JavaScript-based ESAPI: An In-Depth Overview

Marcus Niemietz
marcus.niemietz@rub.de

Practical Work

at

Chair for Network and Data Security
Prof. Dr. Jörg Schwenk

advised through Dipl.-Ing. Mario Heiderich

Partner: OWASP Foundation
`http://www.owasp.org`

2011-04-14

Horst-Görtz Institute Ruhr-University of Bochum

Contents

List of Figures and Listings

List of Figures

List of Listings

1. Introduction

Nowadays there are different companies present that make use of web applications on the Internet. They provide services like enabling users to search the Web, to utilise social networks, or to do shopping [1]. A primary goal of each company should be to generate a high profit so that each web site receives a high commercial relevance. That can begin with upgrading the image of a company or by obtaining direct sales.

The continuous development process shows that new languages like HTML5 [2] and CSS3 [3] will be frequently used in the future. In addition, there are techniques like "Asynchronous JavaScript and XML" available to enable the client to use web applications in an interactive way so that such applications behave more like desktop software. [4].

This development process requires extensive knowledge of web development. One aspect that should not be ignored is the security of these web applications. There are, for example, different business logic flaws that can put a web site at risk [5]. One must pay attention to session handling and managing credit card transactions as well as password recovery.

Some languages like JavaScript have been growing in their functionality. Thus, there are often no security mechanism available to do input validation to protect a user of a web site from the malicious code of an attacker. For the protection of such web applications, new security-relevant code has to be written for each application. This code can have errors in it or can be poorly written. If one considers that there are many problems that are based on faulty or quirky implementations of, for example, browser vendors, the problem of writing secure code is even bigger [6].

So there should be an instance that takes care of this problem. The target is that a developer without a broad security knowledge should write secure applications. This is exactly what this paper is about. A community of security-experienced people is developing an interface to offer possibilities for security and lower-risk applications by ready-made methods.

This paper analyses the JavaScript-based ESAPI as such a tool. It is presented in general and each given assurance criteria is discussed for security reasons. After that improvements on general objectives, redundancy aspects, and old as well as newly defined methods are shown. The paper concludes with an outlook about how the ESAPI affects itself and the future.

2. ESAPI

In the following, information is given regarding the ESAPI in general, an installation guide, and a discussion concerning the ESAPI4JS. Last but not least, the from the OWASP for the project mentioned, assurance criteria are explained [7].

2.1. General information

The not-for-profit worldwide charitable organisation OWASP, which is the abbreviation for “Open Web Application Security Project”, has its focus on improving the security of web applications [8]. One project of the OWASP is called “ESAPI”. It is the short form of “Enterprise Security API”, in which “API” stands for “Application Programming Interface”. According to OWASP, it should allow developers to write secure code very easily without having extensive prior knowledge of web application security [9]. Next to the new development with ESAPI approach, it should be also possible to retrofit security into existing applications. For the reason that the API covers, especially in enterprise business, many security challenges, developers do not have to care about writing parts of their own security-relevant code any-more. All ESAPI files are free and totally open. They are available under the BSD license, which ensures that one can use or modify the project however one likes [10].

All OWASP ESAPI versions have the same basic design to make it possible to develop software for applications, which make use of different programming or scripting languages. Thus, the ESAPI is available as a “Java”, “.NET”, “ASP”, “PHP”, or “JavaScript” implementation [8]. This paper will concentrate on the JavaScript-based part. It is also known under the name “ESAPI4JS “ and is being led and was founded by Chris Schmidt [11]. The current version “0.1.3” of the ESAPI4JS was released in January 2011 [12]. It is a young project and not yet finished as one can see from the version number “0.1.3”. Clear indications for this thesis will be highlighted in Chapter 2.1.1, 2.1.2, and especially in 3.3.

2.1.1. Installation

OWASP provides a sample implementation of the ESAPI4JS [13]. It consists of a step-by-step guide about how to do the installation for basic use. It leads a developer through six steps, beginning with the download and unpacking of the distribution, which is available as a compressed “.tar.gz” or “zip” file. After that, a developer should create a directory called “esapi4js”. Inside the new directory, there should be copied the files of the unpacked ESAPI4JS distribution.

The above-mentioned sample implementation consists of two different parts. The first part is a combination of “script”-tags, which includes the given ESAPI4JS files. It is important to note that one has to follow a strict order of the file inclusion to fulfil some requirements. This was not observed in the sample implementation, so a correct implementation of the first part is given in Listing 2.1.

Listing 2.1: JavaScript files of the ESAPI4JS (filename: index.html - Part 1/2)

```
1 <!-- esapi4js dependencies -->
2 <script type="text/javascript" language="JavaScript" src="http://localhost/esapi/
      esapi4js/lib/log4js.js"></script>
3 <!-- esapi4js core -->
4 <script type="text/javascript" language="JavaScript" src="http://localhost/esapi/
      esapi4js/esapi.js"></script>
5 <!-- esapi4js i18n resources -->
6 <script type="text/javascript" language="JavaScript" src="http://localhost/esapi/
      esapi4js/resources/i18n/ESAPI_Standard_en_US.properties.js"></script>
7 <!-- esapi4js configuration -->
8 <script type="text/javascript" language="JavaScript" src="http://localhost/esapi/
      esapi4js/resources/Base.esapi.properties.js"></script>
```

The second part of the sample implementation shows some possibilities about how to use a few methods, especially the most basic, of the ESAPI4JS. In addition to the first part, an implementation error is also given. The modified and working version is shown in Listing 2.2. The complete implementation is displayed in the Appendix in Listing A.1. First of all, some configuration options are listed that are sourced in the first implementation part. After that the application name is defined in line ten. The next line initialises the API, followed by using the "Logger". In line thirteen, an example of how to use the "Encoder" is shown. Finally, a possible usage of the integrated validator is given from lines fourteen to sixteen.

Listing 2.2: Example of using the ESAPI4JS (filename: index.html - Part 2/2)

```
1 <script type="text/javascript" language="JavaScript">
2     Base.esapi.properties.logging['ApplicationLogger'] = {
3         Level: org.owasp.esapi.Logger.ALL,
4         Appenders: [ new Log4js.ConsoleAppender() ],
5         LogUrl: true,
6         LogApplicationName: true,
7         EncodingRequired: true
8     };
9
10    Base.esapi.properties.application.Name = "My Application v1.0";
11    org.owasp.esapi.ESAPI.initialize();
12    $ESAPI.logger('ApplicationLogger').info(org.owasp.esapi.Logger.EventType.
          EVENT_SUCCESS, 'This is a test message');
13    document.writeln( $ESAPI.encoder().encodeForHTML( "<a href=\"http://owasp-
          esapi-js.googlecode.com\">Check out esapi4js</a>" ) );
14    var validateCreditCard = function() {
15        return $ESAPI.validator().isValidCreditCard( $('CreditCard').value );
16    }
17 </script>
```

2.1.2. Usage

As shown in Listing 2.2, several mechanisms are integrated in the ESAPI4JS. Because there is no documentation about the existing mechanisms for the JavaScript-based ESAPI implementation available, the ESAPI JavaDoc [1] can be used. As already mentioned in Chapter 2.1, all ESAPI versions have the same basic design, so the JavaDoc can be used as a rough guide for ESAPI4JS. To gain a better knowledge

[1] The ESAPI JavaDocs are available under the following URL: `http://owasp-esapi-java.googlecode.com/svn/trunk_doc/index.html`

about the existing methods, one should look into the source code of the file “esapi.js”. An analysis of the file shows that it consists of nearly 3,000 lines of source code and many methods, which are mapped to the typical ESAPI class structure.

