Security Compass

WHITEPAPER

SECURE WEB APPLICATION FRAMEWORK MANIFESTO

Authors: Tom Aratyn, Sahba Kazerooni, Patrick Szeto, & Rohit Sethi

Version 0.08

http://www.securitycompass.com
labs@securitycompass.com

DRAFT

CONTENTS

DRAFT

DRAFT

1 MISSION STATEMENT

The Secure Web Application Framework Manifesto is a document detailing a specific set of security requirements for developers of web application frameworks to adhere to. The manifesto centers around the following beliefs:

- Frameworks that are ‘secure by default’ will yield a dramatic reduction in the number of common web application security vulnerabilities.
- Application security experts should provide, on a regularly basis, updated guidance to framework developers on how to incorporate mechanisms to avoid newly discovered vulnerabilities.

DRAFT

2 INTRODUCTION

Developers are increasingly relying on scaffolding-based systems like Rails and Django to build applications. The number of web application frameworks, scaffolding or otherwise, is constantly growing and it's becoming increasingly clear that securing these frameworks will be a major boon for the future of secure web applications.

In the words of Jeff Williams, we have plenty of "painkillers" for web application framework developers to follow such as lists of vulnerabilities to avoid. The Security Analysis of Core J2EE Patterns was our first attempt at providing "vitamins" or positive advice to framework developers on what they should do to incorporate security into their design. Recognizing that many developers are gravitating to leveraging web application frameworks, we decided it was time to provide a list of positive features that these frameworks should include.

This "Secure Web Application Framework Manifesto" must, of course, be a living document. At any given point, it should provide a minimum baseline of what a web application framework should include to appeal to security-conscious developers. We contend that if such a web application framework is broadly adopted, it will have far reaching effects into web application security.

Adhering to the manifesto is only a starting point. Developers still can, and surely will, introduce vulnerabilities not covered by the manifesto; especially those pertaining to their core domain such as fine-grained authorization. Secure-by-default frameworks are compliments but not substitutes for developer security awareness.

The Manifesto is not an exhaustive specification. It is designed to provide a minimum standard for frameworks to adhere to in order to facilitate development of secure web applications. Some of these features will come with tradeoffs in performance or usability. Security features should be turned on by default with the option to turn them off explicitly. In some cases, the usability or performance trade-offs may be so great that framework developers will turn the features off by default. Such decisions should be the exception and not the norm.

2.1 ACKNOWLEDGEMENTS

We owe a debt of gratitude to Arshan Dabirsiaghi and the entire OWASP Intrinsic Security Working Group. The ISWG aims to measure the security of various frameworks – the inverse of the Secure Web Application Framework Manifesto, which aims to provide the measuring stick itself. Although we initially started this manifesto independent of their work, cross referencing their requirements helped us identify gaps in the Manifesto.

Similarly, James Landis was kind enough to provide us with a similar body of work he put together in defining requirements for a secure web application framework. His ideas also helped shape the manifesto.

We also would like to thank the following individuals for their insight and support in creating the manifesto:

DRAFT

- Jim Manico
- Dinis Cruz
- James McGovern
- Paco Hope
- Paul Johnston

DRAFT

3 REQUIREMENTS

3.1 INJECTION PREVENTION

Confounding data with executable code is the cause of the most pervasive application security problems: Cross Site Scripting (XSS), SQL injection, buffer overflow and several others. The fundamental problem arises when developers can mix user-supplied or user-influenced data, such as HTTP parameters, with static or system-generated code. The resultant data is then executed or otherwise interpreted by a process which can no longer differentiate the code from the data. An obvious example of this principle at work is SQL injection. Pseudo code:

```
bad_query = "select * from accounts where accountid = '" +
user_supplied_value + "'";

DatabaseTool.executeQuery(bad_query);
```

In this example, the database tool has no way to differentiate which parts of the query variable came from the string literal and which parts came from the user supplied value. Most modern programming languages and development frameworks offer a way around this by offering parameterized queries or prepared statements.

Pseudo code:

```
PreparedStatement good_query = "select * from accounts where accountid
= ?";

good_query.setParameter(1, user_supplied_value)

DatabaseTool.executeParameterizedQuery(good_query);
```

DatabaseTool has a few ways to protect against the vulnerability in the second example. For example, the tool could pre-compile the string literal and pass the parameters to the database separately. The database is then responsible for not misinterpreting any portion of the user supplied data as SQL code. Such an approach renders SQL injection impossible.

Another approach is to encode unsafe data in a context relevant format. For example: to mitigate against Cross Site Scripting, a secure web application framework could automatically HTML, HTML attribute, cascading style sheet, or JavaScript encode nearly all non alpha-numeric characters depending on the context. The encoding functions in the OWASP ESAPI project for Java serve as an excellent reference for this approach.

DRAFT

3.1.1 Provide Tools that Output Data Which is Safe from Interpretation by Browsers

Requirement Description

Provide tools that take potentially dangerous data, such as user-supplied input, and outputs the data to Hyper Text Markup Language (HTML), Cascading Style Sheet (CSS), or client-side script. The data should be outputted in such a way that all supported web browsers will not interpret the result as including meta-characters for code. In particular, the output should not contain valid HTML markup, CSS code, or client-side script code such as JavaScript. Tag libraries must, by default, employ these tools when outputting user-supplied data.

