Development of Content Security Policy Detection and Reporting Web Application

Mahwish Naz
Dept. of Computer Science and Engineering
Seoultech University
Seoul, Korea
Email: mahwishnaz488 [AT] gmail.com

Kilhung Lee (Corresponding Author)
Dept. of Computer Science and Engineering
Seoultech University
Seoul, Korea
Email: khlee [AT] seoultech.sc.kr

Abstract—**Attacks detection and prevention is becoming progressively challenging, in Web Application. The Web Applications requisite attack detection interface that can check about the services, analyzed obviate studies, and perform real time monitoring to secure the web applications. There are esteemed threats use for data hack and website defacement such as XSS (Cross-Site Scripting), XSRF (Cross-Ste Request Forgery), XEE (XML External Entity), Code Injections, DOS(Denial of Services) etc. The number of XSS has been increasing with high intensity, so it is compulsory to develop solutions that can detect and report attacks as well as analyze for prevention of modern web applications. Based on this situation this paper proposed a method which is Content Security Policy for detection and reporting vulnerable web applications. Content Security Policy also prevents the exploitation of cross-site scripting vulnerabilities.**

Keywords-Content Security Policy, Features/Permission Policy, Web Application Security.

I. INTRODUCTION

In view of the fact that the rapid development of technology, many activities related to daily life are introducing to the web environment, due to such situation the chance of hacking of web application has increased[1].

During this era, attack detection and prevention is becoming more and more challenging in web applications. The Web Applications requisite attack detection interface that can check about the services, analyzed obviate studies, and perform real time monitoring to secure the web applications. Web Security is a system of protection that protect our websites from attackers.

In this paper we developed a Content Security Policy detection and reporting system which detect and generate reports of vulnerable websites and shows the results in tables using browser as well as we make test scenarios to show how our system restrict vulnerable web application URLs and violate reports. After perform testing we did analysis between the results of our tests using graphs and pie charts. Also add some real time monitoring which helped us to identify actual reporting time as well as the type of violation.

The basic goal of this system is to secure websites from attackers when they put different types of attacks on any web application like Cross-site scripting and cross site request forgery. We detect and block attacks using different policies of Content Security Policy, Features/ Permission policies.

According to these policies we can allow or restrict different Web URLs or images for specific browsers. Our System will test web application URLs and if it found XSS or XSRF threats it will generate JSON report and save results to our database and then make analysis between these results to check which web application has high possibility of attacks or which has low and will show the results in Bar graphs and Pie charts using different colors to makes it efficient.

Cross-Site Scripting attack has the capability to inject inline scripts into the web pages of web applications. This could be complicate in the procedure of detecting and preventing XSS attack. To address these attacks new method is needed to deal with huge quantity of data from many web users. There are three major types of cross-site scripting attack, which is Reflected XSS, Stored XSS, Dom based XSS[1]. Content Security Policy is a Web program procedure which is designed to mitigate Cross-site scripting[2].

Today Content security policy is excellent protector against cross-site scripting. It is a suggestive policy procedure which allow website developers to determine which type of user side information can be loaded and executed by the browsers. The main purpose of CSP is to restrict website abilities to load and execute malicious code[2]. We include Content-security policy HTTP header to enable a CSP. The CSP policy contain single or many directive, and use semicolon to separate these directives such as the listed directive will allow only scripts to be loaded from the same origin as the page itself: script-src ‘self’ as well as the following directive will allow scripts to be loaded from a specific domain like https://example.com.

Basically, CSP execute the same origin policy to ensure that the browser only execute material from authenticate sources. We can also configure a policy by using meta elements.

<meta http-equiv =“Content-Security-Policy”
Content = “script-src ‘self’; default-src ‘none’;”/>

II. RELATED RESEARCH

In this section we will include reviews of previous researches. Firstly, we will discuss some common threats, secondly, some work that aim to protect web applications from threats using CSP policies. In this research there are different methods are used to detect and prevent websites from different attacks. Here we will review threats that can attacks in web applications. Although Cross-site scripting vulnerability is less common than past while in the web directions are still uncontrolled. The interaction of users with web application is increasing with the passage of time, the opportunities for XSS vulnerabilities also increasing in web application[3].

According to the research [4] a method is proposed by the authors which is WAF(Web Application Firewall). By using this methods packets can be filter as well as block danger HTTP requests. The implementation of WAF on web applications is depend on the installation of Mod Security module on a Reverse proxy device. In [5] authors proposed that they can control the content upload on their websites by content restrictions applied on websites. They also propose an implementation regarding content security policy will enable application developers to check the content loading rule for their web applications. Furthermore, we reviewed the Content Security Policy and found it is the last line of defense.

A. Web Security

Web Security is also known as cyber security. It basically means that protecting a web application by detection, preventing and responding to web attacks or
Threats. Web security based on same origin policy which prevents the content loading from outside of origin. The goal of web security to prevent content from
attackers. Web security is an application that secure web applications from unauthorized logins, use as well as editing. Here is the list of some popular technical solutions for testing, building and preventing threats:
Black Box Testing use to finding the input/output validation errors. The Fuzzing Tools(Fuzz is realtime software testing method to find software errors, White Box Testing tools is a type of testing which use for conformation of executed code, web Application Firewall control undesirable traffic in a network, Security or vulnerability scanners use to search and reports vulnerabilities in an organization, and password cracking tools.

- *OWASP:*The Open Web Application Security Project is an organization that helps to provide high security to software, tools and resources community, Networking education and Training without any self benefit.

- *OWASP Web Security Testing Guide:* This project produce the leading web security testing resources for the developer of web applications. Its all about testing like what is testing, why perform testing, when to test, what to test. Testing techniques: Manual inspections and Reviewers, Threat Modeling, Code Review, Penetration Testing.

- *Web Attacks:* Top Vulnerabilities for all web applications includes XSS, XSRF, XEE, SQL injections, Code injections, Command injections, DOS and Distributed DOS. This paper concentrate XSS and little bit of XSRF. There are many ways to protect from Web Applications from different types of attacks but CSP protect security behavior well.

