Global Information Assurance Certification Paper

Content Security Policy in Practice

GCIH

Author: Varghese Palathuruthil, thealmostrealmccoy@gmail.com
Advisor: *Johannes Ullrich*

Accepted: *June, 2018*

Abstract

The implementation of Content Security Policy to leverage web browser capability in protecting a web application from cross-site scripting attack has been a challenge for many legacy web applications. Typical web applications maintained over the years accumulate a number of web pages that do not follow a consistent design. There are no widely available tools to quickly transform legacy web pages to adopt Content Security Policy. The results of this research cover the outcome of implementing a set of tools to address this need.

1. Introduction

Cross-site scripting or XSS continues to rank among the top ten web application vulnerabilities listed by the Open Web Application Security Project. A cross-site scripting attack occurs when an attacker injects a malicious script into a vulnerable page of a web application. The malicious scripts is then executed in the web browser of a different end user, accessing the same vulnerable page of the web application. One response header introduced by web browser vendors to prevent cross-site scripting attack is Content Security Policy. The web application specifies the policy in the response header. The end-user-web-browser enforces the policy. The Content Security Policy is typically listed in the web application response header and can also be applied as a meta-tag in the web pages rendered on the end-user-web-browser. Content Security Policy controls are not exclusive to scripts. The policy can be used to restrict a broad range of resource types including styles, images, and fonts. The Content Security Policy was introduced in September 2016. Since the initial release, Content Security Policy has undergone a few revisions known as levels. Level 3 is the latest working draft of the Content Security Policy.

1.1. Content Security Policy in Theory

The Content Security Policy directive grants additional control to the web application over what locations the end-user-web-browser is permitted to load resources from when loading pages from the web application. The directive uses a whitelist approach. Scripts from any other domain will be blocked. Content Security Policy provides several directives from which to choose so that different media types such as images, objects, frames, fonts, styles, and more could be set to use their own white listed resources. Figure 1 is an example of a Content Security Policy directive, in which all resources from the host web application specifying the Content Security Policy is allowed except for media and object resource types. The end-user-web-browser will block the loading of any audio, video and plugin resources on the rendered pages from any domain including the web application that specifies this Content Security Policy.

Varghese Palathuruthil,
thealmostrealmccoy@gmail.com

```
Content-Security-Policy: default-src 'self'; media-src 'none'; object-src 'none‘
```

Figure 1: Content Security Policy directives

Figure 2 is an example of a Content Security Policy directive, demonstrates how a whitelist can be specified to restrict external resources loaded on the rendered web page of the web application that uses this Content Security Policy. In this case, scripts only from the host web application (‘self') and jquery.com will be permitted to be loaded on the rendered web page.

```
Content-Security-Policy: default-src 'self'; object-src 'none‘; script-src ‘self ‘ http://www.jquery.com
```

Figure 2: Content Security Policy directive with whitelist

Varghese Palathuruthil,
thealmostrealmccoy@gmail.com

Figure 3 is a sample list of directives that could be used to restrict certain types of resources.

Directive	Restricted resource type
connect-src	Restrict resources used for script interfaces.
font-src	Restrict resources used for fonts.
img-src	Restrict resources used for images.
media-src	Restrict resources used for audios and videos.
object-src	Restrict resources used for plugins.
script-src	Restrict resources used for scripts such as JavaScript.
style-src	Restrict resources used for Cascading Style Sheets.
form-action	Restrict domains specified in the action of an html form element.

Figure 3: Sample list of Content Security Policy directives

Figure 4 list the two most commonly used keywords applied to resource-specific directives in a typical Content Security Policy.

Keyword	Restricted resource type
none	Block all resources of this type from being loaded on the rendered page.
self	Restrict loading of resources of this type from only the host web application on the page.

Figure 4: Keywords for Resource-Specific Directives

Varghese Palathuruthil,
thealmostrealmccoy@gmail.com

Content Security Policy provides a reporting feature that is useful for testing the Content Security Policy. When using this feature, policy violations are collected by end-user-web-browser and sent to the configured resource for further analysis. Figure 5 is an example of a Content Security Policy configured to report all violations of the policy.

```
Content-Security-Policy: default-src 'self'; media-src 'none'; object-src 'none‘; report-uri
/csp_test/ContentSecurityPolicyReporter
```

Figure 5: Content Security Policy with reporting option

The value following the keyword report-uri is the URL to which the Content Security Policy-compliant web browser will post violation to the policy. Web browsers post Content Security Policy violations in JSON format. Figure 6 is an example of violation reported by a Content Security Policy-compliant web browser.

```
{"csp-report":
 {"blocked-uri":"self",
  "document-uri":"http://localhost/csp_test/jsp/mouseclick.jsp",
  "original-policy":"report-uri http://localhost/csp_test/ContentSecurityPolicyReporter; default-
src http://localhost; script-src http://localhost
https://cdn.firebase.com/libs/angularfire/0.9.0/angularfire.min.js
https://getbootstrap.com/dist/js/bootstrap.min.js https://code.jquery.com/jquery-1.10.2.js
'nonce-XtiDaiPiuUnFJGV8bcHhnBGzWK5t92LH'",
  "referrer":"http://localhost/csp_test/",
  "script-sample":"onsubmit attribute on DIV element",
  "source-file":"http://localhost/csp_test/jsp/mouseclick.jsp",
  "violated-directive":"script-src"
 }
}
```

Figure 6: Sample Content Security Policy Violation Report

Varghese Palathuruthil,
thealmostrealmccoy@gmail.com

The Content Security Policy-compliant web browser will not block violations to Content Security Policy configured in report-only mode; it will merely report it. Figure 7 is an example of a Content Security Policy in report-only mode:

```
Content-Security-Policy-Report-Only: default-src 'self'; media-src 'none'; object-src 'none‘;
report-uri /csp_test/ContentSecurityPolicyReporter
```

Figure 7: Content Security Policy Report Only Mode

A prominent drawback in level 1 of the Content Security Policy is the way it addresses inline scripts, inline styles, and inline event handlers. Whitelisted directives will not work for inline code on pages. Instead, it will block them from being executed. Figure 8 is an example of a Content Security Policy with a restricted whitelist, blocking inline scripts and styles on the rendered page from being executed in a Content Security Policy-compliant web browser.