Relevant Weaknesses

This requirement mitigates the following weaknesses:

- Cross Site Scripting (XSS)

Implementation Suggestions

The most common implementation of this feature is to escape potentially dangerous characters such that they will not be interpreted by the browser as HTML markup, CSS code, or JavaScript. Most implementations use HTML entities, CSS escaping, and JavaScript escaping respectively.

Knowing which form of escaping to use means understanding where the data will be output. The Open Web Application Security Project (OWASP) Enterprise Security Application Programming Interface (ESAPI) for Java project provides multiple encoding functions and requires the developer to select the correct function depending on context.

If a framework is context aware (i.e. understands whether data will be output to HTML, CSS, or JavaScript) then it can automatically select the correct encoding format. Although not always possible, such functionality is ideally suited for template-based server pages such as ASP, JSP, or PHP. See Google's template system.

An important consideration is which characters to encode versus which characters to leave unencoded. Excessive encoding may mean extra performance and transmission costs, whereas under encoding may mean missing dangerous characters and leaving applications susceptible to attack. Where possible, decide upon a whitelist of valid characters, such as characters within the Unicode Alpha or Numeric classes, and encode all other characters.

Documentation Suggestion

In all user manuals, tutorials, demonstration/sample code, and all other documentation, always use these tools to output data to HTML, JavaScript, or CSS.

DRAFT

The only exception should be for functions that must, by design, output valid HTML markup, JavaScript, CSS – for example, a tag that generates “” and “” markers to denote bold text.

DRAFT

3.1.2 Provide Parameterized Query Functionality for SQL Statements

Requirement Description

Provide tools that allow developers to create static SQL String literals with the ability to bind parameters at runtime. This functionality is commonly referred to as Parameterized Queries or Prepared Statements. Databases must not interpret bound parameters as valid SQL escape sequences, such as an apostrophe to delimit a string. Note that that term “parameterized *query*” does not just refer to Select statements, it also refers to other common SQL statements, such as Insert, Update, and Delete

Relevant Weaknesses

This requirement mitigates the following weaknesses:

- SQL injection

Implementation Suggestions

Two common implementations include:

- Pre-compiling the string literal and transmitting the parameters to the database separately. The database is then responsible for maintaining a distinction between the SQL statement and the parameters

- Contextual escaping, similar to the defense described in “3.1.1Provide Tools that Output Data Which is Safe from Interpretation by Browsers”. Note that any such escaping should account for different possible encoding formats of the underlying database

Documentation Suggestion

In all user manuals, tutorials, demonstration **always** use parameterized queries; **never** use dynamic statements consisting of dynamically concatenated string literals and parameters.

DRAFT

3.1.3 Provide Tools that Output Data Which is Safe from Interpretation by XML Processors

Requirement Description

Provide tools that take potentially dangerous data, such as user-supplied input, and outputs the data to XML. The data should be outputted in such a way that an XML validator, parser, or other processor will not interpret the result as including meta-characters for XML code. In particular, the output should not contain XML element, XML attribute, XML comment, CData, Document Type Definition (DTD), XML Stylesheet, preprocessing, or any other XML tags.

Relevant Weaknesses

This requirement mitigates the following weaknesses:

- XML injection

Implementation Suggestions

The most common implementation of this feature is to escape potentially dangerous characters using XML entities and / or numeric character reference. Unlike "3.1.1Provide Tools that Output Data Which is Safe from Interpretation by Browsers", this requirement applies to a single output type, and does not encompass the same complexities associated with Cross Site Scripting mitigation. See the Open Web Application Security Project (OWASP) Enterprise Security Application Programming Interface (ESAPI) for Java project provides an example of XML encoding.

An important consideration is which characters to encode versus which characters to leave unencoded. Excessive encoding may mean extra performance and transmission costs, whereas under encoding may mean missing dangerous characters and leaving applications susceptible to attack. Where possible, decide upon a whitelist of valid characters, such as characters within the Unicode Alpha or Numeric classes, and encode all other characters.

Documentation Suggestion

In all user manuals, tutorials, demonstration/sample code, and all other documentation, always use these tools to output data to XML.

Pseudo code:

```
xmlText =  "<element>" + SafeXMLFunction(userParameter) + "</element>"
```

The only exception should be for functions that must, by design, output valid XML tags – for example, a function that generates standard Simple Object Access Protocol (SOAP) element tags.

DRAFT

3.1.4 Provide Parameterized Query Functionality for XPath Statements

Requirement Description

Provide tools that allow developers to create static XPath String literals with the ability to bind parameters at runtime, similar to Parameterized Queries for SQL. XPath engines must not interpret bound parameters as valid XML escape sequences, such as an apostrophe to delimit a string.

Relevant Weaknesses

This requirement mitigates the following weaknesses:

- XPath injection

Implementation Suggestions

As with SQL Parameterized Queries, two possible implementations include:

• Pre-compiling the string literal and transmitting the parameters to the XPath engine separately. The XPath engine is then responsible for maintaining a distinction between the XPath statement and the parameters (see this article from the Microsoft Developer Network [MSDN])

• Contextual escaping, similar to the defense described in “3.1.1Provide Tools that Output Data Which is Safe from Interpretation by Browsers”. Note that any such escaping should account for different possible encoding formats of the underlying database

Documentation Suggestion

In all user manuals, tutorials, demonstration/sample code, and all other documentation, always use these tools to perform XPath queries.

DRAFT

3.1.5 Provide Parameterized Query Functionality for LDAP Statements

Requirement Description

Provide tools that allow developers to create static Lightweight Directory Access Protocol (LDAP) String literals with the ability to bind parameters at runtime, similar to Parameterized Queries for SQL. LDAP directories must not interpret bound parameters as valid LDAP escape sequences, such as an asterisk to denote a wildcard character .

