computers & security 67 (2017) 89–106

Available online at www.sciencedirect.com

ScienceDirect

journal homepage: www.elsevier.com/locate/cose

Computers & Security

Invi-server: Reducing the attack surfaces by making protected server invisible on networks

Jaehyun Park, Jiseong Noh [1], *Myungchul Kim, Brent Byunghoon Kang* [*]

School of Computing, Korea Advanced Institute of Science and Technology, 291 Daehak-ro, Yuseong-gu, Daejeon 34141, Republic of Korea

A R T I C L E I N F O

Article history:
Received 4 October 2016
Received in revised form 9 January 2017
Accepted 21 February 2017
Available online 1 March 2017

Keywords:
Invisible authentication
Server security
Covert channel
Secret server
One-time password

A B S T R A C T

The advantage of having remote access motivates network administrators to connect mission-critical servers (e.g., enterprise management systems) as well as public web servers via the Internet, even though connecting these mission-critical servers to the Internet is not recommended. These mission-critical or public servers are accessible from any host on the Internet, allowing cyber attackers to engage the targeted server as part of a process to discover potential exploits and unpatched vulnerabilities. Although it would be difficult to eradicate all the potential vulnerabilities in advance, accessibility to a server can be controlled to limit or minimize the chance of exposing a vulnerable surface. We aimed to address the accessibility issue by designing and prototyping an Invi-server system, in which the IP and MAC addresses of the protected secret server remain invisible from external scanning and eavesdropping trials and even from compromised internal hosts on the network. This Invi-server system can be used as a way to reduce the attack surface of a protected server while allowing authorized users to send and receive packets via the protected server. We also implemented a prototype of the Invi-server system to demonstrate that our proposed system has the ability to reduce the attack surfaces significantly without increasing network performance overhead to any significant extent.

1. Introduction

The Internet provides connectivity anywhere in the world and enables worldwide network services. According to a report published in 2014 (Internet Live Stats), around three billion users were using the Internet in that year which indicates that around 40% of the world population are using the Internet. Furthermore, because of the fast deployment of the IPv6 technology and the pending era of the Internet of Things, the connectivity of the Internet is increasing dramatically.

Because of the advantages offered by worldwide connectivity, many services involving confidential information use the Internet as a medium for their users. For example, companies allow their employees to access the private network remotely from home for increasing productivity. Monitoring services such as the enterprise network management system (i.e., an SNMP aggregator), or equipment monitor of SCADA (Supervisory Control And Data Acquisition) systems are usually connected to the Internet to provide the instant response to incidents (Fernandez and Fernandez, 2005). Even though connecting these mission-critical servers to the Internet is not

[*] *Corresponding author.*
E-mail addresses: jaehyun.park@kaist.ac.kr (J. Park), jiseong.noh@kaist.ac.kr (J. Noh), mck@kaist.ac.kr (M. Kim), brentkang@kaist.ac.kr (B.B. Kang).
[1] Co-first author.

http://dx.doi.org/10.1016/j.cose.2017.02.012
0167-4048/© 2017 Published by Elsevier Ltd.

**This paper includes a few updates based on detailed feedbacks from the author of 'SilentKnock'.

recommended, the convenience of remote access motivates network administrators to continue doing so.

However, providing confidential services via the Internet can be dangerous because the services can be located and targeted by attacks such as a scanning attack or an eavesdropping attack. First, a scanning attack can be carried out to locate Internet-connected devices or services. An attacker tries to collect IP addresses to identify the open ports of the devices or services throughout the Internet, after which they connect to these ports to obtain detailed information about the service running on these ports (e.g., Apache Web servers, and BIND DNS servers). This enables the attacker to compromise the devices by exploiting potential vulnerabilities in the service (e.g., scanning attacks are prevalent on the Internet and are used by human attackers and even self-propagating worms (Shin and Gu, 2010)). The attackers (or worms) can scan the entire IPv4 address range of the Internet in a very short time, even less than 45 minutes, by using a well-crafted scanning tool (Durumeric et al., 2013).

Second, an eavesdropping attack is an attack aimed at listening to a particular secret communication between two entities. For example, an attacker would be able to exploit an Access Point (AP) in a company's enterprise network and can collect all of the network traffic transmitted via the victim AP. Although the communication can be encrypted, knowledge of an IP address and a port number could be useful for the attacker to exploit the target since this information can indicate which service is running. As a result, the above two attacks can create a potential vulnerability to allow attacks, such as a remote shell execution attack (Metasploit), to mission critical servers.

The need to prevent attacks from targeting potential vulnerabilities on a server has led to the proposal of service vulnerability defense techniques such as a firewall, Intrusion Detection System (IDS) (Depren et al., 2005; Mell; Roesch, 1999), and Intrusion Prevention System (IPS) (Koller et al., 2008; Stiawan et al., 2010; Zhang et al., 2004). The proposed techniques filter unauthorized users or traffic to prevent an attack on the server. However, these techniques have certain limitations. In the case of a firewall, controlling access toward the IP address of a client is easily evaded using attack techniques such as IP spoofing (Wang et al., 2011). For IDS and IPS, it is impossible to detect all the different kinds of zero-day attacks (Verizon Business Risk Team).

Because it would be an extensive task to filter all malicious traffic, it would be helpful if we could effectively prohibit illegal traffic toward a *secret server* during the initial stage of an attack. This could be accomplished by hiding the IP address of the *secret server* to reduce direct attack trials because being hidden from the Internet inevitably limits direct scanning. One of the methods to hide the IP address would be to use a proxy-based server hiding technique. This technique would use the proxy server to hide the *secret server* that provides secret services, by locating them on an internal network behind the proxy server. Representative examples of proxy-based techniques are Reverse Proxy (RP) (Kruegel and Vigna, 2003; Reese, 2008) and Virtual Private Network (VPN) (Andersson and Madsen, 2005).

Although the proxy-based technique can limit the number of direct attack paths to the *secret server*, the proxy server itself is exploitable by the attacker because it is exposed to the Internet. For example, in the case of VPN, even though the *secret server* would be able to hide within the private network, the existence of a VPN gateway would create a potentially vulnerable point. An eavesdropper between the VPN gateway and VPN client could obtain the IP address of the VPN gateway, the specific program name (i.e., OpenVPN or Cisco VPN) and even version information, which would provide valuable information for an attacker. After exploiting the proxy server using that leaked information, the attacker would be able to reach the internal servers because the proxy server knows the routing path to these servers. In this state, the *secret server* would again be vulnerable to attacks.

We propose an Invi-server system, a secret server system that prevents a *secret server* on an internal network from being attacked after the Internet-open server that co-locates with the *secret server*, such as a public web server or a proxy server, has been compromised. In the proxy-based technique, the existence of the *secret server* can be revealed by the compromised public-open server. On the other hand, the Invi-server system, which contains the *secret server*, is located in front of the *public server*, rather than behind it. The Invi-server system hijacks requests to the *public server*, as a Man-in-the-Middle (MitM) attack, provides secret services for authenticated requests, by using the MAC and IP addresses of the *public server*, and forwards non-authenticated requests to the *public server*. Because the *public server* does not know about the existence of the Invi-server system, attacking the *secret server* through the *public server* is highly difficult.

We assume an attacker can use an eavesdropping technique to listen to all the traffic between clients and servers. Therefore, all the traffic must be encrypted using an SSL/TLS (or HTTPS) technique. The secret authentication between the client and the Invi-server system is implicitly conducted by verifying the shared secret during an SSL/TLS handshake. However, the method may cause significant overhead to the Invi-server system because an SSL/TLS handshake is computationally intensive and the system would have to perform the SSL/TLS handshake for all the clients, even clients of the *public server*.

To solve this problem, the system first looks up the TCP initial sequence number of a TCP SYN packet and checks whether the number contains an authentication key rather than just a random number before engaging in an SSL/TLS handshake process. This helps the system reduce the overhead because it allows SSL/TLS sessions only with authenticated users (henceforth we refer to this process as *candidate client selection*). The techniques used by the Invi-server system effectively degrade malignant trials with the scanner and eavesdropper.

In short, this paper has the following three contributions.

- Protecting against scanning and eavesdropping attacks using a system designed to have no publicly opened IP address. Because Invi-server does not have its own IP address, direct attacks using the IP address, such as scanning or eavesdropping attacks, are prevented.
- Protecting against an internal attack after the *public server* has been compromised. Because the Invi-server has no direct route from the *public server*, this kind of attack is prevented. Scenario-based analyses are given to compare this hiding technique based on the Invi-server and a proxy-based server.

- Lightweight authentication using *candidate client selection* is proposed. Invi-server improves the authentication performance by using *candidate client selection* to reduce the performance overhead to make an SSL/TLS connection with all client requests. Because Invi-server only causes a slight degradation in the performance of the existing *public server*, deploying Invi-server in networks is feasible for network operators.

The remaining part of this paper is structured as follows. First, background knowledge is provided in Section 2. Next, the threat model and assumptions are given in Section 3. After that, the design of the Invi-server system, including its components and working mechanism, is presented in Section 4. The implementation is described in Section 5. Three attack scenarios in which Invi-server could face threats, i.e., scanning, eavesdropping, and internal attacks, are discussed in Section 6 by comparing the two existing server models (stand-alone and proxy-based). Section 7 provides a detailed evaluation of Invi-server and its performance together with an assessment of its security. Related work is given in Section 8, and the last section concludes the paper.

2. Background

In this section, we describe proxy-based server hiding techniques and briefly explain their limitations from the point of view of security. We subsequently describe two techniques, MitM and TCP covert channel, both of which are used by Invi-server to assist with the understanding of the remaining sections.

2.1. *Proxy-based server hiding techniques*

A proxy is a system that is widely used for caching and server security. Especially, the RP and VPN are used as security measures to restrict the direct access to the servers. Distinct from being directly connected to an external network, proxy-based server hiding techniques receive external requests and forward them to backend servers located behind the proxy. Generally, the proxy server has a public IP address and backend servers have private IP addresses. This means the backend servers are protected against direct attacks from external attackers even though the servers have known vulnerabilities. However, the IP address of the proxy server is open to the Internet which presents another vulnerable surface. Moreover, if the proxy server is compromised, the backend servers could also be compromised.

2.2. *Two attack techniques used in Invi-server*

Invi-server uses the following two techniques: MitM and TCP covert channel. MitM is used to provide a service for legitimate users. On the other hand, TCP covert channel is used to provide *candidate client selection* which enables lightweight authentication for users.

2.2.1. *Man-in-the-Middle attack (MitM)*

The term MitM refers to an attack in which an attacker secretly relays a session between two components. In the case of an ongoing TCP session between a victim client and a victim server, a malicious attacker firstly intercepts the client's request by placing packets on the route used by the session. The attacker then relays all the requests from the client to the server and vice versa. This enables the attacker to gain access to the entire contents of the communication between the client and the server. Likewise, Invi-server uses the MitM to intercept client's sessions. When the client sends a shared secret when performing the SSL/TLS handshake, Invi-server quickly hijacks the session and takes over the service to the client.

2.2.2. TCP covert channel and its use in user authentication The term TCP covert channel refers to the use of a TCP header field capable of leaking certain information (Ahsan, 2002; Murdoch and Lewis, 2005). Contrary to normal data transfer using TCP, the data are inserted into a hardly noticeable field such as the TCP initial sequence number, low-bits of time stamp, packet order, or source port number, all of which are originally partial random values. Among these fields, the TCP sequence number field, especially the initial sequence number, is the most effective covert channel because of its length and randomness. These fields in a covert channel are normally used by an attacker with the intention of leaking the victim's information.

