KnowageLabs / **Knowage-Server** Public

Code | Issues 4 | Pull requests 28 | Actions | Security 1 | Insights

b079a654c1

Knowage-Server / knowageutils / src / main / java / it / eng / spagobi / utilities / filters /

XSSRequestWrapper.java / Jump to

n3ils Fixed a typo.

History

3 contributors

453 lines (353 sloc) | 16.8 KB

```
package it.eng.spagobi.utilities.filters;

import java.io.IOException;
import java.net.MalformedURLException;
import java.net.URL;
import java.util.Iterator;
import java.util.List;
import java.util.regex.Matcher;

import java.util.regex.Pattern;

```

```java
import javax.servlet.ServletInputStream;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletRequestWrapper;

import org.apache.commons.validator.UrlValidator;
import org.apache.log4j.Logger;

import it.eng.spagobi.utilities.whitelist.WhiteList;

public class XSSRequestWrapper extends HttpServletRequestWrapper {

        private static transient Logger logger = Logger.getLogger(XSSRequestWrapper.class);
        private static WhiteList whitelist = WhiteList.INSTANCE;

        public XSSRequestWrapper(HttpServletRequest servletRequest) {
                super(servletRequest);
        }

        @Override
        public String[] getParameterValues(String parameter) {
                String[] values = super.getParameterValues(parameter);

                if (values == null) {
                        return null;
                }

                int count = values.length;
                String[] encodedValues = new String[count];
                for (int i = 0; i < count; i++) {
                        encodedValues[i] = stripXSS(values[i]);
                }

                return encodedValues;
        }

        @Override
        public String getParameter(String parameter) {
                String value = super.getParameter(parameter);

                return stripXSS(value);
        }

        @Override
        public String getHeader(String name) {
                String value = super.getHeader(name);
                return stripXSS(value);
        }

        @Override
```

```java
    public ServletInputStream getInputStream() throws IOException {

        return super.getInputStream();
    }

    public static String stripXSS(String value) {
        logger.debug("IN");
        String initialValue = value;

        if (value != null) {
            // NOTE: It's highly recommended to use the ESAPI library and uncomment th
            // avoid encoded attacks.
            // value = ESAPI.encoder().canonicalize(value);

            // Avoid null characters
            value = value.replaceAll("", "");

            // Avoid anything between script tags
            Pattern scriptPattern = Pattern.compile("<script>(.*?)</script>", Pattern.
            value = scriptPattern.matcher(value).replaceAll("");

            scriptPattern = Pattern.compile("&lt;script&gt;(.*?)&lt;/script&gt;", Patt
            value = scriptPattern.matcher(value).replaceAll("");

            // Avoid anything in a src='...' type of expression
            // Pattern.compile("src[\r\n]*=[\r\n]*\\\'(.*?)\\\'", Pattern.CASE_INSENSI
            // value = scriptPattern.matcher(value).replaceAll("");
            //
            // scriptPattern = Pattern.compile("src[\r\n]*=[\r\n]*\\\"(.*?)\\\"", Patt
            // value = scriptPattern.matcher(value).replaceAll("");

            value = checkImgTags(value);
            value = checkIframeTags(value);
            value = checkAnchorTags(value);
            value = checkVideoTags(value);
            value = checkCSS(value);

            // Remove any lonesome </script> tag
            scriptPattern = Pattern.compile("</script>", Pattern.CASE_INSENSITIVE);
            value = scriptPattern.matcher(value).replaceAll("");

            scriptPattern = Pattern.compile("&lt;/script&gt;", Pattern.CASE_INSENSITIV
            value = scriptPattern.matcher(value).replaceAll("");

            // Remove any lonesome <script ...> tag
            scriptPattern = Pattern.compile("<script(.*?)>", Pattern.CASE_INSENSITIVE
            value = scriptPattern.matcher(value).replaceAll("");

            scriptPattern = Pattern.compile("&lt;script(.*?)&gt;", Pattern.CASE_INSENS
            value = scriptPattern.matcher(value).replaceAll("");

            // Avoid eval(...) expressions
            scriptPattern = Pattern.compile("eval\\((.*?)\\)", Pattern.CASE_INSENSITIV
            value = scriptPattern.matcher(value).replaceAll("");

            // Avoid expression(...) expressions
            scriptPattern = Pattern.compile("expression\\((.*?)\\)", Pattern.CASE_INSE
            value = scriptPattern.matcher(value).replaceAll("");

            // Avoid javascript:... expressions