Relevant Weaknesses

This requirement mitigates the following weaknesses:

- LDAP injection

Implementation Suggestions

As with SQL Parameterized Queries, two possible implementations include:

• Pre-compiling the string literal and transmitting the parameters to the LDAP directory separately. The LDAP engine is then responsible for maintaining a distinction between the LDAP Path statement and the parameters

• Contextual escaping, similar to the defense described in “3.1.1Provide Tools that Output Data Which is Safe from Interpretation by Browsers”. Note that any such escaping should account for different possible encoding formats of the underlying database

Documentation Suggestion

In all user manuals, tutorials, demonstration/sample code, and all other documentation, always use these tools to perform LDAP queries.

DRAFT

3.1.6 Disallow Newline Characters from Untrusted Data in HTTP Response Headers

Requirement Description

For all functions that can modify HTTP response headers, such as a redirect function or the “Set-Cookie” header, disallow newline characters in potentially un-trusted parameters. For example, disallow newline characters inside of a cookie value or URL for a redirect.

Note that the term disallow is purposefully undefined; framework developers should use the best approach to match their needs, such as:

- Strip newline characters out
    - Note that whenever stripping a potentially malicious character, ensure the resuling string is also free from dangerous characters. For example “%%0A0A” would still result in a URL encoded newline character if the “%0A” was stripped out once.
- Cause an error condition
- Replace newline characters with a safe equivalent, such as the literal string “\n” or ”\r”

Common functions that can modify HTTP response headers include:

- Setting HTTP status code
- Setting URL for a redirect
- Setting cookie name, value, path, secure flag, HttpOnly flag, or expiry

Framework developers may wish to provide an option to turn this functionality off for compatibility, performance, or other reasons; however, it must be turned on by default for new applications. Store this configuration setting in a centralized, auditable security settings file.

Relevant Weaknesses

This requirement mitigates the following weaknesses:

- HTTP response splitting

Implementation Suggestions

Use a regular expression to replace carriage returns and line feeds with safe equivalents; namely, the string literals “\n” and “\r”.

Documentation Suggestion

State that affected methods may not work as intended if users intentionally supply newline characters into HTTP response splitting. Indicate that solution addresses Http Response Splitting and, if appropriate, describe how to turn the feature off along with an appropriate warning of the resultant risk.

DRAFT

3.1.7 Provide Option to Disallow Newline Characters in Text File Logging

Requirement Description

Provide an option in logging functionality to automatically disallow writing newline characters in text file-based logs. Modifying all log statements may incur significant overhead, thus this requirement is an option rather than a default setting. Store this configuration setting in a centralized, auditable security settings file.

For HTML-based logging use the tools described in “3.1.1Provide Tools that Output Data Which is Safe from Interpretation by Browsers”. For XML-based logging use the tools described in “3.1.3 Provide Tools that Output Data Which is Safe from Interpretation by XML Processors”.

Relevant Weaknesses

This requirement mitigates the following weaknesses:

- Log injection

Implementation Suggestions

Use a regular expression to replace carriage returns and line feeds with safe equivalents; namely, the string literals “\n” and “\r”.

Documentation Suggestion

Provide clear instructions on how to modify this setting, as well as the security implications of keeping the default value as turned off.

DRAFT

3.2 INPUT VALIDATION

3.2.1 Provide Configurable Validation for All Forms of User-Supplied Input

Requirement Description

Provide a mechanism to validate the content of all user-supplied input without directly modifying other application code. For example, provide a configuration file that allows users to supply regular expressions to validate HTTP parameters for any page.

The types of input to validate must include, at a minimum:

- HTTP request parameter names
- HTTP request parameter values
- HTTP request header names
- HTTP request header values
- URLs
- Cookie names
- Cookie values
- SQL statement results
- Input from a proprietary format, such as Flash Action Message Format
- Remotely accessible Application Program Interfaces (APIs), such as Simple Object Access Protocol (SOAP) or Representational State Transfer (REST) endpoints

Where possible, store the validation configurations setting in a centralized, standard location. Ideally, developers / administrators should be able to make changes to validation logic during application deployment rather than requiring a rebuild.

Optionally, provide a tool that allows security auditors to easily determine which forms of input the application is currently validating through the validation engine.

Optionally, provide sample regular expressions useful for whitelist validation of common data types such as phone numbers, zip codes, etc.

Relevant Weaknesses

This requirement is part of a defense in depth strategy. Although providing configurable validation does not, in itself, mitigate specific vulnerabilities it does help provide defense in depth. Input validation is particularly useful as additional defense for injection attacks, such as:

- Cross site scripting (XSS)
- SQL injection
- XML injection
- XPath injection

DRAFT

- LDAP injection
- HTTP response splitting
- Log injection

In addition, input validation can help against undiscovered input / injection attacks as well as attacks on downstream systems.

Implementation Suggestions

Use a configuration file similar to the Apache Struts Validator plug-in. Note that the Validator plugin only provides input validation for form fields; a secure framework should provide a similar mechanism for *all* forms of user-supplied input.

The architecture of the validation configuration should follow the default application architecture. For example, if the default application uses a single HTML page with many different command parameters to represent different transactions, then the validation framework should allow developers to specify different validation for different commands – distinguishing the “command=” parameter from other parameters.

