**Cross Site Scripting Protection Framework**

**1. Purpose**

The purpose of this document is to provide a comprehensive documentation of the Cross Site Scripting protection framework that was designed and built in house at 18F. Key end to end features will be discussed to include java classes, database tables, tag libraries as well as key components and how these parts work together to achieve the goal of protecting 18F web applications.

**2. Scope**

The scope of this document is restricted to describing how the Cross Site Scripting protection framework is designed and how it is able to achieve its intended goal of protecting 18F websites.

**3. Strategy:**

3.1 Requirement:

There is an implicit and obvious service level agreement (SLA) that requires online ecommerce websites to be secure. One way of achieving this goal is by enabling the FD4 website to be protected from Cross Site Scripting attacks.

This framework was designed to clean specific parameters for specific views and commands. There were a number of options that were considered. These included the following:

1. WCS OOTB Cross Site Scripting protection via wc-server.xml
2. Solution that would clean every parameter by default.
3. Clean only specific parameters for specific View/Commands

The decision to take the 3rd approach was made because the option to do a broad based cleaning of all parameters would be unsuitable, problematic and more than likely would introduce defects. Also there are specific cases where malicious characters may actually be allowed such as the Gift Message box in the order checkout process.

There was also the requirement/need to be able to

1. Dynamically block specific parameters for specific View/Commands
2. Database driven configurations
3. Different options for cleaning
4. Allow exceptions

3.2 OWASP:

The Open Web Application Security Project (OWASP) is an organization that provides various open source software to help make software more secure. Among the suite of software and software libraries built and maintained is the ESAPI library.

https://www.owasp.org

3.3 ESAPI:

ESAPI (The OWASP Enterprise Security API) is a free, open source, web application security control library that makes it easier for programmers to write lower-risk applications.

https://www.owasp.org/index.php/Category:OWASP\_Enterprise\_Security\_API

3.4 CONFIGURATION:

The jar file/library is esapi-2.1.0.jar (version 2.1.0) and needs to be placed in the lib folder on the deployment appserver. In order for the ESAPI library to function there are two important dependencies: ESAPI.properties and validation.properties.

When the ESAPI class files are being loaded into the VM during server startup, the API hunts for the properties files in the following locations in sequence:

1. Specific location as configured as a file
2. Specific location as configured as a resource
3. /opt/IBM/WebSphere/AppServer/profiles/MBPDev01/<FILE>.properties
4. SystemResource Directory/resourceDirectory: .esapi/<FILE>.properties
5. 'user.home' (/home/was) directory
6. file I/O
7. Classpath

The esapi-2.1.0.jar file was modified to include thee ESAPI.properties and the validation.properties files. This was motivated by fact that these files are 1) not normally modified and 2) ease of maintenance. These files are located in the folder /esapi inside the archive file. Please note that if the jar file is downloaded again or if there is an upgrade of the jar file, then these two properties files will need to be either placed inside the new jar file, or placed in one of the aforementioned locations where the API will be looking for it.

A typical URL would look like this:

<https://www.1800flowers-dev.net/webapp/wcs/stores/servlet/LogonForm?catalogId=13302&langId=-1&storeId=20051&URL=%2Ffields-of-europe-90977%3FcategoryId%3D400096025&krypto=0D%2FBoAAJDvEda9Rc2aTm3g%3D%3D&ddkey=http:LogonForm>

View/Command = LogonForm

Sample Parameter = ddkey

Sample ParameterValue = http:LogonForm

Cleaner Class = CUSCleanerImpl.java

If the requirement is to clean the ddkey parameter, then the approach would be to configure the Framework to clean the ddkey parameter when the LogonForm View/Command is invoked with a specific cleaner (CUSCleanerImpl.java).