To gain more clarity about the existing classes, a graph in Figure 2.1 is displayed. It shows the DOM [2] in a depth of four levels of the object “org” from the file “esapi.js”. Furthermore, each node of the last two node levels is a method or includes at least one method.

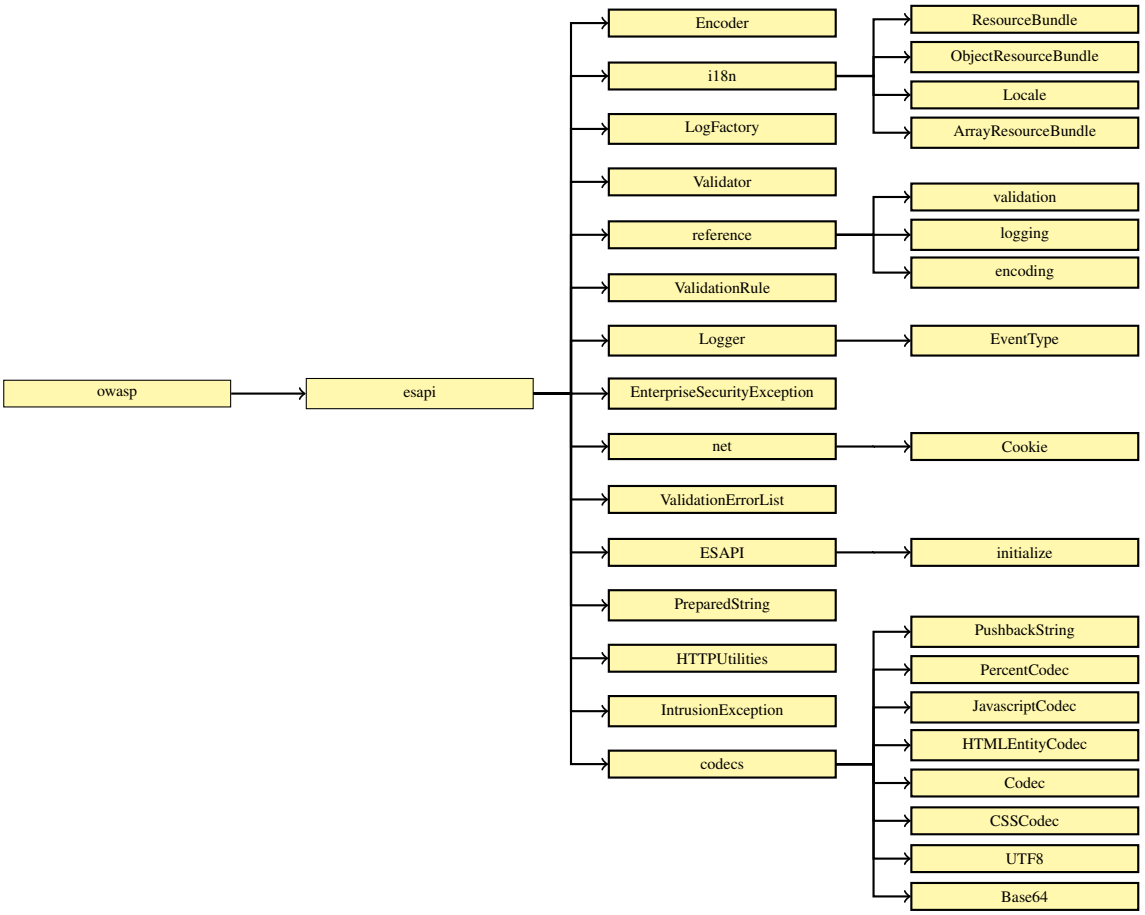

Figure 2.1.: DOM in a depth of four levels of the object “org” from the file “esapi.js”

Countermeasures against DOM-based XSS

One of the three basic types of XSS [3] is called “DOM-based”. Besides reflective and persistent XSS, DOM-based XSS can be used on static web pages, too [14].

[2] “DOM” is the abbreviation for “Document Object Model” and is the standard model for representing HTML or XML content. Thus, one can use JavaScript code to access or modify HTML DOM objects [14]. More information is available under the URL: `http://www.w3.org/DOM/`

[3] Cross-site scripting, also known as “XSS”, is an attack that one can typically find in web applications. Its capability includes the theft of data, changes of the visual appearance, and distributed denial-of-service attacks. [15]

Listing 2.3: An example of a vulnerable JavaScript code (file: domXSS.html)

```
1 <script>document.write(document.URL);</script>
```

An example of a vulnerable JavaScript code that can be used with DOM-based XSS is given in Listing 2.3. In the case where this code is placed on “example.org”, one can inject malicious code by using a URL like “http://example.org?<script>alert(0)</script>”. If there are no filters activated, it causes a web browser to show an alert window with the message “0”.

The ESAPI4JS offers a way to protect a user against this kind of XSS attack. The code of Listing 2.4 can be used as shown in the Appendix in Listing A.1. In this case “encodeForHTML” encodes data by using HTML entity encoding [16]. It makes use of “encode(IMMUNE_HTML, sInput)”, whereby “IMMUNE_HTML” is an array with explicitly allowed characters like “,” or “.”. According to the displayed graph of Figure 2.1, “encodeForHTML” is inside “org.owasp.esapi.reference.encoding.DefaultEncoder”.

Listing 2.4: An example of a vulnerable JavaScript code (file: domXSSsanitized.html)

```
1 document.write($ESAPI.encoder().encodeForHTML(document.URL));
```

2.2. Assurance criteria

The term assurance shows in the field of system security how much one can trust a system. Thus, assurance is a quantifier of trust. The term includes the consideration of the specification, design, and implementation of a system [17]. To achieve a certain degree of assurance, Abadi and Needham released some design principles for robust protocols in 1995 [18]. They used basic principles like “explicitness” and “appropriate action” as well as principles like “naming”, “encryption”, “timeliness”, “recognizing messages and encodings”, and “trust”. These are informal guidelines for a good protocol design.

In addition to that, OWASP also published some coding practices for the ESAPI. They are described in the following subsections [7].

2.2.1. OWASP Top 10

The “OWASP Top 10” web application security risks defined in 2010 are given in the following enumeration.

1. Injection
2. Cross-Site Scripting (XSS)
3. Broken Authentication and Session Management
4. Insecure Direct Object References
5. Cross-Site Request Forgery (CSRF)
6. Security Misconfiguration
7. Insecure Cryptographic Storage
8. Failure to Restrict URL Access
9. Insufficient Transport Layer Protection
10. Unvalidated Redirects and Forwards

OWASP offers some information boxes for each point of the Top 10 list in their "Top 10" pdf file [19]. These boxes answer the questions about vulnerability to the attack and how one can protect an application from it. In addition to that, there is also a box with an examplary attack scenario and a box with some references in it. With regard to the development of web applications, especially for the ESAPI4JS, each "How Do I Prevent" box can be evaluated to fulfil the first assurance criterion.

