Web Security Patterns for Analysis and Design

Takao Okubo
Fujitsu Laboratories Ltd.
Kamikodanaka 4-1-1, Nakahara-ku,
Kawasaki, 211-8588 Japan
okubo@jp.fujitsu.com

Hidehiko Tanaka
Institute of Information Security
Tsuruyacho 2-14-1, Kanagawa-ku,
Yokohama, 221-0835 Japan
tanaka@iisec.ac.jp

Abstract
Although security requirements analysis plays a very significant role in secure software development, it is difficult since it requires much security expertise and man-power. Plain and practical security analysis patterns are required. We have presented a visualized analysis approach for eliciting security requirements by extending misuse cases, and found that some of its results can be pattern candidates. This paper proposes 8 new web security analysis patterns with our analysis approach. The proposed patterns give analysts a way to find a proper pattern for a specific security goal. They are related to security solutions, and also procure some security design possibilities. We have applied these patterns to some case studies and evaluated that they are effective for web security analysis.

Keywords analysis, design, security pattern, web application program

1. Introduction
In this paper we present some web security analysis patterns which have been mined through applying our new security analysis approach.

Security issues have become more and more important in software development. In spite of software developers' efforts, many security incidents are reported every day. These incidents occur because security functionalities are not realized properly, or they are disregarded from the beginning.

One of the reasons why security is disregarded is the lack of security aspects. In general software developments, analysts only have to consider the usual behavior of programs: intended usage by proper users. However, in order to elicit security requirements, analysts have to consider abuse or misuse of the functions from the viewpoint of adversaries. This is one of the biggest differences between security analysis and functional analysis. Currently, for security, another special analysis is done by security experts, who have the expertise of what adversaries want and where and how attacks are performed. They do the analysis from the viewpoint of attackers and find what are the threats for the target system, this is called "threat analysis".

Unfortunately, threat analysis is difficult without security expertise. Analysts have to identify in advance potential vulnerabilities which may occur in the programming stage. There are web program-specific vulnerabilities such as cross-site scripting (XSS)(11), cross-site request forgery (CSRF)(10) and injection attacks(9). But current security analysis methods cannot identify systematically such threats in the later stages. It requires a great deal of manpower, meanwhile the number of security experts is limited.

If there exists an efficient security analysis method, analysts can save the manpower for verifying the coverage of threat identification. If there are security analysis patterns, analysts may be able to skip some analysis steps by reusing the patterns.

We have proposed a new security analysis method which enables identification of potential threats in programming and expand the security coverage with misuse cases(7). Although misuse cases are well visualized and efficient for understanding and verification, they lack expression of assets and they are not useful for asset-based threat analysis. Therefore we have extended misuse cases for asset-based analysis. We have applied our approach to the web domain and successfully mined 8 analysis patterns. The contexts of these patterns are clear, so analysis can find the proper patterns from use case diagrams of the target program.

The target audience of the PLoP workshop are web application developers who have some knowledge about web security such as XSS, and also who are not security experts. Security experts may be able to confirm the correctness of the proposed patterns. Developers with little security knowledge may be able to learn that if they specify the contexts of their application, they can find the possible threats and countermeasures automatically by applying the proposed patterns.

Section 2 reviews current security patterns and their issues. In Section 3.1 the efficient security analysis approach with the extension of misuse cases is presented. In the next Section 3.2 new web security patterns mined with Section 3.1’s approach are presented. Section 4 examines the presented patterns with some case studies. Section 5 concludes this paper.

2. Current Security Patterns
In this section, current security patterns are evaluated in terms of web security. Several security patterns are presented in (6). Some of them can be categorized as analysis patterns and the others as design patterns.

1. Security patterns for mechanisms
   Identification and Authentication (I&A), Authorization, Access control and logging are kind of design pattern. These patterns are already well known and used. However, there are more web security vulnerabilities such as injection attacks (SQL injection(9), pass traversals), cross-site scripting(XSS(11)) and cross-site request forgery(CSRF(10)). Current patterns or frameworks do not provide complete remedy for those vulnere-
Design patterns contain another issue in the security domain. Most design patterns do not have so much critical effect on software quality as on security quality. For example, even without applying "Visitor pattern", it is possible to build software with the same behavior as the software with "Visitor pattern". However if "I&A pattern" is not applied, it may be difficult to keep it secure. Therefore, security design patterns should satisfy the following requirements.