Nevertheless, some research has shown that the covert channels can be used for the purpose of hiding authentication (Houmansadr et al., 2011; Vasserman et al., 2009). Silentknock (Vasserman et al., 2009) assigns an authentication sequence to the TCP initial sequence number for user authentication. Likewise, Cirripede (Houmansadr et al., 2011) uses the TCP initial sequence number for the purpose of evading censorship. However, because the TCP initial sequence number is four bytes long which could be insufficient against brute force attack, above two systems use additional packets or field (Cirripede uses mul-tiple TCP SYN packets and Silentknock uses timestamp field on TCP option which is enabled or not by different operating systems or users) to meet goal of attack re-sistance. These additional fields could reveal details of the use of their system to an eavesdropper, which we do not intend applying to our system. A four-byte-long Message Authentication Code (MAC) could be sufficient to prevent a brute force attack on a relatively slow data link; however, in the case of an internal network attacker with a fast optic link (e.g., a 10-GbE interface), the four-byte MAC can be broken in a few minutes unless additional trial rate limiting techniques are employed.

In the case of Invi-server, it uses the TCP covert channel to select a candidate for use as a *secret client* to attain invisibility and performance advantages. Different from the previous work, Invi-server does not finish user authentication with the validation process of TCP initial sequence number. Invi-server additionally uses SSL/TLS handshake as user authentication process in a covert manner. previous work opens the possibility of using upper layer authentication such as SSH, but the detailed design and usage of covert authentication in the upper layer (such as SSL/TLS) with collaborating with TCP covert channel was not discussed.

Although some research concerning methods that use an SSL/TLS handshake for user authentication as a covert authentication channel has been reported (Karlin et al., 2011; Wustrow et al., 2011), these researchers did not use the TCP initial sequence number for reducing potential authorized clients. A disadvantage of their method is that the method should build all TCP session for each clients, which can cause significant overhead. Therefore, Invi-server effectively uses the TCP covert channel for candidate client selection while authenticating user in SSL/TLS hand-shake with large number of bits used for authentication.

3. Threat model and assumptions

3.1. Threat model

Our research mainly targets scanning and eavesdropping attacks launched by attackers from the Internet. The third kind of attack, namely an internal attack that is launched as a consequence of successfully compromising a publicly accessible server, also forms the focus of our work. The following three kinds of attacks are targeted in this paper.

A scanning attack: An attacker tries to scan all public IP addresses and open ports on the Internet. If a server has a vulnerable service on an open port, the attacker succeeds in compromising the server.
An eavesdropping attack: An attacker tries to listen to the communication between a targeted server and a client to determine the IP address of the client and the types of transport layer and application layer protocols of the services provided by the targeted server. After obtaining this service information, the attacker has an increased possibility of determining the vulnerabilities of the service, thereby improving the chances of compromising the server.
An internal attack after compromising the *public server*: If the attacker succeeds in compromising the publicly opened server, he/she attempts to scan internal IP addresses from the compromised server. This attack is a critical threat for proxy-based server hiding techniques because the proxy server provides a route to hidden servers located in internal networks.

The performance of Invi-server will be analyzed regarding the above three threat models in the attack scenario and analysis presented in Section 6.

3.2. Assumptions

Our target networks on which we deploy the Invi-server are either campus or enterprise networks running various public services but need a confidential service. For instance, a company provides a web service containing company information and their product information. Although the service is publicly accessible by all Internet users, the company needs a secret service for high-level authorized users (hereafter we refer to these users as *secret client*) such as a web community for high-ranking company members. We do not target attackers who can physically access the *secret server*. We also do not assume that the attacker can break the cryptographic methods such as AES or HMAC. We also assume that the public service uses SSL/TLS to prevent eavesdropping of the contents of the session between the client and the server. Lastly, the Invi-server has an SSL/TLS certificate and a private key of the target *public server* to perform an MitM. We additionally assume that no other servers, except for Invi-server and the target *public server*, have this private key.

4. System design

4.1. Overview

The purpose of Invi-server is to provide a secure server free from falling victim to scanning and eavesdropping attacks. These attacks are prevented from occurring, by attaching Invi-server to one of the *public servers* and by enabling Invi-server to use the IP address of the *public server* (Fig. 1). In addition, to allow authorized users to access services on Invi-server, this server distributes special clients to its users, termed *secret clients*. A *secret client* has a 20-byte long shared secret key with Invi-server which is used for authentication. The first 4 bytes of the key are used for *candidate client selection* at the *network bridge* while the remaining bytes are used for *modified SSL/TLS handshake* at the *secret server*, which are described in detail in Section 4.2 and Section 4.3. Invi-server can distinguish between traffic from *secret clients* and that from *non-secret clients* with a customized SSL/TLS handshake process. Three components of Invi-server, namely, i) *network bridge*, ii) *candidate client selection*, and iii) *modified SSL/TLS handshake* enable secure and invisible

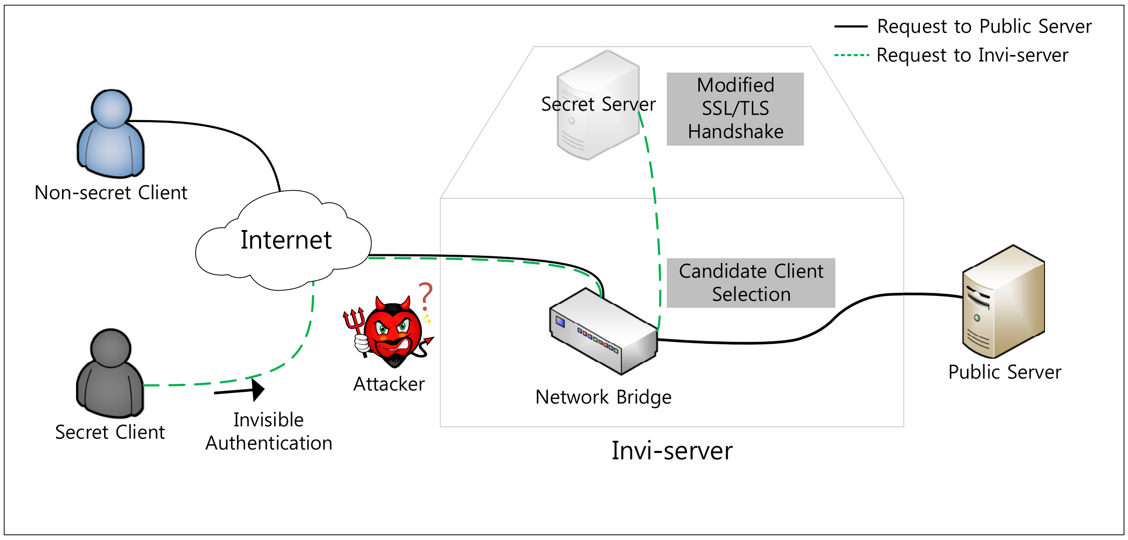


Fig. 1 – Components of Invi-server.

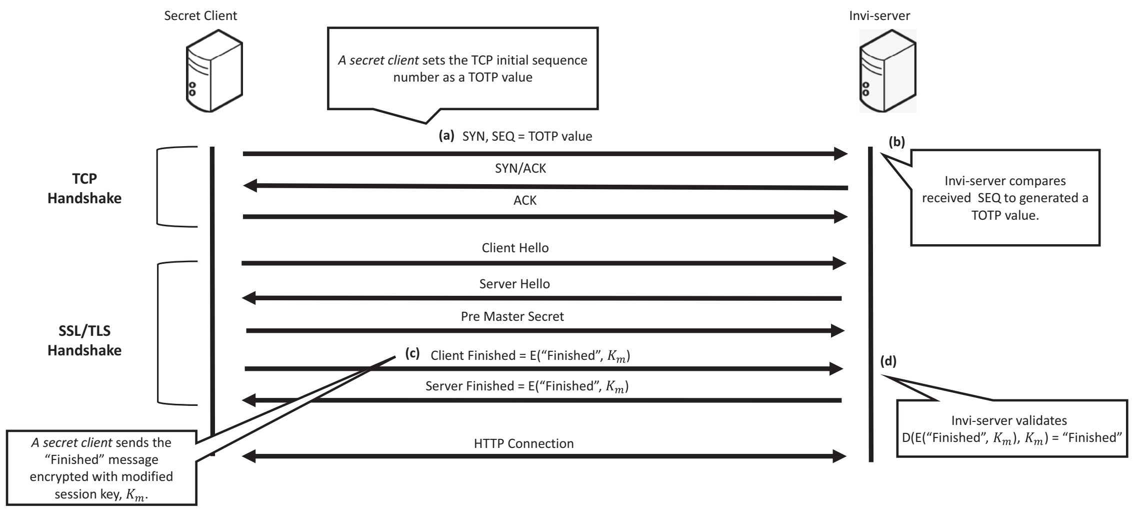

Secret Client
Invi-server
A *secret client* sets the TCP initial sequence number as a TOTP value
(a) SYN, SEQ = TOTP value
SYN/ACK
ACK
TCP Handshake
(b)
Invi-server compares received SEQ to generated a TOTP value.
Client Hello
Server Hello
Pre Master Secret
SSL/TLS Handshake
(c) Client Finished = E("Finished", K_m)
Server Finished = E("Finished", K_m)
(d)
Invi-server validates D(E("Finished", K_m), K_m) = "Finished"
HTTP Connection
A *secret client* sends the "Finished" message encrypted with modified session key, K_m.

Fig. 2 – Invi-server communication process.

connections with *secret clients*. We clarify the role of each component by describing the overall communication procedure between a *secret client* and Invi-server as follows (Fig. 2).

1. A *secret client* sends a TCP SYN packet, which is the initial packet of TCP connection, to the *public server*. The packet differs from a normal SYN packet in that the initial sequence number of the packet contains an one-time password (OTP) for *candidate client selection*.
2. Because Invi-server is located in front of the *public server*, the packet arrives at Invi-server before it reaches *public server*. The *network bridge* in the Invi-server system forwards all TCP SYN packets to a *candidate client selection* module (Section 4.2) to check whether the initial sequence number contains an OTP whereupon the client IP address is inserted into the *candidate client list* to allow subsequent packets from this client. After that, the *network bridge* forwards further packets from this client to the internal *secret server* in the Invi-server system.
3. The *secret client* who succeeds the *candidate client selection* attempts to conduct an SSL/TLS session with the *secret server* through an SSL/TLS handshake. In the process, the *secret client* and the *secret server* conduct the *modified SSL/TLS handshake* rather than a normal SSL/TLS handshake process. The *modified SSL/TLS handshake* contains the client authentication process by a shared key. A detailed description of the *modified SSL/TLS handshake* is in Section 4.3.
4. After a successful SSL/TLS handshake, the *secret client* and Invi-server use encryption to communicate each other using the session key generated in a previous step.

4.2. *Candidate client selection*

The first component of the process followed by Invi-server is *candidate client selection*, which reduces the overhead during client authentication. Invi-server authenticates a *secret client* by the *modified SSL/TLS handshake*. However, this generates the following overhead. Invi-server should hijack all sessions, i.e., those from both *secret* and *non-secret clients* because Invi-server does not know a *secret client* before checking a shared secret by performing SSL/TLS handshaking. Therefore, Invi-server needs to create two sessions to enable it to hijack a session: one for the client and the other for the *public server*. When it receives a request from a *non-secret client*, Invi-server checks all the SSL/TLS handshake requests from the client and if there is no shared secret, it forwards the request to the *public server*. Because it needs two different SSL/TLS sessions to hijack the session, encrypting and decrypting overhead is doubled which causes network delay.

In this state, *candidate client selection* solves the above problem by reducing the burden to create two SSL/TLS sessions (one each for the client and *public server*) by selecting a potential candidate as a *secret client* before the SSL handshake. Fig. 3 shows the impact of this step. As a result of the *candidate client selection*, most of the *non-secret client* requests are forwarded to the *public server* without creating a new session in the Invi-server system. As demonstrated in the figure, Invi-server does not hijack session for non-candidate clients.

