How to Secure Your Website

5th Edition

Approaches to Improve Web Application and Website Security

April 2011

IPA
IT SECURITY CENTER (ISEC)
INFORMATION-TECHNOLOGY PROMOTION AGENCY, JAPAN

Both English and Japanese edition are available for download at:

http://www.ipa.go.jp/security/english/third.html#websecurity (English web page)

http://www.ipa.go.jp/security/vuln/websecurity.html (Japanese web page)

Contents

Preface

Various websites provide a variety of services on the Internet. According to "Communications Usage Trend Survey"[1], as of 2011, it is estimated that more than 90 million people use the Internet in Japan and social interaction through the websites is expected to keep growing.

Meanwhile, the number of security incidents exploiting "security holes" (vulnerabilities) in the websites is also on the rise. Recently, they have become for-profit and are getting more vicious. More than 6,500 website vulnerabilities have been reported[2] to Information-technology Promotion Agency (IPA) since it started receiving the reports in 2005. Especially, "SQL Injection" is one of the most popular vulnerabilities reported and seen as a cause of personal information leakage via websites and virus infection of web pages.

To maintain the safety of your website, you need to take appropriate security measures on each website component. For operating systems or software, you could refer to the security practices common to all users provided by the vendors and make sure to securely configure the settings or apply security patches. Web applications, however, tend to be uniquely customized for each website and you need to secure each web application accordingly. If any security problems are found in a web application already in operation, it is usually difficult to fix them at the design level and you may need to settle for ad-hoc solutions. Nevertheless, remember that the best solution is to try not to create security holes when developing a web application in the first place and achieve the fundamental solution of "vulnerability-free" as much as possible.

This book makes use of vulnerability information on the software products and web applications reported to IPA, picking up the vulnerabilities frequently called to attention or with serious impact, and suggests the fundamental solutions and mitigation measures against them. In addition, it provides some references on how to improve the security of the websites and a few case studies to illustrate where developers may fail to secure web applications.

We hope this book will help you secure your website.

[1] Communications Usage Trend Survey, Ministry of Internal Affairs and Communications (MIC), http://www.soumu.go.jp/johotsusintokei/statistics/statistics05.html (Japanese Only)

[2] Appointed by the Ministry of Economy, Trade and Industry (METI), IPA serves as a national contact to receive reports on security vulnerabilities from the vendors and the general public. For more information, please visit: http://www.ipa.go.jp/security/vuln/report/index.html (Japanese Only)

Organization of This Book

This book mainly covers the computer software security issues that IPA, as the reporting point and analyzing agency designated in the Information Security Early Warning Partnership framework, has regarded as “vulnerability”.

This book consists of three chapters.

Chapter 1 “Web Application Security Implementation” addresses 9 types of vulnerabilities, including SQL injection, OS command injection and cross-site scripting, and discusses threats these vulnerabilities may pose and the characteristics of the websites that might be most susceptible to these vulnerabilities. It also provides fundamental solutions that aim to eliminate the vulnerability altogether and mitigation measures that try to reduce the damage of attacks exploiting the vulnerability.

Chapter 2 “Approaches to Improve Website Security” addresses 7 topics, including web server security and anti-phishing measures, and discusses how to improve the security of the websites mainly from operational perspective.

Chapter 3 picks up 8 types of vulnerability addressed in Chapter 1 and presents case studies, illustrating what may happen to the vulnerable websites with code examples, what is wrong with them and how to fix them.

In the appendix of this book, you will find a checklist you could use to assess the security of your website and a CWE mapping table.

Please note that each solution shown in this book is one example of many other possible solutions and we do not mean to force the use of them. We have performed the simple tests to evaluate the effectiveness of the solutions we provided in this book but we do not guarantee that they produce no unexpected side effects in your environment. Please use this book as a reference to solve the security problems and take appropriate action accordingly to your environment.

Intended Reader

The intended reader of this book is all of those who involved in website operation, such as web application developers and server administrators, regardless of whether one is individual or organization. Especially targeted at the web application developers who have just come to aware of the security issues.

What is Revised in the 5th Edition

In this edition, security issues for mobile websites are added to help understand the problems often faced when designing a website and approaches to fix the problems.

Also, 2 case studies are added to those introduced in the 4th edition and the total of 8 case studies are provided with the code examples to help understand what is wrong and how to fix it.

As for the content of each chapter, some changes, such as layout, have been made to improve readability, but the content and its mapping with the checklist items are unchanged.

Fixing Vulnerabilities
－Fundamental Solution and Mitigation Measure－

The outcome of security measures differ depending on what you do and what you try to achieve. You could focus on the measures to eliminate the cause of vulnerability aiming at fundamental solution, or you could focus on the attacking methods and prevent certain attacks, but you could be still vulnerable to other types of attacks. Either way, what is important is that you correctly understand the nature of the measure you have chosen to take and whether the expected result can be achieved with it.

In this book, we divided the web application security measures into two categories based on their nature: "fundamental solution" and "mitigation measure".

■ Fundamental Solution

Fundamental Solutions discuss "the methods to realize vulnerability-free implementation". By taking fundamental solutions, you could eliminate vulnerabilities and thus expect to nullify the attacks exploiting them.

■ Mitigation Measure

Mitigation measures discuss "the methods to mitigate the damage of attacks". They are different from fundamental solutions because they do not eliminate the cause of vulnerability, but they reduce the impact at each of the following phases, from the attack to its damage.

- Reduce the chance of being attacked
  (e.g. do not give out clues that lead to enable attacks)
- Reduce the possibility that vulnerability is exploited when being attacked.
  (e.g. Sanitize the data that can be used in attacks)
- Minimize the damage when vulnerability is exploited
  (e.g. access control)
- Detect the damage promptly
  (e.g. notification email)

Ideally, it is desired to implement fundamental solutions from at the design phase of a web application. Because mitigation measures do not eliminate the fundamental causes of vulnerability, just implementing mitigation measures is not desirable. Nonetheless, if fundamental solutions are not implemented perfectly, mitigation measures can work as safety net. In some cases, the combined use of fundamental solutions and mitigation measures may work well.

Likewise, when implementing vulnerability countermeasures to the web applications that are already in operation, it is also desirable to implement fundamental solutions. But if it is not possible because of cost, time or some other reasons, mitigation measures can work as a temporary measure.

Some mitigation measures may constrain the behavior of the expected functions. When applying mitigation measures, you should take into account those possible side effects as well.

1. Web Application Security Implementation

This chapter discusses the implementation of web application security, picking up the following nine vulnerabilities[3], and shows threats each vulnerability may pose, what types of websites might be most vulnerable, possible fundamental solutions and mitigation measures.