            scriptPattern = Pattern.compile("javascript:", Pattern.CASE_INSENSITIVE);
            value = scriptPattern.matcher(value).replaceAll("");

            // Avoid vbscript:... expressions
            scriptPattern = Pattern.compile("vbscript:", Pattern.CASE_INSENSITIVE);
            value = scriptPattern.matcher(value).replaceAll("");

            // Avoid onload= expressions
            scriptPattern = Pattern.compile("onload(.*?)=", Pattern.CASE_INSENSITIVE |
            value = scriptPattern.matcher(value).replaceAll("");

            // Avoid onClick= expressions
            scriptPattern = Pattern.compile("onClick(.*?)=", Pattern.CASE_INSENSITIVE
            value = scriptPattern.matcher(value).replaceAll("");

            // Avoid anything between form tags
            Pattern formPattern = Pattern.compile("<form(.*?)</form>", Pattern.CASE_IN
            value = formPattern.matcher(value).replaceAll("");

            // Avoid anything between a tags
            // Pattern aPattern = Pattern.compile("<a(.*?)</a>", Pattern.CASE_INSENSIT
            // value = aPattern.matcher(value).replaceAll("");

            // aPattern = Pattern.compile("<a(.*?/)>", Pattern.CASE_INSENSITIVE | Patt
            // value = aPattern.matcher(value).replaceAll("");

            Pattern aPattern = Pattern.compile("&lt;a(.*?)&lt;/a&gt;", Pattern.CASE_IN
            value = aPattern.matcher(value).replaceAll("");

            // Avoid anything between button tags
            Pattern buttonPattern = Pattern.compile("<button(.*?)</button>", Pattern.C
            value = buttonPattern.matcher(value).replaceAll("");

            buttonPattern = Pattern.compile("<button(.*?/)>", Pattern.CASE_INSENSITIVE
            value = buttonPattern.matcher(value).replaceAll("");

            buttonPattern = Pattern.compile("&lt;button(.*?)&lt;/button&gt;", Pattern.
            value = buttonPattern.matcher(value).replaceAll("");
```

```java

                        // Example value ="<object data=\"javascript:alert('XSS')\"></object>"
                        // Avoid anything between script tags
                        Pattern objectPattern = Pattern.compile("<object(.*?)</object>", Pattern.C
                        value = objectPattern.matcher(value).replaceAll("");

                        objectPattern = Pattern.compile("&lt;object(.*?)&lt;/object&gt;", Pattern.
                        value = objectPattern.matcher(value).replaceAll("");

                        // Remove any lonesome </object> tag
                        objectPattern = Pattern.compile("</object>", Pattern.CASE_INSENSITIVE);
                        value = objectPattern.matcher(value).replaceAll("");

                        objectPattern = Pattern.compile("&lt;/object&gt;", Pattern.CASE_INSENSITIV
                        value = objectPattern.matcher(value).replaceAll("");

                        // Remove any lonesome <object ...> tag
                        objectPattern = Pattern.compile("<object(.*?/)>", Pattern.CASE_INSENSITIVE
                        value = objectPattern.matcher(value).replaceAll("");

                        objectPattern = Pattern.compile("&lt;object(.*?/)&gt;", Pattern.CASE_INSEN
                        value = objectPattern.matcher(value).replaceAll("");

                        if (!value.equalsIgnoreCase(initialValue)) {
                                logger.warn("Message: detected a web attack through injection");
                        }

                }

                logger.debug("OUT");
                return value;
        }

        private static String checkImgTags(String value) {
                logger.debug("IN");
                Pattern maliciousImgPattern = Pattern.compile("&lt;img(.*?)&gt;", Pattern.CASE_INS
                value = maliciousImgPattern.matcher(value).replaceAll("");

                Pattern scriptPattern = Pattern.compile("<img[^>]+(src\\s*=\\s*['\"]([^'\"]+)['\"]
                                Pattern.CASE_INSENSITIVE | Pattern.MULTILINE | Pattern.DOTALL);
                Pattern dataPattern = Pattern.compile("data:image\\/(gif|jpeg|pjpeg|png|svg\\+xml|
                                Pattern.CASE_INSENSITIVE | Pattern.MULTILINE | Pattern.DOTALL);
                Matcher scriptMatcher = scriptPattern.matcher(value);

                while (scriptMatcher.find()) {
                        String img = scriptMatcher.group();
                        String link = scriptMatcher.group(2);

                        Matcher dataMatcher = dataPattern.matcher(link);

                if (!dataMatcher.find()) {
                    try {
                        URL url = new URL(link);
                        String baseUrl = url.getProtocol() + "://" + url.getHost()

                        if (!whitelist.getExternalServices().contains(baseUrl)) {