Candidate client selection occurs when the *secret client* uses the TCP initial sequence number to mark a *secret client* (Fig. 2 (a)). Because this number is originally a 32-bit random number, it is the longest covert channel in the TCP header field. We use the TCP initial sequence number in authentication. Other fields in TCP header (e.g. ACK field, windows size, URG pointer, checksum, and timestamp in TCP option field) may be detected if they are used as the authentication purpose. For example, the TCP standard states the ACK field must be set to 0 in the TCP SYN packet. If the ACK field were set other than zero, an attacker (especially eavesdropper) would notice of the anomaly. Second, the value of the URG pointer is also typically set to zero and is not used unless some special cases. Although the least significant bit of the TCP timestamp option may be used as a covert channel (Giffin et al., 2002), the number of bytes it can contain is limited, and applications use the TCP timestamp option dependent with its operating system. If the *public server* does not use the option, an attacker can catch the difference through comparing the traffic from the *public server* and the *secret server*. Also, if TCP Window size or TCP checksum was used to hold the authentication information, the inherent functions of each congestion control and integrity check would be lost. Therefore, we did not change

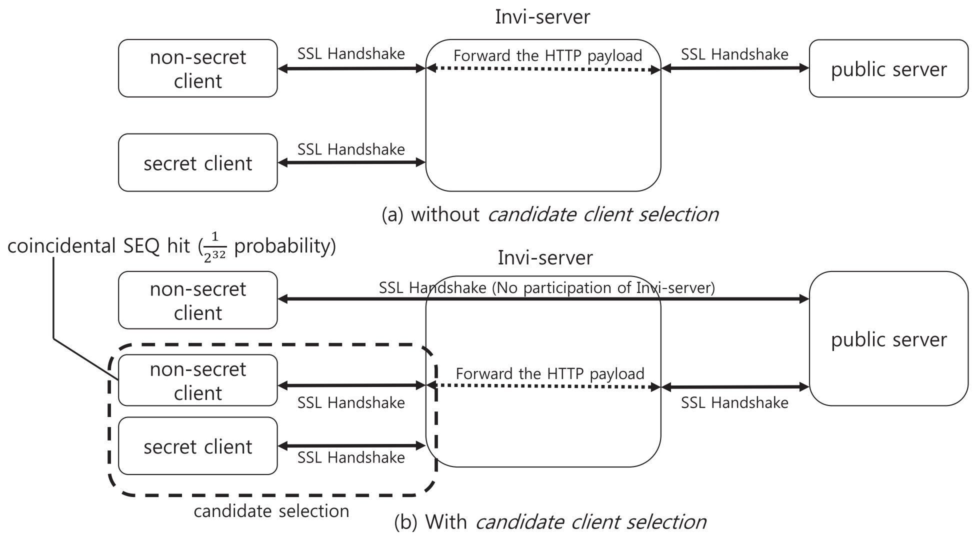


Fig. 3 – Process of *candidate client selection*. (a) Burden of the session MitM for the *non-secret client*. (b) Reduced burden with *candidate client selection*.

the original function of the TCP protocol, but used the TCP initial sequence number to prevent the attacker from being aware of the fact that authentication is taking place.

As the mark identifying the *candidate client* is placed in the same field as the initial sequence number, an attacker who uses eavesdropping to listen to the traffic would not notice the existence of the mark. The *secret client* adds the generated one-time password (OTP) value to the field containing the initial sequence number by using the following equation:

$$\textit{initial\_sequence\_number} = \textit{TOTP}\,(\textit{shared\_key}, \textit{kernel\_time}) \qquad (1)$$

The *secret client* and Invi-server use a time-based one-time password (TOTP) for authentication using *shared_key* and *kernel_time* where *shared_key* is the first 32-bit value of the pre-shared 20 bytes pre-shared secret. This 32-bit value is generated by a pseudo-random function and exchanged securely when the client program is distributed. As described in RFC 6238 (Raihi et al., 2011), the TOTP value is generated as a pseudo-random value. Therefore, an attacker who uses eavesdropping to obtain information about the packet would not notice that the OTP value has been input into the initial sequence number field. After the *candidate client selection* module receives the client's SYN packet with the OTP value, it also calculates the OTP value in the same way, and checks whether these two values are equal (Fig. 2 (b)). If the values are the same, the tuple (srcIPAddr, srcPortNum, dstPortNum) of the request is added to the *candidate client list* and the client request is submitted to the internal *secret server*. If not, the request is rapidly forwarded to the *public server* (before an SSL/TLS handshake). Although a typical TCP flow is identified as a 5-tuple, in the case of Invi-server, 3-tuple is used because destIPAddr is fixed to the IP address of the public server and the transport layer protocol is fixed to TCP. Because *non-secret clients* generate the TCP initial sequence number in a random way, one *non-secret client* per 2^{32} clients accidentally becomes a candidate with a very low probability.

Client packets other than SYN packets are passed to the internal *secret server* when its (srcIPAddr, srcPortNum, dstPortNum) tuple is in the *candidate client list*. Ten seconds after the last packet has arrived from the client, the tuple relating to the client is deleted automatically.

In summary, *candidate client selection* is a filtering process to quickly distinguish candidates of *secret clients*. In order to do this, the *candidate client selection* checks whether an SYN packet, which starts the TCP session, contains OTP or not, rather than checks OTP for all other packets. The remaining packets after the session initiation are checked by the flow information, the 3-tuple of TCP session which is registered in the *candidate client list*. In other words, an SYN packet containing OTP opens the route toward the *secret server* for remaining packets of the flow.

Alternatively, *candidate client list* can be structured using sequence numbers rather than the 3-tuple of flow information. If the initial sequence number of an SYN packet of a *secret client* matches the TOTP calculated by Invi-server, Invi-server registers the sequence number, not the 3-tuple of the flow, in *candidate client list*. After that, the remaining packets from the *secret client* are authenticated by matching (previous sequence number in *candidate client list* + current packet length) with the sequence number of the current packet. If this check passes, Invi-server forwards the traffic to the *secret server* and updates the value of the *candidate client list* with the current sequence number. Both of the methods have trade-off. The 3-tuple based method does not require update of the *candidate client list* for each incoming packets. On the other hand, the sequence number method has advantage in that only the sequence number is stored in the list rather than the 3-tuple. In the current implementation (as described in Section 5), Invi-server have been implemented with the 3-tuple fashion, but both methods can be used.

Note that *secret clients* would not be affected by network address translators (such as wireless APs assigning private IP addresses). Because Invi-server registers the session information of the translated IP address and port number, the *secret clients* can access the Invi-server as a usual way. However, if there are middle boxes which modify the TCP sequence number of the *secret clients*, the *secret clients* may not access Invi-server.

The value of *kernel_time* is the Linux kernel time of the *secret client* and Invi-server. This value can be set manually or synchronized with each other by using a time synchronization protocol such as network time protocol (NTP) (Mills et al., 2010). Since the Invi-server cannot be accessible without time synchronization, the *secret client* and Invi-server perform time synchronization through a trusted NTP server periodically. However, time synchronization may show errors due to delays on the Internet. We set a 30 seconds time step window, which is recommended by the TOTP standard. With this, a *secret client* can pass the *candidate client selection* even if they do not have perfect time synchronization. This time step may lead to a security leak, which we will discuss in Section 7.1.5

4.3. *Modified SSL/TLS handshake*

Once *candidate client selection* succeeds, the client and Invi-server start an SSL/TLS four handshake by using the following procedure. First, the client sends a *client hello* message to Invi-server, after which Invi-server sends a *server hello* and a server certificate, which is the same certificate as that sent by the *public server*. The reason why Invi-server uses the same certificate with a *public server* is that if Invi-server uses different certificate, an eavesdropper may notice that there exists a different service between the client and the *public server*. Because Invi-server works as though it has the same IP address as the *public server*, it is suitable to use a certificate identical to that of the *public server*. Invi-server and the *secret client* use a session key when they communicate using the SSL/TLS protocol. The key is necessary for authentication of the *secret client*. The session key, K_m, is generated by the following equation:

$$K_m = HMAC\_SHA1\,(random\_number, client\_id\_key) \quad (2)$$

HMAC_SHA1 needs 64 bytes of plain text (*random_number*) and 16 bytes for the key (*client_id_key*) to generate a new 16-byte session key. The *random_number* is generated by concatenating *client_random* and *server_random* each of which is a 32-byte pseudo random number, and which is generated during the previous steps in the handshake. The *client_id_key* is the least significant 16 bytes of the shared secret. This key is unique for each client group who has the same access permission for Invi-server. The final step of the SSL/TLS handshake procedure on the client side is sending an encrypted finish message to the server (Fig. 2 (c)). Invi-server decrypts this message by using each *client_id_key* (Fig. 2 (d)). If there is a *client_id_key* that decrypts the finish message correctly, the client request is finally forwarded to the *secret server*.

If the message is not decrypted to the correct plain text with any *client_id_key*, Invi-server establishes a new connection with the *public server* and forwards the entire payload to the *public server*. In situations such as these, Invi-server runs as an MitM attacker, forwards the request from the client to the *public server*, and forwards the response from the *public server* to the client. Note that this situation hardly occurs because only the connection trials passing the *candidate client selection* reach the *modified SSL/TLS handshake* module.

5. Implementation

We verified the performance of Invi-server by building a testbed with a *secret client* and Invi-server prototype. The *secret client* is implemented with a TOTP generation module and a *modified SSL/TLS handshake* module. The Invi-server prototype is implemented with a *network bridge*, a *candidate client selection* module, and a *modified SSL/TLS handshake* module.

5.1. *Secret client*

Implementation of a *secret client* consists of two steps. The first step involves setting the TCP initial sequence number as a TOTP result and the second step is changing the generated SSL/TLS session key for a *modified SSL/TLS handshake*.

5.1.1. *Implementing initial sequence number as TOTP value*

We implemented the TOTP generation module by modifying Linux kernel 3.2.0 of Ubuntu 12.04.1 LTS. In the Linux kernel, a function named *secure_tcp_sequence_number*() in *secure_seq.c* returns a 32-bit random number with which the initial sequence number is generated securely. We modified this function to return a 32-bit TOTP result instead of a random number and generated the TOTP result by using the *current_kernel_time*() function of the *ktime* library to obtain the current kernel time. After obtaining the current kernel time, it is divided by 30 to determine the time synchronization steps as described in the TOTP standard.

5.1.2. *Implementing modified SSL/TLS handshake*

Firefox, which was used for web client program, uses NSS (Network Security Services) as a security library including an SSL/TLS connection. We modified the NSS such that it was possible to use a modified session key as described in Section 4.3. NSS uses PKCS#11 libraries (RSA Laboratories) for their security tokens to generate and save their session keys securely. For implementation, our prototype bypasses the PKCS#11 security token and uses internal key generation function in NSS. We modified *ssl3_keyAndMacDeriveBypass* which is an SSL/TLS key generation function when bypassing PKCS#11.