3.5 Types of Cleaners:

There are two basic ways to mitigate cross site scripting:

1. Escaping malicious characters
2. Removing malicious characters

The ESAPI approach has predefined industry standards that designate certain characters as malicious. This creates complexity when there are exceptions such as the example used above where the ‘:’ is actually allowed. To resolve this, a place holder is used to temporarily replace the character and then after the parameter is escaped using the ESAPI cleaner, the temporary character is then placed back in the string. This is the case with the Custom Cleaner (CUSCleanerImpl.java)

The other approach is the EVIL\_CHAR\_LIST cleaner approach. This is basically the technique of identifying and defining characters that are deemed malicious and removing them from the parameter. With this approach there are a number of advantages:

1. 1800Flowers.com can define what is deemed malicious.
2. The malicious characters (EVIL\_CHAR\_LIST) can be maintained dynamically in the registry (XSTOREATTR)
3. The framework is architecture so that character sequences can be blocked if necessary. E.g. the string ‘logicbomb’ can be cleaned from a parameter if needed.
4. There also may be special cases where there may be exceptions to the rule. These can also be user defined (EVIL\_CHAR\_EXCEPT\_LIST).

|  |  |  |
| --- | --- | --- |
| List | Values | Comments |
| EVIL\_CHAR\_LIST | ),(,&,;,:,>,<,+,\n,%0D,\,%0A,',\*,",%22,%26,%3B,%3b,%3A,%3a,%3E,%3e,%3C,%3c,%2B,%2b,%5C,%5c,%27,%0d,%0a | The actual evil character and their equivalent escape values are maintained. |
| EVIL\_CHAR\_EXCEPT\_LIST | ',-,&,%27,%26 |  |
| XSS\_SECURITY\_FLAG | Y/N (Yes/No) | Toggle the feature on and off dynamically. If value is missing or invalid the default is Y or will default to clean parameters. |

As seen above there is a list that defines character that are sometimes allowed under specific conditions. These exceptions are maintained in the EVIL\_CHAR\_EXCEPT\_LIST in the XSTOREATTR table.

a) package com.fd.commerce.security.xss

Classes:

|  |  |  |
| --- | --- | --- |
| Class Name | Purpose | Comments |
| XSSCleanerFacade.java | General interface between the Controller type code such as FDBaseAction and HttpServletRequest classes. | Convenient collection of methods to handle cross site scripting functionality. |
| AbstractEvilCharCleanerImpl.java | Parent class for all classes that perform evil character cleaning. Abstract and requires extending class to define String cleanEvilChars(String value) method. | Implements CleanerInterface directly. |
| CUSCleanerImpl.java | Cleans a url via character escaping(ESAPI). There is hard coded list of characters that are not encoded for url per business needs such as ‘:’ in the ddkey parameter. These character are would otherwise be encoded. | Implements CleanerInterface directly.  Exceptions=’:,@,=,&’ |
| ECCleanerImpl.java | Uses EVIL\_CHAR cleaning strategy, realizes String cleanEvilChars(String value) method. This removes characters defined as being malicious. | Extends AbstractEvilCharCleanerImpl abstract class. |
| ECXCleanerImpl.java | Uses EVIL\_CHAR cleaning strategy, realizes String cleanEvilChars(String value) method. Uses special FORM TYPE exception list. Used to clean form fields parameters. | Extends AbstractEvilCharCleanerImpl abstract class. URL Exception\_List= ‘',-,&,%27,%26’ |
| ESAPIEncoder.java | This wrapper class uses Singleton design pattern to access the ESAPI.encoder() method. Returns a Encoder class. | Encoder class has methods that performs various type of character encoding/escaping. |
| ESAPIValidator.java | This wrapper class uses Singleton design pattern to access the ESAPI.validator() method. Returns a Validator class. | Validator class has methods that can validate a string to see if it contains malicious code. |
| HTMLCleanerImpl.java | Cleans html characters via character escaping (ESAPI). | Implements CleanerInterface directly. |
| JSCleanerImpl.java | Cleans javascript characters via character escaping (ESAPI). | Implements CleanerInterface directly. |
| NOCleanerImpl.java | Does not perform any parameter cleaning. | Implements CleanerInterface directly. |
| URLCleanerImpl.java | Cleans a url via character escaping(ESAPI). | Implements CleanerInterface directly. |
| URLParamCleanerImpl.java | Uses EVIL\_CHAR cleaning strategy, realizes String cleanEvilChars(String value) method. Uses special URL TYPE exception list. Used to clean URL type values being passed as parameters. | Extends AbstractEvilCharCleanerImpl abstract class. URL Exception\_List= ’&,/,?,=,:,%26,%2F,%2f,  %3F,%3f,%3D,%3d,%3A,%3a’ |
| FDFetchXSSDetailsCmd | Defines Interface WCS command class. Uses DynaCache command cacheing. | Extends CacheableCommand. Requires configuring the cachespec.xml file to include this command. |
| FDFetchXSSDetailsCmdImpl | Responsible for fetching data from the XSSSECURITYCONFIG table quickly and efficiently. Places the configuration information in a HashMap to be used by the cleaner framework. This class uses reflection to construct and store the java class in the cleaner configuration HashMap. | Extends CacheableCommand and implements FDFetchXSSDetailsCmd. |
| FDMBPConstants | Added XSS\_SECURE\_FLAG constant to define the attribute name in XSTOREATTR table. | package com.fd.commerce.mbp.helpers; |
| CleanParamTag | Extends BodyTagSupport. The doAfterBody() method calls the JSCleanerImpl clean method on the body contents of the tag. This escapes/encodes the contents for javascript using ESAPI . | package com.fd.commerce.taglib;    packaged inside the fd.tld tag library. |