B. Cross-site Scripting

In these days cross-site scripting is most popular vulnerability throughout the internet, and effect numbers of users interactions to the web applications[11]. It is such type of threats which assure hackers to attacks any victim account accounts with vulnerable pages of websites. It is a high level of risk able threat[23]. How does XSS work: Vulnerable Web Application will effect by the cross site scripting when the user will load the page. Attackers inject code in vulnerable web application and send link to authenticate user through any message or email. When the user click link and login on his/her valid account then attacker can see, delete or modify account and steal information on the behalf of victim. Basically, victim does not know about the attack.
XSS proof of concept: In this research we confirm that most kind of XSS attacks occure due to execution of javascript and HTML tags by our own browsers.

- *Types of XSS:* XSS threat categories into three types:
  1. Stored: In this type the data will store in database.
  2. Reflected: The code will not store in database, reflect by server.
  3. DOM-based: Code will store as well as execute in the browser.

There are two another types in cross-site scripting

1. Server XSS: Generate an answer when any unauthorized user send data.
2. Client XSS: Show when any unauthorized send data for update.

C. Cross-Site Request Forgery

Furthermore, this study related to CSRF attack damage the trust relationship between a browser and authorized client as well as a web server. Basically the browser trust the actions performed by the user device on the behalf of victim[10]. In this type of attack user cannot understand what type of action, victim has to execute unwanted action on authenticated web application which is sent by attacker. When end user click link then attackers can work on the behalf of end user with authenticated account and can do anything, like transfer fund or delete history etc.

D. Content-Security Policy

Content-security policy is a browser security mechanism that purpose to protect web application from different attacks[11]. It is the last line for defending against cross-site scripting. If XSS prevention fail then we can use CSP for protection by using multiple directives and security headers. For better security W3C web application security recommended by users. It is also supported by modern web browsers.

In 2004 Robert Hansen was a person who deploy a method for restriction of web application contents. This method was firstly executed by Firefox 4 and then other browser also started to execute. In 2012 version1 was published as W3C candidate recommendation and quickly with further versions. In 2014 level 2 was published as well as in 2015 level 3 was developed with the new features.

III. Development of CSP Reporting and Analysis System

A. Implementation

Security in web applications is an issue that needs attention. There are several solutions that can be done to implement security services on web applications. Even though this is not completely perfect, it is a preventive measure from unwanted things. One solution that can be applied is to implement a CSP Policy.

CSP is preventing the exploitation of cross-site scripting vulnerabilities. When an application uses a strict policy, an attacker who finds an XSS bug will no longer be able to force the browser to execute malicious scripts on the page. To achieve our target we implemented CSP, FP, and PP approaches to generate a report and restrict the attacks against web Applications. The two types of attacks used to test attacks are based on a list which is very wild threats in web applications approved by OWASP Top Ten Web Application Security Risks.

Testing: The current research proposed that the implementation of CSP in a web application may improve security functions, so as to prevent damage from attacks that may occur against web application. Testing conducted in this research were determined to be performed under such conditions like by using directives or security headers of CSP, FP/PP which URL can be blocked or which access web applications.

Analysis: After testing, the next step was the analysis to see whether if there were different result between the different conditions like using different directives as well as the security headers. In our analysis According to use of the list of these security headers we use different example websites for the testing like 'http://site.example/scripts.js', 'http://example.com/foo/bar/'. we found CSP (Content Security Policy) is the best. It provides best security.3.

Conclusions: With the help of CSP, FP, and PP secure the web Applications from the attacks of attackers. These all works with the directives. For generating the violation reports using these policies make some test scenario. And then make some analysis between the results of them. Add realtime monitoring also to show the actual time of reporting. And also show which type of security http header block different API or browser with malicious codes.

B. Security Headers

HTTP Headers are a great booster for web security with easy implementation. Proper HTTP response headers help to prevent security vulnerabilities. An HTTP header is a response by a web server to a browser that is trying to access a web page. Here is the list of Security headers:

1. X-Frame-Options: The X-Frame-Options security header helps stop click-jacking attacks. This type of security header can be used to show that browser should load or execute a page or not. Mostly web application uses this type of security header to save from different attacks. Syntax: X-Frame-Options: DENY.
2. Cross-Site Scripting Protection (X-XSS): X-XSS header helps protect websites against script injection attacks. When an attacker injects malicious JavaScript code into an HTTP request for accessing confidential information such as session cookies, at that time HTTP X-XSS-Protection header can stop the browsers from loading web pages. XSS is a very common and effective attack. Syntax: X-XXS-Protection: 1; node-block
3. X-Content-Type-Options: X-Content-Type-Options response header prevents the browser from MIME-sniffing attack a response away from the show content type. A MIME-sniffing vulnerability allows an attacker to inject a malicious resource, suppose an attacker changes the response for an innocent resource, such as an image. With MIME sniffing, the browser will ignore the declared image content type, and

instead of rendering an image will execute the malicious script. Syntax: X-Content-Type-Options: nosniff

4. Strict-Transport-Security: The Strict-Transport-Security Header is also called the HTTP Strict Transport Security header (HSTS). Many websites only have a 301 redirect from HTTP to HTTPS. But that's not enough to keep the website secure because the website is still vulnerable to a man-in-the-middle attack. HSTS prevents an attacker from downgrading the HTTPS connection to an HTTP connection which then allows the attacker to take advantage of insecure redirects. Syntax: Strict-Transport-Security: max-age=<expire-time>

5. Content-Security-Policy: A content security policy (CSP) helps to protect a website and the site visitors from Cross Site Scripting (XSS) attacks and from data injection attacks. By using HTTP Content-Security-Policy response header web application administrators can control all resources like they can set any page should be load or not. Syntax: Content-Security-Policy: <policy-directive>; <policy-directive>

6. Cross-Origin-Embedder-Policy: The HTTP Cross-Origin-Embedder-Policy (COEP)
response header prevents a document from loading any cross-origin resources that don't explicitly grant the document permission (using CORP or CORS). Syntax: Cross-Origin-Embedder-Policy: unsafe-none | require-corp

7. Content-Security-Policy-Report-Only: By using the HTTP Content-Security-Policy-Report-Only response header web application administrators can evaluate with different policies by observing their results. Through HTTP POST request we can send these violation reports consist of JSON documents. Syntax: Content-Security-Policy-Report-Only: <policy-directive>; <policy-directive>

8. CSP-violation: By using the HTTP Content-Security-Policy-Report-Only response header web application administrators can evaluate with different policies by observing their results. Through HTTP POST request we can send these csp-violation reports consist of JSON documents. CSP Violation reports occurred where high possibility of XSS attacks.

