Validation Analysis of Scalable Vector Graphics (SVG) File Upload using Magic Number and Document Object Model (DOM)

Fahmi Anwar[1]
Department of Informatics
Universitas Ahmad Dahlan
Yogyakarta, Indonesia

Abdul Fadlil[2]
Department of Electrical Engineering
Universitas Ahmad Dahlan
Yogyakarta, Indonesia

Imam Riadi[3]
Department of Information Systems
Universitas Ahmad Dahlan
Yogyakarta, Indonesia

***Abstract*—The use of technology is increasing rapidly, such as applications or services connected to the Internet. Security is considered necessary because of the growing and increasing use of digital systems. With the number of threats to attacks on digital form or server systems is required to handle the risk of attacks on the server, the file upload feature. The system usually processes the file upload feature on a website or server with server-side (back-end) validation or filtering of digital object file types or a client-side (front-end) web browser in HTML or Javascript. Filtering techniques for Scalable Vector Graphics (SVG) usually files only see the file extension or Multipurpose Internet Mail Extension (MIME) type of an uploaded file. However, this filtering can still manipulate, for example, in ASCII prefix checking, which has two writes, namely "<?xml" and "<svg ". SVG files do not contain metadata such as image encoded in JPEG or PNG files. This problem can overcome by adding filtering techniques to check the validation of a file with validation of eXtensible Markup Language (XML) using magic numbers and the Document Object Model (DOM). This research developed using the waterfall method and black-box security testing refers to a software security testing method in which security controls, defense, and application design are tested. Handling of security validation for uploading SVG files using file extensions and MIME types has a success rate of 75 percent from the eight tested scenarios while handling using file extensions, magic numbers, and Document Object Model (DOM) produces a success rate of 100 percent from 8 test scenarios. Testing uses a black-box so that handling using the file extension, magic number, and Document Object Model (DOM) is better than using only file extensions and mime types.**

***Keywords*—Magic number; Scalable Vector Graphics (SVG); security; upload; validation**

I. Introduction

The website is an Internet service that can be used by various users in the world, which usually has an upload feature. The file upload feature or file upload is a feature that is generally functionally needed in applications for users [1]. However, without proper filtering, file selection, and validation processes during upload can present a significant security risk for website security [2] with three critical characteristics: integrity, input validation, and correct logic required for security applications DDoS attacks.

Distributed Denial of Service (DDoS) is a network security problem that continues to develop dynamically and increases significantly until now. DDoS is a type of attack performed by draining the network resources by flooding packets with significant intensity until they become overloaded and servers stop functioning. DDoS assault characterization depends on network traffic movement utilizing the Neural Networks and Naïve Bayes Methods. Because of the trials led, it discovered that the aftereffects of exactness in counterfeit Neural Networks were 95.23%, and Naïve Bayes Methods was 99.9%. The trial results show that the Naïve Bayes Methods are superior to Neural Networks. The examination and investigation consequences as proof in the primary cycle [3]. Another research is Artificial Neural Network (ANN) can be used as a viable device for network parcel arrangement with the proper blend of learning, move, concealed layer, and preparing capacities. ANN with two concealed layers gives generally predictable MSE, combination speed, higher right grouping rate at 99.04%, and a Quasi-Newton preparing capacity strategy (Matlab-trainlm) suited for the arrangement task, given the estimation of relapse both in the preparation and approval stage [4]. Another technique in detecting these attacks is monitoring but found several problems [5], including difficulty distinguishing the attack and regular data traffic using Density K-Means Method.

Cyber attacks by sending large data packets that deplete computer network service resources using multiple computers when attacking are called DDoS attacks. Total data packet and essential information in the form of log files sent by the attacker can be observing and captured through the port mirroring of the computer network service. The classification system must distinguish network traffic into two conditions, the first normal condition, and the second attack condition. The Gaussian Naive Bayes classification is a method that can use to process numeric attributes as input and determine two decisions of access that occur on the computer network service [6]: "normal" access or access under "attack" by DDoS as output using Numeric Attribute-based Gaussian Naive Bayes.

Another research about forensic analysis and prevent Cross-Site Scripting (XSS) using the Open Web Application Project (OWASP) Framework covers three essential stages: Attacking stages, Analysis, and Patching. Stages Attacking is doing exercises with Single-Victim-strategy utilizing the

OWASP Xenotix XSS Attack Exploit Framework v6.2 to incorporate assaults Information Gathering, Keylogger, Download spoofer, and Live webcams screen capture to the casualty through the internet browser [7]. Stages Analysis led utilizing Live Forensic by Wireshark, live HTTP Header, and Tcpdump.

Other studies provide reviews of techniques, stages, approaches, and tools to detect web servers is vulnerabilities [8]. Challenges and problems during application security testing prove to provide testers and managers input about application projects [9]. This study also highlighted various authors based on the Open Web Application Security Project (OWASP) Top 10 [10]. As per a report by the Web Application Security Consortium, about 49% of the web applications investigated contain weaknesses of high-hazard levels, and beyond what 13% of the sites can be undermined altogether consequently.