- Proper patterns must be found for certain security requirements.
- Their application should be forced.
- Their application should be easily verified.

### 2. Security requirements patterns

As requirements patterns, the steps for eliciting security requirements are presented in (6). They are Security needs identification, asset valuation, threat assessment, Vulnerability assessment, risk determination, etc. Although they are necessary for eliciting requirements(2)(4), they are not enough for analysts to select the right security approach. More instance patterns are required.

As a result of the analysis of current patterns, we have found that security design patterns require analysis in advance for finding the proper context applying the patterns. Security requirements patterns should be the first step.

### 3. Proposed Web Security Patterns

In this section, we propose some new security requirements patterns for web application programs. They are practical instance patterns, so that analysts can easily find the proper patterns for satisfying security requirements of their programs. Moreover, the proposed analysis patterns are strongly related to the security designs (some of which are design patterns). It solves the issue mentioned in the previous section, that proper patterns must be found for certain security requirements.

#### 3.1 A New Analysis Approach with Extended Misuse Case

The proposed patterns have been mined by applying a new security analysis approach(5). The approach and enables expanding security coverage by improving expressiveness of Sindre and Opdahls’ misuse case approach(7).

1. Adding types of mis-actors

   The original misuse case diagram(7) contains only one kind of mis-actor. But security measures must differ depending on the types of mis-actors:
   - (a) unauthorized person
   - (b) other authorized user (not him/herself)
   - (c) him/herself

   User identification and authentication (I&A) is effective as a measure against abuse of use cases for type (a) of mis-actor, but not for type (c). For type (c), another security measure is required. As just described, there is a risk that some of the measures related to a single mis-actor may be missed. Therefore, our new method(5) defines three types of mis-actor as shown in Figure 1.

2. Additional definition of data asset elements

   Although assets (especially data assets) are essential factors for threat analysis(2)(4), they aren’t explicitly described in the original misuse case diagram. Therefore the proposed method adds a definition of data assets as shown in Figure 2. A UML note in Figure 2 represents a data asset interchanged between an actor and a use case, and an arrow represents a direction of the interchange.

3. Adding types of misuse case endpoints

   In the original misuse case diagram(7), an endpoint of a misuse case points only to a use case. Our new approach(5) allows misuse case endpoints on (a) use cases, (b) channels and (c) actors / clients. Figure 3 shows the extended endpoints. This extension aims at rough assessment of the possibility of each threat. In the analysis stage, fine grained threat assessments like attack tree(4) are difficult to apply because details of the system architecture have not been determined yet. However, in order to elicit security requirements, security analysis needs (even rough) threat assessments.

4. Adding security properties on assets

   Security properties of confidentiality (C), integrity (I), availability (A) are added on the asset elements (data assets/ use cases) in the form of ‘stereotype’ of UML.

The proposed approach is also used for describing the patterns graphically.

#### 3.2 Web Security requirements patterns with the New Approach

Proposed security patterns have been mined through the application of our approach to web programs. We have found that some typical threats and solutions are identified in certain patterns of the
use cases and assets. We have formalized them as web security requirements patterns.

- "Abuse" pattern
- "Leaks of sending data" pattern
- "Tampering sending data" pattern
- "Disclosure of shared data" pattern
- "Disclosure of private data" pattern
- "Tampering of receiving data" pattern
- "ID/Password authentication" pattern
- "Session management" pattern

Each pattern involves the context, problem patterns and solution patterns. The context navigates the right context (use case and asset types) that the pattern should be applied. Problem patterns help identifying potential threats in programs. Solution patterns can be used to select proper countermeasures against certain threats.

In the following sections, two patterns will be described in detail. The pattern description uses the same pattern characteristics formats defined in (6). Refer to the appendix for details of the rest of the patterns.

### 3.2.1 "Abuse" pattern

#### Context
This pattern should be applied if it is undesirable that the target use case is "abused". "Abuse" means an unintended use violating availability of the target use case. It represents the following threats or attacks.

- Unauthorized use: use by a person who is not allowed to use the use case.
- Unintended use: use for unintended purposes such as fraud.
- Attacks such as Denial of service (DoS).