2.2.2. Performance vs. security

The second criterion is to find a balance between the performance and the security of the ESAPI. From the viewpoint of a developer, this might be debatable. A conceivable scenario is a server-based application that is used by thousands of people simultaneously. In compliance with the human response time of a web application, it is a desirable goal to get a fast response from an application [20].

In a security context, the statement does not mean that one should reduce the security to gain more performance. It should rather be understood as a point of a policy to be achieved. Some options to attain more performance by having a constant security level are to use programs with a smaller duration [4]. This means, for example, that implemented algorithms should be chosen thoughtfully if there are alternatives available. Another way is to avoid redundancy or to select the right programming language. Not everything that makes sense in one programming or scripting language makes sense in another language [21].

2.2.3. Training and experience of developers

Attaining knowledge about the current training level of developers plays an important role in the third criterion. There also arises the question about the amount of experience in the web development process [7]. Regardless of the needed programming language skills, there should be a certain degree of IT-security knowledge. If this fundamental prerequisite is not present, one can not assume that security features, such as those integrated in the ESAPI, will be used. To have partial knowledge about the necessary fields of IT-security can also be a problem. It can lead a developer to use security mechanisms in a wrong or insufficient way.

2.2.4. Using tools

The fourth criterion is to decide about which tools to use during and at the end of the whole development process [7]. This assurance criterion is not defined clearly, so it can be interpreted in different ways.

First of all, it is not clear if these tools should help a developer to avoid errors in the field of IT-security or just at certain points, for example to have no code redundancy or syntax errors. If both fits there occurs a second argument. Because security tools are written very generally, one can not assume to gain more security by using them. It can help but it is no replacement for experienced security researchers, which can analyse the entire code more efficiently. In addition to the "training and experience of developers" criterion of Chapter 2.2.3, it is more important to have trained developers with much experience.

As a consequence, the criterion has to be redefined in a clearer way. It would also be useful to evaluate the importance of each criterion.

[4] A comparison between the duration of algorithms can be achieved by using the Landau notation. URL: `http://sse.tggs.kmutnb.ac.th/teaching/ea/slides/landau.pdf`

2.2.5. Unauthorised alterations

One question for this criterion is to check how the code repository is protected against unauthorised alternations. This requires access control mechanisms, which make use of identity establishment to ensure that only authorised users are securely associated with a legitimate entity. One has to prove that the entity is really the entity it claims to be [22].

This act of authorisation is realised with the help of the project hosting on "Google Code". It authenticates a user with her or his mail address against a web server, whereby each user has well-defined privileges to modify the project and thus also the repository [23].

2.2.6. Understanding the code

The topic "Understanding the code" summarises the issue of the needed required work for code check-in and independent review [7]. Badly written code, which is written like it holds obfuscation methods [5] in it, can not be analysed by a researcher as fast as in the case of clearly structured code. A suitable and very important supplement is to set comments into the programming code. The detail level of each comment could vary by checking the code complexity and the effort of refactoring the code after one has misunderstood it two or more times.

ESAPI4JS does not meet this ESAPI criterion. Comments are not contained in some parts of the "esapi.js" file, so it usually takes a lot of time to understand it. This is underlined by the fact that the file contains nearly 3,000 lines of source code. Sometimes the existing comments are also not sufficient. A positive message is that a few comment parts are based on the JavaDoc style guidelines [6]. This makes it easy to understand the code because of the similar basic design of the ESAPI, as already mentioned in Chapter 2.1.

2.2.7. Threat level analyses

A threat is a potential violation of security that can have an undesirable effect on at least one system asset or resource [25]. Therefore, it is important to know what threat level is being accounted for the ESAPI [7].

From this it follows for the ESAPI that one has to prove if existing protection mechanisms work against all and not just some kinds of attacks. The ESAPI and especially the ESAPI4JS are characterised by providing more security, so the assumption of an attacker-resistant software, at least insofar as it can be implemented, is very important. Furthermore, it should be also clear which threat modelling is being used. This is not specified by the OWASP for the ESAPI but is already mentioned at an existing "Threat Risk Modelling" web page [26].

[5] With obfuscation methods, it may also be possible to bypass security measures like input filters or web-based intrusion detection systems. [24]

[6] Oracle provides a web page with "How to Write Doc Comments for the Javadoc Tool" guidelines. It is reachable under the URL: `http://www.oracle.com/technetwork/java/javase/documentation/index-137868.html`

3. Improvements

This chapter discusses general objectives like retrofitting security into existing applications and having the same basic design in all ESAPI implementations. Also highlighted is a way to modify objects and references to this possible countermeasure. Similarly, redundancy aspects such as empty methods or duplicates are focussed. Last but not least are analysed and introduced existing and newly created methods.

3.1. General objectives

In this chapter it will be analysed whether it makes sense to retrofit security into used applications and if one should rely on the same basic design of the ESAPI to reach the same application security level.

3.1.1. Retrofit security

One of the goals of the ESAPI design is to make it easier for developers to retrofit security into applications [8]. This should be enjoyed with caution because there might occur the impression for a developer that she or he can secure an application by only using the ESAPI, or in this paper rather the ESAPI4JS.

In general a security life cycle of an IT-system starts with an analysis of potential violations of security, which is also called “threat” [27]. Such threats should generally be considered from the start of a developing process. If this is not the case, it can be difficult to analyse security issues in a complex application. Therefore, retrofitting security in a existing application is a patchwork and should be only used as a last resort.

3.1.2. Same basic design

One of the advertised benefits of the ESAPI is to have the same basic design [8]. This makes it possible to gain security by using the same structure in each language where the ESAPI is implemented. This is a good thing if one pays attention to a few limitations.

Such limitations can be easily explained by the fact that each language has its own data handling. On the one hand, there are client-side interpreted languages and on the other hand there are server-side interpreted languages. Server-side languages will be compiled and executed by a server where client-side languages like JavaScript rely on the interpretation of third-party tools like a web browser. In the case of JavaScript, a user can deactivate it to inter alia avoid JavaScript-based XSS, too. This approach to gain more security can destroy the execution of ESAPI4JS, which this paper concentrates on.

Therefore, it is important to tell developers that the same security mechanism of the ESAPI, which provides security in a server-side language like PHP [1], will not provide the same security level in each implemented language.

[1] PHP is a widely used scripting language for web applications especially. [28]

3.2. Modification of objects

A typical characteristic of the language JavaScript is the client-side interpretation by a web browser. The source code is therefore open and accessible for a client. For this reason, the question arises as to what will happen if there is, under some circumstances, even a partial filtered XSS vulnerability on a web page using ESAPI4JS. A desired property is to secure all via the ESAPI4JS protected areas regardless if there is a vulnerability on the web page or not.

As described in Chapter 3.2.1, the ESAPI4JS is not hardened for Internet Explorer. An existing countermeasure for new web browsers is given in Chapter 3.2.2.