                            logger.warn("Provided image's src is: " + url + ".
                            value = value.replace(img, "");
                        }

                    } catch (MalformedURLException e) {
                        logger.debug("URL [" + link + "] is malformed. Trying to s
                        if (isValidRelativeURL(link) && isTrustedRelativePath(link
                            logger.debug("URL " + link + " is recognized to be
                        } else {
                            logger.error("Malformed URL [" + link + "]", e);
                            value = value.replace(img, "");
                        }
                    }

                }

            }

            logger.debug("OUT");
            return value;
        }

    private static String checkIframeTags(String value) {
            logger.debug("IN");
            Pattern maliciousTagPattern = Pattern.compile("&lt;iframe(.*?)iframe\\s*&gt;", Pat
            value = maliciousTagPattern.matcher(value).replaceAll("");

            Pattern scriptPattern = Pattern.compile("<iframe[^>]+(src\\s*=\\s*['\"]([^'\"]+)['
                    Pattern.CASE_INSENSITIVE | Pattern.MULTILINE | Pattern.DOTALL);
            Matcher scriptMatcher = scriptPattern.matcher(value);

            while (scriptMatcher.find()) {
                String iframe = scriptMatcher.group();
                String link = scriptMatcher.group(2);

                try {
                    URL url = new URL(link);
                    String baseUrl = url.getProtocol() + "://" + url.getHost();

                    if (!whitelist.getExternalServices().contains(baseUrl)) {
                        logger.warn("Provided iframe's src is: " + url + ". Iframe
                        value = value.replace(iframe, "");
```

```
                                        }

                                } catch (MalformedURLException e) {
                                        logger.debug("URL [" + link + "] is malformed. Trying to see if it
                                        if (isValidRelativeURL(link) && isTrustedRelativePath(link)) {
                                                logger.debug("URL " + link + " is recognized to be a valid
                                        } else {
                                                logger.error("Malformed URL [" + link + "]", e);
                                                value = value.replace(iframe, "");
                                        }
                                }

                        }

                        logger.debug("OUT");
                        return value;
                }

                private static String checkAnchorTags(String value) {
                        logger.debug("IN");
                        Pattern aPattern = Pattern.compile("<a([^>]+)>(.+?)</a>", Pattern.CASE_INSENSITIVE
                        Pattern hrefPattern = Pattern.compile("\\s*href\\s*=\\s*['\"]([^'\"]+)['\"]", Patt

                        Matcher aTagMatcher = aPattern.matcher(value);

                        while (aTagMatcher.find()) {
                                String aTag = aTagMatcher.group();
                                String href = aTagMatcher.group(1);

                                // In <a> tag find href attribute
                                Matcher hrefMatcher = hrefPattern.matcher(href);

                                while (hrefMatcher.find()) {
                                        String link = hrefMatcher.group(1);

                                        try {
                                                URL url = new URL(link);
                                                String baseUrl = url.getProtocol() + "://" + url.getHost()

                                                if (!whitelist.getExternalServices().contains(baseUrl)) {
                                                        logger.warn("Provided anchor's href is: " + url +
                                                        value = value.replace(aTag, "");
                                                }

                                        } catch (MalformedURLException e) {
                                                logger.debug("URL [" + link + "] is malformed. Trying to s
                                                if (isValidRelativeURL(link) && isTrustedRelativePath(link
                                                        logger.debug("URL " + link + " is recognized to be
                                                } else {
```

```
                                        logger.error("Malformed URL [" + link + "]", e);
                                        value = value.replace(aTag, "");