Documentation Suggestion

Demonstrate examples of the validation logic as part of normal application development. Include validation examples in user manuals, tutorials, demonstration/sample code, and all other documentation.

DRAFT

3.2.2 Use Whitelist Validation for File Paths and Names in File Handling Functionality

Requirement Description

For each supported operating system, only allow legal characters in the file paths and file names in the file handling functionality such as open and save. Disallow, for example, null characters. This functionality is particularly important since file handling often relies on lower-level operating system commands. Strings in operating system functions may be null-terminated even if framework strings are not null-terminated.

Relevant Weaknesses

This requirement mitigates the following weaknesses:

- Insecure direct object reference

Implementation Suggestions

N/A

Documentation Suggestion

Document how this feature works, what impact it may have on file handling, and how to turn it off along with an appropriate warning of the resultant risk.

DRAFT

3.2.3 Specify an Encoding Format for Every HTTP Response Page

Requirement Description

Assign a consistent encoding format such as UTF-8 to all HTTP response pages unless there is a specific reason to use a different format. Allow developers to define the default encoding format.

Relevant Weaknesses

This requirement mitigates the following weaknesses:

- Cross site scripting (XSS)
  - Obscure cross-site scripting vectors

Implementation Suggestions

Django provides a configuration option for default character set.

Documentation Suggestion

Document how this feature works, what impact it may have on internationalization, and how to turn it off along with an appropriate warning of the resultant risk.

DRAFT

3.2.4 Do Not Accept Characters with Illegal Byte Sequences or Overly Long Forms for a Given Encoding

Requirement Description

Overly long and malformed characters in variable length encoding formats such as UTF-8 can be used to bypass filters and may sometimes be translated to the proper format after sanitization by a different component or application. Only accept legal character sequences.

Relevant Weaknesses

This requirement mitigates the following weaknesses:

- Filter bypass

Implementation Suggestions

The W3C provides a regular expression to validate UTF-8 characters.

Documentation Suggestion

In reference documentation describe that this feature exists.

DRAFT

3.2.5 Provide Function to Detect HTTP Parameter Tampering

Requirement Description

In some cases end users should not be able to modify certain parameters, such as some hidden form fields. Provide a server-side mechanism that detects tampering of “read-only” parameters without the overhead of storing these parameters on the server.

The framework will not necessarily know about *all* read-only parameters; however, the framework should be able to automatically identify *some* read-only parameters (e.g. hidden form fields with static values), and allow individual developers to identify other read-only parameters. If applied transparently, this feature may break functionality, so provide options to turn the feature on for specific forms or across the application.

Relevant Weaknesses

This requirement mitigates the following weaknesses:

- Parameter Manipulation

Implementation Suggestions

The .Net framework provides a defense in the form of an HMAC for ViewState. The HMAC solution:

- Takes a hash of read-only fields in a form prior to sending them to the client
- Encrypts that hash with a secret key stored on the server
- Adds the hashed and encrypted value as an additional hidden field in the form
- Upon form submission, rehashes and re-encrypts the read only client-supplied parameters and compares the hash with the client-supplied HMAC parameter. Any difference indicates that one or more of the read only parameters were tampered with

Documentation Suggestion

Provide detailed documentation on how to enable this feature and how it works. Include (if applicable) how the framework creates and stores cryptographic keys, how developers can change cryptographic algorithms, and how to configure the feature to work in load balanced environments. Always enable the feature for read-only form fields in samples and tutorials.

DRAFT

3.2.6 Automatically Generate Content Security Policy (CSP) Headers

Requirement Description

Mozilla proposed CSP to help protect against Cross Site Scripting. Although CSP, at the time of this writing, is not fully implemented in most browsers, a secure web application framework should proactively provide this control for when CSP becomes standard.

CSP allows developers to specify which domains a web application allows to host its scripts. A browser that complies with CSP will, when instructed to, only run scripts from the whitelisted domains and avoid executing inline or event handling HTML attribute scripts. CSP also helps protect against Clickjacking by specifying "which sites may embed contents from my site".

Automatically generate CSP headers that restrict script access to the application's domain and prevent inline / event handling HTML attribute scripts unless necessary. Provide tools to easily extend the list of white-listed domains when required. By default, disallow all other sites from being embedded within the application's contents unless necessary.

Relevant Weaknesses

This requirement mitigates the following weaknesses:

- Cross Site Scripting (XSS)
- Clickjacking

Implementation Suggestions

N/A

Documentation Suggestion

Document how this feature works, what impact it may have on embedded scripts, and how to turn it off along with an appropriate warning of the resultant risk.

DRAFT

3.2.7 Automatically Generate Origin Headers

Requirement Description

Mozilla proposed the Origin Header to protect against Cross Site Request Forgery (CSRF). Supporting clients send information about the originating domain of each request to the server.

Where possible, verify that the origin of a request is from the expected domain. In particular, verify that application form fields originate from the application's domain. Generate errors for requests with invalid origins.

Relevant Weaknesses

This requirement mitigates the following weaknesses:

- Cross Site Request Forgery

Implementation Suggestions

N/A

Documentation Suggestion

Document how this feature works, what impact it may have on application integration, and how to turn it off along with an appropriate warning of the resultant risk.

DRAFT

3.2.8 Specify a Default Maximum Payload Size for All Inbound Interfaces

Requirement Description

Check the size of payloads from all inbound interfaces prior to processing. If the payload size exceeds a default maximum generate an error. Allow developers to change the default size and turn off the feature.