Varghese Palathuruthil,
thealmostrealmccoy@gmail.com

```
Content-Security-Policy: default-src 'self'; script-src 'self' https://code.jquery.com/jquery-
1.10.2.js
<html>
<head>
  <title>click demo</title>
  <style>
  p {
    color: red;
    margin: 5px;
    cursor: pointer;
  }
  p:hover {
    background: yellow;
  }
  </style>
  <script src="https://code.jquery.com/jquery-1.10.2.js"></script>
</head>
<body>
<p>First Paragraph</p>
<p>Second Paragraph</p>
<p>Yet one more Paragraph</p>
 <script>
$( "p" ).click(function() {
  $( this ).slideUp();
});
</script>
</body>
</html>
```

Figure 8: Content Security Policy blocking inline scripts

Varghese Palathuruthil,
thealmostrealmccoy@gmail.com

Content Security Policy does provide a directive called unsafe-inline. Adding this directive to the Content Security Policy of a web application will allow inline code, such as scripts, styles, and event handler on the web pages of that particular web application, to be executed. However, this directive does not come with a whitelist option. Using the unsafe-inline keyword will not only allow the execution of legitimate inline scripts, styles, and event handler on the web page but will also permit malicious scripts injected by an attacker into the inline scripts of the web page, thus effectively bypassing the security of Content Security Policy. The solution to this problem is to move all inline scripts, styles, and event handlers on web pages to whitelisted resources such as Javascript or Cascaded Style Sheet files. This is a daunting task for a web application which includes thousands of web pages with thousands of unique inline scripts, styles, and event handler. Level 2 of the Content Security Policy implemented by web browsers addressed this limitation of handling inline scripts by introducing the concept of a nonce. A nonce is a random number or string used once. In level 2 of the Content Security Policy, web applications can set a nonce value as an attribute in the Content Security Policy header response of a web page. The same nonce value is then set as an attribute for all the inline scripts and style tags within that web page. Web browsers that implement level 2 of the Content Security Policy and load such a web page and response header will now allow only those inline scripts and styles with a matching nonce and will block all other injected scripts or styles. Figure 9 is an example of using a unique nonce value in the Content Security Policy directive and the same nonce value in the inline scripts and styles of the rendered page. As a result, the Content Security Policy-compliant web browser will allow the execution of the matching nonce tagged inline scripts and styles in the web browser.

Varghese Palathuruthil,
thealmostrealmccoy@gmail.com

```
Content-Security-Policy: default-src 'self'; script-src 'self' https://code.jquery.com/jquery-1.10.2.js 'nonce-XtiDaiPiuUnFJGV8bcHhnBGzWK5t92LH'

<html>
<head>
  <title>click demo</title>
  <style nonce=”XtiDaiPiuUnFJGV8bcHhnBGzWK5t92LH”>
  p {
    color: red;
    margin: 5px;
    cursor: pointer;
  }
  p:hover {
    background: yellow;
  }
  </style>
  <script src="https://code.jquery.com/jquery-1.10.2.js"></script>
</head>
<body>
<p>First Paragraph</p>
<p>Second Paragraph</p>
<p>Yet one more Paragraph</p>
 <script nonce=”XtiDaiPiuUnFJGV8bcHhnBGzWK5t92LH”>
$( "p" ).click(function() {
  $( this ).slideUp();
});
</script>
</body>
</html>
```

Figure 9: Content Security Policy directive with nonce

Varghese Palathuruthil,
thealmostrealmccoy@gmail.com

This approach addresses the limitation of Content Security Policy on inline scripts to a certain extent. However, level 2 of the Content Security Policy did not address all the issues. The nonce attribute cannot apply to inline event handlers of a web page. As a result, script injection on the inline event handler of a web page can still bypass level 2 of the Content Security Policy. Level 3 of the Content Security Policy proposes a new attribute called unsafe-hashed-attributes to address the limitation of the impact of Content Security Policy on event handlers. Similar to the use of a nonce in inline scripts and styles, level 3 of the Content Security Policy proposes to use matching hash values as a part of the unsafe-hashed-attributes in the response header and the inline event handler of the web page. Note that this proposal is still a work in progress. Level 3 of the Content Security Policy also introduces an additional expression called strict-dynamic for the script directive. Web browsers implementing level 3 of the Content Security Policy will propagate the trust given to a script using a nonce to all the other scripts loaded by this script.