Interface:

|  |  |  |
| --- | --- | --- |
| Class Name | Purpose | Comments |
| CleanerInterface.java | This defines an interface that will be implemented so that a common method can be defined for handling parameter cleaning in XSSCleanerFacde. | Defines a  String clean(String url) method that has to be implemented. |
|  |  |  |

b) package com.fd.commerce.security.xss.beans

Classes:

|  |  |  |
| --- | --- | --- |
| Class Name | Purpose | Comments |
| XSSConfigBean.java | POJO bean class that is used to maintain data retrieved from the XSSSECURITYCONFIG table. |  |
|  |  |  |

3.6 Key Touch Points:

XSSCleanerFacade :

This Façade class contains various methods that accomplish the task of cleaning the parameters being used by the web application. Can be configured for specific stores but the default store\_id used is 0.

There is complexity as a result of the fact that HttpServletRequest parameters are immutable so they cannot be modified programmatically. However WebSphere Commerce has a map of the parameters (TypedProperty which can be retrieved from the Command/View context via context.getRequestProperties()). Belongs to the com.fd.commerce.security.xss. package.

|  |  |  |
| --- | --- | --- |
| Method Name | Purpose and Logic | Comments |
| String clean(String key, String value) | This is generally called by the code that is trying to clean a parameter.  A key is constructed that follows the internal format <View/Command>@@<parameter>  The key will then be used to get the appropriate CleanerInterface implementation class. | Once the appropriate cleaner class is defined then the cleaner.clean(value) method is called and the value is returned. |
| String[] clean(String key, String[] value) | Cleans a parameter that is a String array type. This is done by iteratively calling the clean(String key, value) method defined above. |  |
| CleanerInterface getCleaner(String key) | Checks configuration HashMap for the defined cleaner class for the requested View/Command and Parameter combination. If there is no cleaner class found then NOCleanerImpl class is returned that basically does not clean the parameter. |  |
| Map checkRequestParamMap(  FDHttpServletRequest req, Map map) | Inspects the request url to determine the view/command name and then uses it to iteratively clean all the values that are present in the TypedProperty map. |  |

FDBaseAction :

The preprocess method was modified to gets the TypedProperty map from context and pass this map along with the Request object to the XSS Cleaner Façade which cleans all the parameters in the map and returns the cleaned values. This cleaned map is then placed back in the context which essentially replaces the map that could potentially contain parameters having malicious characters. Belongs to the com.fd.commerce.struts package.

FDHttpServletRequest :

This class extends the HttpServletRequestWrapper class.