C. Reporting API

The Reporting API provides a generic reporting mechanism for web applications to use to make reports available based on various platform features (for example Content Security Policy, Feature-Policy, or feature deprecation reports) in a consistent manner. In this research we generate different reports by Reporting API.

1. Document-policy-violation: Document Policies can be set in the HTTP response headers of a resource. It sets the policy on the document that it's served with. It's possible to create restrictions on image sizes, compression ratios, lazy loading of frames, use of sync API calls, etc. Document Policy is an extension of Permissions Policy and allows to fine-tune a policy for the structure of a document. Document Policies is an extension of Permissions Policy, violations can easily be tracked in URI ports.com account under the section Permissions Policy.

2. Document-policy-violation: Document Policies can be set in the HTTP response headers of a resource. It sets the policy on the document that it's served with. It's possible to create restrictions on image sizes, compression ratios, lazy loading of frames, use of sync API calls, etc. Document Policy is an extension of Permissions Policy and allows to fine-tune a policy for the structure of a document. Document Policies is an extension of Permissions Policy, violations can easily be tracked in URI ports.com account under the section Permissions Policy.

TABLE I. REPORTING API

Directive	**Controlled Resources types**
Blocked-uri	The URI of the resource that was blocked from loading by the Content Security Policy.
Document-uri	The URI of the document in which the violation occurred.
Effective-directive	The effective- directive is the directive which enforcement caused the violation.
Original-policy	Content-Security-Policy-Report-Only HTTP header identify the Original policy.
Referrer	In referrer violation occurred like as document-uri.
script-sample	It is inline script which is injected by attacker in any vulnerable web applications.
Status-code	Resources has global object was instantiated HTTP status code.
Violated-directive	It is a name of the such type of policy which can violated in policy section.

3. Deprecation: Deprecation report Indicates that a Web API or other browser feature being used in the website is expected to stop working in a future release. Indicated by a Report.body property with a Deprecation Report Body return value.

4. Intervention: Intervention report Indicates that a request made by the website has been denied by the browser, e.g. for security or user annoyance reasons. Indicated by a Report body property with a Intervention Report Body return value.

D. Content Security Policy

Content-Security-Policy is a web program procedure which is designed to mitigate cross-site scripting (XSS), that is the top security vulnerability in modern websites [2]. Today, Content Security Policy is highest promising corrective against cross-site scripting. Content security policy is a demonstrative policy procedure which allows websites creator to define which type of client-side material can be loaded and executed by the different type of browsers. By rejected inline scripts and allowing only believe able domains as a origin of external scripts, CSP goals to restrict a website abilities to load and execute malicious code. Content security policy has goals to make the websites secure by block the misuse of the errors the attacker should not be able of loading inline malicious code without controlling a believe able host. Content security policy is the last line of defense against cross-site scripting. If XSS prevention fails, we can use CSP to mitigate XSS by restricting what an attacker can do.

*Implementing the CSP algorithm:*CSP is the best tool that web developers could utilize to restrict their web applications in many ways to mitigating the dangerous of code injection vulnerabilities such as XSS attack and minimizing the entitled with where their applications will be run[1]. CSP is proposed as the first tool of defense against code injection vulnerabilities. A web developer using CSP to build a trusted web application with safe list allowed resources and their origins. In other words, scripts codes that are wanted to execute the web application are allowed from specific origins path. Otherwise, CSP blocks them as shown in Fig. 1.

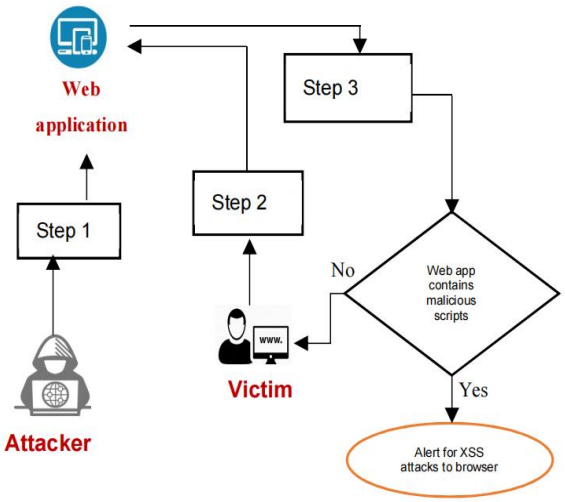


Figure 1. (CSP detection and prevention structure system)

CSP detection and prevention structure system explanations:
Step 1: The attacker injects malicious scripts code into web applications to exploit client-side
Step 2: User request web page to the web server that attacker already injected scripts code
Step 3: When the server response to the users, the page loaded on the user browser and show the report on the browser
Step 4: If CSP detects malicious code Alert for XSS attacks to the browser.

To deploy CSP we include an HTTP response header called Content-Security-Policy-Report-Only.Content-Security-Policy-Report-Only: policy" as an HTTP header to specify our policy. The violation reporting mechanism has been designed to reduce the risk that a malicious website could use violation reports to investigate the performance of other servers and set to the report-uri directive. An example CSP is as follows: default-src 'self'; script-src 'self'; object-src 'none'; frame-src 'none'; base-uri 'none'; For examples the malicious website "https://example.com/foo/bar" allowing as a sourse of example which uses the following policies by disallowing everything from http://evilhackerscripts.com with CSP HTTP header "Content-Security-Policy: default-src 'self'; report-uri csp-hotline.js; csp-reports" as in Fig. 2.

```
$ ./cpp7.sh
*   Trying 127.0.0.1:58080...
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0*
Connected to localhost (127.0.0.1) port 58080 (#0)
> POST / HTTP/1.1
> Host: localhost:58080
> User-Agent: curl/7.81.0
> Accept: */*
> Content-Type: application/csp-report;charset=utf-8
> Content-Length: 248
>
} [248 bytes data]
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< X-Powered-By: Express
< Content-Type: text/html; charset=utf-8
< Content-Length: 431
< ETag: W/"1af-fdakIVtNo63nT3bCcfjjjnX3HJs"
< Date: Sun, 21 Aug 2022 05:40:41 GMT
< Connection: keep-alive
< Keep-Alive: timeout=5
<
{ [431 bytes data]
100   679  100   431  100   248  14350   8257 --:--:-- --:--:-- --:--:-- 23413fu
nction header(name) {
  if (!name) {
    throw new TypeError('name argument is required to req.get');
  }

  if (typeof name !== 'string') {
    throw new TypeError('name must be a string to req.get');
  }

  var lc = name.toLowerCase();

  switch (lc) {
    case 'referer':
    case 'referrer':
      return this.headers.referrer
        || this.headers.referer;
    default:
      return this.headers[lc];
  }
};[object Object]
* Connection #0 to host localhost left intact
```