1.2. Content Security Policy in Practice

The features implemented and proposed in the Content Security Policy is useful in restricting resources loaded by an end-user-web-browser and addressing issues such as cross-site scripting in a web application that integrates Content Security Policy. The problem lies in the feasibility of its implementation in a typical web application. Implementation of Content Security Policy is straightforward on a new web application built from scratch with Content Security Policy in mind. Implementation of Content Security Policy is also less daunting on an existing web application that already follows good practices such as externalizing all scripts and styles code in external files. However, this is not the case for a typical web application, built and maintained over many years, by many different teams that have supported legacy features, while adding new features. Such web applications tend to have hundreds of web pages with custom inline scripts, styles, and event handlers. How do you transform these web pages quickly to benefit from Content Security Policy features? How do you generate and propagate matching nonce and hash values across all the inline script and style codes of such a web application? Does a typical web application maintain an inventory of all external

resources referenced by its web pages so that it could be used to build a successful whitelist for its Content Security Policy?

2. Tools to Build an Effective Content Security Policy

A set of tools is needed to apply a potent Content Security Policy to a legacy web application. The tools work together to cover different aspects of Content Security Policy. The Java programming language was chosen for this research to implement tools to apply Content Security Policy to a test web application. The Java Enterprise Edition or Java EE is the de facto standard in building web applications in Java. A favored implementation of the Java EE servlet container is Apache Tomcat. The web application developed to test these tools follows the Java EE standards and deploys on Apache Tomcat. A Java Servlet Filter was implemented to apply the configured Content Security Policy to the test web application. As the name implies, a Java Servlet Filter can be constructed to map URL patterns of the web application to ensure that these URL requests are screened using the rules implemented in the Java Servlet Filter. Java Servlet Filter is a powerful mechanism that controls the response headers of web pages rendered in a Java EE web application. As a result, the Java Servlet Filter was used to apply the Content Security Policy to the response header of the rendered web pages. Java Servlet Filters like their Java Servlet counterparts have access to the session scope of a Java EE web application. It provides the advantage of generating and maintaining nonce values unique to each session in the test web application. The pages implemented in the test web application use two other standards from the Java EE stack, namely Java Server Pages and Java Server Pages Standard Tag Libraries. Java Server Pages, or JSP, dynamically generates web pages to aids in applying custom rules on the rendered html content that finally displays on the end-user-web-browser. Java Server Pages Standard Tag Libraries, or JSP tags, are used to pack small snippets of Java code that define as tags for reusability. Using JSP tags in conjunction with Servlet filter enables the application of dynamic nonce to inline scripts and styles in the rendered html content that displays in the end-user-web-browser. The rendered html content on the test web pages include references to well-known style and script libraries such as Bootstrap, Angular, and JQuery. Additionally, there are two Java

Varghese Palathuruthil,
thealmostrealmccoy@gmail.com

Servlets implemented for the test web application with vulnerability to cross-site scripting attacks. Apache Ant is a favored tool used to compile and build Java applications. Ant already comes with a wide range of built-in tasks to scan the/an application workspace to prepare and package it into a deployable or executable application. Like typical open source applications, the Ant tool design is highly extensible to build new custom tasks. This feature has been leveraged to develop Ant tasks that will scan an application workspace to compile a list of external styles, scripts, images, and referenced frame resources. This feature also uses this list to prepare a Content Security Policy that could be applied to the application.

2.1. Tool to list external resources referenced by web pages

ContentSecurityPolicyWhiteListCollector is an Ant task written in Java that will scan the target web application codebase and compile the list of resources referenced by web pages. The output will be an xml file (named csp_external_resource_list.xml) that categorizes the list by resource media type such as scripts, frame, link, anchor, image, e.t.c. Figure 10 outlines the details listed in the output xml file.

Name	Description
resourceTypeDetail	The element name
Name	The resource type name sub-element. Examples of value that will be listed here include script, image, anchor, etc.
Pattern	The pattern sub-element to find this resource. The value is character data (CDATA) An example of a value that will be listed here would be <![CDATA[<script.*</script>]]>
fileDetail	The location of the code base sub-element where this resource is listed. The value is character data (CDATA) An example of a value that will be listed here would be <![CDATA[C:/tmp/csptest/src/main/webapp/WEB-INF/jsp/welcome.jsp]]>
Value	The retrieved resource sub-element. The value is character data (CDATA) An example of a value that will be listed here would be <![CDATA[https://oss.maxcdn.com/libs/respond.js/1.4.2/respond.min.js]]>

Figure 10: Elements in the generated xml file of the ContentSecurityPolicyWhiteListCollector Ant task.

Figure 11 is an example of an xml file generated by running the ContentSecurityPolicyWhiteListCollector Ant task.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<resourceType>
<resourceTypeDetail>
<name>script</name>
<pattern><![CDATA[<script.*</script>]]></pattern>
<fileDetail><![CDATA[C:/tmp/csptest/src/main/webapp/WEB-INF/jsp/welcome.jsp]]></fileDetail>
<value><![CDATA[https://oss.maxcdn.com/libs/respond.js/1.4.2/respond.min.js]]></value>
</resourceTypeDetail>
<resourceTypeDetail>
<name>anchor</name>
<pattern><![CDATA[<a.*</a>]]></pattern>
<fileDetail><![CDATA[C:/tmp/csptest/src/main/webapp/WEB-INF/jsp/example.jsp]]></fileDetail>
<value><![CDATA[http://get.adobe.com/reader/]]></value>
</resourceTypeDetail>
<resourceTypeDetail>
<name>image</name>
<pattern><![CDATA[<img.*>]]></pattern>
<fileDetail><![CDATA[C:/tmp/csptest/src/main/webapp/WEB-INF/jsp/xss.jsp]]></fileDetail>
<value><![CDATA[<c:url value='/images/icon_alert.gif'></c:url>]]></value>
</resourceTypeDetail>
<resourceTypeDetail>
<name>link</name>
<pattern><![CDATA[<link.*>]]></pattern>
<fileDetail><![CDATA[C:/tmp/csptest/src/main/webapp/WEB-INF/jsp/xss.jsp]]></fileDetail>
<value><![CDATA[favicon.ico]]></value>
</resourceTypeDetail>
<resourceTypeDetail>
<name>link</name>
<pattern><![CDATA[<link.*>]]></pattern>
<fileDetail><![CDATA[C:/tmp/csptest/src/main/webapp/WEB-INF/jsp/xss.jsp]]></fileDetail>
<value><![CDATA[plugins/datepicker/css/datepicker3.css]]></value>
</resourceTypeDetail>
</resourceType>
```

Figure 11: Sample xml output generated by the ContentSecurityPolicyWhiteListCollector Ant task

Figure 12 is an example of running the ContentSecurityPolicyWhiteListCollector Ant task.

Varghese Palathuruthil,
thealmostrealmccoy@gmail.com