1) SQL Injection

2) OS Command Injection

3) Unchecked Path Parameter / Directory Traversal

4) Improper Session Management

5) Cross-Site Scripting

6) CSRF (Cross-Site Request Forgery)

7) HTTP Header Injection

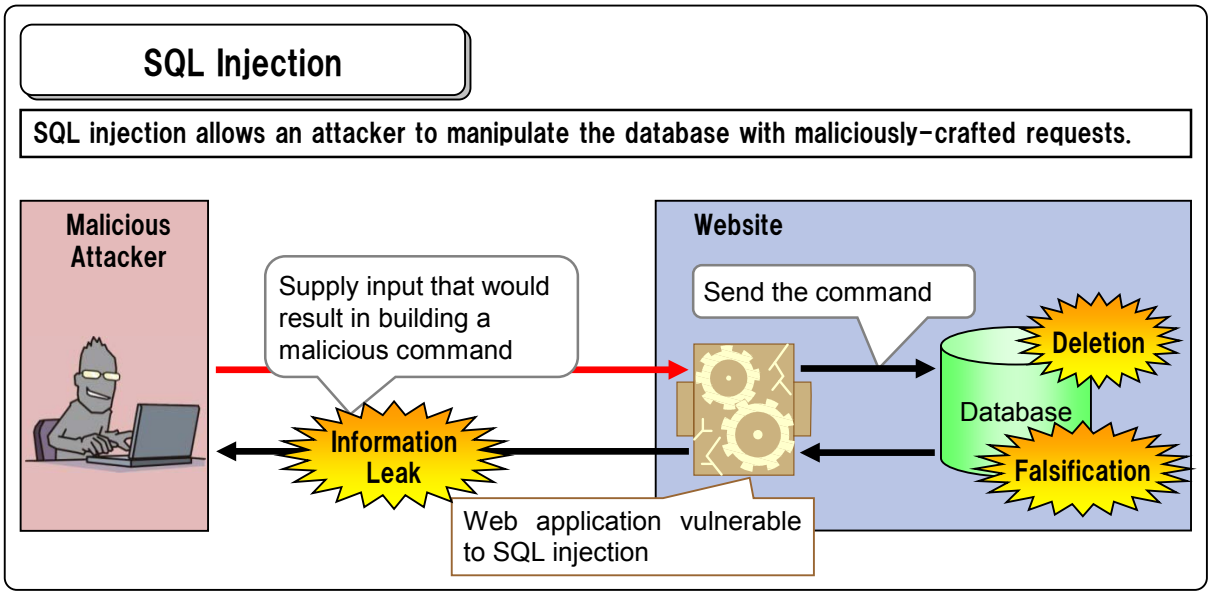
8) Mail Header Injection

9) Lack of Authentication and Authorization

[3] The numbering of the vulnerabilities reflects their severity or impact of possible attacks but does not indicate the priority you should work on to secure your web site. The priority should be examined based on the environment and status of the web site in question.

1.1 SQL Injection

Most of web applications that use a database build an SQL statement (a command to operate the database) based on user input. This means if the SQL statement-building process is not securely guarded, attacking and manipulating the database would become possible. This issue is called "SQL Injection vulnerability" and the attacking method exploiting this vulnerability is called "SQL Injection attack".



■ Possible Threats

This vulnerability could allow malicious attackers to:

- View sensitive data stored in the database

- e.g. Disclosure of personal information

- Falsify and/or delete data stored in the database

- e.g. Falsification of web pages, password change, system shutdown

- Bypass login authentication [4]

All the operations permitted under the privileges of a login account become unauthorizedly possible.

- Execute OS commands using stored procedures

- e.g. System hijacking, making the target PC a bot (launching point) to attack others

■ Websites That Need Special Attention

Regardless of what kind of website it is or who operates it, a website may fall into a victim if a website runs web applications that interact with database[5]. If the website uses the database that stores highly sensitive data, such as personal information, extreme caution is called for.

[4] It will be discussed also in "1.3 Improper Session Management".
[5] Commonly used database engines are: MySQL, PostgreSQL, Oracle, Microsoft SQL Server and DB2.

■ Reported Vulnerabilities[6]

SQL injection vulnerability is more popular than other vulnerabilities and accounts for about 14 percents of website-related vulnerabilities reported to IPA during from the time it started receiving the reports to the end of 2009. Software products, albeit fewer than websites, are also vulnerable to SQL injection and have been reported to IPA as well. The following are some of those software products, which are now fixed against this vulnerability.

- MODx Evolution Vulnerable to SQL Injection
  http://jvndb.jvn.jp/jvndb/JVNDB-2011-000008
- Aipo Vulnerable to SQL Injection
  http://jvndb.jvn.jp/jvndb/JVNDB-2011-000003
- Movable Type Vulnerable to SQL Injection
  http://jvndb.jvn.jp/jvndb/JVNDB-2010-000061

■ Fundamental Solutions

1-(i)-a

☞ Build all SQL statements using placeholders.

Usually, the SQL has a mechanism to build an SQL statement using placeholders. It is a mechanism to put a symbol (placeholder) at the place of the variables in the template of an SQL statement and replacing it with an actual data value mechanically later. Compared to a method where a web application directly builds an SQL statement through concatenation, the method that uses placeholders can eliminate the SQL injection vulnerability since it builds an SQL statement mechanically.

The process of replacing a placeholder with an actual data value is called binding. There are two binding methods: one is a method where an SQL statement is compiled keeping placeholders in it and the database engine replaces them with their corresponding actual data values (static placeholder) and the other is a method where the application's database connection library performs escaping and replaces the placeholders with their corresponding actual data value (dynamic placeholder). With the ISO/JIS standard for SQL, the static placeholder is called the prepared statement.

Both methods will remove SQL injection vulnerability but the static placeholder is more secure since it will eliminate the chance of SQL injection vulnerability in principal. For more information, see 3.2 of this book's supplementary volume, "How to Use SQL Calls to Secure Your Web Sites".

1-(i)-b

☞When building an SQL statement through concatenation, use a special API offered by the database engine to perform escaping and make up the literals in the SQL statement correctly.

When building an SQL statement through concatenation, insert a variable value in the SQL statement

[6] For the latest information, please refer to: http://www.ipa.go.jp/security/vuln/report/press.html (Japanese Only)

in the form of a literal. When inserting a value as the string type, you will bracket the value in single quotes. In that case, you should perform escaping for the string literal to sanitize the special characters (e.g. ‘ to ‘‘ and \ to \\). When inserting a value as the numeric type, makes it processed as a numeric literal (e.g. casting it into the numeric type).

What should be done exactly is different depending on the type and settings of the database engine in use and you should implement what it takes accordingly. Some database engines offer a special API[7] that generates a literal as a string. If your engine has one of those APIs, we recommend to use it. For more information see 4.1 of “How to Use SQL Calls to Secure Your Web Sites”.

This process should be performed not only for the values that may be affected by the external factors but also for all literals that compose an SQL statement.

1-(ii)

☞ **Do not write SQL statement directly in the parameter to be passed to the web application.**

This may sound absurd but it did happen nevertheless and we feel we should warn you not to directly write an SQL statement into the parameters, such as hidden, that are to be passed to the web application.

Specifying an SQL statement in a web application parameter directly could lead to a risk of someone falsifying the value of the parameter and manipulating the database.

■ Mitigation Measures

1-(iii)

☞ **Limit information to display in error message on the web browser.**

If an error message contains the information about database engine name or SQL statements which have caused the error, then malicious users could get useful information for attacking the website. Error messages can be used not only to give tips for attacking but also to show the result of an attack. It is recommended not to show error messages related to the database operation on the user’s web browser.

1-(iv)

☞ **Grant minimum privileges to database accounts.**

If the privileges of the database account that a web application uses to access to the database is higher than necessary, the damage the attack could inflict becomes more serious. Examine the commands the web application needs to interact with the database and give the access account the minimum privileges just enough to execute those commands.

