environment

3.1.1 Trusted Platform Module Simulation Environment Setup The trusted connection test environment in this paper uses the Trusted Platform Module emulator TPM_emulator to replace TPM encryption chips for testing device platform software and hardware integrity. Developed by German researchers, TPM_emulator is currently the most widely used TPM simulator [14]. The software uses Cmake to organize source code and can run on Linux platforms, effectively simulating TPM functions including random number generation, RSA key generation, and digital signatures.

TPM_emulator operation requires cross-platform compilation tools, computation libraries, graphics libraries, TSS software stack, interface management modules, etc. The graphical management interface, as an optionally installed module, can help users better manage the trusted platform.

3.1.2 SDN Trusted Connection Test Topology Setup The SDN trusted connection network topology mainly consists of two controllers and three switches. Among them, Ryu1, OVS1, and OVS3 are trusted devices, while OVS2 and Ryu2 are untrusted devices. Logically, OVS1 and OVS2 directly connect to Ryu1, while OVS3 needs to choose a trusted device to connect to from Ryu1 and Ryu2. The experiment configures it to prioritize connecting to Ryu2. Only when both parties pass trusted authentication simultaneously can the connection be successfully established. The test topology is shown in [Figure 9: see original paper] and [Figure 10: see original paper].

3.1.3 Wireshark Collaboration with OpenFlow Protocol Function Extension Currently, many Wireshark versions can capture and parse OpenFlow messages, but custom-added messages can only be captured without being parsed, which undoubtedly hinders later analysis of trusted connections. Therefore, according to the analysis requirements of each trusted connection module, the OpenFlow protocol in Wireshark-2.0.9 is functionally extended to support parsing of trusted connection information, enabling intuitive analysis of the authentication process and control of communication entity authentication status during the testing phase.

The modification work mainly includes four aspects: a) Add parseable message types: define and match names and message numbers for integrity authentication request messages and integrity check value reply messages to enable normal recognition during packet capture; b) Define and initialize variables in the error message structure to enable smooth content extraction; c) Write custom message parsing functions: based on defined variables, write message parsing functions according to message formats to achieve complete message parsing; d) Add parsing processes for newly added messages: add branches for new message processing in the main function to enable message recognition and processing.

3.1.4 SDN Test Environment Software and Hardware Description

TABLE:5 SDN Test Environment Network Configuration

Device	IP Configuration	Description
Ryu1	192.168.2.40	Trusted controller
OVS1	192.168.2.50	Trusted switch, connects to Ryu1
OVS5	192.168.2.60	Untrusted switch, connects to Ryu1
Ryu2	192.168.2.70	Untrusted controller
OVS6	192.168.2.80	Trusted switch, connects to Ryu2

TABLE:6 SDN Test Environment Hardware Equipment

Device	Operating System	Description
Ryu1	Ubuntu14.04.1	SDN controller
Ryu2	Ubuntu14.04.1	SDN controller
OVS1	Ubuntu14.04.1	SDN switch
OVS5	Ubuntu14.04.1	SDN switch
OVS6	Ubuntu14.04.1	SDN switch

TABLE:7 SDN Test Environment Software Installation

Software Name and Version	Purpose
Modified controller in this paper	SDN controller
OVS-2.4.0	SDN switch
Wireshark-2.0.9 (extended)	Protocol analysis
tpm_emulator 0.7.4	Building trusted platform simulation environment, implementing trusted authentication
libgtk 2.0	
TrouSerS 0.3.8	
tpm_tools 1.3.8	
tpmmanager 0.8.1	

3.2 Test Steps and Results Analysis

3.2.1 SDN Trusted Connection Device Processing Action Test Steps

This test divides devices in the topology diagram into three groups: trusted controller Ryu1—trusted switch OVS1, trusted controller Ryu1—untrusted switch

OVS5, and untrusted controller Ryu2—trusted switch OVS6. The test aims to evaluate switch integrity authentication device processing actions, controller integrity authentication device processing actions, and the interaction information quantity of each test group.

1. Trusted platform simulator operation test;
2. Test switch trusted authentication status, start trusted controller Ryu1:
   - Test trusted switch OVS1: Start OVS1, configure controller Ryu1 address to discover Ryu1; observe Ryu1 authentication results and use Wireshark to capture interaction information during authentication and subsequent processes.
   - Test untrusted switch OVS5: Start OVS5, configure controller Ryu1 address to discover Ryu1; observe Ryu1 authentication results and use Wireshark to capture and record packets during authentication and subsequent processes.
3. Test controller trusted authentication status, start untrusted controller Ryu2: Start trusted switch OVS6, configure controller Ryu2 address to discover Ryu1; use Wireshark to capture and record packets during authentication and subsequent processes, and observe OVS6' s action response after authentication.
4. Conduct comparative analysis of packet interaction across the three test groups.

3.2.2 SDN Trusted Connection Time Test Steps

1. Trusted platform simulator operation test;
2. Start the modified trusted controller Ryu1, start OVS1, configure controller Ryu1 address, and record trusted connection establishment time;
3. Test traditional connection time and compare it with trusted connection establishment time.