```
ant -f csp-tools.xml content_security_policy_white_list_collector
-Dprojectroot=C:/tmp/csp-test
```

Figure 12: Example of running the ContentSecurityPolicyWhiteListCollector Ant task.

2.2. Tool to generate an initial Content Security Policy

The ContentSecurityPolicyGenerator is an Ant task written in Java that will take the xml output from the CSPWhiteListCollector tool to create an initial Content Security Policy for the target web application. The policy will include whitelist. It will not contain nonce as this value will be generated to be unique per user session. The output will be in an xml file. Figure 13 lists the details in the output xml file.

Varghese Palathuruthil,
thealmostrealmccoy@gmail.com

Name	Description
policyHeader	The element holding recurrent features that do not fall under a specific resource type
Key	The sub-element of the policyHeader element indicating whether the policy is live or in report only mode.
Prefix	The sub-element of the policyHeader element indicating the frequent directives applied across all resource types.
Nonce	The sub-element of the policyHeader element indicating whether the tool that feeds in this document to administer the policy should generate a nonce.
policyDetail	The element holding policy specific to a resource type
Type	The sub-element of the policyDetail element indicating the resource type
Policy	The sub-element of the policyDetail element indicating the whitelist policy that will be applied to a specific resource type.

Figure 13: Details in the generated xml file of the ContentSecurityPolicyGenerator Ant task.

Figure 14 is an example of an xml file generated by running the ContentSecurityPolicyGenerator Ant task.

Varghese Palathuruthil,
thealmostrealmccoy@gmail.com

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<contentSecurityPolicy>
<policyHeader>
<key><![CDATA[Content-Security-Policy]]></key>
<prefix><![CDATA[report-uri /csp_test/ContentSecurityPolicyViolationReporter; default-src
'self']]></prefix>
<nonce>true</nonce>
</policyHeader>
<policyDetail>
<type><![CDATA[script-src]]></type>
<policy><![CDATA[ 'strict-dynamic'
https://cdn.firebase.com/libs/angularfire/0.9.0/angularfire.min.js
https://getbootstrap.com/dist/js/bootstrap.min.js https://code.jquery.com/jquery-1.10.2.js
https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular-resource.min.js
https://code.jquery.com/jquery-3.2.1.slim.min.js https://code.jquery.com/ui/1.12.1/jquery-ui.js
https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular-route.min.js
https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js
https://code.jquery.com/jquery-1.12.4.js https://cdn.firebase.com/js/client/2.0.4/firebase.js
https://getbootstrap.com/assets/js/vendor/holder.min.js
https://getbootstrap.com/assets/js/vendor/popper.min.js
https://getbootstrap.com/assets/js/vendor/jquery-slim.min.js]]></policy>
</policyDetail>
<policyDetail>
<type><![CDATA[style-src]]></type>
<policy><![CDATA[ 'strict-dynamic' https://jqueryui.com/resources/demos/style.css
https://getbootstrap.com/dist/css/bootstrap.min.css
https://code.jquery.com/ui/1.12.1/themes/base/jquery-ui.css]]></policy>
</policyDetail>
</contentSecurityPolicy>
```

Figure 14: Sample xml output generated by the ContentSecurityPolicyGenerator Ant task

Varghese Palathuruthil,
thealmostrealmccoy@gmail.com

Figure 15 is an example of running the ContentSecurityPolicyGenerator Ant task.

```
ant -f csp-tools.xml content_security_policy_generator
-DresourceListLocation=C:/tmp/csp-test/csp_external_resource_list.xml
-Dnonce=true
-DreportUri=/csp_test/ContentSecurityPolicyViolationReporter
-DreportOnly=false
```

Figure 15: Example of running the ContentSecurityPolicyGenerator Ant task.