By implementing these measures, security against SQL injection attacks is expected to improve. For more information on SQL injection vulnerability and developing web applications that use database, you

[7] Depending on the execution environment, some API is reported to have vulnerability where it does not perform escaping correctly. In that case, apply security patch or use other measure.

could refer to the following documents as well.

■ References

IPA: How to Use SQL Calls to Secure Your Web Site
http://www.ipa.go.jp/security/vuln/documents/website_security_sql_en.pdf

IPA: 知っていますか？脆弱性（ぜいじゃくせい）「1. SQL インジェクション」
http://www.ipa.go.jp/security/vuln/vuln_contents/sql.html (Japanese Only)
http://www.ipa.go.jp/security/vuln/vuln_contents/sql_flash.html (Japanese Only)

IPA: セキュア・プログラミング講座「SQL 注入: #1 実装における対策」
http://www.ipa.go.jp/security/awareness/vendor/programmingv2/contents/502.html
(Japanese Only)

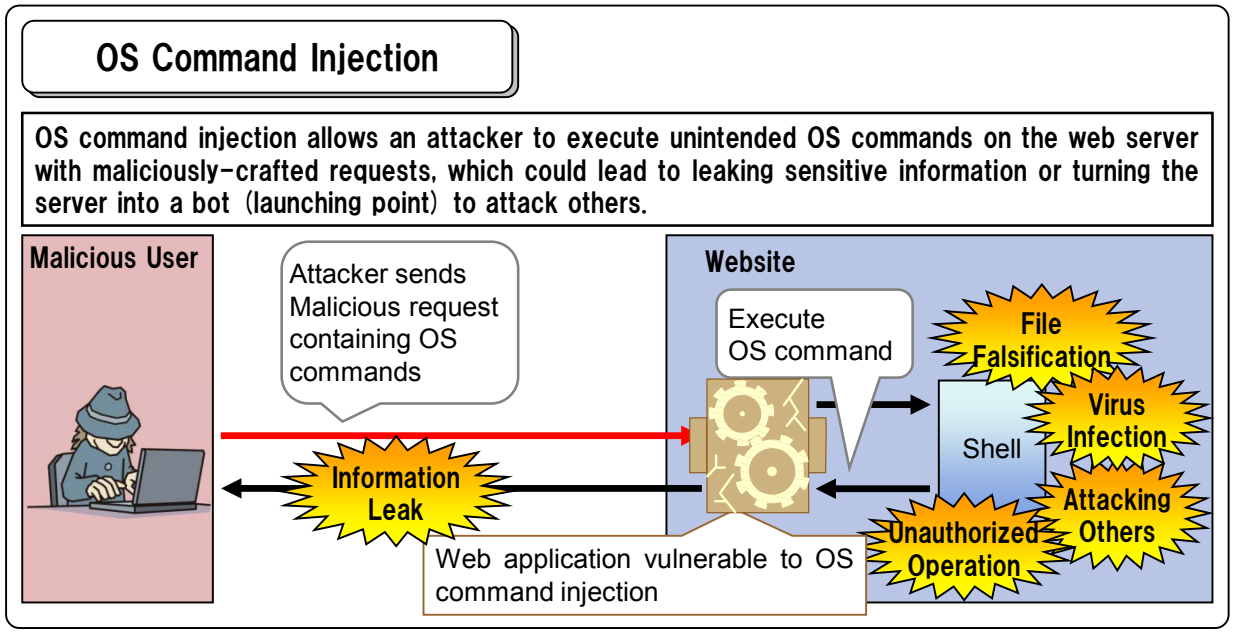
IPA: セキュア・プログラミング講座「SQL 注入: #2 設定における対策」
http://www.ipa.go.jp/security/awareness/vendor/programmingv2/contents/503.html
(Japanese Only)

IPA: Information Security White Paper 2009 Part 2: 10 Major Security Threats – Attacking Techniques Become More and More Sophisticated -)
http://www.ipa.go.jp/security/vuln/documents/10threats2009_en.pdf

IPA: 情報セキュリティ白書 2008 第 2 部「10 大脅威 ますます進む『見えない化』」
http://www.ipa.go.jp/security/vuln/20080527_10threats.html (Japanese Only)

1.2 OS Command Injection

Web applications can be vulnerable in such a way that they allow a remote attacker to execute OS level commands via those applications. This issue is called "OS Command Injection vulnerability" and the attacking method exploiting this vulnerability is called "OS Command Injection attack".



■ Possible Threats

This vulnerability could allow attackers to:

- View, falsify and delete files stored in the server

- e.g. Disclosure of sensitive information, falsification of configuration files

- Maliciously manipulate the system

- e.g. Unintended OS shutdown, adding/deleting user accounts

- Download and execute malicious programs

- e.g. Virus, worm and bot infection, backdoor implementation

- Make the system a launching point to attack others

- e.g. Denial of Service attack, reconnaissance and spamming

■ Websites That Need Special Attention

Regardless of what kind of website it is or who operates it, special attention is needed if a website runs any web applications using the functions that are capable of calling external programs[8].

[8] Examples of functions capable to call external programs:
Perl: open(), system(), eval()
PHP: exec(), passthru(), shell_exec(), system(), popen()

■ Reported Vulnerabilities

OS command injection vulnerability is found mostly in the web application software written in Perl and reported to IPA. The following are some of those software products, which are now fixed against this vulnerability.

- Webservice-DIC yoyaku_v41 Vulnerble to Command Injection
  http://jvndb.jvn.jp/jvndb/JVNDB-2009-000060
- Snoopy Command Injection Vulnerability
  http://jvndb.jvn.jp/jvndb/JVNDB-2008-000074
- Webmin OS Command Injection Vulnerability
  http://jvndb.jvn.jp/jvndb/JVNDB-2007-000730

■ Fundamental Solutions

2-(i)

☞ **Avoid using functions which could call shell commands.**

Some programming languages used to write web applications have the functions that are capable to call shell commands, such as the `open()` function in Perl. The `open()` function takes a file name as its argument and specifying it with "| (pipe)" would call and execute an OS command. That tells it is dangerous to allow external input to be used as its argument. You should avoid the use of these functions that can call shell commands[9] and substitute other functions for them. If you want to write a program to open a file in Perl, you could do it using the `sysopen()` without calling a shell command.

■ Mitigation Measures

2-(ii)

☞ **When using functions which could call shell commands, check all variables that make up the shell parameters and make sure to execute only those that are granted to be executed.**

Check all variables to be used as parameter of the functions capable to call shell commands before they are passed to the parameters to make sure that the system behaves in expected ways. The recommended method is whitelisting, which makes a list of accepted string patterns for a certain parameter and reject all others. If a parameter should be numeric, it will see if a string consists of only numbers. If it finds that the string does not follow the permitted patterns, it will not pass the value to the parameter and cancel the process.

Blacklisting, on the other hand, makes a list of string patterns likely to be used in OS command injection attacks, such as "|", "<" and ">" to rejected and permit all others, but this method has the risk of missing should-have-been-banned items and hence not recommended.

[9] See Corrective Measure #1～#3 in 3.2

By implementing these measures, security against OS command injection attacks is expected to improve. For more information on OS command injection vulnerability, you could refer to the following documents as well.