Figure 2. (CSP test headers and contents)

The violation report may be consisted careful data which is in the loaded web application page. If violation reports fully blocked URL such as session identifiers, IP address as identities. For this objective, the web browser consists only such type of URL that has the real request not the loaded page. This is shown at Fig. 3.

```
mahwi@DESKTOP-S7PRJS3 MINGW64 ~/csp/cspmon
$ node cspmon.js
CSR report server listening on port 58080
mongodb connected
?? A user connected
{
  'csp-report': {
    'document-uri': 'https://example.com/foo/bar',
    referrer: 'https://www.google.com/',
    'violated-directive': 'default-src self',
    'original-policy': 'default-src self; report-uri /csp-hotline.js',
    'blocked-uri': 'http://evilhackerscripts.com'
  }
}
csrpt:[object Object]
document-uri:https://example.com/foo/bar
referrer:https://www.google.com/
violated-directive:default-src self
original-policy:default-src self; report-uri /csp-hotline.js
blocked-uri:http://evilhackerscripts.com
csr_search:[object Object]
mongodb save success
```

Figure 3. (CSP violation report)

E. Content Security Policy HTTP headers

The Content-Security-Policy header allows how browsers can restrict different resources such as JavaScript, CSS, or anything which is the loaded by browser. CSP is
the name of HTTP response header that modern browser uses to enhance the security of web applications, The Content-Security-Policy header has more directives, and all the directives are separated with a semicolon; CSP can be configured in Report-Only mode. In report-only mode, the browser will monitor the policy and report violations but without actually enforcing the restrictions. The owner of web application can receive information through report-uri directive. Although, we can configure a policy by using meta element. <meta http-equiv="Content-Security-Policy" content="default-src 'self'; img-src https://example.com; child-src 'none';">.

F. CSP directives

The Content Security Policy directives are divided into 4 types:

- *Fetch directives*: Content-Security-Policy header use the content-security-policy fetch directives, CSP meta tag or some other html headers. All fetch directives fallback to default-src. That means, if a fetch directive is absent in the CSP header, the user agent (browser) will look for the default-src directive and and take the rules from it.
- *Document Directives:* Content-Security-Policy header use the content-security-policy document directives, or CSP meta tag. Document directives don't fall back to the default-src directive.
- *Navigation directive:* Content-Security-Policy header use the content-security-policy navigation directives, or CSP meta tag, and govern to which location a user can navigate to or submit a form to.
- *Reporting directive:* Content-Security-Policy header use the content-security-policy reporting directives and it also restrict the reporting procedure of CSP violations. Reporting directives don't use default-src directive as a fall back.

G. Content Security Policy level 2

Content security policy is the last line of defense against cross-site scripting. If XSS prevention fails, we can use CSP to mitigate XSS by restricting what an attacker can do. As we can see that in level 2 there are some changes from Level 1, and it also contribute many support many new directives and abilities which are followed below:

1. These summarized changes are here:

1) The result of loaded pages will be ignored.

2) Only a secure data has ability to load in web application and it will be controlled by child-src.

2. The summarized directives are new in level 2.

TABLE II. CSP LEVEL 2 DIRECTIVES.

Directives	Controlled resource type
base-uri	The capability of secure resources is control by this directive.
child-src	child-src instead of script-src is use in CSP level 2.
Form-action	This directive allows the secure resource's capability to submit contents.
SourceFile	SourceFile mostly same like as document-uri.
LineNumber	The number of line where attackers injected the malicious code to hack this website.
ColumnNumber	The number of column where attackers injected the malicious code to hack this website.

H. Content Security Policy level 3

Content Security Policy (CSP) is an implement that can be used by website creators to restrict their web applications in different ways, to mitigating the risk of injection of different vulnerabilities like as cross site scripting(XSS), as well as reduce the freedom with that their applications execute.

I. Feature Policy

The specification of Feature Policy defines a mechanism that allows developers to selectively enable and disable use of various browser features and APIs. Feature Policy is similar to the Content Security Policy but controls feature of browser instead of security behavior. Feature Policy contains directives with indicate the keys and list of sources which limit the use

of features controlled of these directives.Syntax: Feature-Policy: <directive> <allowlist>

J. Permission Policy

Feature Policy is renamed to Permissions Policy. But as of May 2021 no browser supports the Permissions Policy, browsers continue to support Feature Policy. Permissions Policy is also similar to the Content Security Policy but controls features of browser instead of security behavior. Feature-Policy: Feature-Policy: geolocation 'none' Permissions-Policy:Permissions-Policy: geolocation=()

IV. System Test and Reporting Data Analysis

The Content Security Policy(CSP) system tests were performed to show the violation reports, analysis as well as realtime monitoring to check vulnerabilities in different web links using CSP, FP/PP security headers as well as directives and source lists. By using these http security headers with Report-Only we monitor the policy and report violations but without actually enforcing the restrictions.

We test our CSP system as shown in Fig. 4, We use many websites URLs like " https://blog.bluetriangle.om", ttps:// ww.google.com","csp-hotline.js", "https:// vilhackers.om", https://crashtest-security.com", and "https://csplite.com" as input for the testing our CSP system to check whether the Cross-Site Scripting(XSS) attacks can be allowed in these web application or not. Reports will be generated in the same origin because we use directive default-src self.

A. CSP System Test and Reports

Here performed test and get csp-violation report as in Fig. 4.