2.3. Tool to apply Content Security Policy to a response header

The ContentSecurityPolicyFilter is a Java Servlet Filter that will use the xml output from the ContentSecurityPolicyGenerator tool and apply the prepared Content Security Policy from it to the response header of all page URLs mapped to this Java Servlet Filter in the web application. The output from the ContentSecurityPolicyGenerator tool is an xml file. The location of this xml file is a configured initialization parameter of the ContentSecurityPolicyFilter Java Servlet Filter. The content of the xml file is loaded and parsed during the Java Servlet Filter initialization phase typically when the Java EE servlet container deploys the web application or when the container restarts. The parsed content of the xml file is maintained by the Java Servlet Filter for later analysis during the doFilter method invocation. doFilter is a critical method used in a Java Servlet Filter implementation. This method is invoked when the Java Servlet Filter screens request to URLs mapped to this Java Servlet Filter in the web application configuration file. The doFilter method in the ContentSecurityPolicyFilter will analyze the parsed content of the xml file. If the policy has been configured to generate a nonce, then it will examine the web application session scope for such a nonce value. If none exist, it will produce a new nonce value and store it in the web application session scope as an attribute. The Filter updates the parsed Content Security Policy from the xml file with the nonce value and adds the updated content as a header to the response of the rendered page served by the requested URL. Figure 16 is an

example of ContentSecurityPolicyFilter configured in the web application configuration file of a typical Java EE web application.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns=http://xmlns.jcp.org/xml/ns/javaee>
   <description>Series of CSP Tests</description>
   <display-name>CSP Test</display-name>
…
   <filter>
<filter-name>ContentSecurityPolicyFilter</filter-name>
  <filter-class>csptest.filter.ContentSecurityPolicyFilter</filter-class>
  <init-param>
    <param-name>policyLocation</param-name>
    <param-value>/WEB-INF/content_security_policy.xml</param-value>
   </init-param>
  </filter>
   <filter-mapping>
    <filter-name>ContentSecurityPolicyFilter</filter-name>
                <url-pattern>*.jsp</url-pattern>
   </filter-mapping>
   <filter-mapping>
    <filter-name>ContentSecurityPolicyFilter</filter-name>
                <url-pattern>/servlet/*</url-pattern>
   </filter-mapping>
…
</web-app>
```

Figure 16: Sample ContentSecurityPolicyFilter configuration in web.xml

2.4. Tool to synchronize inline scripts and styles with the nonce from Content Security Policy

The NonceTag is an implementation of the Java Server Pages Standard Tag Library. It can only be applied to the contents of Java Server Pages. Pages with the

NonceTag that are requested by end-user-web-browser will scan the enclosed content on the page and update matches for regular expressions such as "<script.*?>", "<style.*?>", or "<link.*?>" with the nonce value retrieved from the web application session scope. Figure 17 is an example of the NonceTag declaration in a typical tag library descriptor file.

```
<taglib>
  <tlib-version>1.0</tlib-version>
  <jsp-version>1.2</jsp-version>
  <short-name>CSPTags</short-name>
...
  <tag>
    <name>nonce</name>
    <tag-class>csptest.tag.NonceTag</tag-class>
    <description>Add nonce attribute to enclosed script and style
tags</description>
  </tag>
</taglib>
```

Figure 17: Sample NonceTag declaration in tag library descriptor file.

Figure 18 is an example of the use of the NonceTag on one of the Java Server Pages in the test web application.

Varghese Palathuruthil,
thealmostrealmccoy@gmail.com

```
<!doctype html>
<%@ taglib prefix="csp" uri="/WEB-INF/taglib/csp.tld" %>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>click demo</title>
  <csp:nonce>
  <style>
  p {
    color: red;
    margin: 5px;
    cursor: pointer;
  }
  p:hover {
    background: yellow;
  }
  </style>
  <script src="https://code.jquery.com/jquery-1.10.2.js"></script>
</head>
<body>
<p>First Paragraph</p>
<p>Second Paragraph</p>
<p>Yet one more Paragraph</p>
<script>
$( "p" ).click(function() {
  $( this ).slideUp();
});
</script>
  </csp:nonce>
</body>
</html>
```

Figure 18: Example usage of NonceTag in a Java Server Page

Varghese Palathuruthil,
thealmostrealmccoy@gmail.com

The resultant output rendered on the end-user-web-browser is listed in Figure 19 with a comparison to the output generated without the NonceTag.

Varghese Palathuruthil,
thealmostrealmccoy@gmail.com

Without NonceTag	With NonceTag

!doctype html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>click demo</title>
<style>
 	
p {
color: red;
margin: 5px;
cursor: pointer;
}
p:hover {
background: yellow;
}
</style>
<script src="https://code.jquery.com/jquery-1.10.2.js"></script>
 	
</head>
<body>
 First Paragraph
 Second Paragraph
 Yet one more Paragraph
<script>
$("p").click(function() {
$(this).slideUp();
});
</script>
</body>
</html>
	!doctype html>
	<html lang="en">
	<head>
	<meta charset="utf-8">
	<title>click demo</title>
	<style nonce="bEypmVNJ4vNWYj45EVJcXaVZNQHWtTif">
	p {
	color: red;
	margin: 5px;
	cursor: pointer;
	}
	p:hover {
	background: yellow;
	}
	</style>
	<script src="https://code.jquery.com/jquery-1.10.2.js" nonce="bEypmVNJ4vNWYj45EVJcXaVZNQHWtTif"></script>
	</head>
	<body>
	 First Paragraph
	 Second Paragraph
	 Yet one more Paragraph
	<script nonce="bEypmVNJ4vNWYj45EVJcXaVZNQHWtTif">
	$("p").click(function() {
	$(this).slideUp();
	});
	</script>
	</body>
	</html>

Figure 19: Comparison of generated output with and without NonceTag usage in a Java Server Page.

Varghese Palathuruthil,
thealmostrealmccoy@gmail.com

2.5. Content Security Policy Violation Reporting

ContentSecurityPolicyViolationReporter is a Java Servlet that accepts the feed of Content Security Policy violations reported by end-user-web-browsers. The URL configured to represent this Java Servlet is specified as the report-uri value in the Content Security Policy identified in the requested page response header. This Java Servlet accepts the policy violation in JSON format and writes it to the application log. Figure 20 is an example of configuring ContentSecurityPolicyViolationReporter Java Servlet in the Java EE web application and Figure 21 is an example of specifying ContentSecurityPolicyViolationReporter Java Servlet in the Content Security Policy directive to report violations.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app>
    <description>Series of CSP Tests</description>
   <display-name>CSP Test</display-name>
...
   <servlet>
      <servlet-name>ContentSecurityPolicyViolationReporter</servlet-name>
      <servlet-class>csptest.report.ContentSecurityPolicyViolationReporter</servlet-
class>
   </servlet>
   <servlet-mapping>
      <servlet-name>ContentSecurityPolicyViolationReporter</servlet-name>
      <url-pattern>/ContentSecurityPolicyViolationReporter</url-pattern>
   </servlet-mapping>
...
</web-app>
```