FDHttpServletRequest(HttpServletRequest request) constructor was defined so that the original url that was invoked can be maintained as the getRequestURL() method sometimes loses information while processing the request.

|  |  |  |
| --- | --- | --- |
| Method Name | Purpose and Logic | Comments |
| String getParameter(String name) | Gets the value of a specified parameter name. Overrides default getParameter() method to include logic that cleans the parameter before it is returned to the caller method. |  |
| String[] getParameterValues(String name) | Gets the String array of values for a specified parameter name. Overrides default getParameterValues() method to include logic that cleans the parameter values before it is returned to the caller method. |  |
| Map getParameterMap() | Gets the Map of values for the current request. Overrides default getParameterMap() method to include logic that cleans the parameter map before it is returned to the caller method. |  |
| Object getAttribute(String name) | Gets the value of a specified attribute name. Overrides default getAttribute() method to include logic that cleans the attribute before it is returned to the caller method. It is designed to only clean the attribute if it is a string i.e.  if (obj instanceof String) | Due to performance issues identified with this method, the feature was commented out. |

4. Oracle Tables:

XSTOREATTR

|  |  |  |  |
| --- | --- | --- | --- |
| Attribute | TYPE | Purpose | Comments |
| XSS\_SECURE\_FLAG | VARCHAR2(254) | Toggle feature on/off |  |
| EVIL\_CHAR\_LIST | VARCHAR2(254) | Defines the list of evil characters. |  |
| EVIL\_CHAR\_EXCEPT\_LIST | VARCHAR2(254) | Defines exceptions to the list of evil characters. | Only used in specific scenarios. |
|  |  |  |  |

XSSSECURITYCONFIG

|  |  |  |  |
| --- | --- | --- | --- |
| Column | Type | Purpose | Comments |
| XSSSECURITYCONFIG\_ID | NUMBER | Unique row identifier |  |
| STORE\_ID | NUMBER | Store Id |  |
| PATH | VARCHAR2(100) | View/Command name |  |
| PARAM\_NAME | VARCHAR2(100) | Parameter name |  |
| CLEANER\_CLASS | VARCHAR2(500) | Name of the cleaner class | Instantiated via reflection. Requires the full qualified class name. |
|  |  |  |  |

5. Miscellaneous

5.1 Reflection:

Map<String, CleanerInterface> map = new HashMap<String, CleanerInterface>();

CleanerInterface cleaner = null;

Class theClass = null;

theClass = Class.forName(CleanerClassName);

cleaner = (CleanerInterface)theClass.newInstance();

The cleaner class created is then placed in the cleanerMap which is subsequently used by the XSS cleaner framework. Please refer to the com.fd.commerce.security.commands. FDFetchXSSDetailsCmdImpl. convertConfigDetailsToMap() method.

5.2 Special Cases:

WCS renders the javascript contents of the web pages before calling FDHttpServletRequest or FDBaseAction classes. This result in a situation where the javascript blocks within the html page will contain malicious code. To solve this, a custom tag was created to clean the applicable pages where this situation occurs.

5.3 SEO Friendly Pages:

These pages require the use of the CleanParamTag tag library solution; such as TopCategoryDisplay.jsp. The CleanParamTag tag belongs to the fd.tld tag library.

A sample page to consider is: [http://www.1800flowers-dev.net/cakesandcookies?cm\_sp='whscheck](http://www.1800flowers-dev.net/cakesandcookies?cm_sp=%27whscheck)