3.2.1. Overwriting DOM properties in IE

In Internet Explorer 6,7,8, and 9, it is possible to overwrite DOM properties by using "id" attributes in markup languages like HTML. An example is shown in Listing 3.1 [29]. It illustrates that one can inter alia redefine the "URL" object such that an alert window will be executed.

Listing 3.1: Redefining the "url" object in IE (filename: ie.html)

```
1 <a id="url" href="javascript:alert(1)">
2 <script>
3 location=url;
4 </script>
```

This procedure can also be transferred to the ESAPI4JS. As shown in Listing 2.2 and Figure 2.1, the tool has an object "org", which includes other objects, respectively methods. By redefining the object "org" with

- `<form id="org">`

at the beginning of the ESAPI4JS code, as in the Appendix in Listing A.1, one can create a JavaScript interpretation error in Internet Explorer. This ensures that the whole JavaScript code and especially the ESAPI4JS files of the vulnerable web page will not be executed any-more. From this it follows that there are no active protection mechanism by the ESAPI any-more.

3.2.2. defineProperty for objects

A countermeasure against the attack described in Chapter 3.2.1 is to use the "JavaScript 1.8.5" integrated function "defineProperty" [30]. It is completely supported by Internet Explorer 9, Firefox 4, Safari 5, and last but not least by Chrome 5 [31]. The syntax looks like

- `Object.defineProperty(obj, prop, descriptor)`

where "obj" is the object with the to be defined or modified property "prop". The "descriptor" can include a "configurable" attribute with the value "true" or "false". It controls if the property of the given object can be deleted or changed [32]. An example is given in Listing 3.2.

Listing 3.2: Define an object with the configurable attribute (filename: ie9secure.html - Part 1/2)

```
1 <script>
2 var org = {};
3 Object.defineProperty(org, "owasp", { value : 1, configurable : false } );
```

```
4 </script>
```

An illustrative example of how the object protection mechanism works is shown in Listing 3.3 [32]. After defining "org.owasp" in Listing 3.2, an alert window is executed with the value "1" in it. Nothing will happen if an attacker wants to "delete" the previously defined object, and similarly a redefinition with "defineProperty" is not possible any-more. This is illustrated by the other commented alert windows.

Listing 3.3: Protection by the configurable attribute (filename: ie9secure.html - Part 2/2)

```
1 <script>
2 alert(org.owasp); // alerts 1
3 delete org.owasp; // nothing happens
4 alert(org.owasp); // alerts 1
5
6 Object.defineProperty(org, "owasp", {value : 0}); // throws a type error
7 alert(org.owasp); // not executed
8 alert(5); // not executed
9 </script>
```

Overall, it shows that the principle "first come first served" fits very well in the "defineProperty" case. If an attacker finds a workable XSS vulnerability before the API code is executed, it is possible to specify the full behaviour of the ESAPI4JS. If there is a vulnerability after the API and if "defineProperty" with "configurable : false" is used, an attacker can not control the behaviour of the previously defined objects anymore. Therefore, one has a tool to protect application objects, and in this case the objects of the API against XSS in one direction, regardless if there is an XSS vulnerability or not.

3.3. Redundancy

The issue of redundancy has already been roughly addressed in Chapters 2.2.2 and 2.2.3. One of the ways in which the term redundancy can be described is to use a definition of Shannon [33]. He defined the term as "the difference between channel capacity and source entropy" [34]. This definition comes from the field of information technology and can also be described as an indicator that occurs if the full information capacity of a communication channel is not used.

Hence, one can suggest for the ESAPI, or in general for applications, that there should not be any empty methods, duplicates, or unessential methods that make at most only use of an existing JavaScript method. Such things are present in the analysed ESAPI4JS. To underline this statements, a few examples for redundancy are given in Chapters 3.3.1, 3.3.2, 3.3.3, and 3.3.4.

3.3.1. Empty methods

As shown in Listing 3.4, an empty method "org.owasp.esapi.Encoder" is defined in the ESAPI4JS. It is also never used inside the "esapi.js" file, so it should be deleted to shrink the ESAPI4JS.

Listing 3.4: Redundancy example: Empty method (filename: esapi.js - lines 348 to 350)

```
1 org.owasp.esapi.Encoder = function() {
2
3 }
```

3.3.2. Duplicates

In general it does not make any sense to use duplicates. In JavaScript one can use the keyword “this” to make a reference to a current object. In the case of the ESAPI4JS, a method “encodeForJavaScript” is defined and after that the same method is redefined with “this.encodeForJavaScript” again. This code is displayed in Listing 3.5.

Listing 3.5: Redundancy example: Duplicate (filename: esapi.js - lines 1998 to 2002)

```
1 encodeForJavaScript: function(sInput) {
2   return !sInput ? null : _javascriptCodec.encode(IMMUNE_JAVASCRIPT, sInput);
3 },
4
5 encodeForJavascript: this.encodeForJavaScript,
```

3.3.3. Unnecessary methods

Repeating something unnecessarily for twice or even more times should be banned from the developing process to avoid redundancy. In the case of the ESAPI4JS, methods that work according to a certain principle in, inter alia, “encodeForURL” or “decodeFromURL” are used. First, it is verified if the input “sInput” is not empty. If it is not empty, “escape” is used to encode a string and “unescape” to decode a string. To avoid redundancy, the code should be deleted because a developer can apply “escape” or “unescape” directly.

Listing 3.6: Redundancy example: Unnecessary methods (filename: esapi.js - lines 2004 to 2010)

```
1 encodeForURL: function(sInput) {
2   return !sInput ? null : escape(sInput);
3 },
4
5 decodeFromURL: function(sInput) {
6   return !sInput ? null : unescape(sInput);
7 },
```

It should be noted that it does make sense to define the above method because the goal is to have the same basic design in ESAPI. However, in this case it provides also the not undesired redundancy.

3.3.4. jQuery-Encoder

jQuery is a JavaScript library that simplifies a developing process with a rapid web development approach [35]. The available methods can be expanded by using plugins. One of these plugins is called “jquery-encoder” [2]. It is also known under the name “jqencoder”. The goal is “to do contextual output encoding on untrusted data” [36].

Chris Schmidt is, as in the case of the ESAPI4JS, the author of the jQuery plugin [37]. He adds few methods to jQuerry via this plugin, which looks similar to the methods of the ESAPI “Encoder” interface. Such methods are “encodeForHTML”, “encodeForHTMLAttribute”, or “encodeForURL” [38].

[2]The jQuery plugin “jqencoder” can be downloaded under the URL: `https://github.com/chrisisbeef/jquery-encoder`

If a developer is using jQuery in a web application and there is just the requirement to have some ESAPI4JS encoding functions, one should use the jQuery plugin instead of the ESAPI4JS. This approach minimises the JavaScript size of a web page and provides, therefore, less redundancy.

3.4. Methods

This chapter takes a look at existing and newly defined methods of the ESAPI. Each method will be individually considered for its implementation.

3.4.1. Analysis of existing methods

Below are listed two eye-catching protection mechanisms of the ESAPI. It will be discussed if they are well-integrated in terms of things like articulation and logic, among other things.

Encoder interface

In some cases, the logic of using a given method plays an important role when using the ESAPI. A good example is to take a look at the "Encoder" interface by taking the Base64 "decode" and "encode" methods. First of all, it should be clear that the ESAPI and especially the ESAPI4JS are designed to provide more application security. This can be interpreted by a developer as a complementary ticket to secure a web application by only using one method of the ESAPI4JS. This assumption is not correct.