According to the 2018 White Hat report, analysis results using data from more than 20,000 applications indicate a decline in security [11]. More than 75% of malicious attacks are mostly cross-site scripting (XSS) attacks [12]. XSS attacks pose serious threats, especially servers in the financial and economic fields. Such attacks pay considerable attention to developing methods of protection against XSS attacks and proactively detecting vulnerabilities. These aspects determine the emergence of scientific research, especially in studying XSS vulnerabilities in graphic content files.

Several vulnerabilities in the exploitable file upload feature [13] include:

- No validation is performed on the client-side or server-side.
- Client-side validation skippable using developer options in the web browser and not using validation checking file contents.
- No validation is performed to check file size, as validation is based only on content type.
- Attacks can be carried out by manipulating file content types.
- It is allowed to use more than one type of file extension.
- Some conditions may use forbidden file extensions along with file extensions not permitted by the application.

Scalable Vector Graphics (SVG) is a language based on the eXtensible Markup Language (XML) for describing two-dimensional vectors and mixed vector/raster graphics. Stylized SVG content can also be scaled to different display resolutions and can be viewed alone or mixed with HTML content or embedded using XML namespaces in other XML languages. SVG also supports dynamic changes with form scripts. The script can create interactive documents and animation using the declarative animation feature or a script [14].

The Document Object Model (DOM) is a programming Application Programming Interface (API) for accessing and modifying XML documents. The DOM defines the document as a logical structure and the various ways of accessing or manipulating the document. XML is also used to represent different formats of information stored in a heterogeneous system and is mostly interpreted as data rather than documents [15].

Previous research that analyzed the main problems related to web applications and Internet services in several web applications from various organizations, such as banking, health care, financial services, retail, developed a systematic grouping of XSS protection techniques [16] such as rules for protecting website graphic content. Web and prevent XSS vulnerabilities [17] and An Analysis of Vulnerability Web Against Attack Unrestricted Image File Upload [18].

Numerous elements make it trying to make sure about applications that have been mulled over to improve application security. Unreliable applications work because of the weaknesses of multiple components. For example, security testing is done past the point of no return in the SDLC, avoiding security testing in light of the delivery surge, spending limitations. All the more usually, the absence of security mindfulness by designers. The lack of designer consciousness of secure coding norms and the absence of spending plans spent on application security are two of the most alarming issues. This present examination's essential objective is for designers and analyzers to comprehend the fundamental weaknesses of record transfer usefulness, prompting assaults, and their particular alleviations for future secure turns of events. This study conducted a series of tests and performed a graphical content vulnerability analysis against XSS attacks. In contrast to image encoded such as JPEG and PNG, which have metadata and can be processed using image processing [19], it can also be classified based on color and pattern values [20]. This study utilizes different magic numbers and DOM to validate SVG files in the file upload feature in the appropriate XML format scriptwriting structure.

II. Research Methodology

File upload is transferring the files (photos, audio files, etc.) to a server on the website. To upload data to the server, the client first initiates communication with the server by initiating a TCP/IP connection from the client to the server called a handshake. In this communication, the client initiates any communication and not the server. When a connection is established between a client and a server, data transfer can occur between them. It does not require port forwarding to send/receive data to/from the server. Now the client needs a file to upload and a form on a Web page where the file is sent to the server. This allows the user to enter one or more files into the form submission like the code shown in Fig. 1.

After the HTML tag from Fig. 1 sends through the server data, it is often processed to save the file to the webserver disk. The server-side script executing the file is received on the server. The server knows how to handle such requests and store data. It saves files to server disks with multiple names and process data by simply extracting some information from them [13]. This study uses the Waterfall model development method. Waterfall model development methods usually suggest a systematic and sequential approach to software development that starts with customer specifications and progresses through planning, modeling, construction, and completed software deployment [21].

Fig. 2 is the stage of the Waterfall Model which reflects the main points of development activities such as:

- The communication contains system services, system limitations, and objectives set after consultation with system users, defined in detail, and used as system specifications.
- Planning contains creating a system to identify and explain the system abstraction and its relationship, estimated processing time, and scheduling.
- Modeling contains a system design that will be made in the form of a flowchart.
- The construction contains designing the system into a program or unit program and then unit testing, which involves verification to ensure whether each unit meets system specifications. Each program unit and existing programs are integrated and tested as system integrity to confirm whether the system requirements have been met. After testing, the new system is deployed to users.
- The deployment contains stages of installation of the system and is used in practice.

Black-box security testing refers to a software security testing method in which security controls, defense, and application design are tested from the outside, with little or no prior knowledge of how the application is internals work. Essentially, black-box testing takes a similar approach to a real attacker. Since black-box security testing does not assume or know the target being tested, it is a technology-independent testing method [22]. This makes it ideal for various situations, mainly when testing for vulnerabilities arising from deployment issues and server configuration errors.

```
<!DOCTYPE html>
<html>
<body>

<form action="upload.php" method="post" enctype="multipart/form-data">
  Select image to upload:
  <input type="file" name="fileToUpload" id="fileToUpload">
  <input type="submit" value="Upload Image" name="submit">
</form>

</body>
</html>
```

Fig. 1. Sample of File upload HTML Tags.