At a minimum, provide payload size checks for:

- HTTP Requests
  - Optionally, provide different configuration for file upload functions to allow for larger payloads
- XML requests (e.g. Simple Object Access Protocol [SOAP])
- Any other programmatic interface (e.g. Remote Method Invocation over Internet Inter-Orb Protocol [RMI IIOP])

Relevant Weaknesses

This requirement mitigates the following weaknesses:

- Denial of Service

Implementation Suggestions

Web servers often provide a maximum configurable HTTP request body size. See the Apache web server LimitRequestBody directive.

Documentation Suggestion

Document how this feature works, what impact it may have on large payloads, and how to turn it off along with an appropriate warning of the resultant risk.

DRAFT

3.3 AUTHENTICATION AND AUTHORIZATION

3.3.1 Enforce Default Deny Policy for Framework Managed Authorization

Requirement Description

Some frameworks elect to provide managed authorization services, such as determining whether a user has sufficient privileges to view a specific page. Ensure that managed authorization services always deny access by default unless explicitly instructed otherwise.

Relevant Weaknesses

This requirement mitigates the following weaknesses:

- HTTP Verb Tampering

Implementation Suggestions

The Spring Security authorization module employs default deny using the RoleVoter for role-based access control.

Documentation Suggestion

Document the default deny behavior when describing the authorization functionality. Provide instructions on how developers can grant access to more users when necessary. If the framework provides a default-accept option, strongly discourage developers from using it and explain the associated risks.

DRAFT

3.3.2 Provide Indirect Object Reference Functionality

Requirement Description

Provide functionality that creates and translates indirect references for a specific file, a set of files, or all files in a particular directory or directories.

Applications often allow users to access sensitive resources such as user-specific files from the application server. Direct object references use the actual file name (e.g. “file=statement1.pdf”) whereas indirect object references provide an independent identifier that the application later translates into an actual filename (e.g. “file=a”, where ‘a’ later translates to statement1.pdf).The problem with the former method is that attackers can sometimes access files that they shouldn’t (e.g. “file=../config.xml”). An indirect object reference renders such an attack impossible because the application only provides access to a specified set of files (e.g. all files in a particular directory, or a predefind list of individual files). Unfortunately, the complexity of creating an indirect object reference for each file that is to be accessed by the end user means that many developers end up favoring direct object references. Providing functionality to automate this task incentivizes developers to rely on indirect object references.

Note that this control applies specifically to resources that require access control. Publicly-accessible static content such as JavaScript or Cascading Style Sheet files that are normally stored on web servers do not necessarily need this protection. On web servers, use operating system or server controls to prevent forcible Path Traversal attacks.

Relevant Weaknesses

This requirement mitigates the following weaknesses:

- Insecure Direct Object Reference

Implementation Suggestions

See the ESAPI Java AccessReferenceMap and .Net’s Web Resource mechanism for examples of this functionality.

Documentation Suggestion

In all user manuals, tutorials, demonstration/sample code, and all other documentation, always use these tools for accessing server-side files except for publicly-accessible static content (e.g. common JavaScript libraries).

DRAFT

3.3.3 Provide a Function That Hashes and Salts Input with Random Bytes

Provide all the functionality necessary for a developer to implement secure authentication. In particular, authentication should use a secure hashing algorithm salted with a fixed length random byte sequence (see). Both the hashing algorithm and salt length should be configurable in case a particular hashing function is defeated in the future. Secure web application frameworks should default to stronger, slower hashing algorithms (e.g. SHA-2) instead of fast algorithms (e.g. MD5 and SHA-1) to mitigate the risk of off-line brute forcing.

Requirement Description

Provide the following two functions:

1) A function that hashes user input using a configurable, strong hashing algorithm (e.g. SHA-2) and adds a configurable-length random salt value
2) A function that checks equality of a plaintext value with a hashed, salted value derived from function 1)

Developers can use these two functions respectively to facilitate securely storing a new password (e.g. new user registration or password reset) and to authenticate a user against a securely stored password.

Relevant Weaknesses

This requirement mitigates the following weaknesses:

- Rainbow tables
- Insecure cryptographic storage

Implementation Suggestions

Jasypt exposes `passwordEncryptor.encryptPassword`() for function 1) and `passwordEncryptor.checkPassword` for function 2). See this explanation for details on how Jasypt stores the salt value and uses it for password comparisons.

Documentation Suggestion

In all user manuals, tutorials, demonstration/sample code, and all other documentation, always use these functions for new user registration, password change, and password-based authentication.

DRAFT

3.4 SESSION MANAGEMENT

3.4.1 Use Cryptographically Secure Random Numbers for Session IDs

Requirement Description

Create session IDs from Cryptogaphically Strong Random Number Generators such as Java's SecureRandom rather than a pseudo random number generator like the rand() function in C.

Relevant Weaknesses

This requirement mitigates the following weaknesses:

- Insufficient entropy in pseudo random number generators

Implementation Suggestions

Tomcat uses SecureRandom numbers for Session IDs by default.

Documentation Suggestion

Describe how the framework generates session IDs in documentation.

DRAFT

3.4.2 Provide Automatic Anti-CSRF Tokens

Requirement Description

Many web application frameworks create or render links and pages derived from form submission pages. Provide an option to transparently add and validate anti-CSRF tokens to form submissions where possible.

Relevant Weaknesses

This requirement mitigates the following weaknesses:

- Cross Site Request Forgery

Implementation Suggestions

Django provides this functionality optionally.