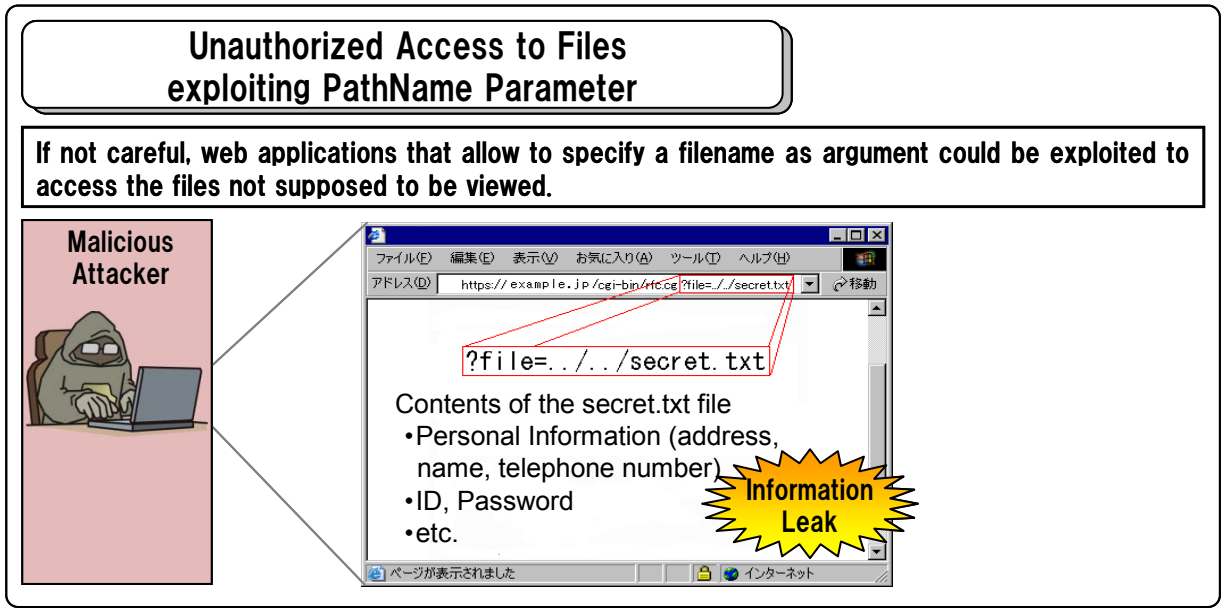
■ References

IPA: 知っていますか？脆弱性（ぜいじゃくせい）5. OS コマンド・インジェクション
http://www.ipa.go.jp/security/vuln/vuln_contents/oscmd.html (Japanese Only)
http://www.ipa.go.jp/security/vuln/vuln_contents/oscmd_flash.html (Japanese Only)

IPA: セキュア・プログラミング講座「コマンド注入攻撃対策」
http://www.ipa.go.jp/security/awareness/vendor/programmingv2/contents/501.html
(Japanese Only)

1.3 Unchecked Path Parameter / Directory Traversal

Some web applications allow to specify the name of files stored on the web server directly using external parameters. If such web application is not carefully programmed, attackers may specify an arbitrary file and have the web application execute unintended operations. This issue is called “Directory Traversal vulnerability” and one of the attacking methods exploiting this vulnerability is called “Directory Traversal attack”.



■ Possible Threats

This vulnerability could allow malicious attackers to:

- View, falsify and delete files stored on the server

- Disclosure of sensitive information
- Falsification and deletion of configuration files, data files and source codes

■ Websites That Need Special Attention

Regardless of what kind of website it is or who operates it, a website may fall into a victim if a web application allows to specify a filename directly using external parameters. If the web server stores sensitive information, such as personal information, as files on the server, extreme caution is called for.

- Examples of web applications that would access the files on the server

- Read web layout templates from the files
- Write user input to a user-specified files

■ Reported Vulnerabilities

Vulnerabilities related to path parameter account only a few percents of all website-related vulnerabilities but keep coming up since we started receiving the reports. The following are some of the software products

with this issue reported to IPA. The vulnerabilities in these products are now fixed.

- MODx Evolution Vulnerable to Directory Traversal
  http://jvndb.jvn.jp/jvndb/JVNDB-2011-000009
- WebCalenderC3 Vulnerable to Directory Traversal
  http://jvndb.jvn.jp/jvndb/JVNDB-2010-000003
- P forum Vulnerable to Directory Traversal
  http://jvndb.jvn.jp/jvndb/JVNDB-2009-000084

■ Fundamental Solutions

3-(i)-a

☞ Do not specify name of files stored on the web server directly using external parameter.

When a web application allows a filename to be specified directly using an external parameter, an attacker could manipulate the parameter specifying arbitrary files and view the file contents that should not be disclosed. For example, in an implementation case where the name of a file stored in the web server is specified in the hidden parameter and that file is used in the web page template, an attacker can output arbitrary file as a web page by manipulating the parameter.

It is recommended that you review the application design and specifications, reconsidering whether it is indeed necessary to allow to specify the name of files stored in the web server in external parameters and alternative methods are available.

3-(i)-b

☞ Use a fixed directory to handle filenames and nullify directory names in filenames.

Suppose that you are to open a file called “filename” in the current directory and if the file-open function is implemented like `open(filename)`, an attacker could access an arbitrary file by specifying the absolute path to the file. To prevent the use of absolute paths, you could use a fixed directory, such as “`dirname`”, and code it like `open(dirname+filename)`. However, just doing that still leaves rooms for directory traversal attacks using “`../`”. To prevent it, you could use an API, such as `basename()`, that extracts only the filename and removes the directory name from a given path like the following: `open(dirname+basename(filename))`[10].

[10] See Corrective Measure in 3.3.

■ Mitigation Measures

3-(ii)

☞ Manage file access permission properly.

If access permission to files on the web server is properly implemented and managed, the web server may be able to prevent attack attempts when a web application tries to open a file in arbitrary directories.

3-(iii)

☞ Check filenames.

When a filename contains the character strings that are used to specify an arbitrary directory, such as "/", "`../`" and "`..\`", cancel the process. Note that if you are using URL encoding and decoding, the URL encoded values like "`%2F`", "`..%2F`" and "`..%5C`" or double encoded values like "`%252F`", "`..%252F`" and "`..%255C`" can be interpreted as valid input values for a filename. Make sure to conduct checking at the appropriate timing.

By implementing these measures, security against attacks abusing path parameters is expected to improve. For more information on this vulnerability, you could refer to the following documents as well.