As displayed in the proof of concept of Listing 3.7, an attacker can execute malicious code by using a given method of the ESAPI4JS. First of all the JavaScript alert-window code will be encoded such that the Base64 output is:

- `PHNjcmlwdD5hbGVydCgwKTwvc2NyaXB0Pg==`

This output will be decoded to the previously encoded code and afterwards written into the source code of the web page. The source code will be interpreted and executed by a web browser such that an alert window occurs.

Listing 3.7: JavaScript code execution using the Base64 method (filename: base64.html)

```
1 document.writeln( org.owasp.esapi.codecs.Base64.decode( org.owasp.esapi.codecs.
      Base64.encode( "<script>alert(0)<\/script>" ) ) );
```

A way to prevent this logical attack is to use "$ESAPI.encoder().encodeForHTML" as a wrapper around the "decode" and "encode" methods. As discussed in Chapter 2.2, "encodeForHTML" encodes data by using HTML entity encoding. It is therefore necessary that a user is aware of this issue. This can be explicitly mentioned in the ESAPI documentation.

Clickjacking

The Java-based ESAPI offers, in contrast to the ESAPI4JS, a clickjacking protection mechanism. This mechanism can be reached in Java via "org.owasp.esapi.filters.ClickjackFilter" [39]. Apart from the desire that a clickjacking protection mechanism should be also implemented in the ESAPI4JS, it should be also clear that the given mechanism is not a solution to prevent clickjacking attacks.

Clickjacking is a term that includes attacks like “basic clickjacking” [40], “likejacking” [41], “stroke-jacking” [42], and “cursorjacking” [43]. A clickjacking attack forces a victim to unintentionally click on an obviously invisible web page, which is usually framed [44]. At least in the case of cursorjacking, the “HTTP X-FRAME-OPTIONS” header does not protect a user from clickjacking any-more. This is because this HTTP header manages with its value if a web page can be framed or not. If the header detects a violation the framed page will not be loaded by the web browser [45]. Cursorjacking does not use such a framing technique; it changes just the way the mouse pointer looks.

Thus, one should rename the expression “ClickjackFilter” to “FramingFilter”. Currently, there is no way to handle all clickjacking attacks, so the class “ClickjackFilter” should be deleted.

3.4.2. Creating new methods

For the reason that the ESAPI offers security features for enterprise applications and that it, as well as the ESAPI4JS, includes a credit card validation process, it should be obvious to also implement some validation processes for other important business flows [46].

International Bank Account Number

The International Bank Account Number, which is the abbreviation for “IBAN”, is an international standard for identifying bank accounts in European Union countries [47]. Each bank account number consists of thirty-four letters maximum [48].

An example for a valid IBAN is [47]:

- Paper Format: GB29 NWBK 6016 1331 9268 19
- Electronic Format: GB29NWBK60161331926819

The first pair consists of two characters. It shows the country where the account is held. After that there are two check digits displayed followed by up to thirty basic characters of the bank account number (BBAN). The BBAN can consist of numeric, alphabetic, or alphanumeric characters. Norway has the shortest account number length with fifteen characters, so the BBAN consists of eleven characters [49]. This will be treated as the infimum [3] for the BBAN with the already mentioned supremum of thirty characters [50].

For the reason that the IBAN can be designed similarly to the credit card validation process, a regular expression is given for the usually applied electronic format. It can be added to the “Base.esapi.properties.js” file of ESAPI4JS:

- `^[a-zA-Z]{2}[0-9]{2}[a-zA-Z0-9]{11,30}$`

The first character matches the beginning of a line. Similarly, the last characters match the end of a line. Among them is the above-discussed IBAN design specified. The expression checks if there are two large or small letters, two numbers from zero to nine, and at least eleven, up to a maximum of thirty, non-case-sensitive alphanumeric characters.

[3] Infimum is also known as “greatest lower bound”. Correlated to that, to that supremum is known as “least upper bound”.

Identity card

An identification card can be a document to verify a person's personal identity [51]. In Germany an identity card is valid up to ten years after it is issued.. Since November 2010 there has been a new identity card available, so old cards are valid till 2020 [52]. Since this is a very long period in the information technology business, there will be introduced a process to verify if the numbers of an old German identity card are valid or not.

An example for "Angelika Mustermann" [53] is given in the following:

- Example: `1220001518D<<6408125<1110078<<<<<<<0`

As displayed, there is a certain structure for the identification numbers, which can be generalised [54]. At first there are ten numeric characters followed by a "D" for the nationality "Deutsch" and two less-than signs. After that there are seven numeric characters with one less-than sign. Finally, it ends with seven numeric characters, seven less-than signs, and one numeric sign. In total it consists of thirty-six characters. This is shown in the following regular expression and can be used as is in the case of the "International Bank Account Number" in Chapter 3.4.2:

- `^[0-9]{10}D<<[0-9]{7}<[0-9]{7}<<<<<<<[0-9]{1}$`

A mathematical formal form looks like [54]:

- $x_1x_2x_3x_4x_5x_6x_7x_8x_9c_1D << y_1y_2y_3y_4y_5y_6c_2 < z_1z_2z_3z_4z_5z_6c_3 <<<<<<< c_{total}$

where

- $c_1 \equiv 7 \times x_1 + 3 \times x_2 + 1 \times x_3 + 7 \times x_4 + 3 \times x_5 + 1 \times x_6 + 7 \times x_7 + 3 \times x_8 + 1 \times x_9 \pmod{10}$
- $c_2 \equiv 7 \times y_1 + 3 \times y_2 + 1 \times y_3 + 7 \times y_4 + 3 \times y_5 + 1 \times y_6 \pmod{10}$
- $c_3 \equiv 7 \times z_1 + 3 \times z_2 + 1 \times z_3 + 7 \times z_4 + 3 \times z_5 + 1 \times z_6 \pmod{10}$
- $c_{total} \equiv 7 \times x_1 + 3 \times x_2 + 1 \times x_3 + 7 \times x_4 + 3 \times x_5 + 1 \times x_6 + 7 \times x_7 + 3 \times x_8 + 1 \times x_9 + 7 \times c_1 + 3 \times y_1 + 1 \times y_2 + 7 \times y_3 + 3 \times y_4 + 1 \times y_5 + 7 \times y_6 + 3 \times c_2 + 1 \times z_1 + 7 \times z_2 + 3 \times z_3 + 1 \times z_4 + 7 \times z_5 + 3 \times z_6 + 1 \times c_3 \pmod{10}$

with

- $x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, y_1, y_2, y_3, y_4, y_5, y_6, z_1, z_2, z_3, z_4, z_5, z_6, c_1, c_2, c_3, c_{total} \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

As shown above, there are different checksum characters "c". They can be used to verify if the number was typed in correctly. If one uses the regular expression together with the formal form of the identitiy card numbers, a method for the ESAPI4JS can be created.