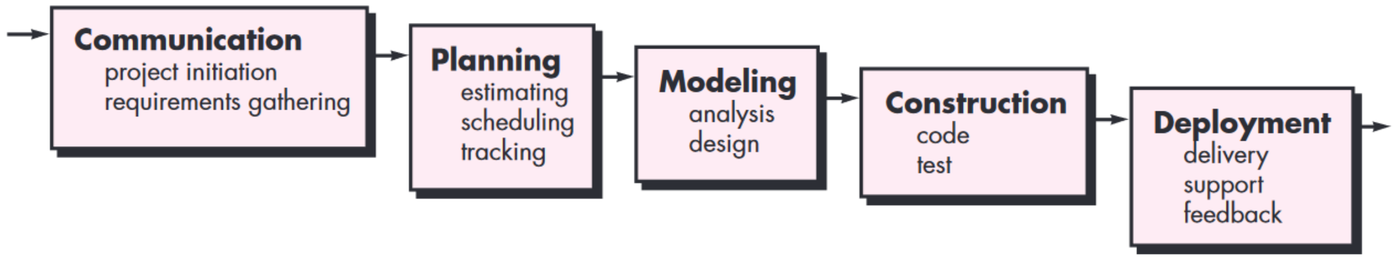

Fig. 2. Waterfall Method.

III. Result and Discussion

This study uses Waterfall as a method of developing the implementation of handling the SVG file upload validation using magic numbers and DOM and black-box security testing refers to a software security testing method with the following stages.

A. *Communication*

The communication stage contains the preparatory stages for making system services, system boundaries, and implementation objectives in detail, according to the system is specifications. SVG files contain tags in the form of XML with file extensions ".svg" and MIME "image/svg+xml" but in checking the ASCII prefix, there are two standards, namely "<?xml" and "<svg ". Bug #79045 in PHP about Incorrect SVG MIME types detected, PHP when rendered SVG may be without an XML header, will return the mime type “image/svg” instead of “image/svg+xml”. However, IANA doesn't list “image/svg” as an existing MIME type, nor does the browser display it [23].

According to SVG standards, the MIME-type returned must always be “image/svg+xml”, PHP must also behave consistently for a single file type. If this is used to distinguish a properly header-fed SVG from non-strict ones, it should be done as part of a file validity check, not a mime type check. A previously reported bug #76543 is indicated as "Not a bug" because it is considered an upstream error with libmagic. However, libmagic returns an svg that doesn't have an xml header as “text/plain”. This indicates that the error is in the adaptation PHP took to eliminate the error [24]. Expected result for MIME type of SVG is “image/svg+xml” but actual result is “image/svg”. SVG files do not contain metadata such as image encoded in JPEG or PNG files.

B. *Planning*

The planning stage contains steps that are carried out using a sample made as presented in Fig. 3, which includes the SVG code as in Fig. 4 with the prefix "<svg " while Fig. 5 contains the SVG code with the prefix "<?xml".

Fig. 3 displays the SVG code presented in Fig. 4 and Fig. 5, showing the same visual appearance with different magic number values.

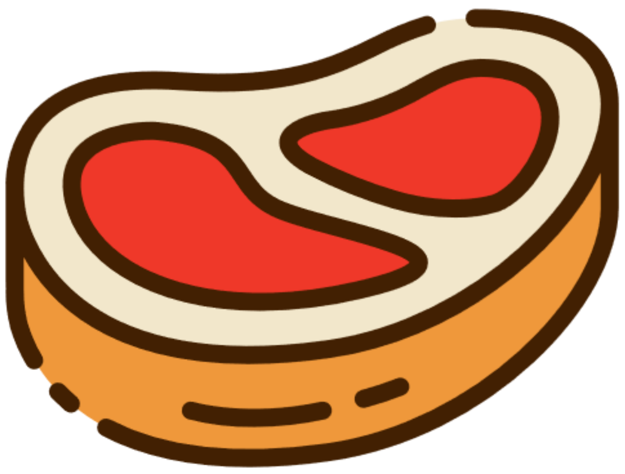

Fig. 3. Sample SVG File.

1) SVG Code with SVG tag first: Fig. 4 contains the SVG code that uses the standard prefix "<?xml" in the form of ASCII hexadecimal value 3C 73 76 67 20, which is the SVG 1.1 standard (Second Edition), which became the W3C recommendation on 16 August 2011 [13]. However, many SVG code writings use the prefix "<svg" in ASCII form, which is hexadecimal 3C 3F 78 6D 6C in Fig. 5.