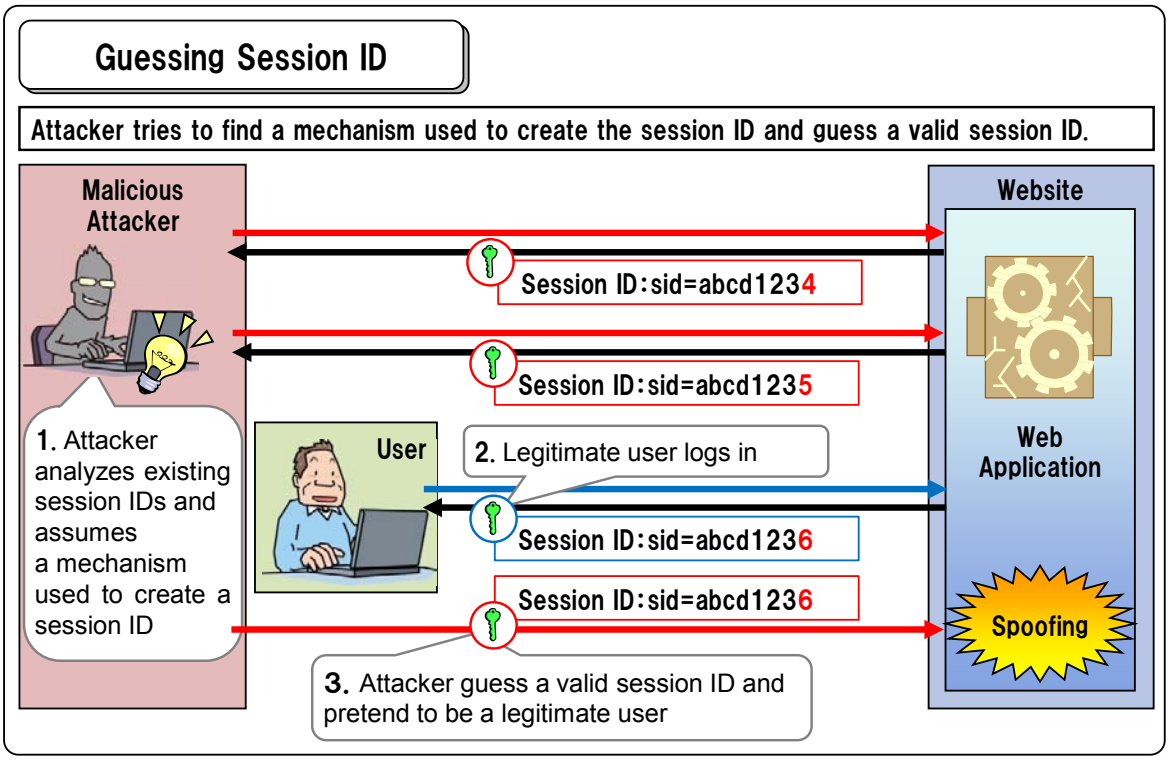
■ References

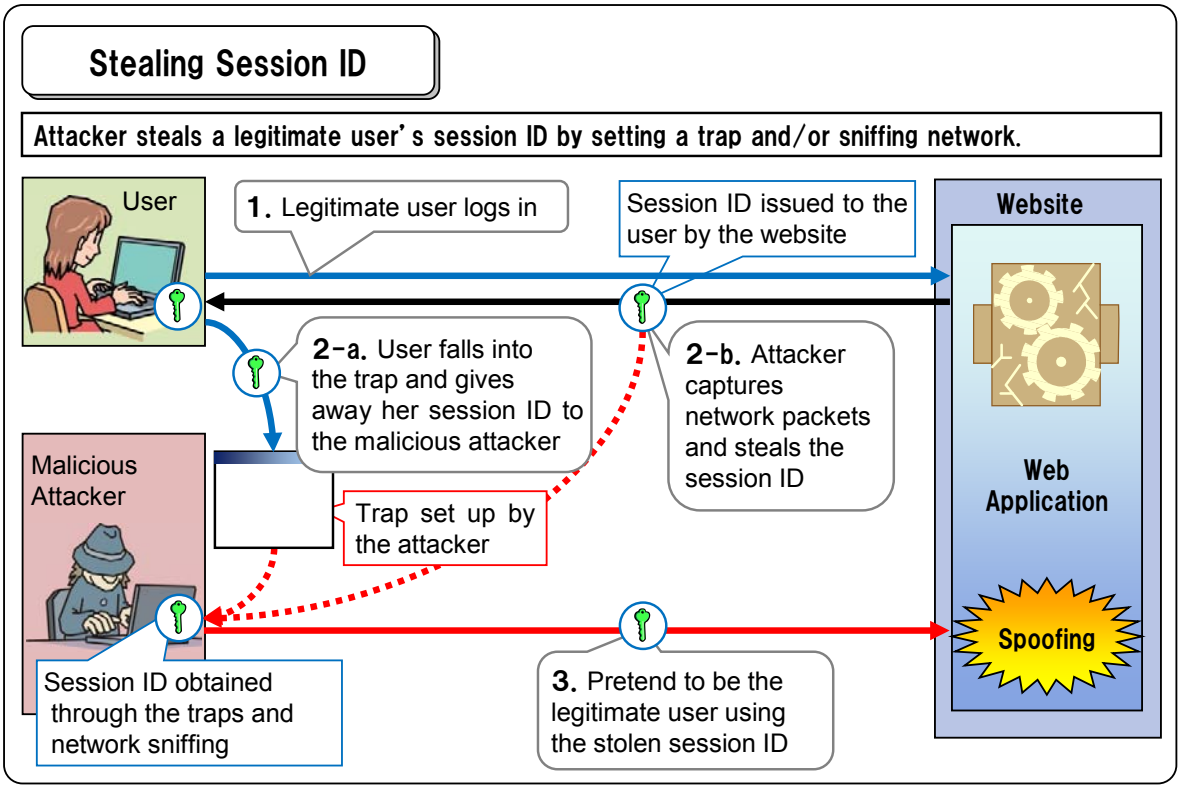
IPA: 知っていますか？脆弱性（ぜいじゃくせい）4. パス名パラメータの未チェック／ディレクトリ・トラバーサル
http://www.ipa.go.jp/security/vuln/vuln_contents/dt.html (Japanese Only)
http://www.ipa.go.jp/security/vuln/vuln_contents/dt_flash.html (Japanese Only)

IPA: セキュア・プログラミング講座「プログラムからのファイル流出対策」
http://www.ipa.go.jp/security/awareness/vendor/programmingv2/contents/402.html
(Japanese Only)