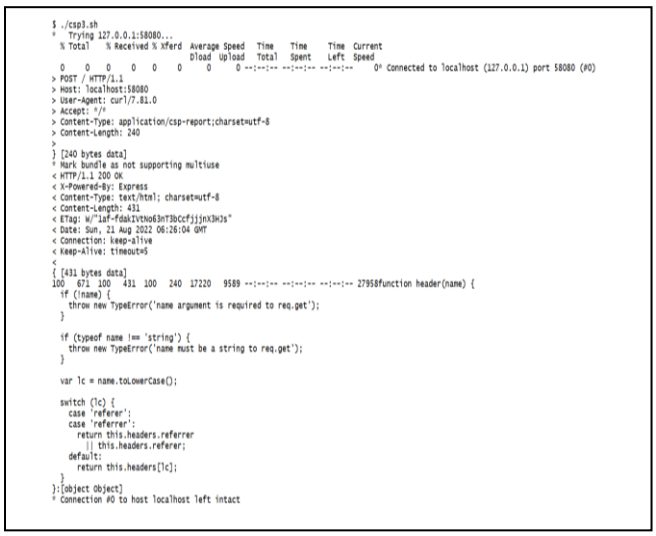


Figure 4. (CSP violation test)

After the implementing of the system we found some reports has been generated by our CSP system. The results are shown in Fig. 5 which shows that CSP violation in "https://blog.bluetriangle.com", "https://crashtest-security.com", and "https://csplite.com" web applications has been occured which is due to injection of Cross-Site-Scripting attack by any attacker as well as our CSP System block some website URLs contents to load in the browser like "https://evilhackers.com". This website contents have been blocked by our system.

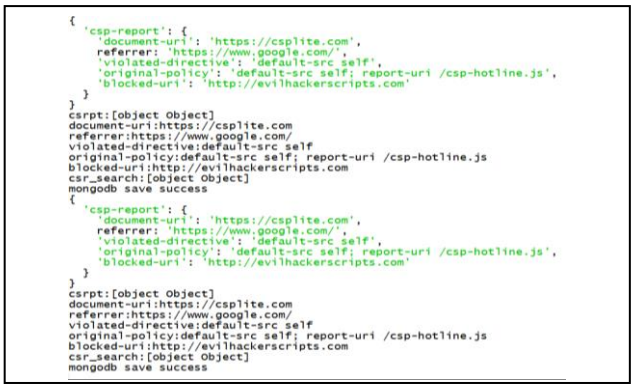


Figure 5. (CSP violation Report)

The report-uri directive used to send violation reports to inform the application's owner of incompatible markup. When report-uri is enabled the browser send a JSON blob whenever the browser detects a violation from the CSP. That JSON blob is the report. In the Fig. 5 is random violation reports from different upper mentioned websites.

B. CSP Results

Fig. 6 shows the results of the web applications tests that how much possibility of XSS injection in these URLs or to test whether this website is vulnerable or not like attackers attacked this web URL for their specific purposes. There are some tests performed to show that how our CSP system violate reports. Because here is attackers inject Cross-Site-Scripting attacks. In this test scenario violation occurred in different URIs which shown in bar graph and Pie chart. The results showed that risk of injection malicious codes in these web URLs between 0 to 120 using different colors for convenient output.

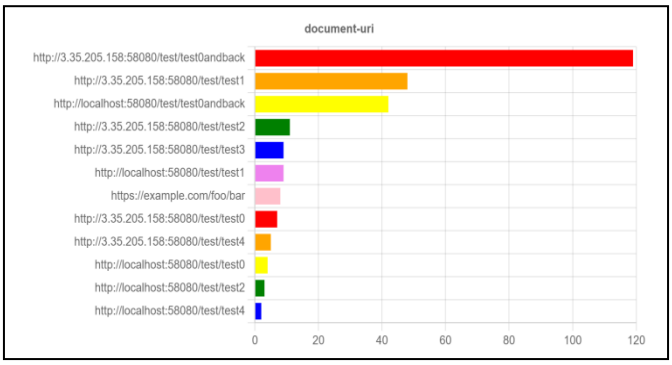


Figure 6. (Document-uri CSP Report)

a) CSP List

CSP List shows that in which documents or web links violation has been occurred or which web URL has been blocked by our CSP system. In the case of CSP Lists we can see that the violation reports are occurred in different web application URLs which we used as a input for the testing our CSP system like "https://google.com", "https://blog. bluetriangle.com", "https://examples.com/foo/bar", "https:// crashtest.com", "https://evilhackerscript.com", "https:// csplite.com", and "https/owasp.org" in the same origin because we use default-src self.

Our system showed the results of CSP violation reports in the table form rather than the JSON form as well as Our System blocked the loading content from different web application links such as "https://example.com/foobar", "https://localhost: 58080/test/test1" etc in these test scenarios. In this list of vulnerable web applications attackers can easily inject any type XSS like stored, reflected, or Dom-based XSS using malicious code like inline Javascript, CSS, any inner HTML tages, images and video etc to damage these websites.

b) CSP Extended Lists

CSP Report Data Summary

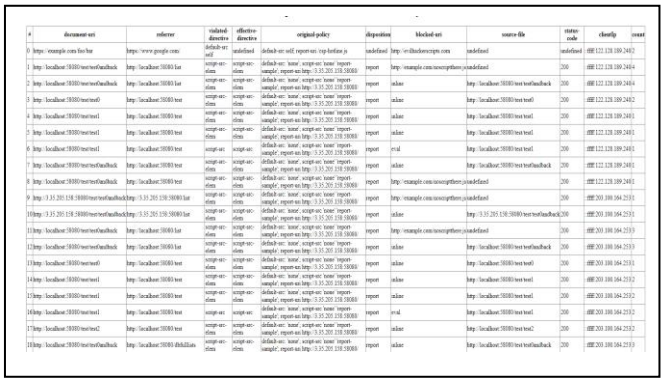
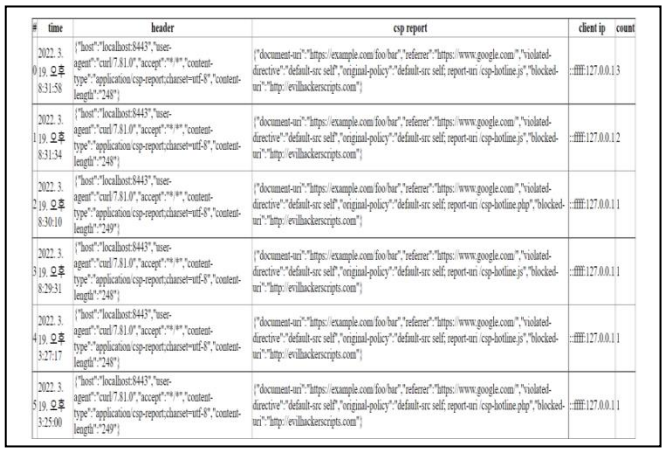