Security property "availability" is required for use cases of the target program. This pattern does not care whether there is a data asset or not.

Table 3.2.1 and Figure 4 shows the context. Analysts can find the applicability of this pattern by verifying whether the use-case diagram of their program contains the matching pattern of Figure 4.

#### Problem
See Figure 5, Figure 6 and Figure 7. Consequently, there exist the following threats in the program.

1. Unintended use of the use case by an unauthorized user (including abuse and Denial of Service (DoS) attack).

#### Solution
See Figure 8, Figure 9 and Figure 10. Solutions are the following:

1. I&A is useful for preventing unintended use of the use case by an unauthorized user.
2. Server authentication such as SSL/TLS and providing server confirmation methods (such as Not to hide the address-bar of web browsers) are useful for preventing or detecting spoofing of the use case function.
3. I&A is useful for preventing repudiation by an authorized user.
4. Logging is useful for detecting abuse and repudiation.
Consequences By applying the problem pattern, analysts can easily find all the typical web threats in the general use cases. They can also use it to confirm their result of the threat analysis. By applying the solution pattern, they can identify design of the security functions as countermeasures against the threats. Some of them can be related to the security design pattern (see "See also").

See Also The following patterns are related to this pattern.
- I&A(6)
- Logging(6)

3.2.2 "Leaks of sending data" pattern

Context This pattern can be applied when there are some "confidential" data assets sent from actors to use cases. Table 3.2.2 and Figure 11 shows the context.

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>condition</th>
<th>security</th>
<th>data flow</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use case type</td>
<td>any</td>
<td>any</td>
<td>n/a</td>
</tr>
<tr>
<td>Data asset</td>
<td>required</td>
<td>confidentiality</td>
<td>send</td>
</tr>
</tbody>
</table>

Table 2. "Leaks of sending data" pattern: context.

Problem From Figure 12, we can find that, there exists the following threats in the program.
1. Disclosure sending data on the client enabled by vewing caches.
2. Disclosure sending data on the client enabled by shoulder hacking.
3. Disclosure of the sending information on the channel enabled by man in the middle (MITM) attacks.
4. Disclosure of the sending information on the server enabled by injection attacks(9).
5. Disclosure of the sending information on the server enabled by XSS(11) attacks.
6. Disclosure of the sending information on the server enabled by inappropriate access control.

**Solution** See Figure 13. Solutions are as the following.

1. Disabling caches is useful for preventing disclosure data on the client enabled by viewing caches.
2. Hiding the data is useful for disclosure data on the client enabled by shoulder hacking.
3. Encryption of communication (such as SSL/TLS) is useful for preventing disclosure on the channel enabled by MITM attacks.
4. Countermeasures against injection attacks (such as use of parameter binding mechanism) are useful for preventing disclosure on the server.
5. Countermeasures against XSS (such as sanitizing output) are useful for disclosure on the server.
6. Correct access control are useful for preventing disclosure on the server.

**See Also** The following patterns are related to this pattern.

- Secure Channels (SSL)(6)

4. Case Studies

4.1 Web Bookstore

We had some case studies with applying our patterns to several sample application programs. This paper presents one of these applications, which is a web bookstore such as Amazon.com. It provides three use cases:

1. Search books
   Customers search books they want to buy. They may want to keep the search keywords and search result secret.

2. View/edit personal information
   Customers are required to input their personal information such as name, address and telephone number. They are allowed to view and edit their own information on the web, but not other customers’ information.

3. Order books
   Customers can order books. The order list sent from customers to the server must not be seen or modified by any other persons.

