Vulnerability Detection Mechanism Based on Open API for Multi-User’s Convenience

Sung Soo Kim, Da Eun Lee, Choong Seon Hong
Department of Computer Science and Engineering
Kyung Hee University
446-701, Republic of Korea
E-mail: {mjs0514, daeunlee, cshong}@khu.ac.kr

Abstract—Vulnerability Detection Tools (VDTs) have been researched and developed to prevent problems with respect to security. Such tools identify vulnerabilities that exist on the server in advance. By using these tools, administrators must protect their servers from attacks. They have, however, different results since methods for detection of different tools are not the same. For this reason, it is recommended that results are gathered from many tools rather than from a single tool but the installation which all of the tools have requires a great overhead. In this paper, we propose a novel vulnerability detection mechanism using Open API and use OpenVAS for actual testing.

Keywords—Open API; OpenVAS; Vulnerability Detection Tool;

I. INTRODUCTION

If we consider some cases of various security incidents, there are some commonalities. Except for special cases, hackers focus on vulnerabilities in servers that exist on the application service layer. If it is the update server, it is easier to distribute malwares. Update server like antivirus-program update server. As it becomes easier for malware to be spread out the influence that can affect to the system is even more serious. Although it is not the updated server, it is still possible for attackers to attack the server through the vulnerabilities. Therefore, the administrators must consider about various security issues from service design step in order to protect the servers from attacks. So far, various platforms and tools have been developed through great efforts and researches in security-related companies and research institutions. Among them, VDTs can be one of solutions that can handle the security problems by realizing and responding to vulnerabilities. Thus, the administrator should install VDTs and periodically inspect states of the servers.

Vulnerability Detection (VD) can be classified into three types. First, System VD detects overall weak points about host Personal Computers (PC). Second, Web VD detects vulnerabilities about web sites. Finally, the Software VD detects vulnerabilities through Fuzzing techniques which are the techniques to find out security vulnerabilities through induction of systematic failure of software by entering random data repeatedly. Since VD methods are respectively different it is better to use the suitable method depending on situation and the purpose. Furthermore, they have different results since methods for detection of different tools are not the same. For this reason, it is recommended that results are gathered from many tools rather than from a single tool. However, installing all the various tools requires a great overhead. In addition, we can use the information collection, application search, network mapping and vulnerability discovery service through 12 pieces of different security scanners in HackerTarget.com web site. However, an additional fee of 156$ per year is required to use up to 25 times a day[1].

In this paper, we propose a novel vulnerability detection mechanism using the Open API and use Open Vulnerability Assessment System (OpenVAS) for actual testing. When the independent application is developed, it would be helpful to reduce the overhead and cost by loading a function of vulnerability detection without separation of implementation.

The rest of this paper is organized as follows. In Section II, we review some related works. Section III, presents necessary preliminaries. The proposed architecture and the actual process of detection mechanism according to scenarios are presented in Section IV. In Section V, we show some simulation results and evaluate our proposal comparing with other service in relation to several kinds parameters. Finally we conclude the paper and mention about future works in Section VI.

II. RELATED WORK

A. Open API

Open API is a standard interface made to allow the developer to access the proprietary software application programmatic control function. Briefly, the API is a set of requirements for controlling how a single application and mutual communication with the others, allows the developer to access an area of the internal functions of the program [2].

Since the Open API supports, it is easier to develop new application services independently. In addition, actual applied cases are increasing and the research for the Open API became active recently. In [3], the authors emphasize the need of independent Open API that enables the development of application which are operated in the future integrated network. In [4], since the mash-up services are increasingly growing by Open API, authors propose the ontology modeling in order to generate automatically considering user’s comfort and purpose what the users wanted. As a result, the Open API is the largest
factors where services appear as the mash-up services that are combined services of various field.

B. OpenVAS

The Open Vulnerability Assessment System (OpenVAS) is a framework which provides several services using comprehensive and powerful vulnerability scanning and vulnerability management solution. Fig. 1 shows OpenVAS Architecture. OpenVAS is composed of Greenbone Security Assistant (GSA), OpenVAS CLI, OpenVAS Manager and OpenVAS Scanner. GSA and the CLI are applications that drive the Manager via the OpenVAS Management Protocol (OMP). GSA provides a user interface to the web browser, CLI provides an "OMP" tool which is the command line interface and build for driving the OpenVAS Manager. The OpenVAS Manager is the main service, that consolidates plain vulnerability scanning into a full vulnerability management solution, controls the Scanner via OTP. Finally, Scanner that is the core of OpenVAS and it is conducted the actual Network Vulnerability Tests (NVTs) that are provided with daily update through the OpenVAS NVT Feed or commercial feed service. NVT Feed work flow is shown in Fig. 2. [5]

III. PRELIMINARIES

In this first part of section, explain how to design the URI before the implementation for the Open API. In second part, in detail, discuss correlation of "config value" and "Config ID". Finally we explain the identifier of OpenVAS, and demonstrate the process of scanning.