An example implementation is given in the Appendix in Listing A.2. The "idCard" method created there can be executed with the following exemplary code:

- `document.write(idCard("1220001518D<<6408125<1110078<<<<<<<0"));`

The method returns "true" if the form was entered correctly and if the computed checksums are equal to the one of the input string. The "true" or "false" value will be written on the web page with "document.write". In the example, the method "idCard" returns the value "true".

International Standard Book Number

ISBN is the abbreviation for “International Standard Book Number” and is ISO Standard 2108. It was developed as a unique international identification system and can be found in, inter alia, books. It shows that it has a high commercial relevance [55]. Originally, it was developed as a “(British) Standard Book Number” in 1968 [56]. Before 2007 each ISBN was ten digits long; since 2007 it had a length of thirteen numbers.

An example of what a valid thirteen-characters-long ISBN looks like [57]:

- 978-3-8362-1194-9

Different ways are used to separate the numbers of an ISBN, so the following regular expression only matches such a structured ISBN as displayed above:

- `^[0-9]{3}-[0-9]{1}-[0-9]{4}-[0-9]{4}-[0-9]{1}$`

The expression checks if there is the right sequence of numeric characters and hyphens. Similar to the identity card of Chapter 3.4.2, a check digit is also given. In the case of an ISBN, it is the last number.

Formally, it can be described as

- $x_1x_2x_3 - x_4 - x_5x_6x_7x_8x_9 - x_{10}x_{11}x_{12} - c_{total}$

where [58]

- c_{total} = (10 - (($x_1 + x_3 + x_5 + x_7 + x_9 + x_{11} + 3 \times (x_2 + x_4 + x_6 + x_7 + x_{10} + x_{12}$)) mod 10)) mod 10

with

- x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}, x_{12}, $c_{total} \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

A JavaScript-based implementation of the ISBN for the ESAPI4JS is given in the Appendix in Listing A.3. The method “isbn” created there can be executed with the following exemplary code:

- `document.write(isbn("978-3-8362-1194-9"));`

The method returns the value “true” if the regular expression was matched and if the checksum was computed and compared correctly. Otherwise, it returns “false”.

4. Conclusion and outlook

In Chapter 2 general information and assurance criteria are discussed regarding the ESAPI as a web application security tool. There it is addressed that this paper is particularly concerned with ESAPI4JS. Regarding this JavaScript-based tool, an installation and usage guide with an implementation example is given in Chapter 2.1. This guide shows that it is easy to implement countermeasures against, for example, DOM-based XSS. Chapter 2.2 illustrates assurance criteria in the form of coding practices, which are available for the ESAPI and thus also for the ESAPI4JS. They are analysed in detail to determine if the assurance criteria are fulfiled.

Chapter 3 demonstrates possible improvement aspects. First if all, there are general objectives introduced in Chapter 3.1. They include an analysis of the important points “retrofit security” and “same basic design” of ESAPI4JS. After that the modification of objects is discussed in Chapter 3.2. It includes an attack for overwriting DOM properties in Internet Explorer and a countermeasure by using “defineProperty”. Chapter 3.3 displays some typical examples for redundancy like “empty methods”, “duplicates”, and “unnecessary methods”. As a redundancy aspect the “jQuery-Encoder” is also given as a plugin for the JavaScript library jQuery.

Last but not least, existing and newly defined methods for the ESAPI4JS are listed in Chapter 3.4. In the analyses of the existing methods, the “Encoder” interface with a proof of concept in the “Base64” case and the not-yet-implemented ESAPI4JS clickjacking protection mechanism of the Java-based ESAPI are analysed. The new methods verify an IBAN, a German identity card number, and an ISBN for its validity.

Overall, it can be said in the outlook that the main work is done, so the ESAPI4JS gives a positive first impression. Thus, it might be a promising project in the future. A correct usage of the methods can provide, in a case where there are no XSS vulnerabilities on the web page, more security or a lower risk in new or even existing web applications, as advertised by the OWASP.

However, a closer look shows that it still needs much work to meet the self-imposed requirements. Comprehensible discussed examples are, inter alia, the inadequate documentation and incomplete implementation regarding the native program ESAPI. The newly defined methods show that it is easy to customise ESAPI4JS for one’s own needs.

For the future, it could be advisable, in addition to the improvements explained in this paper, to program more verification methods. It should be possible to download them separately as plugins for a specific integration to avoid redundancy or unused source code. Examples are the “Value Added Tax (VAT) Identification Number” [59] and the “Universal Product Code” (UPC) [60]. Such a work can be listed on an already available project roadmap [61]. It is not maintained any-more but it should be active to increase the development process by inspiring other new potential developers.

A. Appendix

A.1. ESAPI

Listing A.1: Example of using the ESAPI4JS (filename: index.html)

```
<!-- esapi4js dependencies -->
<script type="text/javascript" language="JavaScript" src="http://localhost/esapi/
    esapi4js/lib/log4js.js"></script>
<!-- esapi4js core -->
<script type="text/javascript" language="JavaScript" src="http://localhost/esapi/
    esapi4js/esapi.js"></script>
<!-- esapi4js i18n resources -->
<script type="text/javascript" language="JavaScript" src="http://localhost/esapi/
    esapi4js/resources/i18n/ESAPI_Standard_en_US.properties.js"></script>
<!-- esapi4js configuration -->
<script type="text/javascript" language="JavaScript" src="http://localhost/esapi/
    esapi4js/resources/Base.esapi.properties.js"></script>

<script type="text/javascript" language="JavaScript">
    Base.esapi.properties.logging['ApplicationLogger'] = {
        Level: org.owasp.esapi.Logger.ALL,
        Appenders: [ new Log4js.ConsoleAppender() ],
        LogUrl: true,
        LogApplicationName: true,
        EncodingRequired: true
    };

    Base.esapi.properties.application.Name = "My Application v1.0";
    org.owasp.esapi.ESAPI.initialize();
    $ESAPI.logger('ApplicationLogger').info(org.owasp.esapi.Logger.EventType.
        EVENT_SUCCESS, 'This is a test message');
    document.writeln( $ESAPI.encoder().encodeForHTML( "<a href=\"http://owasp-
        esapi-js.googlecode.com\">Check out esapi4js</a>" ) );
    var validateCreditCard = function() {
        return $ESAPI.validator().isValidCreditCard( $('CreditCard').value );
    }
</script>
```