Documentation Suggestion

Where possible, turn this feature on in all user manuals, tutorials, demonstration/sample code, and all other documentation. Explain how the feature works and the risk associated with not using it.

DRAFT

3.4.3 Automatically Reset Session IDs After Authentication

Requirement Description

Change the Session ID of a user after successful authentication. Note that this feature requires knowledge of when authentication occurs. Such knowledge is trivial in framework-managed authentication but more difficult if developers elect to use custom or third party authentication. Provide a hook for the developer to tell the framework when authentication occurs in cases where the framework can't make that determination automatically (e.g. `user.hasAuthenticated()`).

If developers can associate server-side state with a session then retain that state when the session ID changes.

In some cases, particularly when working with legacy components, changing session IDs after authentication may break functionality. Provide an option to disable this functionality.

Relevant Weaknesses

This requirement mitigates the following weaknesses:

- Session fixation

Implementation Suggestions

Spring Security has built-in session fixation defense.

Documentation Suggestion

Always keep this functionality enabled on in all user manuals, tutorials, demonstration/sample code, and all other documentation. Explain how the feature works and the risk associated with not using it.

DRAFT

3.4.4 Apply HttpOnly Flag to Session ID Cookie by Default

Requirement Description

Append the "HttpOnly" flag to session cookies by default, with an option to turn that feature off.

Relevant Weaknesses

This requirement mitigates the following weaknesses:

- Cross Site Scripting (XSS)

Implementation Suggestions

The .Net framework provide the ability to add the HttpOnly flag to cookie flags, although the feature isn't enabled by default.

Documentation Suggestion

Always keep this functionality enabled on in all user manuals, tutorials, demonstration/sample code, and all other documentation. Explain how the feature works and the risk associated with not using it.

DRAFT

3.4.5 Provide Configuration Option to Apply Secure Flag to Session ID Cookie

Requirement Description

Most development frameworks make the default assumption that the application works over plaintext HTTP. In cases where the framework can be sure that the application uses SSL for the entire session (e.g. if the application container has SSL enabled), append the "secure" flag to session cookies by default, with an option to turn that feature off.

For cases where the framework cannot be sure that the application uses SSL for the entire session (e.g. a separate hardware device provides SSL and proxies plaintext HTTP to the application server), provide a simple configuration option for developers to add the "secure" flag to session cookies.

Relevant Weaknesses

This requirement mitigates the following weaknesses:

- Session hijacking

Implementation Suggestions

The .Net framework provide the ability to add the secure flag to cookies, although the feature isn't enabled by default.

Documentation Suggestion

Always keep this functionality enabled on in all user manuals, tutorials, demonstration/sample code, and all other documentation. Explain how the feature works and the risk associated with not using it.

DRAFT

3.4.6 Provide Configurable Inactive and Absolute Session Timeouts

Requirement Description

Provide an option to define both inactive (i.e. after a period of inactivity) and hard/absolute session timeout (i.e. period of time, regardless of amount of activity). Provide default values for both values. Provide an option to turn either or both timeouts.

Relevant Weaknesses

This requirement mitigates the following weaknesses:

- Session hijacking

Implementation Suggestions

Java Servlet containers provide configurable inactive timeout values.

Documentation Suggestion

Explain how both features work and the risk associated with not using them.

DRAFT

3.4.7 Provide a Configuration Option to Tie Session IDs to an IP Address, Subnet, or a List of IP Ranges

Requirement Description

Some application developers opt to correlate a session ID to the client's IP address. After session generation, the application verifies that each request comes from the expected IP address thereby mitigating the risk of session hijacking. Provide an option to seamlessly deliver this functionality.

In practice, session IP correlation on large networks is difficult if not impossible due to a variety of networking features – in particular, proxy servers such as AOL proxy. Provide options to help address this by, for example, allowing developers to specify a subnet length and verifying that each request comes from the same subnet. For example, if a developer configures session subnet correlation with a 24 bit subnet, then the application should permit requests from *10.1.1*.3 and *10.1.1*.5 to access the same session but it should not allow requests from *10.1.2*.3 to access the same session.

To deal with known proxy servers, such as AOL proxy, the framework should also allow developer to specify one or more lists of IP ranges. If a clients IP falls into one of ranges then ensure that all future requests for that same session come from the same list. For example, if one request comes from the set of AOL proxy IPs then all future requests for that session should come the AOL proxy IPs.

Turn this feature off by default. Each application may require considerable time to configure for this feature to work properly.

Relevant Weaknesses

This requirement mitigates the following weaknesses:

- Session hijacking

Implementation Suggestions

N/A

Documentation Suggestion

Document how the feature works and how to turn it on. Provide a tutorial or other guidance on how to setup this feature for an application such that it doesn't break availability.

DRAFT

3.5 XML SPECIFIC

3.5.1 Disable the Following Unsafe Features by Default

Requirement Description

Over the years, security researchers have discovered several vulnerabilities in XML libraries – particularly parsers and validators. Disallow dangerous functionality be default, namely:

- External entity resolution
- DTDs defined internally within XML files
- XML Stylesheet Language Transforms (XSLTs) processing instructions within an XML document's prolog
- XSLT extensions that provide direct access to the operating system, such as Java runtime objects or .Net System.Diagnostics.Process
    - Ideally, take a default deny approach to XSLT extensions and only allow known safe extensions with the option to turn on other extensions

Relevant Weaknesses

This requirement mitigates the following weaknesses:

- External entity attacks
- XSLT command injection
- XML bombs

Implementation Suggestions

N/A

Documentation Suggestion

Always have the unsafe features turned off in sample code, unless explicitly necessary. Explain the potential risk of turning any of these features on.