A. Open API Design

Uniform Resource Identifier (URI) of the Open API was designed using Representational State Transfer (RESTful) scheme prior to development of the Open API. Only the necessary functions on the scenario were designed, we assume that procedures of authentication to use the Open API are already conducted. Since the URI distinguish is capital from small letters use only lower cases. Also only nouns to prevent confusion are used. The Open API URI for the function of OpenVAS is shown in Table 1.

B. Correlation of config value and Config ID

In this part, before the discuss correlation of config value and Config ID, have to understand about them. The "config" variable is extracted from the URI. This variable carries the integer value, between 0 and 7. If the integer value is not in the range, the process is added into exception handler. The other hand, The "Config ID" is identifier which is required to create a Task on OpenVAS. Config ID affects the scanning rate according to the string, and it is corresponding to "config value" so those values define the scanning method. The detail description of correlation of config value and Config ID is shown in Table 2.

<table>
<thead>
<tr>
<th>HTTP Method</th>
<th>Definition</th>
<th>Open API URI</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>POST</td>
<td>Create</td>
<td>/apis/openvas/scanning/config={id}</td>
<td>Create/start scan</td>
</tr>
<tr>
<td>GET</td>
<td>Read</td>
<td>/apis/openvas/report/config={id}</td>
<td>Get the report</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>config value (parameter of URI)</th>
<th>OpenVAS Config ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>8715c877-47a0-438d-98a3-27c7a6ab2196</td>
<td>Discovery</td>
</tr>
<tr>
<td>1</td>
<td>085569cc-73ed-11df-83c3-002264764cea</td>
<td>empty</td>
</tr>
<tr>
<td>2</td>
<td>daba56c8-73ec-11df-4a5e-002264764cea</td>
<td>Full and fast</td>
</tr>
<tr>
<td>3</td>
<td>698f691e-7489-11df-9d8c-002264764cea</td>
<td>Full and fast ultimate</td>
</tr>
<tr>
<td>4</td>
<td>70825c4-7489-11df-8094-002264764cea</td>
<td>Full and very deep</td>
</tr>
<tr>
<td>5</td>
<td>74db13df-7489-11df-91b9-002264764cea</td>
<td>Full and very deep ultimate</td>
</tr>
<tr>
<td>6</td>
<td>2d3f05dc-5ba1-11e3-bf43-406186ea4fc5</td>
<td>Host Discovery</td>
</tr>
<tr>
<td>7</td>
<td>bbca7412-a950-11e3-9109-406186ea4fc5</td>
<td>System Discovery</td>
</tr>
</tbody>
</table>
C. Scanning Process

In this section, we introduce a series of procedures for scanning in OpenVAS. The detailed description of the “OpenVAS Identifiers” are shown in Table 3. First of all, Config ID and Target ID are required to scan a process. By using these, Config ID and Target ID, the system can create the Task ID, in order to start the scanning process. Target ID is generated from the client IP address. Generated Task has a unique ID. After scanning, OpenVAS marks this ID as finished and generates the report with “Report ID”. It is possible to confirm the results of the reports through the Report ID.

<table>
<thead>
<tr>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Config ID</td>
<td>Identifier for the detection method</td>
</tr>
<tr>
<td>Target ID</td>
<td>Identifier for the detected target</td>
</tr>
<tr>
<td>Task ID</td>
<td>Identifier for a task</td>
</tr>
<tr>
<td>Report ID</td>
<td>Identifier for a results report</td>
</tr>
</tbody>
</table>

IV. Architecture and Mechanism

This section mainly consists of two parts. At first, we show Open API server architecture followed by the explanation on components of it. Then we discuss about the proposed mechanism in detail for efficient operation via Open API.

A. Open API Server Architecture

Fig. 3 shows the overall architecture of Open API server. As it is shown in the figure, architecture is made up of several layers. Among them, the application layer is the external layer of the server. The others are internal layers which include library layer, adaptor layer, Open API layer. Security library and binary tools as open source are located in the library layer at the bottom of the architecture and the actual processes about security are running on this layer. We only consider about vulnerability detection tools in this paper. Adaptor layer plays a role of connecting the layer of Open API to the library layer. Open API layer has the server container to provide Open API and module that provides Open API is mounted on the server. Finally, application layer is the layer of servers which develop the application through the Open API.