A.2. Improvements

Listing A.2: JavaScript method to verify the correctness of an identification card number

```
<script>
/**
 * The method idCard verifies if sInput is a valid ID card string.
 * It returns ''true'' if the string is valid - otherwise it returns ''false''.
 * @param sInput {String} a <code>String</code> with the ID card numbers
```

```
 */
idCard = function( sInput ) {
  // Defining a regular expression to check the right input format
  var pattern = new RegExp("^[0-9]{10}D<<[0-9]{7}<[0-9]{7}<<<<<<<[0-9]{1}$");
  // True if pattern matches
  if (pattern.test(sInput)) {
    // Checksum verification of character nine
    if (((7* sInput.charAt(0)+3*sInput.charAt(1)+1*sInput.charAt(2)+7*sInput.
        charAt(3)+3*sInput.charAt(4)+1*sInput.charAt(5)+7*sInput.charAt(6)+3*
        sInput.charAt(7)+1*sInput.charAt(8))%10) == sInput.charAt(9)) {
      // Checksum verification of character nineteen
      if (((7* sInput.charAt(13)+3*sInput.charAt(14)+1*sInput.charAt(15)+7*sInput
          .charAt(16)+3*sInput.charAt(17)+1*sInput.charAt(18))%10) == sInput.
          charAt(19)) {
        // Checksum verification of character twenty-seven
        if (((7* sInput.charAt(21)+3*sInput.charAt(22)+1*sInput.charAt(23)+7*
            sInput.charAt(24)+3*sInput.charAt(25)+1*sInput.charAt(26))%10) ==
            sInput.charAt(27)) {
          // Checksum verification of the last character
          if (((7* sInput.charAt(0)+3*sInput.charAt(1)+1*sInput.charAt(2)+7*
              sInput.charAt(3)+3*sInput.charAt(4)+1*sInput.charAt(5)+7*sInput.
              charAt(6)+3*sInput.charAt(7)+1*sInput.charAt(8)+7*sInput.charAt(9)
              +3* sInput.charAt(13)+1*sInput.charAt(14)+7*sInput.charAt(15)+3*
              sInput.charAt(16)+1*sInput.charAt(17)+7*sInput.charAt(18)+3*sInput.
              charAt(19)+1* sInput.charAt(21)+7*sInput.charAt(22)+3*sInput.charAt
              (23)+1*sInput.charAt(24)+7*sInput.charAt(25)+3*sInput.charAt(26)+1*
              sInput.charAt(27))%10) == sInput.charAt(35)) {
            // Return true if everything was calculated correctly
            return true;
          } else {
            return false;
          }
        } else {
          return false;
        }
      } else {
        return false;
      }
    } else {
      return false;
    }
  } else {
    return false;
  }
}
</script>
```

Listing A.3: JavaScript method to check an ISBN

```
<script>
/**
 * The method isbn verifies if sInput is a valid ISBN.
 * It returns ''true'' if the string is valid - otherwise it returns ''false''.
 * @param sInput {String} a <code>String</code> with the ID chard numbers
 */
isbn = function( sInput ) {
  // Defining a regular expression to check the right input format
```

```
9   var pattern = new RegExp("^[0-9]{3}-[0-9]{1}-[0-9]{4}-[0-9]{4}-[0-9]{1}$");
10  // True if pattern matches
11  if (pattern.test(sInput)) {
12    // Checksum verification of character sixteen
13    if (((10-((parseInt(sInput.charAt(0))+parseInt(sInput.charAt(2))+parseInt(
          sInput.charAt(6))+parseInt(sInput.charAt(8))+parseInt(sInput.charAt(11))+
          parseInt(sInput.charAt(13)))+3*(parseInt(sInput.charAt(1))+parseInt(
          sInput.charAt(4))+parseInt(sInput.charAt(7))+parseInt(sInput.charAt(9))+
          parseInt(sInput.charAt(12))+parseInt(sInput.charAt(14))))%10)%10) ==
          sInput.charAt(16)) {
14      // Return true if everything was calculated correctly
15      return true;
16    } else {
17      return false;
18    }
19  } else {
20    return false;
21  }
22 }
23 </script>
```

Bibliography

[1] (2011, March) Alexa Top 500 Global Sites. Alexa Internet, Inc. [Online]. Available: http://www.alexa.com/topsites

[2] (2011, March) HTML5. World Wide Web Consortium. [Online]. Available: http://dev.w3.org/html5/spec/Overview.html

[3] (2001, May) Introduction to CSS3. World Wide Web Consortium. [Online]. Available: http://www.w3.org/TR/css3-roadmap/

[4] Thomas Powell, *AJAX: The Complete Reference*. Mcgraw-Hill Professional, 2008, ch. Introduction to Ajax, p. 3.

[5] Jeremiah Grossman, "Seven Business Logic Flaws That Put Your Website At Risk," October 2007, p. 2. [Online]. Available: http://www.whitehatsec.com/home/assets/WP_bizlogic092407.pdf

[6] Mario Heiderich. (2011, March) html5security - Project Hosting on Google Code. [Online]. Available: http://code.google.com/p/html5security/

[7] (2011, March) ESAPI Assurance - OWASP. [Online]. Available: http://www.owasp.org/index.php/ESAPI_Assurance

[8] OWASP. (2011, March) Project Information: OWASP Enterprise Security API Project. [Online]. Available: http://www.owasp.org/index.php/EASPI

[9] Jeff Williams. OWASP EU Summit 2008 (Portugal) - ESAPI. OWASP. [Online]. Available: http://www.youtube.com/watch?v=QAPD1jPn04g

[10] Karl Franz Fogel, *Producing Open Source Software: How to Run a Successful Free Software Project*. O'Reilly Media, 2005, pp. 240–241.

[11] Chris Schmidt. User:Chris Schmidt. OWASP. [Online]. Available: http://www.owasp.org/index.php/User:Chris_Schmidt

[12] (2011, March) Downloads - owasp-esapi-js - Project Hosting on Google Code. OWASP. [Online]. Available: http://code.google.com/p/owasp-esapi-js/downloads/list

[13] (2011, March) GettingStarted - owasp-esapi-js - Information on installation and basic use - Project Hosting on Google Code. [Online]. Available: http://code.google.com/p/owasp-esapi-js/wiki/GettingStarted

[14] Marcus Niemietz, *Authentication Web Pages with Selenium*. AVM - Akademische Verlagsgemeinschaft München, 2010, ch. DOM-based cross-site scripting, pp. 26–27.

[15] ——, *Authentication Web Pages with Selenium*. AVM - Akademische Verlagsgemeinschaft München, 2010, ch. Cross-site scripting, p. 23.

[16] (2011, Jan) MitigatingDOMBasedXSS - owasp-esapi-js - Mitigating DOM Based XSS with ESAPI4JS - Project Hosting on Google Code. [Online]. Available: http://code.google.com/p/owasp-esapi-js/wiki/MitigatingDOMBasedXSS

[17] Ahmad-Reza Sadeghi, Biljana Cubaleska, "Introduction to System Security," in *System security*,

2010, p. 64.

[18] Martín Adabi, Roger Needham, "Prudent Engineering Practice for Cryptographic Protocols," 1995.

[19] OWASP, "OWASP TOP 10 - 2010," 2010, p. 6. [Online]. Available: http://owasptop10.googlecode.com/files/OWASP%20Top%2010%20-%202010.pdf

[20] Anders Johansen, "Probing human response times," 2008. [Online]. Available: http://arxiv.org/pdf/cond-mat/0305079v2

[21] Jason Voegele. (2011, March) Programming Language Comparison. [Online]. Available: http://www.jvoegele.com/software/langcomp.html

[22] Messaoud Benantar, *Access Control Systems: Security, Identity Management and Trust Models*. Springer, 2005, ch. Elemens of System Security, pp. 3–4.

[23] (2011, March) owasp-esapi-js - Project Hosting on Google Code. [Online]. Available: http://code.google.com/p/owasp-esapi-js/

[24] Mario Heiderich, Eduardo Alberto Vela Nava, Gareth Heyes, *Web Application Obfuscation*. Syngress Media, 2010, ch. Introduction, p. 2.