Before applying our pattern, the extended use case to which data assets and security properties are added must be drawn. Figure 14 presents the extended use case diagram. After drawing the diagram, each part of the diagram is compared to the contexts of our patterns. If it matches one of the contexts, the pattern of the context can be applied. In Figure 14, at the “View/edit user information” use case part, “Abuse” pattern (Section 3.2.1), “Leaks of sending data” pattern (Section 3.2.2), “Tampering sending data” pattern (Section A.1), “Disclosure of private data” pattern (Section A.3) and “Tampering receiving data” pattern (Section A.4) can be applied. At the “Search books” part, “Leaks of sending data” pattern (Section 3.2.2) and “Disclosure of private data” pattern (Section A.3) can be applied. At the “Order books” part, “Abuse” pattern (Section 3.2.1), “Leaks of sending data” pattern (Section 3.2.2) and “Tampering
<table>
<thead>
<tr>
<th>Threat</th>
<th>asset</th>
<th>mis-actor</th>
<th>location</th>
<th>solution</th>
<th>solution type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abuse</td>
<td>view/edit personal info</td>
<td>unauthorized person</td>
<td>server</td>
<td>I&amp;A</td>
<td>prevent</td>
</tr>
<tr>
<td>Abuse</td>
<td>view/edit personal info</td>
<td>unauthorized person</td>
<td>server</td>
<td>logging</td>
<td>detect</td>
</tr>
<tr>
<td>Leaks</td>
<td>order books personal info</td>
<td>other person</td>
<td>client</td>
<td>countermeasure against XSS</td>
<td>prevent</td>
</tr>
<tr>
<td>Leaks</td>
<td>search keyword</td>
<td>other person</td>
<td>channel</td>
<td>encryption</td>
<td>prevent</td>
</tr>
<tr>
<td>Leaks</td>
<td>order personal info.</td>
<td>other person</td>
<td>server</td>
<td>countermeasure against injection</td>
<td>prevent</td>
</tr>
<tr>
<td>Disclosure</td>
<td>search result personal info</td>
<td>other person</td>
<td>client</td>
<td>disabling caching</td>
<td>prevent</td>
</tr>
<tr>
<td>Disclosure</td>
<td>search result personal info</td>
<td>other person</td>
<td>channel</td>
<td>encryption</td>
<td>prevent</td>
</tr>
<tr>
<td>Disclosure</td>
<td>search result personal info</td>
<td>other person</td>
<td>server</td>
<td>access control</td>
<td>prevent</td>
</tr>
<tr>
<td>Tampering</td>
<td>order personal info.</td>
<td>other person</td>
<td>client</td>
<td>I&amp;A</td>
<td>prevent</td>
</tr>
<tr>
<td>Tampering</td>
<td>order personal info.</td>
<td>other person</td>
<td>channel</td>
<td>encryption</td>
<td>prevent</td>
</tr>
<tr>
<td>Tampering</td>
<td>order personal info.</td>
<td>other person</td>
<td>server</td>
<td>countermeasure against injection</td>
<td>prevent</td>
</tr>
<tr>
<td>Tampering</td>
<td>order personal info.</td>
<td>other person</td>
<td>server</td>
<td>countermeasure against XSS</td>
<td>prevent</td>
</tr>
<tr>
<td>Tampering</td>
<td>order personal info.</td>
<td>other person</td>
<td>server</td>
<td>countermeasure against CSRF</td>
<td>prevent</td>
</tr>
<tr>
<td>Tampering</td>
<td>order personal info.</td>
<td>other person</td>
<td>server</td>
<td>logging</td>
<td>detect</td>
</tr>
</tbody>
</table>

Table 3. The analysis result of the first iteration.

Sending data pattern (Section A.1) can be applied. The result of the first analysis iteration is presented on the Table 4.1.

In this case it requires the second iteration. In the first iteration another asset “password” has been added to the system, so we have to consider threats to passwords. If analysts choose ID/password authentication and password management, “ID/password” pattern (Section A.5) and “Session management” pattern (Section A.6) may be applied. Therefore, threats and solutions on Table 4.1 are added to the result of the first iteration.

4.2 Discussion

Efficiency of patterns Usual security analysis requires the following steps (6)(2)(4).
1. Use case definition
2. Asset identification
3. Asset valuation
4. Threat identification
5. Threat assessment & risks
6. Security solution elicitation
7. Security solution determination

With the presented patterns, some of the steps 4 to 6 can be skipped by matching the pattern context to the given use cases and assets. Analysts can obtain typical web threats and solutions from patterns. However, the threats in the presented patterns are only typical ones. Analysts may have to check if there are others.

Sufficient condition The case study result of the “web bookstore” points that the presented the sufficiency of threat identification for the presented patterns is good. It enables threats referred in web threat classifications such as (8) to be identified when applying the patterns.