6. Appendix A:

XSSSECURITYCONFIG table values (ww42) (3/13/2015)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| XSSSECURITYCONFIG\_ID | STORE\_ID | PATH | PARAM\_NAME | CLEANER\_CLASS |
| 1000 | 0 | Search | searchKeywords | com.fd.commerce.security.xss.ECCleanerImpl |
| 1001 | 0 | JoinPassportHomePageView | fromPage | com.fd.commerce.security.xss.ECCleanerImpl |
| 1002 | 0 | ReLogonFormView | ddkey | com.fd.commerce.security.xss.CUSCleanerImpl |
| 1003 | 0 | registerPrimePassportsUserUrl | orderId | com.fd.commerce.security.xss.ECCleanerImpl |
| 1004 | 0 | AnalyticsPageView | category | com.fd.commerce.security.xss.URLCleanerImpl |
| 1005 | 0 | AnalyticsPageView | pagename | com.fd.commerce.security.xss.URLCleanerImpl |
| 1006 | 0 | AnalyticsPageView | ddkey | com.fd.commerce.security.xss.CUSCleanerImpl |
| 1007 | 0 | Logon | ddkey | com.fd.commerce.security.xss.CUSCleanerImpl |
| 1008 | 0 | Logon | ddkeyVal | com.fd.commerce.security.xss.CUSCleanerImpl |
| 1009 | 0 | Logon | URL | com.fd.commerce.security.xss.URLParamCleanerImpl |
| 1010 | 0 | Logon | SEOURL | com.fd.commerce.security.xss.URLParamCleanerImpl |
| 1011 | 0 | LogonForm | ddkey | com.fd.commerce.security.xss.CUSCleanerImpl |
| 1012 | 0 | LogonForm | SEOURL | com.fd.commerce.security.xss.URLParamCleanerImpl |
| 1013 | 0 | LogonForm | ddkeyVal | com.fd.commerce.security.xss.CUSCleanerImpl |
| 1014 | 0 | AjaxLogonForm | ddkeyVal | com.fd.commerce.security.xss.CUSCleanerImpl |
| 1015 | 0 | AjaxUserRegistrationForm | ddkey | com.fd.commerce.security.xss.CUSCleanerImpl |
| 1016 | 0 | CatalogRequestView | ddkey | com.fd.commerce.security.xss.CUSCleanerImpl |
| 1017 | 0 | CatalogRequestView | address3 | com.fd.commerce.security.xss.URLCleanerImpl |
| 1018 | 0 | UserRegistrationAdd | URL | com.fd.commerce.security.xss.URLParamCleanerImpl |
| 1019 | 0 | AjaxLogonForm | ddkey | com.fd.commerce.security.xss.CUSCleanerImpl |
| 1020 | 0 | EmailSignUpThankYouView | ddkey | com.fd.commerce.security.xss.CUSCleanerImpl |
| 1021 | 0 | EmailSignUpThankYouView | emailAddress | com.fd.commerce.security.xss.ECCleanerImpl |
| 1022 | 0 | EmailSignUpThankYouView | email | com.fd.commerce.security.xss.ECCleanerImpl |
| 1023 | 0 | EmailOptIn | emailAddress | com.fd.commerce.security.xss.ECCleanerImpl |
| 1024 | 0 | EmailOptIn | email | com.fd.commerce.security.xss.ECCleanerImpl |
| 1025 | 0 | AjaxUserRegistrationForm | ddkeyVal | com.fd.commerce.security.xss.ECCleanerImpl |
| 1026 | 0 | SubmitCatalogRequest | firstName | com.fd.commerce.security.xss.ECXCleanerImpl |
| 1027 | 0 | SubmitCatalogRequest | lastName | com.fd.commerce.security.xss.ECXCleanerImpl |
| 1028 | 0 | SubmitCatalogRequest | address1 | com.fd.commerce.security.xss.ECXCleanerImpl |
| 1029 | 0 | SubmitCatalogRequest | address2 | com.fd.commerce.security.xss.ECXCleanerImpl |
| 1030 | 0 | SubmitCatalogRequest | address3 | com.fd.commerce.security.xss.ECXCleanerImpl |
| 1031 | 0 | SubmitCatalogRequest | city | com.fd.commerce.security.xss.ECXCleanerImpl |
| 1032 | 0 | SubmitCatalogRequest | state | com.fd.commerce.security.xss.ECXCleanerImpl |
| 1033 | 0 | SubmitCatalogRequest | country | com.fd.commerce.security.xss.ECXCleanerImpl |
| 1034 | 0 | SubmitCatalogRequest | zip | com.fd.commerce.security.xss.ECXCleanerImpl |
| 1035 | 0 | CatalogRequestView | zip | com.fd.commerce.security.xss.ECXCleanerImpl |
| 1036 | 0 | LogonForm | URL | com.fd.commerce.security.xss.URLParamCleanerImpl |
|  |  |  |  |  |