[25] M. A. Bishop, *Computer Security: Art and Science*. Macmillan Technical Publishing, 2002, ch. Building Systems with Assurance, p. 498.

[26] OWASP. (2011, March) Threat Risk Modeling - OWASP. [Online]. Available: http://www.owasp.org/index.php/Threat_Risk_Modeling

[27] Ahmad-Reza Sadeghi, Biljana Cubaleska, "Introduction to System Security," in *System security*, 2010, p. 86.

[28] (2011, March) PHP: Hypertext Preprocessor. The PHP Group. [Online]. Available: http://www.php.net/

[29] (2008, November) <malicious></markup>: HTML Form Controls reviewed: Index. [Online]. Available: http://maliciousmarkup.blogspot.com/2008/11/html-form-controls-reviewed.html

[30] (2011, March) New in JavaScript 1.8.5 - MDC Doc Center. Mozilla Developer Network. [Online]. Available: https://developer.mozilla.org/en/JavaScript/New_in_JavaScript/1.8.5

[31] (2011, March) Browser compatibility - defineProperty - MDC Doc Center. Mozilla Developer Network. [Online]. Available: https://developer.mozilla.org/en/JavaScript/Reference/Global_Objects/Object/defineProperty#Browser_compatibility

[32] (2011, March) Configurable attribute - defineProperty - MDC Doc Center. Mozilla Developer Network. [Online]. Available: https://developer.mozilla.org/en/JavaScript/Reference/Global_Objects/Object/defineProperty#Configurable_attribute

[33] (2011, March) Claude Shannon. Soylent Communications. [Online]. Available: http://www.nndb.com/people/934/000023865/

[34] Jack P. Hailman, *Coding and Redundancy: Man-Made and Animal-Evolved Signals*. Harvard Univ Pr, 2008, ch. Redundancy, p. 165.

[35] (2011, March) jQuery: The Write Less, Do More, JavaScript Library. [Online]. Available: http://jquery.com

[36] (2011, March) README.textile at master from chrisisbeef/jquery-encoder - GitHub. [Online]. Available: https://github.com/chrisisbeef/jquery-encoder/blob/master/README.textile

[37] Chris Schmidt. (2011, March) chrisisbeef's Profile - GitHub. [Online]. Available: https:

//github.com/chrisisbeef

[38] ——. (2011, Feb) Yet Another Developer's Blog: jQuery-Encoder updated. [Online]. Available: http://yet-another-dev.blogspot.com/2011/02/jquery-encoder-updated.html

[39] OWASP. (2011, March) ClickjackFilter (ESAPI 2.0 rc10 2.0_rc10 API). [Online]. Available: http://owasp-esapi-java.googlecode.com/svn/trunk_doc/latest/org/owasp/esapi/filters/ClickjackFilter.html

[40] Marcus Niemietz, "UI Redressing: Attacks and Countermeasures Revisited," Jan 2011, p. 8. [Online]. Available: http://ui-redressing.mniemietz.de/uiRedressing.pdf

[41] Sophos. (2010) Facebook Worm - Likejacking. [Online]. Available: http://nakedsecurity.sophos.com/2010/05/31/facebook-likejacking-worm/

[42] Marcus Niemietz, "UI Redressing: Attacks and Countermeasures Revisited," Jan 2011, p. 11. [Online]. Available: http://ui-redressing.mniemietz.de/uiRedressing.pdf

[43] Eddy Bordi. (2010, August) Proof of Concept - CursorJacking (noScript). [Online]. Available: http://static.vulnerability.fr/noscript-cursorjacking.html

[44] Marcus Niemietz, "UI Redressing: Attacks and Countermeasures Revisited," Jan 2011, p. 7. [Online]. Available: http://ui-redressing.mniemietz.de/uiRedressing.pdf

[45] M. Corporation. (2010) The X-Frame-Options response header. [Online]. Available: https://developer.mozilla.org/en/the_x-frame-options_response_header

[46] (2011, March) Validator - JavaDoc - org.owasp.esapi Interface Validator. OWASP. [Online]. Available: http://owasp-esapi-java.googlecode.com/svn/trunk_doc/1.4.4/org/owasp/esapi/Validator.html

[47] (2011, March) EuropeBanks.Info - IBAN Account Numbers and Payments in Europe. [Online]. Available: http://www.europebanks.info/ibanguide.htm

[48] D. Bundesbank. (2011, March) Bundesbank - Zahlungsverkehr - Single Euro Payments Area (SEPA). [Online]. Available: http://www.bundesbank.de/zahlungsverkehr/zahlungsverkehr_sepa.en.php#sepa3

[49] (2011, March) IBANdocs - International Bank Account Number (IBAN). TBG 5 Finance. [Online]. Available: http://www.tbg5-finance.org/ibandocs.shtml/

[50] (2011, March) International Bank Account Number - Wikipedia, the free encyclopedia. Wikipedia. [Online]. Available: http://en.wikipedia.org/wiki/International_Bank_Account_Number#List_of_valid_IBANs_by_country

[51] (2011, March) Identity document - Wikipedia, the free encyclopedia. Wikipedia. [Online]. Available: http://en.wikipedia.org/wiki/Identity_document

[52] New ID card. Bundesdruckerei. [Online]. Available: http://www.bundesdruckerei.de/en/products/products_idSystem/idSystem_documents/documents_idcard/index.html

[53] (2007, June) File:MustermannPA.jpg - Wikimedia Commons. Wikimedia. [Online]. Available: http://commons.wikimedia.org/wiki/File:MustermannPA.jpg

[54] (2011, March) Prüfziffern: Deutscher Personalausweis. [Online]. Available: http://www.pruefziffernberechnung.de/P/Personalausweis-DE.shtml

[55] (2008, December) ISO 2108:2005 - Information and documentation – International standard book number (ISBN). International Organization for Standardization. [Online]. Available: http://www.iso.org/iso/catalogue_detail?csnumber=36563

[56] Hartmut Walravens, *ISBN - International Standard Book Number. Bibliography: Literature on the ISBN and ISMN (International Standard Music Number) from all over the world*. Simon Verlag für Bibliothekswissen, 2010, ch. Preface, p. 9.

[57] (2011, March) Sichere Webanwendungen. Das Praxisbuch - Das Buch von Galileo Computing. Galileo Computing. [Online]. Available: http://www.galileocomputing.de/katalog/buecher/titel/gp/titelID-1784

[58] (2011, March) International Standard Book Number - Wikipedia, the free encyclopedia. Wikipedia. [Online]. Available: http://en.wikipedia.org/wiki/International_Standard_Book_Number#ISBN-13

[59] (2011, March) EUROPA site - Validation. European Commission's Taxation and Customs Union Directorate-General. [Online]. Available: http://ec.europa.eu/taxation_customs/vies/faqvies.do

[60] (2011, March) UPC Bar Code - Universal Product Code. The New York Times Company. [Online]. Available: http://sbinfocanada.about.com/od/insurancelegalissues/g/upc.htm

[61] Chris Schmidt. (2010, January) Roadmap - owasp-esapi-js - OWASP ESAPI4JS Roadmap - Project Hosting on Google Code. OWASP. [Online]. Available: http://code.google.com/p/owasp-esapi-js/wiki/Roadmap