2) SVG Code with XML tag first: The two standards of the prefix "<?xml" and "<svg " are the standards that are often used and then sampled according to the file extension parameters, magic number, and XML format as in Table I.

```
<svg height="512" viewBox="0 0 128 128"
width="512"
xmlns="http://www.w3.org/2000/svg">
    <g>
        <path d="m7.157 61.039s-8.924-
30.295 31.234-27.477 36.453-15.94 66.258-
11.57c38.141 5.591 6.432 57.391-37.553
63.545-43.805 6.128-56.88-14.101-59.939-
24.498z" fill="#f2e7cb" />
        <path d="m64.006 50.121c-11.747-
6.451 2.471-8.9 2.471-8.9 18.195-2.442
20.9-11.164 32.278-10.175 20.09 1.747 9.23
22.95.881 26.146 0 0-7.734 2.76-19.6-
.637a87.6 87.6 0 0 1 -16.03-6.434z"
fill="#ef3829" />
        <path d="m33.546 43.2s-19.075-.978-
15.81 13.551 25.466 24.3 55.5 17.283-7.336-
11.407-16.483-15.978-4.44-13.196-23.207-
14.856z" fill="#ef3829" />
        <path d="m121.7 38.052c0 17.53-
24.409 43.261-54.6 47.485-43.805 6.128-
56.88-14.1-59.939-24.5a26.44 26.44 0 0 1 -
.844-6.82v19.859a26.436 26.436 0 0 0 .844
6.961c3.059 10.4 16.134 30.626 59.939 24.5
30.14-4.217 54.516-29.867 54.6-47.4v-
20.086z" fill="#ef983b" />
        <g fill="#422002">
```

Fig. 4. Sample SVG Script with SVG Tag First.

```
<?xml version="1.0" standalone="no"?>
<svg height="512" viewBox="0 0 128 128"
width="512"
xmlns="http://www.w3.org/2000/svg">
    <g>
        <path d="m7.157 61.039s-8.924-
30.295 31.234-27.477 36.453-15.94 66.258-
11.57c38.141 5.591 6.432 57.391-37.553
63.545-43.805 6.128-56.88-14.101-59.939-
24.498z" fill="#f2e7cb" />
        <path d="m64.006 50.121c-11.747-
6.451 2.471-8.9 2.471-8.9 18.195-2.442
20.9-11.164 32.278-10.175 20.09 1.747 9.23
22.95.881 26.146 0 0-7.734 2.76-19.6-
.637a87.6 87.6 0 0 1 -16.03-6.434z"
fill="#ef3829" />
        <path d="m33.546 43.2s-19.075-.978-
15.81 13.551 25.466 24.3 55.5 17.283-7.336-
11.407-16.483-15.978-4.44-13.196-23.207-
14.856z" fill="#ef3829" />
        <path d="m121.7 38.052c0 17.53-
24.409 43.261-54.6 47.485-43.805 6.128-
56.88-14.1-59.939-24.5a26.44 26.44 0 0 1 -
.844-6.82v19.859a26.436 26.436 0 0 0 .844
6.961c3.059 10.4 16.134 30.626 59.939 24.5
30.14-4.217 54.516-29.867 54.6-47.4v-
20.086z" fill="#ef983b" />
        <g fill="#422002">
```

Fig. 5. Sample SVG Script with XML Tag First.

TABLE I. SAMPLES OF FILES UPLOAD

No.	File Extension	Magic Number	XML/DOM
1	✕	✕	✕
2	✓	✕	✕
3	✕	✓	✕
4	✕	✕	✓
5	✓	✓	✕
6	✓	✕	✓
7	✕	✓	✓
8	✓	✓	✓

Table I contains eight SVG samples prepared for SVG validation in the system by removing some of the three parameters (validation file extension, the magic number, and XML) from the file, as in Table II.

TABLE II. SVG TYPES FILES

File Extension	ASCII		Magic Number	
	Start of File	End Of File	Start of File	End Of File
svg	<?xml	</svg>	3C 3F 78 6D 6C	3C 2F 73 76 67 3E
svg	<svg	</svg>	3C 73 76 67 20	3C 2F 73 76 67 3E

Table II contains the types of information contained in the SVG file used based on the prefix tag "<?xml" with the magic number value "3C 3F 78 6D 6C" and MIME "text/xml" while "<svg " has a magic number value "3C 73 76 67 20" and MIME is" image/svg". The scenarios in Table III described as follows :

3) PHP Code in TXT file: Fig. 6 illustrates a PHP file renamed file extension from ".php" to ".txt" with file contents like Fig. 7. In this scenario, do not use the file extension, magic number, and XML format.

Fig. 7 contains PHP code by displaying PHP settings or server information by changing the file extension from ".php" to ".txt".

4) PHP Code in SVG file: Fig. 8 illustrates a PHP file renamed file extension from ".php" to ".svg" with file contents like Fig. 9. In this scenario, use the SVG file extension but do not use the magic number at the start of the file ("3C 3F 78 6D 6C" or "3C 73 76 67 20") and the end of the file (3C 2F 73 76 67 3E) and XML format.

Fig. 9 contains PHP code by displaying PHP settings or server information by changing the file extension from ".php" to ".svg".

5) PHP Code in SVG tag TXT file: PHP Code file renamed file extension from ".php" to ".txt" with file contents like Fig. 10. In this scenario, use the magic number at the start of the file ("3C 3F 78 6D 6C" or "3C 73 76 67 20") and the end of the file (3C 2F 73 76 67 3E) but do not use XML format and the SVG file extension.