5.2. *Invi-server*

Fig. 4 illustrates the architecture of Invi-server. Invi-server contains a *network bridge* with a *candidate client selection* module, and a *secret server* with a *modified SSL/TLS handshake* module. The *network bridge* is implemented as a software bridge (linux bridge, brctl) with customized Netfilter hook handler functions for *candidate client selection*. It also acts as a spoofer who makes the IP and MAC addresses of all packets leaving the Invi-server into those of the *public server*. We also modified the OpenSSL library to support our SSL/TLS key generation algorithm. Invi-server is implemented on a Linux system running kernel version 3.2.0–44 (Ubuntu 12.04.1) and OpenSSL 1.0.1e. Apache web server version 2.4.4 is used as the web server

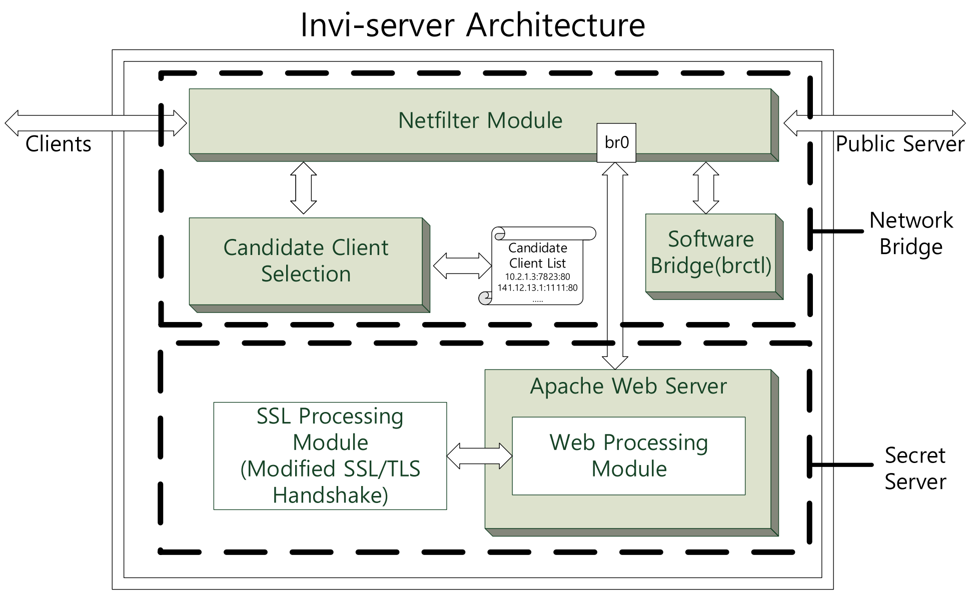


Fig. 4 – Invi-server architecture.

application and uses our modified OpenSSL. We also tested that OpenSSH work on our modified OpenSSL. We believe that other services based on SSL/TLS can also work on Invi-server, such as VPN and IMAP services.

5.2.1. *Network bridge*

Network bridge is located in front of the *public server*. The *network bridge* is implemented based on Linux Bridge (using brctl, which provides a software bridge) and custom functions with the Netfilter module.

We enabled the *secret server* to receive requests from authenticated clients with its existence remaining undetected by using the bridge interface (br0) to assign the *secret server* the same IP address as the *public server*.

Although the two machines in our system have the same IP addresses, having different MAC addresses can allow eavesdroppers to identify which packet is sent from which device. We set the MAC address of the br0 interface to the same address as the *public server*. Thus, two different machines use the same IP address, which causes IP address conflict. We created a concrete rule set that ensures that only one host can receive incoming packets at any given time even though they have the same IP and MAC addresses.

More specifically, the concrete rule set is as follows. br0 accepts authorized packets (SYN packet with OTP or in *candidate client list*) and drops the others. Conversely, the interface connected to the *public server* drops authorized packets and accepts the others. With these two rules, the *network bridge* prevents the *secret server* and the *public server* from receiving same packets. For broadcasting traffic, br0 drops all incoming and outgoing packets of the internal *secret server*. The reason for dropping all the broadcasting traffic is to prevent the possibility of the Invi-server being exposed. However, if all the broadcasting traffic is being dropped, Address Resolution Protocol (ARP) cannot work. It causes the *secret server* could not send packet to *secret client*. In order to solve this problem, whenever a *secret client* is successfully authenticated, the *network bridge* registers the IP address to its MAC address mapping information of the *secret client* in the br0's ARP table.

5.2.2. *Custom functions with Netfilter module for candidate client selection*

All the packets that are passed to or from the *public server* pass through the bridge. Packets that are received by an interface are identified and checked for an error and sent to a Netfilter hook, named NF_INET_PRE_ROUTING which is the first hook after the packet arrives at the host. This hook is used as incoming packet inspection with a user-defined handler function along with the NF_INET_PRE_ROUTING hook. We implemented a customized function named *preRoutingHookEntryFunc*(), which compares the current one time password value generated by Equation (1) with the TCP initial sequence number field value of packets and marks the source IP address, source port number, and destination port number of the matched packet on the *candidate_client_list*. If the incoming packet is not a TCP packet, it is regarded as a forwarded packet. Algorithm 1 presents the pseudo-code of this function.

Algorithm 1 Candidate client selection on prerouting hook

```
pkt = Incoming packet on prerouting hook
if pkt.prot = TCP and pkt.tcp_hdr = SYN and pkt.ip_hdr.dest = PUBLIC_SERVER_ADDR then
    cur_time = get_current_time();
    TOTP_result = generate_TOTP(shared_key,cur_time)
    if pkt.tcp_hdr.seq_num = TOTP_result then
        add_candidate_client_list(packet.ip_header.src,
        packet.tcp_header.src, packet.tcp_header.dst)
    end if
end if
```

After the packet passes through the NF_INET_PRE_ROUTING hook, it enters the Routing Decision Module that actually forwards the packet to its destination. In our implementation, each packet is copied and passed into either the NF_INET_LOCAL_IN or NF_INET_FORWARD hook. Once a packet arrives at NF_INET_LOCAL_IN, the packet proceeds to the *secret server* in the local host unless it is dropped inside the invoked function associated with the hook. On the other hand, if a packet is NF_INET_FORWARD, it is forwarded to another interface unless it is dropped. We implemented a kernel module that consists of function sets that handle each hook. On NF_INET_LOCAL_IN hook, a function named *localInHookEntryFunc*() is called, which looks up the *candidate_client_list*, and accepts the packet if its source IP address is on the list, or drops it otherwise. The accepted packet on this function is passed to the hook and delivered to a local process. On the NF_INET_FORWARD hook, *forwardHookEntryFunc*() is called and performs the opposite action. The accepted packet is passed through another hook named NF_INET_POSTROUTING and finally leaves the host. These functions ensure that only one copy of duplicated packets is received at the destination.

5.2.3. Modified SSL/TLS handshake

Packets that are passed to the NF_INET_LOCAL_IN hook are special as they do not reach the *public server*. Those packets are delivered to and handled by a local process. OpenSSL 1.0.1e was used with some modification in order to support the key generation formula described in Section 4.3. We changed OpenSSL to use the modified session key to receive the *client finish* message.

6. Attack scenario and analysis

In this section, we provide three attack scenarios for Invi-server, whereas the other server models, i.e., stand-alone and proxy server models, are evaluated together. The three attack scenarios are a scanning attack, an eavesdropping attack, and an internal attack after compromising the *public server*. An attacker would "succeed" in attacking the *secret server* in each of the three attack scenarios, by achieving the following goals.

- A scanning attack: Finding IP addresses and open ports of services that could be accessed by the attacker.
- An eavesdropping attack: Finding metadata which contains an IP address and the port information of the sender or receiver, or protocol information of the communication of a *secret server*.
- An internal attack after compromising a *public server*: a *secret server* is accessible (or scannable) after compromising a publicly opened server (a *public server*).

The server models and the three attacks associated with them are shown in Fig. 5. In our scenario, we assume that all server models contain *public* and *secret servers*, and have known web vulnerabilities, such as SQL-injection, XSS, or other types of vulnerabilities (for example CVE 2013–2251, which is a vulnerability of Apache Struts that enables the execution of arbitrary files). Thus, if the attacker successfully scans or uses eavesdropping to obtain the service information of the targeted server, we assume the attacker would be able to compromise the targeted server. We discuss the three attacks for each of the three server models in detail in the remaining part of this section. The results of the attacks are summarized in Table 1.

6.1. Scanning attack

Fig. 5 (a) shows the three attack scenarios and the network architecture of server model based on a stand-alone server. As shown in the figure, a *public server* which is opened to all users and a *secret server* which is open to limited users are on the same network. Note that both of the two servers have public

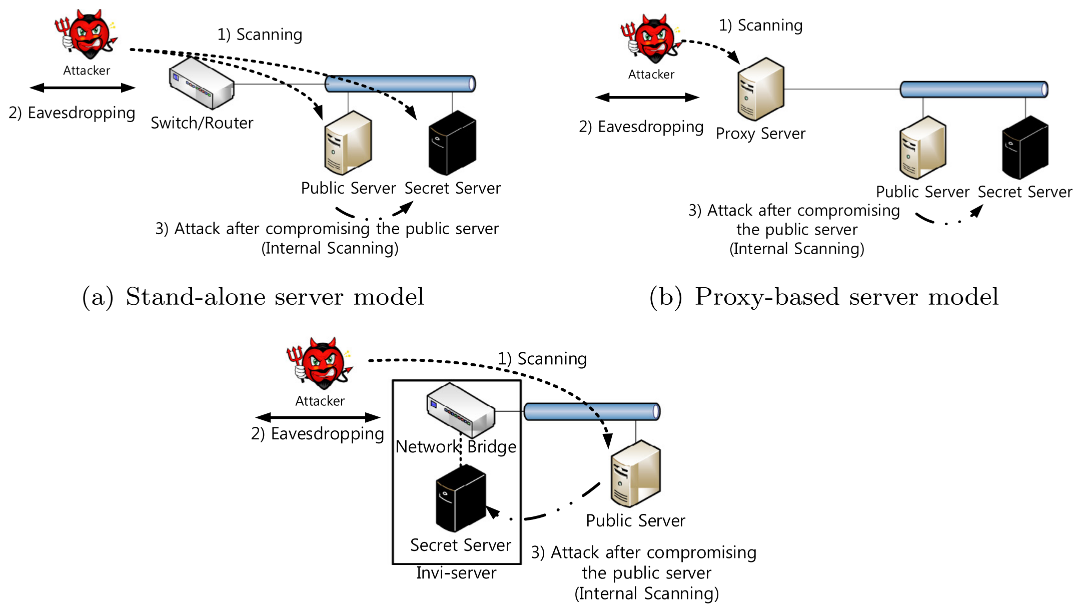

Attacker
1) Scanning
2) Eavesdropping
Switch/Router
Public Server
Secret Server
3) Attack after compromising the public server (Internal Scanning)
(a) Stand-alone server model
Attacker
1) Scanning
2) Eavesdropping
Proxy Server
Public Server
Secret Server
3) Attack after compromising the public server (Internal Scanning)
(b) Proxy-based server model
Attacker
1) Scanning
2) Eavesdropping
Network Bridge
Secret Server
Invi-server
Public Server
3) Attack after compromising the public server (Internal Scanning)
(c) Invi-server model

Fig. 5 – Network topologies of three server models and three attack scenarios for each server model.

Table 1 – Attack possibility of each three attack scenarios.

Attack type	Stand-alone server model	Proxy server model	Invi-server
Scanning attack	The public server and the secret server are scannable	The public server and the proxy server are scannable	Only the public server is scannable
Eavesdropping attack	Metadata of the public server and the secret server are revealed	Metadata of the proxy server is revealed	Metadata of the public server is revealed
Internal attack after compromising a public server	More vulnerable than external scanning attack	The secret server is scannable	The secret server is unscannable

IP addresses which can be accessed by any Internet users. The *secret server* adopts application layer ID/PW-based authentication as is the case for many other web servers. In this state, an attacker succeeds in performing IP and port scanning on both the *public server* and the *secret server*. The use of several vulnerability-scanning tools (Kals et al., 2006; Metasploit) enables the attacker to find vulnerabilities in the *secret server* by using the scanned IP address of this server.

In the proxy server model, the proxy server has a public IP address and forwards requests to the *public server* behind the proxy server. Fig. 5 (b) shows three attack scenarios and the network architecture of the proxy server model. The internal servers, i.e., the *public server* and the *secret server*, communicate with each other using their internal private IP addresses in the same subnet, whereas the proxy server has both a public IP address, which it uses for external communications, and a private IP address, for internal communications. In this situation, the proxy server forwards traffic of authenticated users to the *secret server* by determining the users' IP address (a well-known proxy server, Apache mod_proxy, adopts this method). Scanning the proxy server reveals the following two types of services: the service running on the *public server* of which requests are forwarded by the proxy server and services running on the proxy server itself such as the SSH server. Neither of these two services reveals direct attack routes for the *secret server* because the attacker is not authorized to access the *secret server*. Therefore, in the proxy server case, an attacker cannot attain the goal of scanning a *secret server*.