Figure 7. (CSP Extended Lists)

Fig. 7 shows that in which documents or web links violation has been occurred or which web URL has been blocked by our CSP system. In the case of CSP Extended Lists we can see that the violation reports are occurred in "https://examples.com/foo/bar","https://localhost:58080/test/testback","https://localhost:58080/test/test0","https://localhost:58080/test/test1", "https://localhost:58080/test/test2" in the same origin because we use in our system default-src self. As well as Our System block the loading content from different web application links such as "https://evilhackerscripts.com", "https://google.com" etc in these test scenarios. Here we use many directives of Content-Security-policy to control the behaviour of browser.

c) CSP Full List

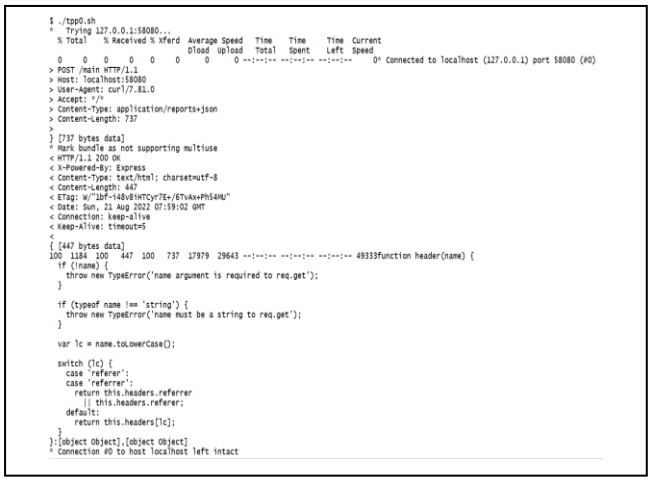


#	time	header	csp report	client ip	count
0	2022. 3. 19. 오후 8:31:58	{"host":"localhost:8443","user-agent":"curl/7.81.0","accept":"*/*","content-type":"application/csp-report;charset=utf-8","content-length":"248"}	{"document-uri":"https://example.com/foo/bar","referrer":"https://www.google.com/","violated-directive":"default-src self","original-policy":"default-src self; report-uri /csp-hotline.js","blocked-uri":"http://evilhackerscripts.com"}	::ffff:127.0.0.1	3
1	2022. 3. 19. 오후 8:31:34	{"host":"localhost:8443","user-agent":"curl/7.81.0","accept":"*/*","content-type":"application/csp-report;charset=utf-8","content-length":"248"}	{"document-uri":"https://example.com/foo/bar","referrer":"https://www.google.com/","violated-directive":"default-src self","original-policy":"default-src self; report-uri /csp-hotline.js","blocked-uri":"http://evilhackerscripts.com"}	::ffff:127.0.0.1	2
2	2022. 3. 19. 오후 8:30:10	{"host":"localhost:8443","user-agent":"curl/7.81.0","accept":"*/*","content-type":"application/csp-report;charset=utf-8","content-length":"249"}	{"document-uri":"https://example.com/foo/bar","referrer":"https://www.google.com/","violated-directive":"default-src self","original-policy":"default-src self; report-uri /csp-hotline.php","blocked-uri":"http://evilhackerscripts.com"}	::ffff:127.0.0.1	1
3	2022. 3. 19. 오후 8:29:31	{"host":"localhost:8443","user-agent":"curl/7.81.0","accept":"*/*","content-type":"application/csp-report;charset=utf-8","content-length":"248"}	{"document-uri":"https://example.com/foo/bar","referrer":"https://www.google.com/","violated-directive":"default-src self","original-policy":"default-src self; report-uri /csp-hotline.js","blocked-uri":"http://evilhackerscripts.com"}	::ffff:127.0.0.1	1
4	2022. 3. 19. 오후 3:27:17	{"host":"localhost:8443","user-agent":"curl/7.81.0","accept":"*/*","content-type":"application/csp-report;charset=utf-8","content-length":"248"}	{"document-uri":"https://example.com/foo/bar","referrer":"https://www.google.com/","violated-directive":"default-src self","original-policy":"default-src self; report-uri /csp-hotline.js","blocked-uri":"http://evilhackerscripts.com"}	::ffff:127.0.0.1	1
5	2022. 3. 19. 오후 3:25:00	{"host":"localhost:8443","user-agent":"curl/7.81.0","accept":"*/*","content-type":"application/csp-report;charset=utf-8","content-length":"249"}	{"document-uri":"https://example.com/foo/bar","referrer":"https://www.google.com/","violated-directive":"default-src self","original-policy":"default-src self; report-uri /csp-hotline.php","blocked-uri":"http://evilhackerscripts.com"}	::ffff:127.0.0.1	1

Figure 8. (CSP Full List)

In Fig. 8, we can see that the full reports which is violated by our CSP system. Here CSP report is occurred in JSON form where document-uri where the CSP violation is occurred because "https://example.com/foo/bar" web application is vulnerable web application. CSP violation is also occurred in the "https://www.google.com/" in the same origin because violated-directive is default-src is self. we set orignal-policy report-uri in our system which is specified by the Content-Security-Policy-HTTP security header as well as our system blocked the resources such as javascript, innerHTML tages, images, videos etc of web application "https://evilhackerscripts.com".