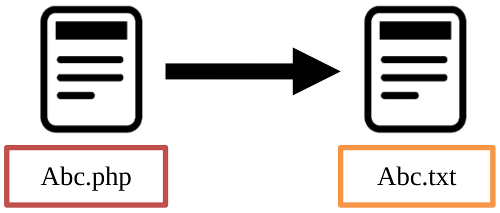

Fig. 6. Illustration of Changing the File Extension from PHP to TXT.

```
<?php
phpinfo();
?>
```

Fig. 7. PHP Code in TXT File.

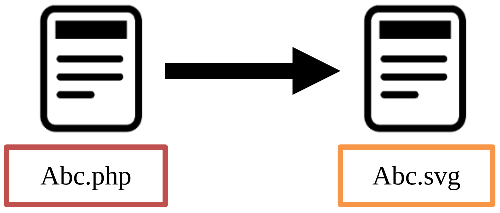

Fig. 8. Illustration of Changing the File Extension from PHP to SVG.

```
<?php
phpinfo();
?>
```

Fig. 9. PHP Code in SVG File.

```
<svg height="512" viewBox="0 0 128 128" width="512"
xmlns="http://www.w3.org/2000/svg">
<?php
phpinfo();
?>
```

Fig. 10. PHP Code in SVG Tag TXT File.

Fig. 10 contains PHP code by displaying PHP settings or server information with SVG tag by changing the file extension from ".php" to ".txt".

6) XML tag in TXT file: SVG tags with XML tag first or magic number value "3C 3F 78 6D 6C" with file contents like Fig. 11. In this scenario, use XML format but do not use the SVG file extension and magic number at the start of the file ("3C 3F 78 6D 6C" or "3C 73 76 67 20") and the end of the file (3C 2F 73 76 67 3E).

7) PHP Code in SVG tag first: Fig. 12 contains PHP code by displaying PHP settings or server information in SVG code. In this scenario, use the file extension and magic number ("3C 3F 78 6D 6C" or "3C 73 76 67 20") but do not use or invalid XML format.

8) XML Code in SVG file: Fig. 13 contains XML tags in SVG file. In this scenario, use the file extension and XML format but do not use the magic number at the start of the file ("3C 3F 78 6D 6C" or "3C 73 76 67 20") and the end of the file (3C 2F 73 76 67 3E).

9) SVG Code in TXT file: Fig. 14 contains SVG tags in the TXT file. In this scenario, do not use the SVG file extension but use the magic number at the start of the file ("3C 3F 78 6D 6C" or "3C 73 76 67 20") and the end of the file (3C 2F 73 76 67 3E) but XML format.

10)SVG Code with SVG/XML first tag in SVG file: Fig. 15 contains SVG tags in SVG file extension. In this scenario, use the file extension and magic number at the start of the file ("3C 3F 78 6D 6C" or "3C 73 76 67 20") and the end of the file (3C 2F 73 76 67 3E) and valid XML format.

```
<?xml version="1.0" standalone="no"?>
<svg height="512" viewBox="0 0 128 128" width="512"
xmlns="http://www.w3.org/2000/svg">
    <g>
        <path d="m7.157 61.039s-8.924-30.295 31.234-
27.477 36.453-15.94 66.258-11.57c38.141 5.591 6.432
57.391-37.553   63.545-43.805   6.128-56.88-14.101-
59.939-24.498z" fill="#f2e7cb" />
..................................................
        </g>
    </g>
</svg>
```

Fig. 11. XML Tag in TXT File.

```
<?xml version="1.0" standalone="no"?>
<svg height="512" viewBox="0 0 128 128" width="512"
xmlns="http://www.w3.org/2000/svg">
    <g>
.................................................................................................
    </g>
</svg>
<?php
phpinfo();
?>
```

Fig. 12. PHP Code in SVG Tag First.

```
<?xml version="1.0" encoding="UTF-8"?>
<svg height="512" viewBox="0 0 128 128" width="512"
xmlns="http://www.w3.org/2000/svg">
<note>
  <to>Fahmi</to>
  <from>Anwar</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

Fig. 13. XML Code in SVG File.

```
<svg height="512" viewBox="0 0 128 128" width="512"
xmlns="http://www.w3.org/2000/svg">
    <g>
        <path d="m7.157 61.039s-8.924-30.295 31.234-
27.477 36.453-15.94 66.258-11.57c38.141 5.591 6.432
57.391-37.553   63.545-43.805   6.128-56.88-14.101-
59.939-24.498z" fill="#f2e7cb" />
.................................................
        </g>
    </g>
</svg>
```

Fig. 14. SVG Code in TXT File.