In the case of the Invi-server, it is unscannable because it does not have its own IP address and only requests from authorized clients who succeed in passing *candidate client selection* are forwarded to Invi-server. Therefore, direct scanning and finding vulnerabilities are prevented because Invi-server is invisible to unauthorized clients.

6.2. *Eavesdropping attack*

The use of eavesdropping to listen to a communication of the *secret server* in the stand-alone server model reveals information containing the IP address of both the client and server, and the service port (which is 80 for a web service). An attacker obtaining this information can launch a vulnerability-scanning attack.

The use of eavesdropping to intercept the communication between a proxy server and its client does not directly reveal the main subject (IP address of either the *public server* or *secret server*) of the communication. Rather than that, it reveals the IP address of the proxy server itself. It means that the eavesdropping does not reveal the communication metadata of the *secret server*. The internal IP addresses of the *public server* and the *secret server* are not exposed to the eavesdropper. Thus, eavesdropping also fails in the case of the proxy server model.

In the case of Invi-server, although the attacker obtains the communication information between the *secret client* and the *secret server*, the metadata reveals the IP address of the *public server*. Therefore, the attacker tends to mistakenly recognize that only the *public server* exists in the network. Moreover, requests of the attacker for vulnerability scanning go to the *public server* and not to the Invi-server because he/she does not have authentication.

6.3. *Internal attack after compromising a public server*

As mentioned above, a stand-alone server model is vulnerable to scanning and eavesdropping attacks. If an attacker compromises the *public server*, the possibility of compromising the *secret server* can increase significantly for the following reasons. First, if the *secret server* and the *public server* are in the same subnet, as in Fig. 5 (a), the attacker can infer the IP address of the *secret server* by using broadcast packets (e.g., Address Resolution Protocol (ARP) of IPv4 or Neighbor Discovery Protocol of IPv6) generated by the *secret server*. Thus, the attacker can determine the existence of *secret server* without scanning or eavesdropping. Second, an internal attack is more vulnerable than an external attack because firewalls are usually located on the border between public and private networks. Therefore, an attack from the *public server* cannot be filtered by the firewall which makes it easier to establish the vulnerabilities of the *secret server*.

As described in the previous section, direct scanning or eavesdropping fails to succeed in their goals in the case of the proxy server model. However, there exist several paths to compromise the *secret server* behind the proxy server as shown in Fig. 6. The first path involves compromising the *public server* by exploiting its possible vulnerabilities and scanning the *secret server* with the compromised public server. This is possible since the *secret server* is accessible if an attacker were to reach the internal network. If the existence of the internal network were to become known, the situation would no longer differ from the stand-alone case in terms of the scanning capability. An attacker could find a vulnerability of the *secret server*, which they could then exploit.

The second path entails compromising or bypassing the proxy server to access the *secret server* directly. If the proxy server has a running service with a vulnerability such as a vulnerable

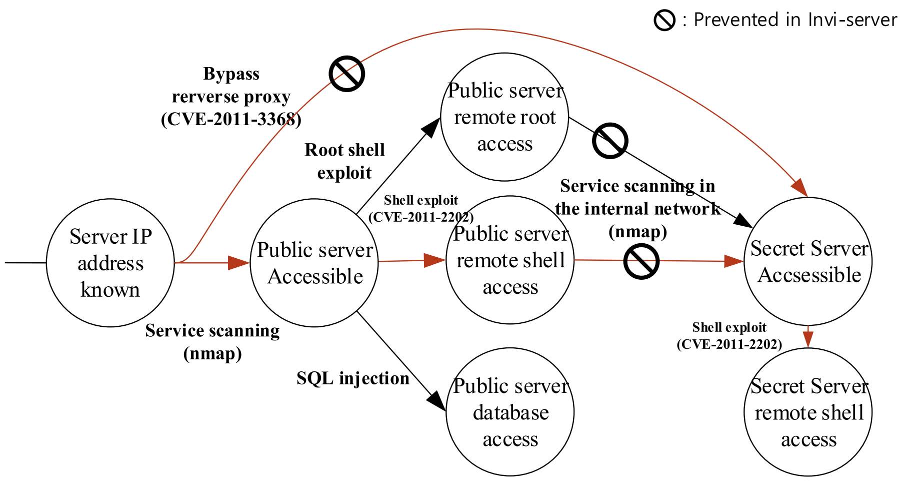

⃠ : Prevented in Invi-server
Bypass rerverse proxy (CVE-2011-3368)
Root shell exploit
Public server remote root access
Service scanning in the internal network (nmap)
Shell exploit (CVE-2011-2202)
Server IP address known
Public server Accessible
Public server remote shell access
Secret Server Accsessible
Service scanning (nmap)
SQL injection
Public server database access
Shell exploit (CVE-2011-2202)
Secret Server remote shell access

Fig. 6 – Attack paths to the *secret server* in the proxy server model. Text in each circle presents states of an attacker's capability after attacks (arrows) are conducted. The red arrows are attack paths which experimented with reported vulnerabilities of the proxy server model. Marked arrows are attack paths prevented in Invi-server.

SSH server, the attacker could compromise the proxy server. Moreover, there also exists a reverse proxy bypass vulnerability which is quite a dangerous vulnerability that enables an attacker to directly access any servers in the internal network without compromising any internal servers.

On the other hand, Invi-server eliminates the potential paths for attacks that exist in the proxy server model. Invi-server does not have its own IP address; thus, internal network scanning via a compromised *public server* does not reveal the existence of the *secret server*. This eliminates the first path of the reverse proxy case. The second path is also cut because Invi-server is installed as a bridge form. Because a *network bridge* performs MAC layer routing rather than application layer routing, the possibility of bypassing the bridge is low. This supports the fact that there were no bypass vulnerability reports about *brctl*, one of the widely used software bridges in Linux systems, which Invi-server adopts as a bridge application. Presumably, the *network bridge* software has no reported vulnerabilities because the input size of the program is quite fixed and small (<64 bytes).

6.4. *Experiment of the attack scenario*

We investigated whether the attack paths in Fig. 6 are feasible in a real environment by building a network topology as in Fig. 5 (b) and (c) with virtual machines. We use the mod_proxy module in Apache HTTP server 2.2.4 as a proxy server and run a web application using Apache Struts2 version 2.3.15. We test the two vulnerability paths from the "server IP address known" state to the "secret server remote shell access" state which contains intermediate states and red arrows. We first access the *secret server* using a reverse proxy bypass vulnerability (CVE-2011-3368) as the upper red arrow in the figure. After that, we successfully obtain a shell of the *secret server* with the remote program execution vulnerability of Apache Struts2 (CVE-2013-2251).

We experiment with the second path, which is the lower path in the figure. We first compromise the *public server* with the remote code execution vulnerability of Apache Struts2. After that, IP address and open port scanning for the internal IP addresses is conducted in the *public server* using *nmap*, a scanning tool. In our experiment, the private IP address of the *secret server* is revealed by the scanning resulting in a web service on the *secret server* becoming accessible. Finally, we reach the remote shell of the *secret server* using the Apache Struts2 vulnerability.

On the other hand, when we try to compromise Invi-server, neither of the above two paths enables us to do so. As mentioned above, bypassing the bridge software vulnerability has not reported yet. Moreover, internal network scanning with a compromised *public server* and *nmap* does not reveal any clue of the existence of Invi-server. Consequently, Invi-server is robust to all three of these attacks, whereas neither the stand-alone server model nor the proxy server model fails to prevent all of these attacks.

7. Evaluation

7.1. *Security evaluation*

Next, we demonstrate the advantage in terms of security aspects of Invi-server by considering the possibility of a brute force attack on Invi-server in this section. Next, difficulty of inferring the existence of secret communication is carefully considered. Furthermore knocking methods which uses TCP covert channel as their authentication method are compared with Invi-server. Some security issues containing possible threats revealing the Invi-server, problems under time synchronization, and forward secrecy are also considered on this section.

7.1.1. *Possibility of brute force attack on Invi-server*

There are two cases in which users who are not approved as the *secret client* send requests to Invi-server; an unintentional user, usually a *public client* who wants to send a request to a *public server* and an intentional user, or an attacker, who wants to access Invi-server through a brute force attack. In both of these cases, there is the possibility of the *candidate client selection* succeeding because the TCP initial sequence number is originally a random number. The possibility of this case succeeding is $1/2^{32}$ because the sequence number is 32 bits long.

After the *candidate client selection*, the unintentional user or the attacker may succeed with the modified SSL/TLS handshake because of the coincidental generation of the same session key. The probability of having the same session key is much higher than the case of *candidate client selection*, which is $1/2^{128}$. Thus, the entire probability of both of *candidate client selection* and *modified SSL/TLS handshake* succeeding is $1/2^{32} \times 1/2^{128} = 1/2^{160}$, which means that coincidental access to Invi-server is almost impossible.

Furthermore, Invi-server can easily find an attacker who continually tries to access it. The situation is improved by the fact that there are many existing detection schemes continuously try to connect to a specific target (Jung et al., 2004; Nychis et al., 2008). Moreover, application layer authentication such as an ID/Password-based login can prevent coincidental access of the critical data on Invi-server. If the application layer authentication of a client fails more times than a specified threshold, Invi-server transparently forwards subsequent requests from the client to the *public server*.

7.1.2. Difficulty of inferring the existence of secret communication

We argue that Invi-server has advantage in terms of difficulty of inferring the existence of their secret communication. This is attributable to the existence of the *public server* which hides the communication between Invi-server and a *secret client*. Consequently, an attacker cannot distinguish communication between Invi-server and a *secret client* from the communication between the *public sever* and *public client*. In the following, we will show how two types of communication, a *public client* to a *public server* and a *secret client* to a *secret server*, are indistinguishable.

We show that these two types of communication are indistinguishable by focusing on possible points that could be used to distinguish them. Since the application layer of both types of communication is encrypted, we only focus on their differences the MAC layer (MAC header), IP layer (IP header), transport layer (TCP header), and session layer (SSL/TLS handshake).

First, because Invi-server changed its MAC and IP addresses to the same as that of the *public server*, the information on the MAC and IP layers are indistinguishable. Second, Invi-server changes its TCP initial sequence number to an OTP value for *candidate client selection* which may inadvertently enable an attacker to distinguish between the two types of communication. However, the changed TCP initial sequence number is hard to distinguish because the OTP value is pseudo-randomly generated.

Lastly, the modified SSL/TLS handshake of Invi-server can provide a clue for differentiating between the two types of communication. However, the process conducted during the modified SSL/TLS handshake is exactly the same as that of a normal SSL/TLS handshake process except for the way the session key is changed. Even though the session key is changed, an attacker cannot guess whether this has occurred because the session key is originally hidden from the attacker in the Diffi–Hellman key exchange protocol. Therefore, there are no clues to distinguish the two types of communication from the attacker's point of view.

7.1.3. A comparison to knocking methods

We compare the Invi-server with knocking methods because both methods use the network layer covert channel for authenticating users. The comparison in this section shows the advantages of Invi-server, which involves not only hiding the authentication method used in secret communication, but also hiding the existence of the secret communication itself.