Necessary condition The presented patterns, by themselves, cannot give the reasoning about what kind of threats are required for the given program. The reasoning has to rely on “security expertise” because the patterns are based on the result of human analysis, though it is visualized for easy understanding. To prove the necessary condition, we need more empirical evaluation for our analysis method and patterns.

5. Conclusions

This paper proposed some new requirements patterns for web security. The proposed patterns solves one of the security-related problem: how to find a proper pattern for a certain security goal. Each
context of the pattern gives analysts the information whether it should be applied to their programs or not. Contexts, problems, and solutions are represented visually which helps the understanding of analysts and stakeholders.

The proposed patterns also provide security design possibilities. They appear in the "Solution" characteristics of patterns. Some of the security design candidates, such as I&A are related to the current security design patterns, which are presented in "See also".

Our case study shows that if the use case and asset analysis is done, it is easy for analysts to find and apply our patterns. As a result analysts obtain sufficient threats and suggestion of their solutions.


As mentioned in Section 4.2, though the presented patterns have the ability to identify sufficient threats and solutions, currently it is difficult to prove that the proper threats are identified for the given situation. A more empirical evaluation is required, which is future work.

References

[1] BRAZ, F., FERNANDEZ, E. B., AND VANHILST, M. Eliciting security requirements through misuse activities. In accepted for the 2nd Int. Workshop on Secure Systems Methodologies using Patterns (SPattern'07). In conjunction with the 4th International Conference on Trust, Privacy & Security in Digital Business (TrustBus'07) (Turin, Italy, 2008).


Table 4. The analysis result of the second iteration.

<table>
<thead>
<tr>
<th>Threat</th>
<th>asset</th>
<th>mis-actor</th>
<th>location</th>
<th>solution</th>
<th>solution type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disclosure</td>
<td>password</td>
<td>unauthorized person</td>
<td>client</td>
<td>hiding password</td>
<td>prevent</td>
</tr>
<tr>
<td>Disclosure</td>
<td>session token</td>
<td>unauthorized person</td>
<td>client</td>
<td>countermeasure against XSS</td>
<td>prevent</td>
</tr>
<tr>
<td>Disclosure</td>
<td>password</td>
<td>unauthorized person</td>
<td>client</td>
<td>cookie management</td>
<td>prevent</td>
</tr>
<tr>
<td>Disclosure</td>
<td>session token</td>
<td>unauthorized person</td>
<td>channel</td>
<td>SSL</td>
<td>detect</td>
</tr>
<tr>
<td>Brute force attack</td>
<td>password</td>
<td>unauthorized person</td>
<td>server</td>
<td>strong password spec.</td>
<td>prevent</td>
</tr>
<tr>
<td>Dictionary attack</td>
<td>password</td>
<td>unauthorized person</td>
<td>server</td>
<td>strong password spec.</td>
<td>prevent</td>
</tr>
<tr>
<td>Session fixation</td>
<td>session token</td>
<td>unauthorized person</td>
<td>server</td>
<td>countermeasure against session fixation</td>
<td>prevent</td>
</tr>
<tr>
<td>Session hijack</td>
<td>session token</td>
<td>unauthorized person</td>
<td>server</td>
<td>countermeasure against session hijack</td>
<td>prevent</td>
</tr>
</tbody>
</table>

Figure 15. "Tampering sending data" pattern: context.


A. Appendix: Web Security Requirements Patterns (rest 6 patterns)

A.1 "Tampering sending data" pattern

Context This pattern can be applied to a program where some data assets required integrity are sent from actors to use cases. Table A.1 and Figure 15 shows the context.

Characteristic condition security data flow

<table>
<thead>
<tr>
<th>Use case type</th>
<th>any</th>
<th>any</th>
<th>n/a</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data asset</td>
<td>required</td>
<td>integrity</td>
<td>send</td>
</tr>
</tbody>
</table>

Table 5. "Tampering sending data" pattern: context.

Problem See Figure 16 and Figure 17. Consequently, there exist the following threats in the program.

1. Tampering sending data on the channel enabled by MITM attacks.
2. Tampering sending data on the server enabled by injection attacks, XSS and CSRF.
3. Tampering sending data on the server enabled by user spoofing.
4. Repudiation of sending data by a user him/herself.

Solution See Figure 18 and Figure 19.
1. Encryption such as SSL/TLS is useful for preventing MITM attacks which enables tampering on the channel.