```
<svg height="512" viewBox="0 0 128 128" width="512"
xmlns="http://www.w3.org/2000/svg">
    <g>
        <path d="m7.157 61.039s-8.924-30.295 31.234-
27.477 36.453-15.94 66.258-11.57c38.141 5.591 6.432
57.391-37.553   63.545-43.805   6.128-56.88-14.101-
59.939-24.498z" fill="#f2e7cb" />
.................................................
        </g>
    </g>
</svg>
```

Fig. 15. SVG Code with SVG/XML First Tag in SVG File.

C. Modeling

The modeling stage contains the process carried out by the system in the form of a flowchart, as shown in Fig. 16 for general validation and Fig. 17 for new validation.

1) General Validation SVG File: Fig. 16 is an SVG image validation flowchart utilizing file extensions and MIME type using the function mime_content_type($file) in PHP programming language.

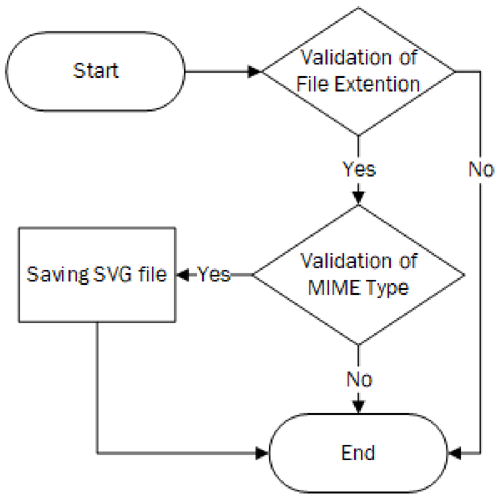

Fig. 16. Flowchart of General Validation SVG File.

2) New Validation SVG File: Fig. 17 is an SVG image validation flowchart utilizing file extensions, magic numbers, and DOM using the PHP programming language.

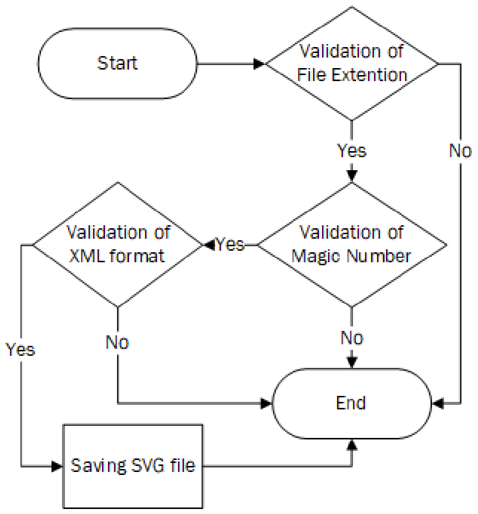

Fig. 17. Flowchart of New Validation SVG File.

D. Construction

The construction stage contains the system's steps in the form of a flowchart converted into Algorithm 1.

1) General Validation SVG File: Algorithm 1 contains the PHP programming algorithm, which functions to filter or validate SVG images by utilizing file extensions and mime using the PHP programming language. The algorithm created is tested using black-box testing with the test scenarios presented in Table III.

Algorithm 1 General Validation SVG File Upload

```
<?php
$filePath = "Abc.svg";
$errors = array();
$fileExtensionWhitelist = array("SVG");
$imageFileExtension = pathinfo($filePath, PATHINFO_EXTENSION);

if (!in_array(strtoupper($imageFileExtension), $fileExtensionWhitelist)){
 array_push($errors,"File Extension is not allowed");
}

if(mime_content_type($filePath)!="image/svg+xml"){
 array_push($errors,"MIME is not allowed");

}

if(empty($errors)){
 echo "SVG is valid";
}else{
 foreach ($errors as $data) {
 echo $data."<br />";
 }
}
?>
```

The algorithm tested using black-box testing by validation types with the results in Table III. All test scenarios have been tested by comparing the expected results with the actual results, then the conclusion of the desired results has been successful or appropriate. The test results in Table III contain one scenario that matches expectations and 6 scenarios that do not match expectations from 8 scenarios so that the success rate is 75%.

2) New Validation SVG File: Algorithm 2 contains the PHP programming algorithm, which functions to filter or validate SVG images by utilizing file extensions, magic numbers, and DOM using the PHP programming language.

Algorithm 2 New Validation SVG File Upload

```
<?php
$filePath = "Abc.svg";
$errors = array();
$fileExtensionWhitelist = array("SVG");
$imageFileExtension = pathinfo($filePath, PATHINFO_EXTENSION);

if (!in_array(strtoupper($imageFileExtension), $fileExtensionWhitelist)){
 array_push($errors,"File Extension is not allowed");
}

function magicNumberStartOfFile($filename) {
 if(file_exists($filename)){
 $handle = fopen($filename, 'r');
 $bytes = strtoupper(bin2hex(fread($handle, 5)));
 fclose($handle);
 return $bytes;
 }else{
 return false;
 }
}

function magicNumberEndOfFile($filename) {
 if(file_exists($filename)){
 $handle = fopen($filename, 'r');
 fseek($handle, -6, SEEK_END);
 $bytes = strtoupper(bin2hex(fread($handle, 6)));
 fclose($handle);
 return $bytes;
 }else{
 return false;
 }
}

$magicNumberStartOfFileWhitelist = array("3C3F786D6C", "3C73766720");
if (!in_array(magicNumberStartOfFile($filePath), $magicNumberStartOfFileWhitelist)){
 array_push($errors,"Magic Number (Start of File) is not allowed");
 }