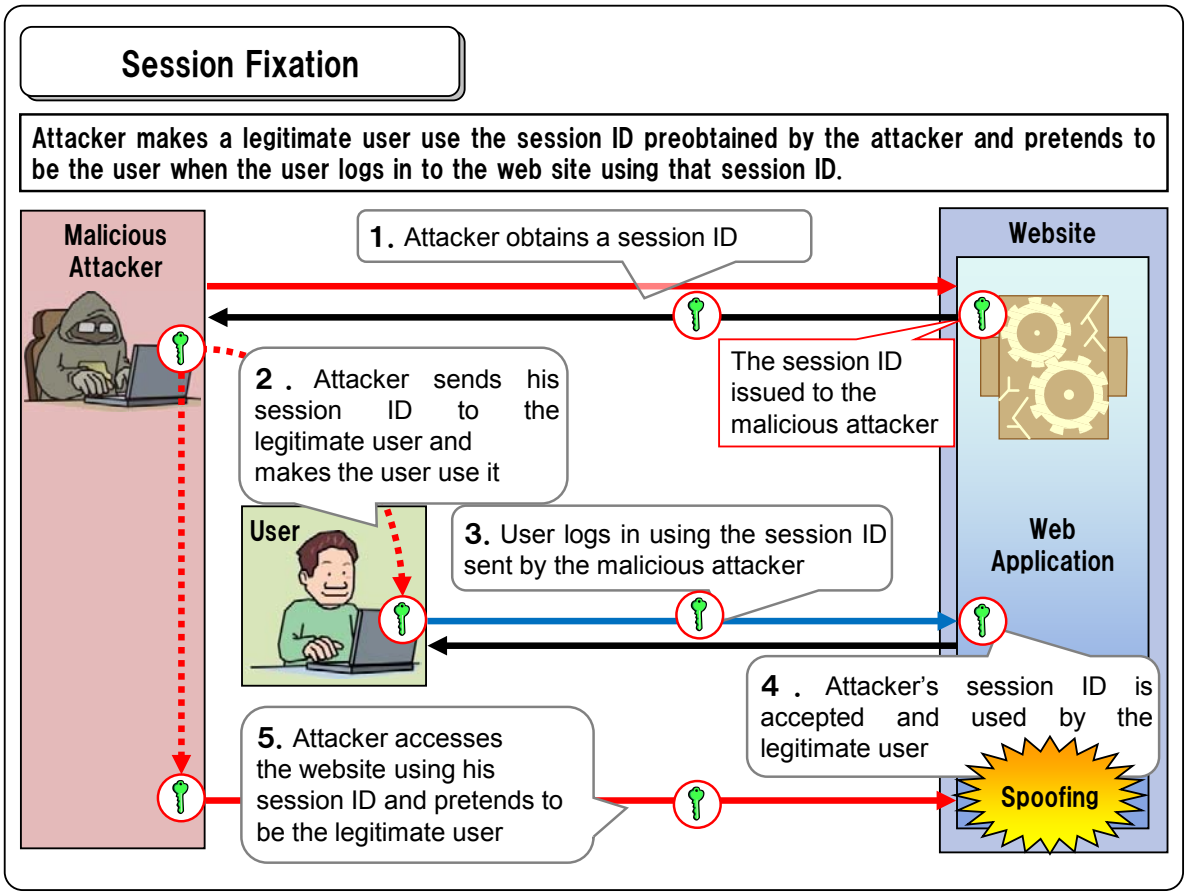
1.4 Improper Session Management

Some web applications issue session ID, which is the information to identify the user, to manage sessions. If session ID is not created and managed properly, an attacker could steal the session ID of a legitimate user and gain unauthorized access to the services pretending to be the legitimate user. The attacking method exploiting this vulnerability in session management is called “Session Hijacking”.





In addition to guessing or stealing session IDs, there is another attack exploiting improper session management called “Session Fixation”. It occurs when an attacker prepares a session ID and has a target user use the session ID in some way[11] and the target user who is unaware of it logs into the website. If successful, the attacker could pretend to be the targeted user using his or her session ID, which has been set up by the attacker, and access the website.



■ Possible Threats

If an attack exploiting improper session management succeeds, an attacker could pretend to be a legitimate user and do the operations permitted to that user. For example, it could allow to:

- **Access the services normally available only for the users who have properly logged in**
  - e.g. Unauthorized money transfer, purchasing unintended goods, canceling the membership against the user's will

[11] This becomes possible when session management is implemented in such a way that:
1. a web application uses the POST method and sets session ID in a hidden parameter to pass it around.
2. a web application sets session ID in a cookie and the user's web browser is capable to set a cross-domain cookie, which is an issue called “Cookie Monster” (*1).
3. a web application sets session ID in a cookie and the web application server is vulnerable to “Session Adoption” (*2).
4. a web application is vulnerable to cross-site scripting (discussed later in 1.5).

*1 “Multiple Browser Cookie Injection Vulnerabilities” http://www.westpoint.ltd.uk/advisories/wp-04-0001.txt
*2 “Session Fixation Vulnerability in Web-based Applications” http://www.acrossecurity.com/papers/session_fixation.pdf

- **Add and modify information normally permitted only for the users who have properly logged in**
  - e.g. Unauthorized change of application settings (passwords, administrator functions etc.), writing inappropriate entries
- **View information normally available only for the users who have properly logged in**
  - e.g. Unauthorized access to personal information, webmails, members-only bulletin board

■ Websites That Need Special Attention

Regardless of what kind of website it is or who operates it, special attention is needed with all the websites that require user login. If the website offers sensitive services, such as making online payment, ramification would be huge and extreme caution is called for.

- **Websites offering online payment**
  - e.g. Online banking, online trading, online shopping, online auction
- **Websites dealing with not-to-be-disclosed/private information**
  - e.g. Job-hunting websites, community websites, webmails
- **Other websites that might offer login feature**
  - e.g. Access to administrator functions, members-only bulletin board, blogs

■ Reported Vulnerabilities

Reports related to improper session management account only a few percents of all website-related vulnerabilities but it keeps coming up since we started receiving the reports. The following are some of the software products with this issue reported to IPA. The vulnerabilities in these products are now fixed.

- e-Pares Vulnerable to Session Fixation
  http://jvndb.jvn.jp/jvndb/JVNDB-2010-000023
- Active! mail 2003 Session ID Disclosure Vulnerability
  http://jvndb.jvn.jp/jvndb/JVNDB-2009-000076
- Predictable Session ID Vulnerability in Serene Bach
  http://jvndb.jvn.jp/jvndb/JVNDB-2009-000035

■ Fundamental Solutions

4-(i)

☞ **Make session ID hard to guess.**

If session ID is generated using a simple algorithm, such as time-based one, it is easy for attackers to

predict what the next session ID would be[12]. If the session ID is obtained by an attacker, it allows the attacker to pretend to be the legitimate user and gain unauthorized access to the services limited to the legitimate user. Make a session ID generation algorithm harder to guess using a mechanism like pseudo random number generators.

When using a web application that offers the session management mechanism, as long as using the mechanism, you do not have to generate session IDs on your won. It recommended not to develop a mechanism to manage session ID but to use a web application product that offers the mechanism.

4-(ii)

☞ Do not use URL parameter to store session ID.

If session ID is set in a URL parameter, the user's browser will forward the session ID-embedded URL to the next website it is accessing through the Referer. If a malicious attacker intercepts it, s/he could hijack the session. Store session ID in a cookie or hidden parameter using the POST method to pass it around.

Some web application servers may automatically turn to use a URL parameter when the user's browser is set to reject cookies. In that case, changer the server settings and try to disable the feature.

4-(iii)

☞ Set the secure attribute of the cookie when using HTTPS.

The cookie has the secure attribute which lets the cookie set with this attribute be sent over HTTPS only. If the secure attribute is not set, an HTTPS cookie can be sent over unencrypted HTTP channels as well and attackers could obtain cookie information by sniffing the channels. When using HTTPS, make sure to set the secure attribute. In addition, if you use a cookie in the HTTP communication as well, create a new cookie, separate from the one used in the HTTPS communication.

4-(iv)-a

☞ Start a new session after successful login.

Some web applications start a session issuing a session ID before the user logs in, possibly when the user first accesses the website, and keep using the same session. This method, however, is vulnerable to session fixation. You should avoid it and better start a new session after the user has successfully logged in (manage the session with a new session ID). Make sure to disable the old session ID when replacing it with the new session ID[13]. This will ensure that a malicious person cannot access a session newly created after the user logs in, even if the person tries to access it with the old session ID s/he has managed to obtain.

[12] See Common Mistakes #1～#2 in 3.4

[13] When the pre-login session information needs to be succeeded by the post-login session, be careful about how to copy the session information. If you shallow-copy an object variable, the pre-login session and post-login session will share and refer to the same data, thus presenting a risk that a person using the pre-login session ID could access and falsify the post-login session data. This risk itself can be vulnerability. You could take the deep-copy approach but some of the problems still remain. We recommend that you disable the pre-login session when the user login is successfully done.