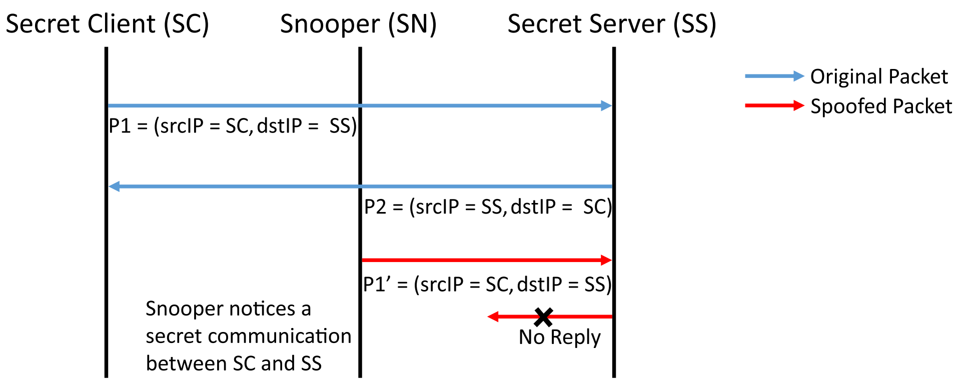
Port knocking is a network layer authentication method that opens a port in the server after some port sequences are received correctly in the form of an initial packet from the client. Krzywinski (2003) proposed this idea in 2003; however, weakness was found when an attacker tried a replay attack after watching the port sequence.

The problem of port knocking occurs because the authentication is only performed by a static sequence of ports and IP address. Several mechanisms have been proposed (Degraaf et al., 2005; Graham-Cumming, 2004; Worth) to solve this problem by improving the security by introducing cryptography into the port sequences (e.g., randomly changing the port sequences as in one time pass code). However, these studies were limited in terms of hiding the presence of authentication method and the existence of secret communication channel between the client and the *secret server*, because they did not use the standard TCP field or additional packets for authentication. These mechanisms would enable a local network snooper to easily find a system with a port knocking method and its existence of secret channel along with the IP addresses of the clients and the *secret servers*.

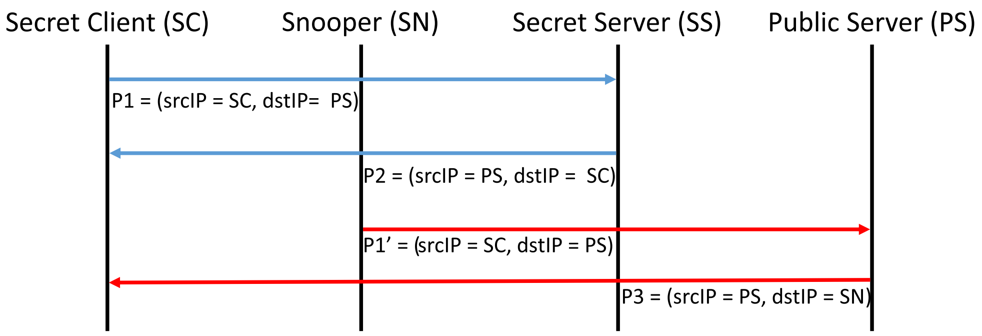
PROVIDE (Koch and Bestavros, 2016) proposes a DNS-based key (the knocking sequences) distribution method for clients of knocking methods. However, this method could be broken by the attacker who contacts the DNS server because its target attacks are limited to horizontal IP scanners.

The TCP initial sequence number field as authentication method was proposed by introducing Silentknock (Vasserman et al., 2009), so that in Silentknock the authentication method can be hidden from external observers, whereas in the port knocking (Krzywinski, 2003), the authentication method can be snooped by the attacker. An RFC information draft (Kirsch and Grothoff, 2015) also has proposed a method to embedding an authentication token to TCP initial sequence number with the similar manner of Silentknock. The most improved scheme among the knocking methods would be Silentknock, however, by design Silentknock does not hide the existence of communication channel between the client and the *secret server*.

In the following, we will provide more detailed comparison. (In our discussion, we will use Silentknock as a representative knocking scheme.) One of the main difference between knocking methods and Invi-server is the existence of the *public server*. We use the IP and MAC addresses of the public server in reply packets to the authorized users. Unauthorized users will also receive the packets with the same IP and MAC addresses. Since the contents can be encrypted, the snooper may not be able to distinguish between packets from Invi-Server and those from public-server, both of which have the same IP and MAC addresses (of the *public server*).


Secret Client (SC)
Snooper (SN)
Secret Server (SS)
Original Packet
Spoofed Packet
P1 = (srcIP = SC, dstIP = SS)
P2 = (srcIP = SS, dstIP = SC)
P1' = (srcIP = SC, dstIP = SS)
Snooper notices a secret communication between SC and SS
No Reply

(a) Inferring the secret communication of knocking method


Secret Client (SC)
Snooper (SN)
Secret Server (SS)
Public Server (PS)
P1 = (srcIP = SC, dstIP= PS)
P2 = (srcIP = PS, dstIP = SC)
P1' = (srcIP = SC, dstIP = PS)
P3 = (srcIP = PS, dstIP = SN)

(b) Inferring the secret communication of Invi-server

Fig. 7 – Inferring the existence of secret communications.

Moreover, Invi-server offers an advantage in a client enumeration attack. We define a client enumeration attack as enumeration by a powerful snooper of the IP addresses of clients who connect to the *secret server* and the compromising of one of the vulnerable clients when they connect the *secret server*. If the number of enumerated IP addresses of the clients increases, the possibility of successful attacks also increases.

In the following, we will show how knocking schemes fail to hide the existence of their *secret server*. Next, we compare the security of the knocking scheme and Invi-server from the viewpoint of a client enumeration attack.

Inferring the existence of a secret server using a knocking method: The advantage of a *secret server* using a knocking method is that a snooper cannot infer which network authentication method is used. This advantage is attributable to the fact that the authentication cannot be distinguished with a normal TCP handshake (Vasserman et al., 2009). However, an attacker can infer the existence of a *secret server* and some authentication method by the following process.

First, a snooper (SN) uses eavesdropping to access a TCP SYN packet from a *secret client* (SC) who successfully connects to a *secret server* (SS) (Fig. 7 (a)). After that, the attacker replays the captured packets with spoofed IP address to the SS. Since replaying a normal TCP SYN packet results a response from the server, if there is no response, the attacker can infer some authentication scheme exists in the SS and the SC is an authorized user. Although this would not destroy the authentication of the *secret server* SS, it would be easier to attack the SCs using the enumerated IP addresses. The attacker can enumerate the IP addresses by collecting the IP addresses of clients who successfully connect to the *secret server*.

On the other hand, in the case of Invi-server, a snooper in Fig. 7 (b) cannot infer the existence of any authentication scheme. If the attacker replays the TCP SYN packets of a client of a *secret client* SC or a *public client*, the request always succeeds in connecting to a *public server*. From the point of view of the attacker, the attacker thinks there is only a *public server* because the captured packet of the SS and the replied packet from PS are hardly distinguishable as described in Section 7.1.2.

Client enumeration attack: Since Invi-server and the knocking scheme use a shared key, it is hard to break the authentication itself. Rather than that, an attacker can enumerate the IP address of their clients and try to compromise the clients rather than directly attack the *secret server*. As described previously, clients of a *secret server* using a knocking scheme can be enumerated by an eavesdropper. In the case of Invi-server, packets from both the *secret server* and *public server* are enumerated without identifying the source of the packets. Since the payload is encrypted, the attacker cannot distinguish whether the reply is from the *secret server* or from the *public server*. In this case, the attacker would have to enumerate all possible IP addresses of both *public clients* and *secret clients*. Assuming the number of *secret client* of the knocking scheme is N_S, the burden of attacking the client of knocking scheme, B_{knock}, can be calculated by Equation (3) where k is a constant value.

$$B_{knock} = k/N_s \tag{3}$$

In the case of Invi-server, there are additional clients known as *public clients* who are the users of the *public server*. Assuming that the number of *public clients* is N_p, the burden of attacking Invi-server clients, B_{Invi}, is calculated by Equation (4).

$$B_{Invi} = B_{knock} \times (N_s + N_p)/N_s = k \times (N_s + N_p)/N_s^2 \tag{4}$$

If N_p is zero which means there are no *public clients*, B_{Invi} is equal to B_{knock}. If N_p increases, B_{Invi} increases concurrently which means the hardness of attacking a client increases. Because N_p is larger than N_s, a client of Invi-server is protected against a client enumeration attack than a client of the knocking scheme. This evidently shows that installation of Invi-server into a *public server* will provide enhanced security against these attacks. For example, if Invi-server has 50 clients while its attached *public server* has 10,000 clients, the possibility to find a *secret client* among the enumerated IP addresses of clients is around 0.4%.

7.1.4. *Possible threats revealing the Invi-server*

In Section 6 and Section 7.1, we have described that Invi-server is difficult to be detected. However, the Invi-server may not cover all kinds of attacks. Here we describe possible threats that Invi-server can be found and how to respond these threats.

Eavesdropping on the back and forth of Invi-server concurrently: Invi-server cannot be found with one of the external scanning, eavesdropping and internal attack. However, it could be detected if both external and internal eavesdropping and internal eavesdropping occur at the same time. For example, suppose an attacker sniffs a packet between an Invi-server and a *secret client* on one of switches or routers, and compromise a *public server* at the same time. If the attacker observes the packet P1 (*srcIPAddr* = *secret_client*, *dstIPAddr* = *public_server*) and the reply packet P2 (*srcIPAddr* = *public_server*, *dstIPAddr* = *secret_client*) in the switches or routers while the P1 and P2 are not generated in the *public server*, she may notice the existence of Invi-server and suspect that the *srcIPAddr* is a *secret client*'s IP address.

This attack also can be performed by observing the ARP Table without sniffing the packets directly. If there is ARP information for a *secret client*'s IP address in switch or router between Invi-server and *secret client*, but there is no information in the compromised *public server*, there is suspicious about the existence of Invi-server.

Imposing heavy load on the public server and analyzing the response time: If an attacker imposes heavy traffic such as denial of service (DoS) attack on a *public server*, the response time difference between from the *public server* and from the Invi-server grows and she can suspect the existence of the Invi-server. This attack is easier to perform because she does not need to compromise the *public server* unlike the above mentioned attacks. As a defense against this attack, the Invi-server *network bridge* can add some delay by performing timing analysis on the traffic going to and from the *public server*. Alternatively, if the traffic delay from the *public server* exceeds a certain threshold, Invi-server can be disabled. These issues will be resolved in future research.

Getting the IP addresses of secret clients with external channel: In addition to above two attacks, if the fact of using the Invi-server and the IP address of a *secret client* is exposed on external path (e.g., a personal e-mail containing the information is compromised), the attacker may collect these facts and discover the existence of a *secret client*.

Possible defense: If the IP address of a *secret client* is exposed, an attacker can eavesdrop the TCP initial sequence number of the *secret client* and spoof it to pass the *candidate client selection*. However, even if the attacker passes the *candidate client selection*, the attacker must pass the *modified SSL/TLS handshake* to access the *secret server*. In order to access the *secret server*, the attacker can try a brute force attack to find out the remaining 16 bytes of the *shared_key*. To prevent such attack, if attack attempts are periodically detected, in other word authentication failure in the *modified SSL/TLS handshake* for a particular *shared_key* exceeds a certain threshold, Invi-server revokes the *shared_key*. When the revocation is performed, an administrator of the Invi-server sends the revocation facts securely to the users who used the *shared_key* and recommends to change the IP addresses of the users.

7.1.5. *Problems under time synchronization*

As described in Section 4.2, on the *candidate client selection* process, the *secret client* and the Invi-server perform Time-based OTP generation procedure with Linux kernel_time. However, due to the time step, the OTP value may remain unchanged within the same time step Δt. An attacker who can collect initial TCP sequence number will notice that the TCP sequence number is used as an authentication method.

In the current state, we set a limit on the connection to the Invi-server within Δt so that the same TCP sequence would not be used again. In other words, a *secret client* is allowed to send one SYN packet within Δt. This degrades the usability of Invi-server. In order to compensate this, OTP could be generated with additional sequence bits. The sequence bits start with 0 when the new time step starts, and increase by 1 for each time of sending a new SYN packet. This OTP could be implemented by revising the Equation (1) to TOTP(shared_secret | sequence_bits, kernel_time) where the mark "|" means concatenation. Invi-server would pre-calculate possible OTP sequences of which the sequence_bits ranges from zero to certain threshold. A hash table can be used for fast matching of a client's OTP with pre-calculated OTPs of set of the sequence bits.