See Also The following patterns are related to this pattern.
• I&A(6)
• Logging(6)
• Secure Channels (SSL)(6)

A.2 "Disclosure of shared data" pattern

Context This pattern can be applied to the program which shared data assets are sent from use cases to actors. Table A.2 and Figure 20 shows the context. "Shared" data means that its access is allowed to the actors of the same role. "C1" stereotype in Figure 20 represents that the data is required "shared" level of confidentiality.

<table>
<thead>
<tr>
<th>Characteristic condition</th>
<th>security</th>
<th>data flow</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use case type</td>
<td>any</td>
<td>any</td>
</tr>
<tr>
<td>Data asset</td>
<td>required</td>
<td>confidentiality (C1)</td>
</tr>
</tbody>
</table>

Table 6. "Disclosure of shared data" pattern: context.

Problem See Figure 21.
1. Disclosure shared data on the client enabled by vewing caches.
2. Disclosure shared data on the client enabled by shoulder hacking.
3. Disclosure shared data on the channel enabled by MITM attacks.
4. Disclosure shared data on the server enabled by user spoofing and injection attacks.

Solution See Figure 22.
1. Disabling caches is useful for preventing disclosure data on the client enabled by vewing caches.
2. Hiding the data is useful for disclosure data on the client enabled by shoulder hacking.
3. Encryption such as SSL/TLS is useful for preventing disclosure on the channel enabled by MITM attacks.
4. I&A is useful for preventing disclosure on the server enabled by user spoofing.
5. Countermeasures against injection attacks are useful for preventing disclosure on the server.

See Also The following patterns are related to this pattern.
• I&A(6)

A.3 "Disclosure of private data" pattern

Context This pattern can be applied to the program which private data assets are sent from use cases to actors.
Table A.3 and Figure 23 shows the context. "Private" data means that its access is allowed to the owner only. "C2" stereotype in Figure 23 represents that the data is required "private" level of confidentiality.

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>condition</th>
<th>security</th>
<th>data flow</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use case type</td>
<td>any</td>
<td>any</td>
<td>n/a</td>
</tr>
<tr>
<td>Data asset</td>
<td>required</td>
<td>confidentiality (C2)</td>
<td>receive</td>
</tr>
</tbody>
</table>

Table 7. "Disclosure of private data" pattern: context.

Problem See Figure 24.
1. Disclosure shared data on the client enabled by vewing caches.
2. Disclosure shared data on the client enabled by shoulder hacking.
3. Disclosure shared data on the channel enabled by MITM attacks.
4. Disclosure shared data on the server enabled by user spoofing and injection attacks.
5. Disclosure private data on the client enabled by elevation of privilege.
Solution  See Figure 25.

1. Disabling caches is useful for preventing disclosure data on the client enabled by viewing caches.
2. Hiding the data is useful for disclosure data on the client enabled by shoulder hacking.
3. Encryption such as SSL/TLS is useful for preventing disclosure on the channel enabled by MITM attacks.
4. I&A is useful for preventing disclosure on the server enabled by user spoofing.
5. Countermeasures against injection attacks are useful for preventing disclosure on the server.
6. Access control is useful for preventing disclosure on the client.

See Also  The following patterns are related to this pattern.
- I&A(6)
- Access Control(6)
- Secure Channels (SSL)(6)

A.4 "Tampering receiving data" pattern

Context  This pattern can be applied to a program where some data assets which require integrity are sent from use cases to actors. Table A.4 and Figure A.4 shows the context.

<table>
<thead>
<tr>
<th>Characteristic condition</th>
<th>security data flow</th>
<th>Data asset</th>
<th>Use case type</th>
</tr>
</thead>
<tbody>
<tr>
<td>n/a</td>
<td></td>
<td>required</td>
<td>any</td>
</tr>
<tr>
<td>n/a</td>
<td></td>
<td>integrity</td>
<td>any</td>
</tr>
<tr>
<td>n/a</td>
<td></td>
<td>receive</td>
<td>any</td>
</tr>
</tbody>
</table>

Table 8. "Tampering of receiving data" pattern: context.

Problem  See Figure 27 and Figure 28.