DRAFT

3.6 CRYPTOGRAPHY

3.6.1 Provide Tools for Transparent Database Encryption

Requirement Description

Provide configuration options to seamlessly encrypt columns within a database when the framework handles database interaction. Object Relational Mapping (ORM) libraries in particular should allow developers to configure column-level encryption.

See “Encrypt Passwords and Keys Stored in Configuration Files” for more information on how to protect the encryption key.

Relevant Weaknesses

This requirement mitigates the following weaknesses:

- Insecure cryptographic storage

Implementation Suggestions

Hibernate in Java provides seamless, configurable database encryption.

Documentation Suggestion

Document how the feature works and how to turn it on. Provide a tutorial or other guidance on how to use this feature.

DRAFT

3.6.2 Provide Configurable Cryptographic Algorithms

Requirement Description

Hard-coding specific encryption algorithms and parameters such as key size may leave applications vulnerable to common attacks if a particular algorithm is ever compromised. Allow developers to configure the algorithm and parameters such as key strengths.

Favor modes with secure random Initialization Vectors (IVs) rather than modes without IVs such as Electronic Code Book (ECB).

Relevant Weaknesses

This requirement mitigates the following weaknesses:

- Insecure cryptographic storage

Implementation Suggestions

See Java's security provider architecture.

Developers in non-compiled languages such as PHP, Ruby, and Python often favor scripts written in that programming language rather than static configuration files. Frameworks written in such languages should decouple application code from specific encryption algorithms, either by introducing a static configuration file or calling a utility class (e.g. HashingUtility.hash() rather than SHA2.hash()).

Documentation Suggestion

Clearly explain how to configure cryptographic algorithms. Provide strong default options (such as NIST approved algorithms and parameters).

DRAFT

3.6.3 Follow the TLS Protection Cheatsheet for TLS/SSL Implementations

Requirement Description

Attackers have discovered several attacks on TLS/SSL implementations and X509 certificates: downgrade attacks, plaintext injection during renegotiation null prefix attacks, circumventing Online Certificate Status Protocol (OCSP) controls, and several others.

Reuse libraries that already account for these attacks rather than writing new libraries. Note that some of the attacks, such as null prefix attacks, are actually attacks against the client; however, these attacks also apply to the server during mutual authentication.

Providing an exhaustive set of requirements for TLS/SSL is beyond the scope of this manifesto. Consult the Transport Layer Protection Cheatsheet for comprehensive guidance. SSL Labs maintains an SSL Server Rating guide that provides guidelines around certificate type, key size, cipher strength, key exchange algorithm, and protocol.

Relevant Weaknesses

This requirement mitigates the following weaknesses:

- Insecure communications

Implementation Suggestions

N/A

Documentation Suggestion

Clearly explain how to configure TLS. Provide specific guidance on how to deploy a server for optimum TLS/SSL security.

DRAFT

3.7 CONFIGURATION SECURITY

3.7.1 Encrypt Passwords and Keys Stored in Configuration Files

Requirement Description

Web application frameworks often store plaintext system passwords and keys in configuration files. For example, several frameworks use plaintext configuration files for database connection strings, database encryption keys, Lightweight Directory Access Protocol (LDAP) connection strings, keystore passwords, and other values. Attackers who are able to exploit other vulnerabilities are sometimes able to view the contents of files.

Provide native support for encrypted properties in configuration files.

Relevant Weaknesses

This requirement mitigates the following weaknesses:

- Broken authentication (plaintext credential storage)

Implementation Suggestions

Developers will always run into the problem of providing some sort of password or key to decrypt encrypted credentials. While no solution is perfect, frameworks can employ one of several password / key storage options:

- Store a private key, unique for each machine, as a binary file that can only be accessed by the application server. While this control succeeds in preventing attackers from viewing plaintext passwords in configuration files, it does not prevent attackers from first accessing the binary key and then the configuration file using the same exploit. This should be the minimum security option. See Weblogic.
- Store the decryption key / password in a file, similar to the preceding option. Use operating system controls to ensure that file is only accessible by a separate launching process – not the application server. The launching process can then pass in the key / password as a command line argument when launching the application server. This way, a user who exploits the application server may not necessarily have access to the decryption key itself.
- Support passphrases from an environment variable and/or web-form. This solution takes more work and possibly manual intervention but greatly decreases the risk of an attacker being able to find plaintext passwords in configuration files. See Jasypt
- Leverage a distributed service, such as the .Net Data Protection API (DPAPI) User Store key storage. This restricts key access to a particular user, so other users on the same machine (including local administrators) cannot access that key.

DRAFT

Documentation Suggestion

Always use the most secure possible credential storage in all user manuals, tutorials, demonstration/sample code, and all other documentation. Explain how the features work and the risk associated with not using them.

DRAFT

3.8 FILE UPLOAD

3.8.1 File Upload Tools Should Supports Pluggable Anti Malware Scanning Solutions

Requirement Description

Framework-managed file upload tools should facilitate safe file uploads by providing configuration options to support for library-based third party anti-virus scanning solutions, such as Clam AV.

Relevant Weaknesses

This requirement mitigates the following weaknesses:

- Malicious file execution
- Unrestricted file upload

Implementation Suggestions

N/A

Documentation Suggestion

Document how to use the pluggable anti-malware feature. Provide Application Programming Interface (API) details on how third parties can hook into the framework.