$magicNumberEndOfFileWhitelist = array("3C2F7376673E");
if (!in_array(magicNumberEndOfFile($filePath), $magicNumberEndOfFileWhitelist)){
 array_push($errors,"Magic Number (End of File) is not allowed");
 }

libxml_use_internal_errors(TRUE);
$dom = new DOMDocument;
$dom->Load($filePath);
if ($dom->validate()) {
array_push($errors,"XML format is not valid");
var_dump(libxml_get_errors());
}

if(empty($errors)){
 echo "SVG is valid";
}else{
 foreach ($errors as $data) {
 echo $data."<br />";
 }
}
?>
```

TABLE III. BLACK-BOX TESTING OF GENERAL VALIDATION

No.	Scenarios of Validation			Expected Result	Actual Result
	File Extension	*Magic Number*	*Document Object Model*		
1	×	×	×	Uploaded failed	[✓] Succeed [ ] Failed
2	✓	×	×	Uploaded failed	[✓] Succeed [ ] Failed
3	×	✓	×	Uploaded failed	[✓] Succeed [ ] Failed
4	×	×	✓	Uploaded failed	[✓] Succeed [ ] Failed
5	✓	✓	×	Uploaded failed	[ ] Succeed [✓] Failed
6	✓	×	✓	Uploaded failed	[ ] Succeed [✓] Failed
7	×	✓	✓	Uploaded failed	[✓] Succeed [ ] Failed
8	✓	✓	✓	Uploaded succeed	[✓] Succeed [ ] Failed

All test scenarios have been tested by comparing the expected results with the actual results, then the conclusion of the desired results has been successful or appropriate. The algorithm tested using a black-box of general validation with the results in Table IV.

The test results in Table IV contain six scenarios that match expectations and all scenarios that match expectations from 8 scenarios so that the success rate is 100%.

TABLE IV. BLACK-BOX TESTING OF NEW VALIDATION

No.	Scenarios of Validation			Expected Result	Actual Result
	File Extension	*Magic Number*	*Document Object Model*		
1	×	×	×	Uploaded failed	[✓] Succeed [ ] Failed
2	✓	×	×	Uploaded failed	[✓] Succeed [ ] Failed
3	×	✓	×	Uploaded failed	[✓] Succeed [ ] Failed
4	×	×	✓	Uploaded failed	[✓] Succeed [ ] Failed
5	✓	✓	×	Uploaded failed	[✓] Succeed [ ] Failed
6	✓	×	✓	Uploaded failed	[✓] Succeed [ ] Failed
7	×	✓	✓	Uploaded failed	[✓] Succeed [ ] Failed
8	✓	✓	✓	Uploaded succeed	[✓] Succeed [ ] Failed

E. Deployment

This research can be applied to the SVG file upload algorithm using magic numbers and DOM after other validations, such as validating file extensions to check SVG validation in the file upload process by performing superior filtering with validation of writing XML structures so that they can filter. SVG text that conforms to a standard XML format is incompatible.

IV. Conclusion

This research produces an application that can provide security in uploading files to web-based applications, especially SVG files. The Waterfall method is used to develop or build software because of the many preparatory stages before the software development stage. Handling of security validation for uploading SVG files using file extensions and MIME types has a success rate of 75% from the eight tested scenarios while handling using file extensions, magic numbers, and Document Object Model (DOM) a success rate of 100% from 8 test scenarios. Testing uses a black-box so that handling using the file extension, magic number, and Document Object Model (DOM) is better than using only file extensions and mime types. Subsequent research work is that the proposed method must be validated by various unique classifications of SVG files or other file formats.

Acknowledgment

This research is supported by Direktorat Riset dan Pengabdian Masyarakat, Direktorat Jenderal Penguatan Riset dan Pengembangan Kementerian Riset, Teknologi, dan Pendidikan Tinggi Republik Indonesia. Surat Kontrak Pelaksanaan Penelitian Kementerian Riset dan Teknologi/ Badan Riset dan Inovasi Nasional (KEMENRISTEK/BRIN) Tahun Tunggal Tahun Anggaran 2020 Nomor: PTM-019/SKPP.TT/LPPM UAD/VI/2020.

References

[1] H. Chen, L. J. Zhang, B. Hu, S. Z. Long, and L. H. Luo, “On Developing and Deploying Large-File Upload Services of Personal Cloud Storage,” Proc. - 2015 IEEE Int. Conf. Serv. Comput. SCC 2015, pp. 371–378, 2015, doi: 10.1109/SCC.2015.58.

[2] X. Li and Y. Xue, “A survey on web application security,” Nashville, TN USA, 2011, [Online]. Available: http://isis.vanderbilt.edu/sites/default/files/main_0.pdf.