1. Tampering receiving data on the channel enabled by MITM attacks.
2. Tampering receiving data on the server enabled by injection attacks and user spoofing.
3. Repudiation of receiving data by a user him/herself.

Solution  See Figure 29 and Figure 30.

1. Countermeasures against injection attacks, XSS and CSRF are useful for preventing tampering on the server.
2. Encryption such as SSL/TLS is useful for preventing MITM attacks which enables tampering on the channel.

Figure 25. Disclosure of private data pattern: solution.

Figure 26. "Tampering of receiving data" pattern: context.

Figure 27. "Tampering of receiving data" pattern: problem (1).

Figure 28. "Tampering of receiving data" pattern: problem (2).

A.5 "ID/Password authentication" pattern

Context  This pattern can be applied to the program where ID/password authentication has been introduced. ID/password authentication is the major authentication method in the web application domain. Application of this pattern may be examined after the introduction of I&A is determined with other patterns, Table A.5 and Figure 31 shows the context.

3. Original Assurance such as digital signature, time assurance and I&A are useful for preventing repudiation.
Figure 29. “Tampering of receiving data” pattern: solution (1).

Figure 30. “Tampering of receiving data” pattern: solution (2).

Figure 31. “ID/Password authentication” pattern: context.

Figure 32. “ID/Password authentication” pattern: problem.

Figure 33. “ID/Password authentication” pattern: solution.

**Table 9.** “ID/Password authentication” pattern: context.

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>condition</th>
<th>security</th>
<th>data flow</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use case type</td>
<td>any</td>
<td>any</td>
<td>n/a</td>
</tr>
<tr>
<td>Data asset</td>
<td>use ID/password</td>
<td>any</td>
<td>any</td>
</tr>
<tr>
<td>Other</td>
<td>authentication</td>
<td>n/a</td>
<td>n/a</td>
</tr>
</tbody>
</table>

Problem

See Figure 32. Data Asset “password” has been newly added to the program. Therefore additional threats may be identified for the new assets. The new identified threats are:

1. Brute force attacks on a password which enables user spoofing.
2. Dictionary attacks on a password which enables user spoofing.
3. Disclosure of a password on the client and the channel.

Solution

See Figure 33.

1. Hiding passwords is useful for preventing disclosure on the client.
2. Encryption of communication is useful for preventing disclosure of passwords on the channel.
3. Strong password specification which rejects passwords with short length and a small repertoire of characters is useful for preventing brute force attacks.
4. Rejecting passwords which are easy to guess such as dictionary words, phone numbers and birth dates is useful for preventing dictionary attacks and guessing passwords.

See Also

The following patterns are related to this pattern.

- Password design and use(6)
A.6 "Session management" pattern

**Context**
This pattern can be applied to the program where session management has been introduced. Session management is usually used for maintaining web session data, especially for authenticated users. Therefore application of this pattern may be examined after the introduction of I&A is determined with other patterns, Table A.6 and Figure 34 shows the context.

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>condition</th>
<th>security</th>
<th>data flow</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use case type</td>
<td>any</td>
<td>any</td>
<td>n/a</td>
</tr>
<tr>
<td>Data asset</td>
<td>any</td>
<td>any</td>
<td>any</td>
</tr>
<tr>
<td>Other</td>
<td>use session management</td>
<td>n/a</td>
<td>n/a</td>
</tr>
</tbody>
</table>

**Table 10. "Session management" pattern: context.**

**Problem**
See Figure 35. By introducing session management, session token is identified as a new data asset. Therefore analysts have to consider threats for the new asset. The new threats for a session token are:

1. Guessing a session token which enables user spoofing.
2. Session fixation attacks which enable user spoofing.
3. Session hijack attacks which enable user spoofing.
4. Disclosure session tokens on the client.
5. XSS which enables disclosure session tokens on the client.

6. Disclosure session tokens on the channel with MITM attacks.

**Solution**
See Figure 36.

1. Strong (random) session token generation which makes tokens difficult to guess.
2. Countermeasures which prevent session fixation.
3. Countermeasures which prevent session hijack.
4. Secure cookie management such as setting secure attribute, proper domain and expiration date are useful for preventing disclosure session tokens on the client.
5. Countermeasures against XSS useful for preventing disclosure on the client.
6. Encryption of communication useful for preventing disclosure session tokens on the channel.