7.1.6. *Forward secrecy*

The session key K_m generated in the *modified SSL/TLS handshake* cannot provide forward secrecy since both values of the *client_random* and *server_random* are exposed in plaintext. If an attacker, who obtained a shared_key of a *secret client* and saves a *client_random* and a *server_random* in a session, she could decrypt the data of the session.

In order to provide forward secrecy, a scheme to perform ephemeral Diffie–Hellman (DHE) key exchange can be considered. As an example, how ECDHE (Elliptic Curve DHE) keys can be exchanged is described as follows. Suppose that a *secret client* and Invi-server have a pre-shared elliptic curve domain (such as Curve25519 (Bernstein, 2006)). When the *modified SSL/TLS handshake* starts, the *secret client* and Invi-server generates a 32-byte private key and a 32-byte public key in the pre-shared ECDHE domain. Then, the *secret client* inserts the generated public key into the *client_random* value and sends it to the Invi-server. Likewise, the Invi-server inserts its public key into *server_random* and sends it to the client. After the *server_hello*, the *secret client* and Invi-server generate *shared_secret* with each other's exchanged public key. If the concatenated value of the 32-byte *shared_secret* and a 32-byte zero is used instead of the *random_number* in Equation (2), the *session_key* K_m could provide forward secrecy.

7.2. *Performance evaluation*

In this section, the performance of Invi-server is evaluated. Since Invi-server is attached to the existing *public server*, the packet processing time caused by the *network bridge* and *candidate client selection* module incur performance degradation to the *public server* which may create a service obstacle. The additional packet processing time of the Invi-server attached to a *public server* compared to a single *public server* is as follows:

$$T_{public\_overhead} = T_{network\_bridge} + T_{candidate\_client\_selection} \tag{5}$$

In order to measure this overhead, we measure the time taken in two different modules; the *network bridge* module (Section 6) and the *candidate client selection* module (Section 7.2.2).

On the other hand, the additional packet processing time in the Invi-server compared to single *public server* is as follows:

$$T_{Invi\_overhead} = T_{network\_bridge} + T_{candidate\_client\_selection} + T_{modified\_SSL\_TLS\_handshake} \tag{6}$$

When comparing Equations (5) and (6), the time difference between two round trip times (RTTs), from a client to the *secret server* and from a client to the *public server*, occurs only in a *modified SSL/TLS handshake* module. If the time difference is significant, an attacker could find the existence of Invi-server by eavesdropping and analyzing the packets to Invi-server and the *public server*. Therefore, we check the processing overhead in the modified SSL/TLS handshake module. Note that this overhead only occurs during handshake time because after the handshake, the modified SSL/TLS module works exactly same as the normal SSL/TLS module.

We built a testbed comprising a *secret client*, the Invi-server, and a *public server*, with a topology similar to that in Fig. 1. Note that the clients in our testbed are directly attached to Invi-server, which is different from the figure. The hardware specification of each component in our testbed is as follows:

- Secret/public client : Intel Core i7-2600 3.40GHz, 8 GB RAM, Realtek Gigabit Ethernet card
- Invi-server : Intel Core i5-3570 3.40GHz, 4 GB RAM, two Intel Gigabit Ethernet card
- Public server : Intel Pentium E5700 3.00GHz, 4 GB RAM, Realtek Gigabit Ethernet card

7.2.1. *Network bridge overhead*

After installing Invi-server, additional overhead is incurred for all packets because of the packet bridging overhead in the *network bridge*. In order to obtain the time of the overhead, the RTTs of the client to the *public server* with and without a *network bridge* were measured using a ping test, hrping (cFos Software).

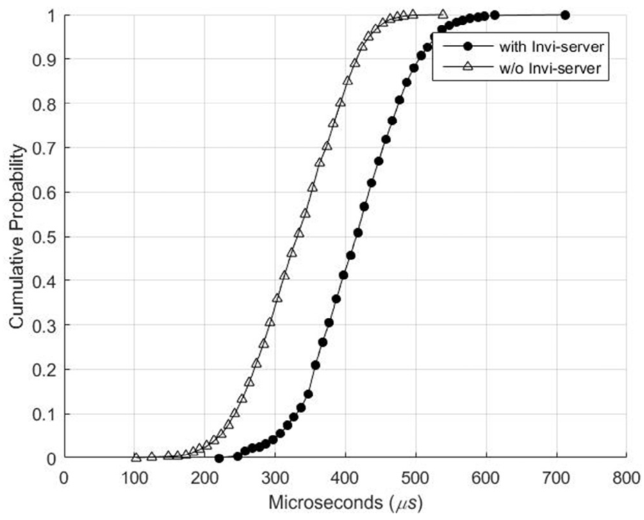


Fig. 8 – Round trip time to a *public server* without a *network bridge* (triangular marks) and to a *public server* with a *network bridge* (circular marks).

The triangular marks in Fig. 8 show the cumulative distribution of the RTT of the request from a client to a public server without passing through the *network bridge*. The RTT result without the *network bridge* is on average 330 μs. This RTT result is smaller than the RTT from the client to the *public server* with the *network bridge* which shows average 415 μs. The difference between these two results is 85 μs which is referred to as the *network bridge* overhead.

Since the average RTT on the Internet is around 100 ms according to a report (Internet Traffic Report), RTT generated by the bridge is negligible. Furthermore, if we use a custom hardware bridge based on Application-Specific Integrated Circuits (ASIC) rather than a software bridge, the overhead could be expected to be much lower than it currently is. We leave it as future work to implement the Invi-server *network bridge* as a hardware bridge.

7.2.2. *Candidate client selection overhead*

In the *candidate client selection* module, there are two different situations in which incoming packets are processed. If a packet is a TCP SYN packet, the *candidate client selection* module generates an OTP value and compares it with the initial sequence number of the packet. If the packet is not a TCP SYN packet, Invi-server searches the *candidate client list* to check whether the (srcIPAddr, srcPortNum, dstPortNum) tuple is in the list. If the number of tuples in the *candidate client list* increases, the searching time increases. We measured the processing time of the two above-mentioned cases by sending packets 1000 times for each case using a customized packet generator in the *secret client*.

Table 2 shows the resulting processing time of the two cases. The resulting processing time of the first case is 9 μs on average. In the meanwhile, the resulting processing time of the second case varies according to the number of tuples in the *candidate client list*. The result shows that the processing time linearly increases as the number of tuples increases. If the *candidate client list* contains 100,000 tuples, which means that 100,000

Table 2 – Processing time of candidate client selection module.

	T_{SYN} (μs)	T_{Others} (μs)		
		1000 tuples	10,000 tuples	100,000 tuples
Average	8.93	1.47	13.91	125.67
Deviation	1.54	0.66	1.30	12.50

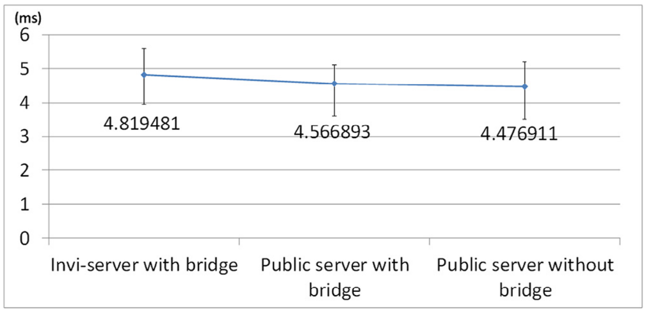


Fig. 9 – SSL/TLS handshake time comparison between Invi-server, public server with bridge, public server without bridge.

concurrent candidate clients are connected to Invi-server, the processing time is around 126 μs. This result means that the overhead of the candidate client selection module is also negligible compared to the RTT of each packet (around 0.12% of the average RTT of the Internet).

7.2.3. *Modified SSL/TLS handshake overhead*

Since the time difference of the connection time to Invi-server and the *public server* after the SSL/TLS handshake is almost the same, we only estimate the SSL/TLS handshake time to check the overhead resulting from the modified SSL/TLS handshake module. The SSL/TLS handshake time was estimated for the three cases: Invi-server, *public server* with the *network bridge*, and *public server* without the *network bridge*.

We used apache benchmark (ab) version 2.3 in both the *public client* and the *secret client* to check the handshake time. The elapsed time between client_hello to server_finish was tested by transmitting the handshake packets 1000 times in each of the three cases and the results are plotted in Fig. 9. Invi-server shows an average of 4.81 ms for an SSL/TLS handshake. This value does not lead to a large difference between the *public server* with the *network bridge* (average 4.56 ms) and the *public server* without the *network bridge* (average 4.47 ms). The performance overhead of the modified SSL/TLS handshake module is approximately 0.25 ms (0.25% of the average RTT of the Internet) that is enough to conceal the existence of Invi-server from attackers conducting a timing analysis to the difference in the response time.

8. Related work

A detailed comparison with other network level layer authentication such as proxy-based scheme (in Section 6), and knocking method (in Section 7.1.3) has been presented in previous sections. In this section we describe other research related to our Invi-Server mechanism; network randomization and decoy routing.

8.1. *Network randomization*

After the advent of the Software-Defined Network (SDN) scheme and the Openflow protocol (McKeown et al., 2008), which is a well-designed SDN interface, controlling network-wide policies, such as routing and security, has become easier and this leverages network randomization techniques.

Network randomization studies (Duan et al., 2013; Jafarian et al., 2012) claim that convictions about either constantly changing IP addresses of hosts or routing policies in a network would help decrease the success possibility of scanning, DDoS, and worm propagation attacks. As one of the network randomization studies, Random Host Mutation (RHM) (Jafarian et al., 2012) is a proposed method that constantly changes the IP addresses of hosts and effectively diminishes the effect by using a scanning attack and self-propagating worms. Another study, Random Route Mutation (RRM) (Duan et al., 2013), proposes constantly changing the routing policies of the network to reduce the possibility of eavesdropping and DDoS attacks for certain routes in the network.

The problem of network randomization studies is that vulnerability scanning would still be possible by exploiting the Domain Name Service (DNS). The network randomization technique involves updating the current IP address of a host to the DNS server for remote access whenever the IP address changes. An attacker can obtain the current IP address of the target host via a DNS query and can launch vulnerability scanning of the host. On the other hand, in our proposed system, an attacker who knows the IP address of the internal host to which Invi-server is attached will hardly be able to access our Invi-server, nor is vulnerability scanning of the Invi-server system likely to be conducted. Another problem with network randomization studies is that the entire network infrastructure requires an SDN environment to enable the proposed methods to be used. In our proposed system, however, the system is simply deployed by attaching it to one of the *public servers* in the network. In view of a network administrator who wants to provide a secret service to clients, deploying the Invi-server system is much easier than deploying an SDN environment to set up the network randomization technique.

8.2. *Decoy routing*

In 2011, three similar papers about decoy routing were published (Houmansadr et al., 2011; Karlin et al., 2011; Wustrow et al., 2011). Decoy routing is designed to evade censorship when the client is in control of censorship that blocks some IP addresses because of political reasons. If a client in the censorship area using a decoy routing system sends a request for converting their destination to a blocked destination, the decoy router installed in ISP converts the destination he wants to reach. Because the user's request should remain invisible from the censorship, all three papers use TCP and an SSL/TLS covert channel for authenticating clients.