4-(iv)-b

☞ **Issue a secret after login and authenticate the user with it whenever the user moves around the web site.**

Issue a secret separate from the session ID and set it in the cookie after the user has logged in successfully, and check whether the secret and the value in the cookie presented by the user's browser are the same at all web pages the user visits within the website[14]. Just like the Fundamental Solution 4-(i) "Make session ID hard to guess", use a secure mechanism, such as pseudo random number generators to issue a secret, or encrypt it.

In case of the following situations , this measure is unnecessary.

- The fundamental measure 4-(iv)-a is being implemented.
- A session ID is issued only after login in the first place.

■ Mitigation Measures

4-(v)

☞ **Use random session ID.**

If the session ID is fixed for each user, an attacker can perform session hijacking attacks anytime without time limitation once the attacker obtains the session ID. Do not use a fixed session ID and create a new session ID each time the user logs in.

4-(vi)

☞ **Set the cookie's expiration date with care when storing session ID in cookie.**

A cookie is retained by the browser till its expiration date. If an attacker manages to steal cookies exploiting the browser's vulnerability, the attacker could gain access to all the cookies retained at that time. When creating a cookie, set the expiration date appropriately.

For example, set a short expiration date and make sure that the browser does not retain the cookie more than necessary.

If the cookie does not need to be retained, you could skip setting the expiration date (expires=), which results in destroying the cookie when the browser is closed. This method may not yield the expected result, however, if the user keeps using the same browser retaining the cookie along the way.

By implementing these measures, security against session hijacking attacks is expected to improve. For more information on session management, you could refer to the following documents as well.

[14] Some web application servers automatically take this approach.

■ References

IPA: 知っていますか？脆弱性（ぜいじゃくせい）6. セッション管理の不備
http://www.ipa.go.jp/security/vuln/vuln_contents/session.html (Japanese Only)
http://www.ipa.go.jp/security/vuln/vuln_contents/session_flash.html (Japanese Only)

IPA: セキュア・プログラミング 「セッション乗っ取り」
http://www.ipa.go.jp/security/awareness/vendor/programmingv2/contents/302.html
(Japanese Only)
http://www.ipa.go.jp/security/awareness/vendor/programmingv2/contents/303.html
(Japanese Only)
http://www.ipa.go.jp/security/awareness/vendor/programmingv2/contents/304.html
(Japanese Only)
http://www.ipa.go.jp/security/awareness/vendor/programmingv2/contents/305.html
(Japanese Only)
http://www.ipa.go.jp/security/awareness/vendor/programmingv2/contents/306.html
(Japanese Only)

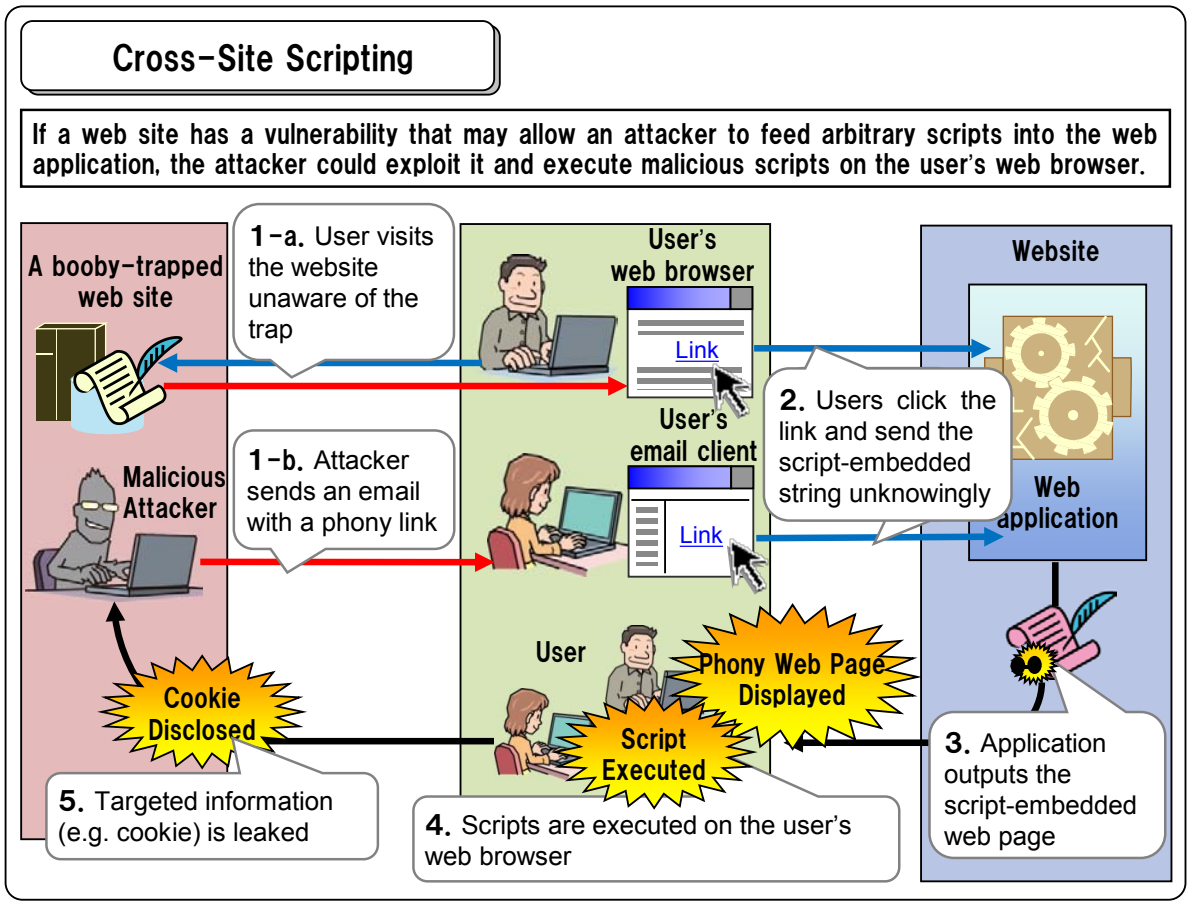
IPA: セッション管理
http://www.ipa.go.jp/security/awareness/administrator/secure-web/chap6/6_session-1.html
(Japanese Only)

IPA: セッション管理の留意点
http://www.ipa.go.jp/security/awareness/administrator/secure-web/chap6/6_session-2.html
(Japanese Only)

産業技術総合研究所 高木浩光: 「CSRF」と「Session Fixation」の諸問題について
http://www.ipa.go.jp/security/vuln/event/documents/20060228_3.pdf (Japanese Only)

1.5 Cross-Site Scripting

Some web applications output a web page based on user input or HTTP header information, such as search results, the user registration confirmation page, bulletin boards and web statistics reports. If this process is not security-conscious, an attacker could embed malicious content, such as arbitrary scripts, into the output web page. This issue is called "Cross-Site Scripting vulnerability" and one of the attacking methods exploiting this vulnerability is called "Cross-Site Scripting attack". It may not harm the website itself but it would affect the safety of the visitors of the website.



■ Possible Threats

This vulnerability could allow malicious attackers to:

- Display a phony web page on the legitimate website

- e.g. Confusion caused by false information
- e.g. Disclosure of sensitive information through phishing attacks

- Steal cookies retained by the web browser

- If session ID is stored in the stolen cookie, it could lead to spoofing[15]
- If personal information is stored in the stolen cookie, the sensitive data would be disclosed.