                                }
                        }
                }

        }

        logger.debug("OUT");
        return value;
}

private static String checkVideoTags(String value) {
        logger.debug("IN");
        Pattern maliciousPattern = Pattern.compile("&lt;video(.*?)video&gt;", Pattern.CASE
        value = maliciousPattern.matcher(value).replaceAll("");

        Pattern scriptPattern = Pattern.compile("<video(.+?)</video\\s*>", Pattern.CASE_IN
        Pattern srcAttributePattern = Pattern.compile("\\s*src\\s*=\\s*['\"]([^'\"]+)['\"]
        Matcher matcher = scriptPattern.matcher(value);

        while (matcher.find()) {
                String video = matcher.group();
                String betweenVideoTags = matcher.group(1);

                Matcher srcMatcher = srcAttributePattern.matcher(betweenVideoTags);

                while (srcMatcher.find()) {
                        String link = srcMatcher.group(1);

                        try {
                                URL url = new URL(link);
                                String baseUrl = url.getProtocol() + "://" + url.getHost()

                                if (!whitelist.getExternalServices().contains(baseUrl)) {
                                        logger.warn("Provided anchor's href is: " + url +
                                        value = value.replace(video, "");
                                }

                        } catch (MalformedURLException e) {
                                logger.debug("URL [" + link + "] is malformed. Trying to s
                                if (isValidRelativeURL(link) && isTrustedRelativePath(link
                                        logger.debug("URL " + link + " is recognized to be
                                } else {
                                        logger.error("Malformed or untrusted URL [" + link
                                        value = value.replace(video, "");
                                }
                        }
                }

        }

        logger.debug("OUT");
        return value;
    }

    private static String checkCSS(String value) {
        logger.debug("IN");
        Pattern cssUrlPattern = Pattern.compile("url\\s*\\(['\"]?([^'\"\\)]+)['\"]?\\)", P
        Pattern cssUrlDataPattern = Pattern.compile("data:image\\/(gif|jpeg|pjpeg|png|svg\
                Pattern.CASE_INSENSITIVE | Pattern.MULTILINE | Pattern.DOTALL);
        Pattern domElementID = Pattern.compile("(#[a-zA-Z0-9\\_\\-]+)", Pattern.CASE_INSEN
        String domId = "";

        Matcher urlMatcher = cssUrlPattern.matcher(value);

        while (urlMatcher.find()) {
            String cssUrl = urlMatcher.group();
            String link = urlMatcher.group(1);

            Matcher dataMatcher = cssUrlDataPattern.matcher(link);
            Matcher domIdMatcher = domElementID.matcher(link);

            if (domIdMatcher.find()) {
                domId = domIdMatcher.group();
                if (domId.length() > 50) {
                    logger.warn("Provided url attribute with Id is: " + domId
                    value = value.replace(cssUrl, "");
                }
            }

            if (!dataMatcher.find()) {
                try {
                    URL url = new URL(link);
                    String baseUrl = url.getProtocol() + "://" + url.getHost()

                    if (!whitelist.getExternalServices().contains(baseUrl)) {
                        logger.warn("Provided CSS url attribute is: " + ur
                        value = value.replace(cssUrl, "");
                    }
                } catch (MalformedURLException e) {
                    logger.debug("URL [" + link + "] is malformed. Trying to s
                    if (isValidRelativeURL(link) && isTrustedRelativePath(link
                        logger.debug("URL " + link + " is recognized to be
                    } else if (link.equals(domId)) {
                        return value;
                    } else {
                        logger.error("Malformed or untrusted URL [" + link
```

```java
						value = value.replace(cssUrl, "");
					}
				}
			}

		}

		logger.debug("OUT");
		return value;
	}

	private static boolean isValidRelativeURL(String url) {
		String absoluteUrl = "http://mynonexistingserver.something.smt:99999" + url;
		UrlValidator urlValidator = new UrlValidator();
		if (urlValidator.isValid(absoluteUrl)) {
			return true;
		} else {
			return false;
		}
	}

	private static boolean isTrustedRelativePath(String url) {
		List<String> relativePaths = whitelist.getRelativePaths();
		Iterator<String> it = relativePaths.iterator();

		while (it.hasNext()) {
			if (url.startsWith(it.next())) {
				return true;
			}
		}
		return false;
	}

}
```