C. FP/ PP Test and Results



```
$ ./tpp0.sh
*   Trying 127.0.0.1:58080...
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0* Connected to localhost (127.0.0.1) port 58080 (#0)
> POST /main HTTP/1.1
> Host: localhost:58080
> User-Agent: curl/7.81.0
> Accept: */*
> Content-Type: application/reports+json
> Content-Length: 737
>
} [737 bytes data]
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< X-Powered-By: Express
< Content-Type: text/html; charset=utf-8
< Content-Length: 447
< ETag: W/"1bf-148v8iHTCyr7E+/6TvAx+Ph54MU"
< Date: Sun, 21 Aug 2022 07:59:02 GMT
< Connection: keep-alive
< Keep-Alive: timeout=5
<
{ [447 bytes data]
100  1184  100   447  100   737  17979  29643 --:--:-- --:--:-- --:--:-- 49333function header(name) {
  if (!name) {
    throw new TypeError('name argument is required to req.get');
  }

  if (typeof name !== 'string') {
    throw new TypeError('name must be a string to req.get');
  }

  var lc = name.toLowerCase();

  switch (lc) {
    case 'referer':
    case 'referrer':
      return this.headers.referrer
        || this.headers.referer;
    default:
      return this.headers[lc];
  }
};[object Object],[object Object]
* Connection #0 to host localhost left intact
```

Figure 9. (FP/PP Test)

Figure 10. (FP/PP Test Report)

Fig. 9 shows that the scenario for the testing of FP/PP policy to generate the violation reports and blocked urls which has some malicious code injected by the attackers. Fig. 10 shows the results of the violation reports of the web application "https://example.com/foo/bar" There are many type of reports are generated like document-policy-violation, Corp and Coep etc which showed that the Attacker injected Cross-site-Request-forgery (CSRF) attacks to hack this website. where document-write is not allowed in this document due to the document-policy violation and it block the url "https://site.example/img.jpg". In this case we can say that in line number 12 attackers injected the malicious code to hack this website.

a) FP/PP Data List

FP/PP Data List shows that in which documents or web links violation has been occurred or which web URL has been blocked by our FP/PP policy using many directives as well as source lists. Our results of this policy shows that in web applications "https://site.example", "https://www.example", "https://site.example/foo/bar","https://csplite.com","https//www.owasp.com","https://www.csp.com","https://www.fp/pp.com" attacker attack the Cross-site-scripting.

b) FP/PP Extended and Full Data List

FP/PP Extended and Full Data List shows that the FP/PP violation reports occurred in different web application which we use as a input for the testing using different types of browsers like Mozilla and chrome etc. FP/PP violation is occurred in the URL " https://site.example", as well as FP/PP block the contents of "https://site2.example/scripts.js.

D. Realtime Monitoring of CSP/FP/ PP

The HTTP Content-Security-Policy-Report-Only response header allows web developers to experiment with policies by monitoring (but not enforcing) their effects. These violation reports consist of JSON documents sent via an HTTP POST request to the specified URI. We get some results by realtime monitoring the violation occurred reports as well as blocked URIs using horizontal bar graph as well as the pie chart bu using different colors for web application links to elaborate in efficient way.

a) Realtime CSP Monitoring Reports

In Fig. 11, we use real-world websites examples to test our system to see whether our system generate the reports against XSS or CSRF attacks. We found that here in website URLs "https://www.google.com/" and "https://blog.bluetriangle.com" CSP violation has been occurred in 4:21:59 and 4:22:50 respectively. Also our system blocked the "https://evilhackerscripts.com" in both reports. In this case we can see that when we test any web application URLs then we found new reports of web application where there are some chances to attacks from attackers. Our CSP system generate realtime reports when system found vulnerabilities in web applications.

Figure 11. (CSP Realtime Monitoring)

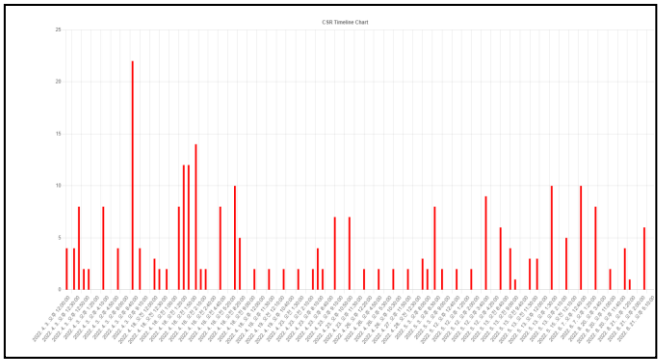


Figure 12. (CSP Realtime Monitoring)

Fig. 12 shows the results when analyzing the violated reports, blocked vulnerable APIs due to the security header restriction according to the number of tests as well as according to the actual time. We can clearly notice that every violated reports is shown with the violation actual time as well as violation

date in the form of vertical bar graph using the same color which is red.

b) Realtime FP/PP Monitoring Reports

We found that here in website URLs https://site.example/ img.jpg, FP/PP violation report has been occurred in 7:39:52 and 7:42:46 respectively as well as our system blocked the "https://evilhackerscripts.com" in both reports. In this case we can see that when we test any web application URLs then we found new reports of web application where there have some chances to attacks from attackers. We can clearly notice that the CSRF attacks or malicious code has been injected in line number 11 in both reports.

Realtime FP/PP Monitoring Reports shows that the analysis between these reports that which web application URLs has allow to attacker to inject malicious code. "http://3.35.205.158", has the high possibility of Cross-site-scripting(XSS) attacks which is 0 to 80 percent while "http://localhost", has low rate of cross-site-scripting(XSS) attack. Here is only 0 to 10 percent chance to inject malicious code from attackers and load contents to web browsers in both bar graph and pie chart.

E. Analysis of Reporting Data with Web Security

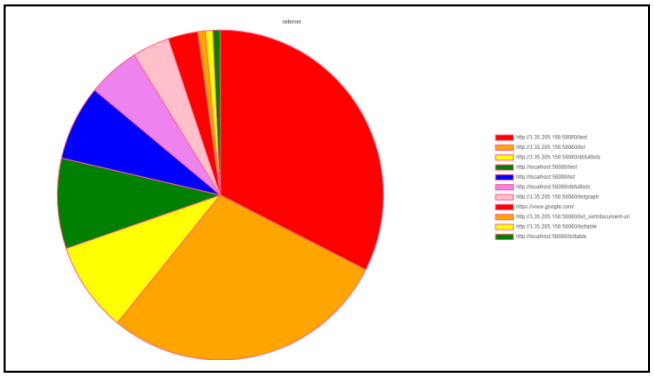


Figure 13. (CSP Analysis of Report Violations)

The author in [8] selected img-src for Images, script-src for JavaScript, XSLT, style-src for Stylesheets (CSS), connect-src for Targets of XMLHttpRequest, default-src for Contents w/o explicit directives. The syntax of CSP allows the inclusion of multiple directives for the same content type (e.g., script-src) in the same header. However, we observed in all the tested browsers a weird, unexpected difference in the treatment of inline scripts between the following two policies:

1. img-src www.example.com;
2. img-src www.example.com; default-src *.