Figure 20: Sample configuration of ContentSecurityPolicyViolationReporter Java Servlet in web.xml

Varghese Palathuruthil,
thealmostrealmccoy@gmail.com

```
Content-Security-Policy: report-uri
http://localhost/csp_test/ContentSecurityPolicyViolationReporter;
default-src self;
script-src self
https://cdn.firebase.com/libs/angularfire/0.9.0/angularfire.min.js
https://getbootstrap.com/dist/js/bootstrap.min.js
https://code.jquery.com/jquery-1.10.2.js
'nonce-bEypmVNJ4vNWYj45EVJcXaVZNQHWtTif';
style-src self
https://jqueryui.com/resources/demos/style.css
https://getbootstrap.com/dist/css/bootstrap.min.css
https://code.jquery.com/ui/1.12.1/themes/base/jquery-ui.css
'nonce-bEypmVNJ4vNWYj45EVJcXaVZNQHWtTif'"
```

Figure 21: Example of configuring ContentSecurityPolicyViolationReporter Java Servlet in a Content Security Policy directive.

3. Findings

The impact of Content Security Policy on the Java EE web application was tested using a Content Security Policy-compliant web browser such as the latest version of Firefox. The debugging tool bundled with the Firefox web browser provided information about Content Security Policy violation. The Content Security Policy violations reported by the configured Java Servlet in the web application logs were reviewed. The findings from the tests are detailed below.

3.1. Impact on XSS vulnerable Java Servlets

The Content Security Policy violations reported by the configured Java Servlet in the web application logs proved that the Content Security Policy specified in the response header of the XSS vulnerable Java Servlets was effective in suppressing the loading of malicious input payload in a Content Security Policy-compliant web browser. Figure 22

is a snippet from the web application logs reporting the Content Security Policy violations.

```
2018-04-28 07:11:45,224 DEBUG [http-nio-10148-exec-9]
[csptest.report.ContentSecurityPolicyViolationReporter]
{"csp-report":
{"blocked-uri":"self",
"document-uri":"http://localhost/csp_test/servlet/ReflectedXSS?
userInput=%3Cscript%3Ealert(%27Hello%20World!%27)%3C/script%3E",
"line-number":8,
"original-policy":"report-uri
http://localhost/csp_test/ContentSecurityPolicyViolationReporter;
default-src http://localhost;
script-src http://localhost
https://cdn.firebase.com/libs/angularfire/0.9.0/angularfire.min.js
https://getbootstrap.com/dist/js/bootstrap.min.js https://code.jquery.com/jquery-1.10.2.js
'nonce-WTSD5pvnzxCVKv8MrzKvtC626fQjVtnt';
style-src http://localhost https://jqueryui.com/resources/demos/style.css
https://getbootstrap.com/dist/css/bootstrap.min.css
'nonce-WTSD5pvnzxCVKv8MrzKvtC626fQjVtnt'",
"referrer":"http://localhost/csp_test/",
"script-sample":"alert('Hello World!')",
"source-file":"http://localhost/csp_test/servlet/ReflectedXSS?
userInput=%3Cscript%3Ealert(%27Hello%20World!%27)%3C/script%3E",
"violated-directive":"script-src"}}
```

Figure 22: Content Security Policy violation reported for XSS vulnerable Java Servlets.

3.2. Impact on external resources referenced

The tests proved that external resources such as third-party JavaScript libraries and Cascading Style Sheets correctly whitelisted in the Content Security Policy specified in the response header loads without any restrictions by a Content Security Policy-

compliant web browser. As a result, these test pages rendered correctly on the web browser.

3.3. Impact of nonce on inline scripts

The test proved that inline scripts and styles with nonce value attributes that match the nonce value specified in the Content Security Policy in the response header of the rendered page were correctly whitelisted and were allowed to load and execute in a Content Security Policy- compliant web browser. The same results were seen when the 'strict-dynamic' attribute was included in the Content Security Policy and used in conjunction with the nonce attribute. The generation and usage of a nonce in Content Security Policy for this research was session-based. The ContentSecurityPolicyFilter inspected the session scope of the Java EE test web application and generated new nonce value if none existed for this user session. The generated nonce persist during the lifetime of that particular session. As a result, nonce value specified in the Content Security Policy in the requested page response header are unique per user session. This uniqueness of nonce value per user session maintains the random nature of the nonce value, improving the defense of the nonce value against exploits. The randomness of nonce value was tested using two separate instances of Content Security Policy-compliant web browsers requesting the same page from the test web application. The test resulted in the creation of two distinct sessions by the web application, and two unique nonce values returned in the response header as part of the Content Security Policy. Figure 23 is a comparison of nonce values generated by two separate user sessions to the test web application.