[3] A. Yudhana, I. Riadi, and F. Ridho, “DDoS classification using neural network and naïve bayes methods for network forensics,” Int. J. Adv. Comput. Sci. Appl., vol. 9, no. 11, pp. 177–183, 2018, doi: 10.14569/ijacsa.2018.091125.

[4] I. Riadi, A. W. Muhammad, and Sunardi, “Neural network-based ddos detection regarding hidden layer variation,” J. Theor. Appl. Inf. Technol., vol. 95, no. 15, pp. 3684–3691, 2017.

[5] A. Iswardani and I. Riadi, “Denial of service log analysis using density K-means method,” J. Theor. Appl. Inf. Technol., vol. 83, no. 2, pp. 299–302, 2016.

[6] A. Fadlil, I. Riadi, and S. Aji, “DDoS Attacks Classification using Numeric Attribute-based Gaussian Naive Bayes,” Int. J. Adv. Comput. Sci. Appl., vol. 8, no. 8, pp. 42–50, 2017, doi: 10.14569/ijacsa.2017.080806.

[7] A. Kurniawan, I. Riadi, and A. Luthfi, “Forensic analysis and prevent of cross site scripting in single victim attack using open web application security project (OWASP) framework,” J. Theor. Appl. Inf. Technol., vol. 95, no. 6, pp. 1363–1371, 2017.

[8] S. B. Almi, “Web Server Security and Survey on Web Application Security,” Int. J. Recent Innov. Trends Comput. Commun., vol. 2, no. 1, pp. 114–119, 2014, [Online]. Available: http://ijritcc.org/IJRITCC Vol_2 Issue_1/Web Server Security and Survey on Web Application Security.pdf.

[9] A. Jaiswal, G. Raj, and D. Singh, “Security Testing of Web Applications: Issues and Challenges,” Int. J. Comput. Appl., vol. 88, no. 3, pp. 26–32, 2014, doi: 10.5120/15334-3667.

[10] OWASP, “OWASP Top 10 2017 - The Ten Most Critical Web Application Security Risks Release Candidate 2,” pp. 1–25, 2017, [Online]. Available: https://www.owasp.org/images/b/b0/OWASP_Top_10_2017_RC2_Final.pdf.

[11] WhiteHatSec, “2018 Application Security Statistics Report - The Evolution of the Secure Software Lifecycle,” 2018. https://info.whitehatsec.com/Content-2018StatsReport_LP.html.

[12] V. Subramaniyaswamy, G. Venkata Kalyani, and N. Likhitha, “Securing web applications from malware attacks using hybrid feature extraction,” Int. J. Pure Appl. Math., vol. 119, no. 12, pp. 13367–13385, 2018.

[13] K. Pooj and S. Patil, “Understanding File Upload Security for Web Applications,” Int. J. Eng. Trends Technol., vol. 42, no. 7, pp. 342–347, 2016, doi: 10.14445/22315381/ijett-v42p261.

[14] W3C, “Scalable Vector Graphics (SVG) 1.1 (Second Edition),” W3C, 2011. https://www.w3.org/TR/SVG11/ (accessed Oct. 20, 2020).

[15] S. D. Ankush, “XSS Attack Prevention Using DOM based filtering API XSS Attack Prevention Using DOM based fitering API,” 2014.

[16] M. Johns, B. Engelmann, and J. Posegga, “XSSDS: Server-side detection of Cross-site Scripting attacks,” Proc. - Annu. Comput. Secur. Appl. Conf. ACSAC, pp. 335–344, 2008, doi: 10.1109/ACSAC.2008.36.

[17] D. Zubarev and I. Skarga-Bandurova, “Cross-Site Scripting for Graphic Data: Vulnerabilities and Prevention,” Conf. Proc. 2019 10th Int. Conf. Dependable Syst. Serv. Technol. DESSERT 2019, pp. 154–160, 2019, doi: 10.1109/DESSERT.2019.8770043.

[18] I. Riadi and E. I. Aristianto, “An Analysis of Vulnerability Web Against Attack Unrestricted Image File Upload,” Comput. Eng. Appl. J., vol. 5, no. 1, pp. 19–28, 2016, doi: 10.18495/comengapp.v5i1.161.

[19] I. Riadi, A. Fadlil, and T. Sari, “Image Forensic for detecting Splicing Image with Distance Function,” Int. J. Comput. Appl., vol. 169, no. 5, pp. 6–10, 2017, doi: 10.5120/ijca2017914729.

[20] R. A. Surya, A. Fadlil, and A. Yudhana, “Identification of Pekalongan Batik Images Using Backpropagation Method,” J. Phys. Conf. Ser., vol. 1373, no. 1, 2019, doi: 10.1088/1742-6596/1373/1/012049.

[21] R. S. Pressman and B. R. Maxim, Software Engineering : a practitioner’s approach, 8th ed. New York: McGraw-Hill Education, 2014.

[22] J. Bau, E. Bursztein, D. Gupta, and J. Mitchell, “State of the art: Automated black-box web application vulnerability testing,” Proc. - IEEE Symp. Secur. Priv., pp. 332–345, 2010, doi: 10.1109/SP.2010.27.

[23] IANA, “Media Types,” 2020. https://www.iana.org/assignments/media-types/media-types.xhtml.

[24] Sloth at 0k dot vc, “PHP :: Bug #79045 :: Incorrect svg mimetypes detected,” 2019. https://bugs.php.net/bug.php?id=79045 (accessed Nov. 21, 2020).