Their experiments revealed that the first policy allows the execution of inline scripts, but the second one does not. On the other hands we use many directives to mitigate the risk of XSS attacks and violate reports. We use document-uri, referrer, violated-directive, orignal-policy, blocked-uri, client-ip as well as count and show all results in the form of Json, tables, and graphs.

CONCLUSION

In this research paper, a CSP system was developed for the generating reports and blocking the contents loading of vulnerable web applications. For generating the violation reports we used many of CSP directives and source lists to allow or blocks the web application. We performed some test to check our CSP system that our system violates the reports or not as well as block the vulnerable websites or not and showed all data in the form of tables. And after tests we make analysis between the results of them and showed the results of web applications in horizontal bar graph and also using Pie Charts. The results showed that the used web applications URLs as an input to check the threats are vulnerable by XSS as well as by CSRF.

We also set the real-time monitoring to show the actual time of violation reports and blocked URLs and saved all data to our database. We also use FP/PP policies to secure web application from the hackers. By implementing CSP approach we got more expected results. Finally, we can say that the Content Security Policy is the last line to mitigate Cross-Site-Scripting (XSS) and defense from attacks and secure the web applications.

ACKNOWLEDGEMENT

This research was supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education (Grants No. NRF-2020R1A6A1A03042742).

REFERENCES

[1] Chen, H.-C., Nshimiyimana, A., Damarjati, C., & Chang, P.-H. (2021). Detection and prevention of cross-site scripting attack with combined approaches. 2021 International Conference on Electronics, Information, and Communication (ICEIC).

[2] Weichselbaum, L., Spagnuolo, M., Lekies, S., & Janc, A. (2016). CSP is dead, long live CSP! on the insecurity of Whitelists and the future of content security policy. Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security.

[3] Hoffman, A. (2020). Web application security: Exploitation and countermeasures for modern web applications. O'Reilly Media.

[4] Muzaki, R. A., Briliyant, O. C., Hasditama, M. A., & Ritchi, H. (2020). Improving security of web-based application using ModSecurity and reverse proxy in web application firewall. 2020 International Workshop on Big Data and Information Security (IWBIS).

[5] Stamm, S., Sterne, B., & Markham, G. (2010). Reining in the web with content security policy. Proceedings of the 19th International Conference on World Wide Web - WWW '10.

[6] Lavrenovs, A., & Melon, F. J. (2018). HTTP security headers analysis of top one million websites. 2018 10th International Conference on Cyber Conflict (CyCon).

[7] Lepofsky, R. (2014). Web application vulnerabilities and countermeasures. The Manager's Guide to Web Application Security: 47-79.

[8] Calzavara, S., Rabitti, A., & Bugliesi, M. (2016). Content security problems? Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security.

[9] Stamm, S., Sterne, B., & Markham, G. (2010). Reining in the web with content security policy. Proceedings of the 19th International Conference on World Wide Web - WWW '10. https://doi.org/10.1145/1772690.1772784

[10] Johari, R., & Sharma, P. (2012). A survey on web application vulnerabilities (SQLIA, XSS) exploitation and security engine for SQL Injection. 2012 International Conference on Communication Systems and Network Technologies.

[11] Sahin, M., & Sogukpinar, I. (2017). An efficient firewall for web applications (EFWA). 2017 International Conference on Computer Science and Engineering (UBMK).

[12] Shrivastava, A., Choudhary, S., & Kumar, A. (2016). XSS vulnerability assessment and prevention in web application. 2016 2nd International Conference on Next Generation Computing Technologies (NGCT).

[13] Yusof, I., & Pathan, A.-S. K. (2014). Preventing persistent cross-site scripting (XSS) attack by applying pattern filtering approach. The 5th International Conference on Information and Communication Technology for The Muslim World (ICT4M).

[14] Babiker, M., Karaarslan, E., & Hoscan, Y. (2018). Web application attack detection and forensics: A survey. 2018 6th International Symposium on Digital Forensic and Security (ISDFS).

[15] Huang, H.-C., Zhang, Z.-K., Cheng, H.-W., & Shieh, S. W. (2017). Web application security: Threats, countermeasures, and Pitfalls. Computer, 50(6), 81-85.

[16] Hadpawat, T., & Vaya, D. (2017). Analysis of prevention of XSS attacks at client side. International Journal of Computer Applications, 173(10), 1-4.

[17] Some, D. F., Bielova, N., & Rezk, T. (2017). On the content security policy violations due to the same-origin policy. Proceedings of the 26th International Conference on World Wide Web.

[18] Patil, K., & Shah, R. (2018). A measurement study of the sub resource integrity mechanism on real-world applications. International Journal of Security and Networks, 13(2), 129.

[19] S.Choudhary, A., & L. Dhore, M. (2012). CIDT: Detection of malicious code injection attacks on web application. International Journal of Computer Applications, 52(2), 19-26.

[20] Mitropoulos, D., Louridas, P., Polychronakis, M., & Keromytis, A. D. (2019). Defending against web application attacks: Approaches, challenges and implications. IEEE Transactions on Dependable and Secure Computing, 16(2), 188-203.

[21] Yadav, D., Gupta, D., Singh, D., Kumar, D., & Sharma, U. (2018). Vulnerabilities and security of web applications. 2018 4th International Conference on Computing Communication and Automation (ICCCA).

[22] Products - content security policy. Report URI. (n.d.). Retrieved May 14, 2022, from https://report-uri.com/products/content_security_policy

[23] OWASP, "OWASP Top Ten," OWASP, 2020. [Online]. Available: https://owasp.org/www-project-top-ten/. [Accessed 31 07 2020].