3.2.3 Trusted Platform Simulator Operation Test Results The TPM simulation environment runs successfully, allowing viewing of platform version numbers, public keys, and other content, with the TPM_emulator manager starting successfully ([Figure 11: see original paper]).

3.2.4 SDN Trusted Connection Device Processing Action Test Results and Analysis SDN trusted connection device processing action test results for the three groups are shown in .

TABLE:8 SDN Trusted Connection Test Results

Test Group	Challenger	Prover	Authentication Result	Device Processing Action
1	Ryu1	OVS1	Success	Continue subsequent configuration queries; periodically send OFPT_ECHO messages to probe connection status
2	Ryu1	OVS5	Failure	Send authentication failure information; stop sending packets and drop connection ([Figure 12: see original paper])
3	Ryu2	OVS6	Failure	Only unilaterally establish connection; controller does not respond to its messages; switch unilaterally connects; controller does not respond to its information; stop receiving and drop connection

The three groups of test results and corresponding device processing actions are consistent with the trusted connection process design, demonstrating that SDN trusted connections can complete device integrity authentication during connection establishment. Additionally, the interaction information quantity of each test group is shown in [Figure 13: see original paper].

After Group 1 authentication completes, the connection is successfully established and periodically exchanges echo messages to maintain the connection. Group 2 shows no subsequent configuration information interaction due to authentication failure and the connection is dropped, so Group 2 no longer appears in OpenFlow messages afterward. In Group 3, OVS6 attempts to connect to Ryu2, but the connection is dropped because Ryu2 is untrusted, with no subsequent information interaction.

3.2.5 SDN Trusted Connection Time Test Results and Analysis Compared with the traditional connection flow, the trusted connection process adds steps. Although the connection occupation time also changes slightly slightly, tests compare the time between the added trusted connection process and the traditional connection process.

The test results ([Figure 14: see original paper]) show that although trusted connection time increases, it remains within an acceptable range compared to traditional connection time and does not affect subsequent processes.

4 Conclusion

Software-defined network trusted connections analyze the security requirements between the control layer and data forwarding layer, identifying the imperfect integrity authentication mechanism between them as one of the urgent security issues to be addressed. Tests show that the trusted connection proposed in this paper implements device integrity authentication before device connection, effectively preventing malicious connections from compromised devices. However, as software-defined networks develop rapidly and systems continue to update, besides the device integrity authentication issue between controllers and switches, other authentication problems should continue to be solved by selecting applicable and mature security technologies.

References

[1] Shu Z, Wan J, Li D, et al. Security in software-defined networking: threats and countermeasures [J]. Mobile Networks & Applications, 2016, 21(5): 1-14.

[2] Bawany N Z, Shamsi J A, Salah K. DDoS attack detection and mitigation using SDN: methods, practices, and solutions [J]. Arabian Journal for Science & Engineering, 2017, 42(2): 425-441.

[3] Zuo Qingyun, Zhang Haisu. Security analysis and research of SDN based on OpenFlow [J]. Information Network Security, 2015(2): 26-32.

[4] Zhang Tuanli, Lv Guanghong, Yang Peilin. Survey on SDN reliability based on OpenFlow [J]. Electronic Science and Technology, 2016, 29(2): 177-181.

[5] Zhou Ruikang. Research on trusted network system based on SDN [D]. Beijing: Beijing University of Technology, 2015.

[6] Pan Qiuyue. Trusted improvement of STP protocol for trusted switch based on Open vSwitch [D]. Beijing: Beijing University of Technology, 2014.

[7] Zhang Chaokun, Cui Yong, Tang Heyi, et al. Research progress on software-defined networking (SDN) [J]. Journal of Software, 2015, 26(1): 62-81.

[8] Ma Wenting. Research on key technologies of SDN controller based on OpenFlow [D]. Beijing: Beijing University of Posts and Telecommunications, 2015.

[9] Chi Yaping, Wang Quanmin. Research and simulation design of trusted platform module based on USBkey [J]. Journal of Beijing Electronic Science

and Technology Institute, 2007, 15(4): 13-15.

[10] Zhang Huanguo, Chen Lu, Zhang Liqiang. Research on trusted network connection [J]. Chinese Journal of Computers, 2010, 33(4): 706-717.

[11] Feng Dengguo, Qin Yu, Wang Dan, et al. Research on trusted computing technology [J]. Journal of Computer Research and Development, 2011, 48(8): 1332-1349.

[12] Wen Bowei. Research on trusted computing platform technology application [D]. Xi' an: Shaanxi Normal University, 2013.

[13] Wang Xiaoming. Research on trusted network remote attestation [D]. Jinan: Shandong University, 2015.

[14] Luo Hongda, Dong Zengshou, Yang Wei. Design of trusted computing experimental platform based on TPM emulator [J]. Journal of Taiyuan University of Science and Technology, 2013, 34(5): 337-341.

Note: Figure translations are in progress. See original paper for figures.