Varghese Palathuruthil,
thealmostrealmccoy@gmail.com

```
<html>
<head>
...
<link rel="stylesheet"
href="https://code.jquery.com/ui/1.12.1/themes
/base/jquery-ui.css"
nonce="2sxB2ahLG9CYbfv7HavyUuemMP6B
opjB">
  <script src="https://code.jquery.com/jquery-
1.12.4.js"
nonce="2sxB2ahLG9CYbfv7HavyUuemMP6B
opjB"></script>
  <script
src="https://code.jquery.com/ui/1.12.1/jquery-
ui.js"
nonce="2sxB2ahLG9CYbfv7HavyUuemMP6B
opjB"></script>
  <script
nonce="2sxB2ahLG9CYbfv7HavyUuemMP6B
opjB">
  $( function() {
...
  } );
  </script>
</head>
<body>
<div class="ui-widget">
...
</div>
…
</body>
</html>
```

```
<html>
<head>
...
<link rel="stylesheet"
href="https://code.jquery.com/ui/1.12.1/theme
s/base/jquery-ui.css"
nonce="zfWJot8LcgQRBtLQcKLBaoeKg9sET
Hyr">
  <script src="https://code.jquery.com/jquery-
1.12.4.js"
nonce="zfWJot8LcgQRBtLQcKLBaoeKg9sET
Hyr"></script>
  <script
src="https://code.jquery.com/ui/1.12.1/jquery-
ui.js"
nonce="zfWJot8LcgQRBtLQcKLBaoeKg9sET
Hyr"></script>
  <script
nonce="zfWJot8LcgQRBtLQcKLBaoeKg9sET
Hyr">
  $( function() {
...
  } );
  </script>
</head>
<body>
<div class="ui-widget">
...
</div>
…
</body>
</html>
```

Figure 23: Example of two separate nonce values generated in two separate user sessions

Varghese Palathuruthil,
thealmostrealmccoy@gmail.com

3.4. Impact on inline event handlers

The test proved that whitelisting and the application of nonce could not be applied to inline event handlers on the rendered web pages. As a result, the Content Security Policy specified in the response header of the web page ensured that a Content Security Policy-compliant web browser blocked inline event handlers from loading or executing on the rendered web page. Figure 24 is a snippet from the web application log reporting Content Security Policy violation against inline event handler on the web page.

```
2018-04-28 07:31:17,599 DEBUG [http-nio-10148-exec-5]
[csptest.report.ContentSecurityPolicyViolationReporter]
{"csp-report":
{"blocked-uri":"self",
"document-uri":"http://localhost/csp_test/jsp/autocomplete.jsp",
"original-policy":
"report-uri http://localhost/csp_test/ContentSecurityPolicyViolationReporter;
default-src http://localhost;
script-src 'strict-dynamic'
https://cdn.firebase.com/libs/angularfire/0.9.0/angularfire.min.js
https://getbootstrap.com/dist/js/bootstrap.min.js
https://code.jquery.com/jquery-1.10.2.js
'nonce-dDJdk8EZmAyQrzftCUWdxj5NWuZGnnhQ';
style-src https://jqueryui.com/resources/demos/style.css
https://getbootstrap.com/dist/css/bootstrap.min.css
'nonce-dDJdk8EZmAyQrzftCUWdxj5NWuZGnnhQ'",
"referrer":"http://localhost/csp_test/",
"script-sample":"onfocusin attribute on DIV element",
"source-file":"http://localhost/csp_test/jsp/autocomplete.jsp",
"violated-directive":"script-src"}}
```

Figure 24: Example of Content Security Policy violation reported against inline event handler on a web page.

This limitation may be addressed when the proposed unsafe-hashed-attributes for inline event handlers in level 3 of the Content Security Policy are widely adopted by web browsers. In the meantime, integration with the current Content Security Policy would require code refactoring. Figure 25 is an example of using HTML 5 data attribute as a workaround to inline event handler on the web page to comply the web page with the Content Security Policy.

```
<html>
<body>
<a id="foo" data-param1="xyz" data-param2="abc" data-param3="efg"
href="javascript:void(0)">Test</a>
<script nonce="2sxB2ahLG9CYbfv7HavyUuemMP6BopjB">
$(document).ready(function() {
$('#foo').bind('click', function() {
  alert($(this).attr('data-param1') + $(this).attr('data-param2') + $(this).attr('data-
param3'));
});
});
</script>
</body>
</html
```

Figure 25: Example of workaround to inline event handlers on web page to comply with Content Security Policy.

3.5. Processing embedded JSP Tags

In a typical Java Server Page of a Java EE web application, tag libraries are often used to specify resource location. It has been a challenge for the ContentSecurityPolicyWhiteListCollector tool to compile the list of resources referenced when the resources are specified using Java Server Pages Tag Libraries. Figure 26 is an example of tag library usage in a Java Server Page to specify a resource and Figure 27 is the corresponding example of the rendered content on the end-user-web-browser. When the ContentSecurityPolicyWhiteListCollector tool scans a Java Server Page with a tag

library specified in Figure 26, it is difficult for the ContentSecurityPolicyWhiteListCollector tool to predict what the generated value for the resource URL would be on the end-user-web-browser.

```
<script src='<c:url value="/js/myscripts.js"></c:url>'></script>
```

Figure 26: Example of JSP tag library used to specify a resource location.

```
<script src='http://localhost/js/myscripts.js'></script>
```