For example Cirripede (Houmansadr et al., 2011) uses the TCP initial sequence number for client authentication. On the other hand, Telax (Wustrow et al., 2011) and decoy routing (Karlin et al., 2011) use the SSL nonce field, which is a random field in the SSL protocol. Although they use a covert authentication mechanism similar to Invi-server, the goal of decoy routing is quite different from that of Invi-server. Their goal is to allow the client unrestricted Internet access, whereas that of Invi-server is to ensure the server to be protected remains invisible to clients who do not have the pre-shared key (that is used to generate OTP number). Furthermore, since the primary goal of decoy routing systems is to provide service for many unspecified persons who would like to bypass the censorship (rather than to support a few authorized users), these decoy routing methods can obviously be detected by an attacker (who can also be a user of decoy routing service) because of the obtrusive packets; for example, Cirripede client sends 12 SYN packets for key distribution with the server.

All the three routers described above were rarely adopted in the real world. Further, a study suggested that Decoy routers could be detected (Schuchard et al., 2012). Recently, a new decoy router has been proposed (Bocovich and Goldberg, 2016). Slitheen introduces a better mimic technique to defend timing analysis attack. However, this requires significant resources of decoy routers and makes it hard to deploy in the real world.

In contrast, the primary goal of Invi-server is to ensure the protected server remain invisible to unauthorized accesses, while allowing pre-authorized users to engage and access the protected server.

9. Conclusions

Invi-server can be used to reduce the attack surface of a protected server by using the network access control and can be used to make scanning and eavesdropping attacks difficult. The proposed system reduces the attack surfaces by using both a *network bridge* and an authentication mechanism embedded in the TCP initial sequence number and the SSL/TLS handshake, which provides invisibility for Invi-server. The attack scenarios show that the system removes the paths to reach Invi-server and increases the difficulty of launching attacks from the internal network and by using an external scanner and eavesdropper. Our experiments with the prototype show that the Invi-server system produces less than 0.5 ms overhead when using the Invi-server as a web server. In the future, we plan to implement the Invi-server *network bridge* as a hardware bridge for performance purposes. We also plan to implement Invi-server client as an application version without kernel modification for improved usage. The problems under time synchronization and forward secrecy will also be considered in the future work.

Acknowledgments

We appreciate discussions and detailed feedbacks from Dr. Vasserman, the main author of 'SilentKnock' paper. This research was a part of the project titled “SMART-Navigation project”, funded by the Ministry of Oceans and Fisheries, Korea. This work was also supported by the ICT R&D program, MSIP/IITP (R0126-16-1005, Development of High Reliable Communications and Security SW for Various Unmanned Vehicles).

REFERENCES

Ahsan K. Covert channel analysis and data hiding in TCP/IP, Ph.D. thesis, University of Toronto; 2002.

Andersson L, Madsen T. Provider provisioned Virtual Private Network (VPN) terminology, Tech. Rep.; 2005 doi:10.17487/RFC4026.

Bernstein DJ. Curve25519: New Diffie-Hellman Speed Records, International Workshop on Public Key Cryptography, Springer; 2006, pp. 207–228, doi:10.1007/11745853_14.

Bocovich C, Goldberg I. Slitheen: perfectly imitated decoy routing through traffic replacement, in: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, ACM; 2016, pp. 1702–1714, doi:10.1145/2976749.2978312.

cFos Software, Ping utility hrping; 2013, http://www.cfos.de/en/ping/ping.htm. [Accessed 07 March 2017].

Degraaf R, Aycock J, Jacobson M. Improved port knocking with strong authentication, in: 21st Annual Computer Security Applications Conference (ACSAC’05); 2005, IEEE; doi:10.1109/csac.2005.32.

Depren O, Topallar M, Anarim E, Ciliz MK. An intelligent intrusion detection system (IDS) for anomaly and misuse detection in computer networks. Expert Syst App 2005;29:713–22. doi:10.1016/j.eswa.2005.05.002.

Duan Q, Al-Shaer E, Jafarian H, Efficient random route mutation considering flow and network constraints, in: IEEE Conference on Communications and Network Security (CNS), IEEE (2013), doi:10.1109/cns.2013.6682715.

Durumeric Z, Wustrow E, Halderman JA, ZMap: Fast Internet-wide scanning and its security applications, in: USENIX Security, Citeseer (2013).

Fernandez JD, Fernandez AE. SCADA systems: vulnerabilities and remediation. J Computing Sci Colleges 2005;20:160–8.

Giffin J, Greenstadt R, Litwack P, Tibbetts R. Covert messaging through TCP timestamps, in: International Workshop on Privacy Enhancing Technologies, Springer; 2002, pp. 194–208. doi:10.1007/3-540-36467-6_15.

Graham-Cumming J. Practical secure port knocking. Dr. Dobb’s Journ 2004;29:51–3.

Houmansadr A, Nguyen GT, Caesar M, Borisov N. Cirripede: circumvention infrastructure using router redirection with plausible deniability, in: Proceedings of the 18th ACM conference on Computer and communications security, ACM; 2011. doi:10.1145/2046707.2046730.

Internet Live Stats. Internet user statistic in the world; 2014, http://www.internetlivestats.com/internet-users/ [Accessed 07 March 2017].

Internet Traffic Report, Average response time for Internet traffic; 2016, http://www.internettrafficreport.com/ [Accessed 07 March 2017].

Jafarian JH, Al-Shaer E, Duan Q. Openflow random host mutation: transparent moving target defense using software defined networking, in: Proceedings of the first workshop on Hot topics in software defined networks, ACM; 2012. doi:10.1145/2342441.2342467.

Jung J, Paxson V, Berger AW, Balakrishnan H. Fast portscan detection using sequential hypothesis testing, in: IEEE Symposium on Security and Privacy, IEEE; 2004. doi:10.1109/SECPRI.2004.1301325.

Kals S, Kirda E, Kruegel C, Jovanovic N. Secubat: a web vulnerability scanner, in: Proceedings of the 15th international conference on World Wide Web, ACM; 2006. doi:10.1145/1135777.1135817.

Karlin J, Ellard D, Jackson AW, Jones CE, Lauer G, Mankins DP, et al., Decoy routing: toward unblockable internet communication, in: USENIX Workshop on Free and Open Communications on the Internet; 2011.
Kirsch J, Grothoff C. Tcp stealth; 2015.
Koch W, Bestavros A. Provide: hiding from automated network scans with proofs of identity, in: Hot Topics in Web Systems and Technologies (HotWeb), 2016 Fourth IEEE Workshop on, IEEE; 2016, pp. 66–71, doi:10.1109/HotWeb.2016.20.
Koller R, Rangaswami R, Marrero J, Hernandez I, Smith G, Barsilai M, et al., Anatomy of a real-time intrusion prevention system, in: International Conference on Autonomic Computing (ICAC), IEEE; 2008. doi:10.1109/ICAC.2008.24.
Kruegel C, Vigna G. Anomaly detection of web-based attacks, in: Proceedings of the 10th ACM conference on Computer and communications security, ACM; 2003. doi:10.1145/948109.948144.
Krzywinski M. Port knocking from the inside out. SysAdmin Mag 2003;12:12–7.
McKeown N, Anderson T, Balakrishnan H, Parulkar G, Peterson L, Rexford J, et al. OpenFlow: enabling innovation in campus networks. ACM SIGCOMM Comp Comm Rev 2008;38:69–74. doi:10.1145/1355734.1355746.
Mell P. Understanding intrusion detection systems. EDPACS 2011;29:1–10. http://dx.doi.org/10.1201/1079/43273.29.5.20011101/31414.1.
Metasploit. Penetration testing software; 2016, https://www.metasploit.com/ [Accessed 07 March 2017].
Mills D, Martin J, Burbank J, Kasch W, Network time protocol version 4: protocol and algorithms specification (2010). doi:10.17487/RFC5905.
Murdoch SJ, Lewis S. Embedding covert channels into TCP/IP, in: Information Hiding, Springer, (2005), pp. 247–261, doi:10.1007/11558859_19.
Nychis G, Sekar V, Andersen DG, Kim H, Zhang H, An empirical evaluation of entropy-based traffic anomaly detection, in: Proceedings of the 8th ACM SIGCOMM conference on Internet measurement, ACM (2008). doi:10.1145/1452520.1452539.
Raihi DM, Machani S, Pei M, Rydell J. TOTP: time based one time password algorithm, 2011. doi:10.17487/RFC6238.
Reese W. Nginx: The high-performance web server and reverse proxy. Linux J 2008;2008:2.
Roesch M. Snort: lightweight intrusion detection for networks, in: 13th Systems Administration Conference (LISA); 1999.
RSA Laboratories. Pkcs #11: cryptographic token interface standard; 2004, https://www.cryptsoft.com/pkcs11doc/STANDARD/pkcs-11v2-20.pdf. [Accessed 07 March 2017].
Schuchard M, Geddes J, Thompson C, Hopper N. Routing around decoys, in: Proceedings of the 2012 ACM conference on Computer and communications security, ACM; 2012, pp. 85–96. doi:10.1145/2382196.2382209.
Shin S, Gu G. Conficker and beyond: a large-scale empirical study, in: Proceedings of the 26th Annual Computer Security Applications Conference, ACM; 2010. doi:10.1145/1920261.1920285.
Stiawan D, Abdullah AH, Idris MY. The trends of intrusion prevention system network, in: International Conference on Education Technology and Computer (ICETC), IEEE; 2010. doi:10.1109/ICETC.2010.5529697.
Vasserman EY, Hopper N, Tyra J. SilentKnock: practical, provably undetectable authentication. Int J Info Sec 2009;8:121–35. doi:10.1007/s10207-008-0070-1.
Verizon Business Risk Team. Data breach investigations report; 2015, https://msisac.cisecurity.org/whitepaper/documents/1.pdf [Accessed 07 March 2017].
Wang Z, Qian Z, Xu Q, Mao Z, Zhang M. An untold story of middleboxes in cellular networks, ACM SIGCOMM Comp Comm Rev 2011;41:374–85. doi:10.1145/2018436.2018479.
Worth D. COK: cryptographic one-time knocking. USA: Black Hat; 2004.
Wustrow E, Wolchok S, Goldberg I, Halderman JA. Telex: anticensorship in the network infrastructure, in: USENIX Security Symposium; 2011.
Zhang X, Li C, Zheng W. Intrusion prevention system design, in: Computer and Information Technology, International Conference on, IEEE Computer Society; 2004. doi:10.1109/CIT.2004.1357226.

Jaehyun Park received his B.S. degree in computer science from Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Korea, in 2012. He received his M.S. degree in computer science from KAIST in 2014. Currently, he is a Ph.D. student in School of Computing at KAIST. His research interests include security on mobility management, network security, and security issues in Software-defined Networking (SDN).

Jiseong Noh received the B.S. degree in Computer Science and Engineering from Soongsil University, South Korea in 2012. He also received the M.S. degree in Information Security from KAIST (Korea Advanced Institute of Science and Technology), South Korea in 2014. He is pursuing a Ph.D degree in the School of Computing at KAIST. His research interests include software-defined networking (SDN), SDN security, secure network authentication, and trusted execution environment (TEE) applications.

Myungchul Kim received his B.A. in Electronics Engineering from Ajou University in 1982, M.S. in Computer Science from the Korea Advanced Institute of Science and Technology (KAIST) in 1984, and Ph.D. in Computer Science from the University of British Columbia, Vancouver, Canada, in 1993. Currently, he is with the faculty of KAIST as a Professor. He has served as a member of Program Committees for many conferences including IWTCS, IEEE ICDCS, and IFIP FORTE. He has published over 100 conference proceedings, book chapters, and journal articles in the areas of computer networks, wireless mobile networks, protocol engineering, and network security.

Brent Byunghoon Kang is currently an associate professor at the GSIS (Graduate School of Information Security) at KAIST (Korea Advanced Institute of Science and Technology). Before KAIST, he has been with George Mason University as an associate professor in the Volgenau School of Engineering. Dr. Kang received his Ph.D. in Computer Science from the University of California at Berkeley, and M.S. from the University of Maryland at College Park, and B.S. from Seoul National University. He has been working on systems security area including OS kernel integrity monitor, trusted execution environment, hardware-assisted security, botnet malware defense, and DNS analytics.