DRAFT

3.8.2 File Upload Tool Should Provide Options to Disallow Saving Outside of a Specified Directory

Requirement Description

Framework-managed file upload tools should disallow saving a file outside of a configurable specified directory and, optionally, any subdirectories (see the Unix chroot command).

Relevant Weaknesses

This requirement mitigates the following weaknesses:

- Unrestricted file upload

Implementation Suggestions

You may wish to allow developers to specify absolute paths or relative paths to the application's root. Restrict file upload to a relative path by default (e.g. /app/uploads directory).

Documentation Suggestion

In all user manuals, tutorials, demonstration/sample code, and all other documentation, always use this feature with file upload. Document the risks associated with turning this feature off.

DRAFT

3.8.3 Provide a File Upload Tool that Supports Pluggable Content Validation

Requirement Description

Framework-managed file upload tools should facilitate safe file uploads by providing configuration options to support for library-based third party solutions that validate the contents of a particular file type. For example, a PDF validator might ensure that a given file is indeed a PDF and does not contain any executable code or dangerous PDF extensions.

Relevant Weaknesses

This requirement mitigates the following weaknesses:

- Malicious file execution
- Unrestricted file upload

Implementation Suggestions

N/A

Documentation Suggestion

Document how to use the pluggable validation feature. Provide options to associate different validation libraries for different extensions. Provide Application Programming Interface (API) details on how third parties can hook into the framework.

DRAFT

3.9 MISCELLANEOUS

3.9.1 Provide Security Specific Logs and Log All Attack Points Specified in AppSensor

Requirement Description

Provide a security-specific log and turn it on by default. Automatically log potential attacks using all of the attack points documented in the OWASP AppSensor Project.

Ensure consistent use of event IDs (e.g. SE5 for source change of IP during session). Developers should be able to log to the security-specific log as well.

Relevant Weaknesses

This requirement mitigates the following weaknesses:

- Insufficient application intrusion detection

Implementation Suggestions

See the AppSensor code.

Documentation Suggestion

Explain what the security log is, how it works, how and what to add to it, and the format for log entries. Expose details of the log format so that log analysis / Security Event Manager (SEM) tools can detect potential attacks.

DRAFT

3.9.2 Automatically Generate X-Frame-Options Header

Requirement Description

Browsers such as Internet Explorer 8+ support the X-FRAME-OPTIONS header. Automatically set the X-FRAME-OPTIONS value to DENY by default or SAMEORIGIN if the application requires nested frames from within the same application.

Relevant Weaknesses

This requirement mitigates the following weaknesses:

- Clickjacking

Implementation Suggestions

N/A

Documentation Suggestion

In all user manuals, tutorials, demonstration/sample code, and all other documentation, always use this feature. Document the risks associated with turning this feature off or providing an overly broad policy.

DRAFT

3.9.3 Provide Arithmetic Utilities that Protect Against Integer and Floating Point Overflow and Underflow

Requirement Description

Many programming languages such as Java are vulnerable to Integer and floating point overflow and underflow. Provide libraries that encapsulate basic arithmetic operations (e..g addition, subtraction, multiplication, division) and throw errors / exceptions upon overflow or underflow conditions.

Relevant Weaknesses

This requirement mitigates the following weaknesses:

- Numeric overflow

Implementation Suggestions

N/A

Documentation Suggestion

Always use these encapsulation functions when performing normal arithmetic in all user manuals, tutorials, demonstration/sample code, and all other documentation.

DRAFT

3.9.4 Provide Support for Pluggable Anti-Automation

Requirement Description

Provide a mechanism for application administrators to make use of third-party anti-automation techniques such as CAPTCHA on certain pages. Which type of anti-automation mechanism and the implementation of that technique should be configurable. Provide an Application Programming Interface (API) to allow third party providers to plug-in anti automation into the framework. Using a pluggable architecture will promote loose coupling and allow developers to change anti-automation techniques with minor impact to the rest of the application. Developers should be able to change anti-automation techniques because attackers often find ways to break anti-automation.

Relevant Weaknesses

This requirement mitigates the following weaknesses:

- Brute force attacks
- User enumeration

Implementation Suggestions

N/A

Documentation Suggestion

Always use anti-automation for user registration and forgot password in all user manuals, tutorials, demonstration/sample code, and all other documentation. Document how to change the automation provider. Provide Application Programming Interface (API) details on how third parties can hook into the framework.

DRAFT

3.9.5 Return Generic Error Pages by Default

Requirement Description

Generate error pages devoid of application details such as stack traces by default. In order to facilitate troubleshooting, add detailed error messages to an error log and optionally include a reference number to the log in the error page.

Relevant Weaknesses

This requirement mitigates the following weaknesses:

- Missing error handling

Implementation Suggestions

Developers can implement generic error pages in ASP.Net through configuration.

Documentation Suggestion

Always keep this feature turned on in all user manuals, tutorials, demonstration/sample code, and all other documentation.

DRAFT

3.9.6 Centralized Security Configuration Options

Requirement Description

Many of the requirements in this document require configuration options (e.g. “3.2.1Provide Configurable Validation for All Forms of User-Supplied Input”). Consolidate as many security-relevant configuration options into a single security configuration file. Consolidated security configuration helps facilitate auditing.

Relevant Weaknesses

- N/.A

Implementation Suggestions

N/A

Documentation Suggestion

Describe all security configuration options in documentation.

DRAFT