Figure 27: Example of rendered content of a JSP tag on the end-user-web-browser

4. Recommendations for Future Research

The tools developed for this research have room for improvement The Content Security Policy is a standard that has been evolving over the last few levels. For instance, directives such as frame-src have been deprecated in level 2 of the Content Security Policy while new directives such as form-action, and child-src are added. Similarly, the Content Security Policy Reporting directive report-uri will be replaced with report-to in a future version of the policy. These changes to the specification and corresponding web browser support needs to be kept in mind when building and maintaining tools to apply Content Security Policy to a web application. One of the most critical enhancement needed to Content Security Policy is whitelisting inline event handlers. Level 3 of the Content Security Policy proposed the unsafe-hashed-attributes to whitelist event handlers in a similar manner that the nonce attribute did to whitelist inline scripts and styles. If and when web browsers adopt and support the unsafe-hashed-attributes for Content Security Policy, tools developed to implement Content Security Policy to a web application need to follow suit. The collection of Content Security Policy violations developed as part of this research were rudimentary. JSON data reported by end-user-web-browser writes to the application logs. This feature could be enhanced and integrated with an external database so that the indexed reports are available for further analysis.

Varghese Palathuruthil,
thealmostrealmccoy@gmail.com

5. Conclusion

Content Security Policy is a step in the right direction to address issues such as cross-site scripting. The directives specified in the Content Security Policy could use the compiled list of resources referenced in a web application as a whitelist to block unauthorized references. Unique nonce value per session could be used to allow sanitized inline script execution. However, Content Security Policy is an evolving policy with a lot of challenges. Tools such as the ones covered in this paper will continue to be developed and eventually gain traction as this policy matures.

Varghese Palathuruthil,
thealmostrealmccoy@gmail.com

References

Klein, A. (2002, June). Cross Site Scripting Explained. Retrieved from https://crypto.stanford.edu/cs155/papers/CSS.pdf

OWASP Top 10 - 2017. Retrieved from https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf

OWASP Top 10 - 2013. Retrieved from https://storage.googleapis.com/google-code-archive-downloads/v2/code.google.com/owasptop10/OWASP%20Top%2010%20-%202013.pdf

OWASP Top 10 - 2010. Retrieved from https://storage.googleapis.com/google-code-archive-downloads/v2/code.google.com/owasptop10/OWASP%20Top%2010%20-%202010.pdf

Barth, A., Sterne, B. (2015, February). Content Security Policy 1.0. Retrieved from https://www.w3.org/TR/CSP1/

West, M., Barth, A., Veditz, D. (2016, December 15). Content Security Policy Level 2. Retrieved from https://www.w3.org/TR/CSP2/

West, M. (2016, September 13). Content Security Policy Level 3. Retrieved from https://www.w3.org/TR/CSP3/

Weichselbaum, L., Spagnuolo, M., Lekies, S. (2016, October 16). CSP Is Dead, Long Live CSP! On the Insecurity of Whitelists and the Future of Content Security Policy. Retrieved from https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/45542.pdf

Kotowicz, K., Gross, S., Vela Neva, E., A., Johns, M., Lekies, S. (2017, October 30). Code-Reuse Attacks for the Web: Breaking Cross-Site Scripting Mitigations via Script Gadgets. Retrieved from https://acmccs.github.io/papers/p1709-lekiesA.pdf

Patil, K., Braun, F. (2015, May 5). A Measurement Study of the Content Security Policy on Real-World Applications. Retrieved from http://ijns.femto.com.tw/contents/ijns-v18-n2/ijns-2016-v18-n2-p383-392.pdf

Varghese Palathuruthil,
thealmostrealmccoy@gmail.com

Doupé, A., Weidong, C., Jakubowski, M., H., Peinado, M., Kruegel, C., Vigna, G. (2013, November). deDacota: Toward Preventing Server-Side XSS via Automatic Code and Data Separation. Retrieved from http://sefcom.asu.edu/publications/dedacota-ccs2013.pdf

Patil, K., Vyas, T., Braun, F., Goodwin, M., Liang, Z. (2013, July). Poster: UserCSP-User Specified Content Security Policies. Retrieved from https://cups.cs.cmu.edu/soups/2013/posters/soups13_posters-final1.pdf

Java Platform, Enterprise Edition (Java EE). Retrieved from http://www.oracle.com/technetwork/java/javaee/overview/index.html

JavaServer Pages Technology. Retrieved from http://www.oracle.com/technetwork/java/javaee/jsp/index.html

JavaServer Pages Standard Tag Library. Retrieved from http://www.oracle.com/technetwork/java/index-jsp-135995.html

Apache Ant. Retrieved from https://ant.apache.org/

Java Servlet Filters. Retrieved from http://www.oracle.com/technetwork/java/filters-137243.html

OWASP Content Security Policy Cheat Sheet. Retrieved from https://www.owasp.org/index.php/Content_Security_Policy_Cheat_Sheet

JavaScript Object Notation. Retrieved from https://www.json.org

Apache Tomcat. Retrieved from http://tomcat.apache.org/

JQuery. Retrieved from https://jquery.com/

Bootstrap. Retrieved from https://getbootstrap.com/

Angular. Retrieved from https://angular.io/

HTML data-* Attributes. Retrieved from https://www.w3schools.com/tags/att_global_data.asp

CSP: script-src. Retrieved from https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Security-Policy/script-src

CSP: report-uri. Retrieved from https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Security-Policy/report-uri

Tools developed for this research. https://github.com/thealmostrealmccoy123/csp-test

Varghese Palathuruthil,
thealmostrealmccoy@gmail.com