B. Vulnerability Detection Mechanism

In this last part of the section, we discuss about how to detect the vulnerability through Open API. Once the request for scanning reaches to server using Open API, the request is delivered to security adaptor on lower layer. Next, the adaptor attempt to connect to OpenVAS. The connection is made in which the adaptor calls the function implemented using omp provided by OpenVAS CLI. At this moment, the config value received from the URI is forwarded to the function. It is used to create a unique task name for the requesting client. For example, the IP address of the client is 1.1.1.1. If you pass a value of 0 in config, task name becomes 1.1.1.1_0. URI doesn't pass the IP address of the client but it can be obtained from request object. The function, for the connection, asks to OpenVAS Manager using the unique task name to know whether task already exists or not. The next step will be described as categorized into three cases. All of the procedures are shown in Fig. 4.

1) First time to request: Since the client requests Open API for the first time, there is no matched target. This means, there is no tasks. So, in this case, OpenVAS firstly creates a target. Afterwards, it creates a new task by the rules already mentioned in the previous part of this section.

2) Request with a new config value: There is certainly a matched target. Because it is not the first time that requests are made. So, in this case, we can skip a step which creates a target. However, tasks have specific name according to config values. If request for a new config value is made, a task needs to be created.
3) Request with the same config value : There already exists the task that is matched to the name of the request. Because already made a target and request to same config value. As both the target is already created and the same request for the same config value, it is not necessary to follow all steps to create a target and a task.

As a result, A Task ID is available after all steps and the query is sent to the manager using Task ID to find out whether it is now running or not. If the task is not running, it is able to start scanning. Otherwise scanning process cannot be executable until the running task is finished

V. Simulation and Evaluation

In this section, we evaluated the proposed mechanism as categorized into three cases that were presented in Section 4. We used JAVA on Eclipse as integrated development environment for simulation and install Ubuntu 14.04.3 LTS as the operating system. The first part of this section shows simulation results of each case. The second part represents the assessment method, comparison and analysis through these methods.

A. Simulation result

Fig. 5 shows the simulation results of each case. Sequentially the first scenario is in relation to the first time to request. The result shows the case when we enter 0 as config value. Because any requests were formerly made, a target and a task are created, which is shown in the first part of the Fig. 5.

The second scenario is about a request with a new config value. The result represents when we enter 5 as config value. Since a target was already created by the previous request, it only produces a task without any steps of generating targets. Although the same value 0 is entered as config value, the result seems to be different from the first scenario. This is because a task already exists so nothing is created. Finally, the last box in the figure shows that if the task is running, the process ends up without scanning.

B. Comparison and analysis

We assume that Open API service using the proposed mechanism is provided. In terms of service, we compare the proposed mechanism to HackerTarget.com and analyze the results appeared from the comparison. Table 4 shows the parameters used in the comparison as well as the results. API of HackerTarget is not published and it is available to use the service after paying for them. However, it has an advantage that there are twelve tools that can be immediately used. On the other hand, the proposed mechanism only provides a single tool without any costs. In addition to cost and the number of tools, there is also one big difference between these. It is the range of object to be scanned.

If we consider only from the nearsighted point of view, the larger realm of target seems to be better but the attackers could take the advantage of it to collect information about the target. This means that we can’t ensure it is always outperforming. In order to cope with such problems, HackerTarget does not only monitor the traffics but it also uses the policy that finds out and cuts off the users who maliciously use services.

![Fig. 5. Simulation results of each case](image)

Meanwhile, the proposed mechanism limits the range of scanning targets to only requested clients, which results in more secured services. Furthermore, it has the advantage in terms of cost for monitoring since it does not require additional cost for it.

VI. Conclusion

In this paper, we make it possible to use the tools to detect vulnerability through Open API. The proposed mechanism gives the solution to the problem the existing tools face, which is every vulnerability can’t be completely detected by only a single tool. Moreover, there is no need to concern about additional efforts and costs to use various kinds of tools. In addition, the range of scanning target is limited to support stable services as well as to provide convenience to many users.

In future work, we will extend VDTs as well as the security tools of other types. Furthermore, with respect to complement, we will improve for the problems of degradation and system paralysis caused by many requests.
ACKNOWLEDGMENT

This work was supported by Institute for Information & communications Technology Promotion(IITP) grant funded by the Korea government(MSIP) (R0126-15-1009, Development of Smart Mediator for Mashup Service and Information Sharing among ICBMS Platform) *Dr. CS Hong is the corresponding author

REFERENCES