[15] The same problem mentioned in "Possible Threats" in "1.4 Improper Session Management".

– Make the browser save arbitrary cookies

- Attacker could launch the session fixation attack making the user use arbitrary session ID[16].

■ Websites That Need Special Attention

Regardless of what kind of website it is or who operates it, all websites should be cautious about this vulnerability. If the website manages the login session using the cookie or includes the web pages that the phishers tend to pick on, such as a login page and a user registration page asking for personal information, extra caution should be taken.

– Web page features likely having cross-site scripting vulnerability

- Provide user-input confirmation (e.g. login, user registration and survey)
- Display the user input values to prompt the user to re-enter data after erroneous input
- Show search results
- Show error messages
- Provide comment/entry feature (blogs, bulletin boards) etc.

■ Reported Vulnerabilities

Cross-site scripting vulnerability is more popular than other website vulnerabilities and accounts for about 40 percents of all the vulnerabilities reported during from the time IPA started receiving the report to the end of 2009. Many software products have been also reported being vulnerable to cross-site scripting. The following are some of those products, which are now fixed against this vulnerability.

- EC-CUBE Vulnerable to Cross-Site Scripting
  http://jvndb.jvn.jp/jvndb/JVNDB-2011-000011
- Cross-Site Scripting Vulnerability in Multiple Rocomotion Products
  http://jvndb.jvn.jp/jvndb/JVNDB-2011-000006
- SGX-SP Final and SGX-SP Final NE Vulnerable to Cross-Site Scripting
  http://jvndb.jvn.jp/jvndb/JVNDB-2011-000002

■ Countermeasures

In this book, we divide the countermeasures against cross-site scripting vulnerability into three categories based on the nature of the web application.

1) Measures for the web applications that do not permit HTML text input

2) Measures for the web applications that permit HTML text input

3) Measures common to all web applications

The web applications applicable to 1) would be those that offer the features which do not require the use of HTML tags, such as search engine or user registration. Most web applications will fall under this category.

[16] For more information on “Session Fixation”, please refer to the page 17

The web applications applicable to 2) could be those that require some freedom in terms of data presentation, such as blogs or bulletin boards. For example, HTML text input may be permitted to implement a function that lets the users choose the font size or color of the user entry.

Measures under the category 3) are required for the both types of web applications.

1.5.1 Measures for Web Applications That Do Not Permit HTML Text Input

■ Fundamental Solutions

5-(i)

☞ Perform Escaping for everything to be outputted to the web page.

To prevent cross-site scripting, perform escaping for all web page elements, such as the contents and the value of the HTML attributes. One way to implement escaping is to replace the special characters used to control the layout of a web page, such as "<", ">" and "&", with the HTML entities "`<`", "`>`" and "`&`" respectively. If the web application needs to output the HTML tags, make sure to enclose all attribute values in double quotation marks, then perform escaping by replacing the double quotation mark contained in the attribute values with the HTML entity "`&quot;`".

In terms of vulnerability prevention, the data that must go through escaping process are input character strings passed to the web application by the external entity, the values retrieved from database or files and those generated from arithmetic operation on character strings. However, you could make sure not to miss anything by taking more consistent approach where all text elements of a web page are to go through escaping process regardless of whether it is necessary[17].

The output process that needs to include escaping process is not limited to that for the HTTP response. When changing the contents of a web page dynamically, for example using the `document.write` method in JavaScript or the `innerHTML` property, the same process is required.

5-(ii)

☞ When outputting URLs in HTML, permit only those that start with certain patterns, such as "http://" and "https://".

A URL can start with not only "`http://`" or "`https://`" but also with "`javascript:`". If a URL of the resources or the images to be inserted into an HTML page is dynamically created based on the external input, an attacker could launch cross-site scripting attacks by embedding a script into the URL. For example, if a web application creates a HTML output page by setting a URL specified by the user like `<a herf=`*`user input URL`*`>`, the attacker could insert a script by strings that start with "`http://`" or "`https:://`" for the URL value. Take a whitelist approach where only the strings that start with `http://` or `https://` are allowed for the URL value.

[17] See 3.5.2.

5-(iii)

☞ **Do not dynamically create the content of the <script>...</script> tag.**

If the value for the `<script> ... </script>` tag is dynamically created based on the external input, arbitrary scripts could be inserted in there. You could check and nullify risky scripts but it is recommended not to let the application dynamically set the value for the `<script> ... </script>` tag for it would be difficult to determine which scripts are indeed dangerous ones for sure.

5-(iv)

☞ **Do not allow to import stylesheets from arbitrary websites**

☞ Scripts can be written into stylesheets using a function like `expression()`. That means that malicious scripts can be inserted into the web page if the website design allows to import a stylesheet from arbitrary websites. You could check the imported stylesheet and nullify dangerous scripts but you would better not to let the application use external stylesheets for it would be difficult to clear them for absolute sure.

■ Mitigation Measures

5-(v)

☞ **Check input values.**

Make the web application have a function to check input values and ask the user to re-enter when they do not follow certain rules. Know that this cannot prevent a case where the input values are crafted to generate a script string through arithmetic operation after they have passed the input check. Therefore, you should not relay solely on this countermeasure.

1.5.2 Measures for Web Applications That Permit HTML text Input

■ Fundamental Solutions

5-(vi)

☞ **Create a parse tree from the HTML text input and extract only the necessary elements that do not contain scripts.**

Parse the HTML text input and extract only the elements permitted in the predefined whitelist. When implementing this measure, think it through carefully for it requires complex programming and the processing load would be high.

■ Mitigation Measures

5-(vii)

☞ Nullify script strings in HTML text input.

Identify script strings included in HTML text input and nullify them. We recommend you nullify those strings by replacing them with harmless strings. For example, if you would like to replace "`<script>`" or "`javascript:`" with something harmless, you could add an character to those strings like "`<xscript>`" or "`xjavascript:`". Alternatively, you could delete script strings altogether but which may present a new risk that removing them will put together a dangerous string in turn[18] and is not recommended.

This measure poses the difficulty of extracting all the dangerous strings for sure. Because some web browsers interpret a string like "`java&#09;script:`" or "`java(linefeed)script:`" as "`javascript:`", a simple pattern matching would not do the job. Thus, it is not recommended to relay on this kind of blacklist approach.

1.5.3 Measures common to all web applications

■ Fundamental Solutions

5-(viii)

☞ Set the charset parameter of the HTTP Content-Type header.

You can set the character code (charset) in the `Content-Type` filed of the HTTP header like: "`Content-Type:text/html; charset=UTF-8`". When the charset parameter is absent from the `Content-Type` header field, the browser assumes the character code based on its own rule and processes the strings with the assumed character set to display them on the web browser. For example, some browsers are known to use a particular character code when the first part of the HTML text contains a certain character string.

If the charset is not specified, an attacker could exploit this browser behavior, have the browser choose the particular character set intentionally by inserting the certain character string and embed the character strings that would emerge as scripts when they are processed with that character set.

For example, if the character string "`+ADw-script+AD4-alert(+ACI-test+ACI-)+AdsAPA-/script+AD4-`" is inserted into the HTML text, some browsers would recognize it as a string encoded by UTF-7. If this string is processed using UTF-7, it becomes "`<script>alert('test')</script>`" and the script will be executed.

Even if you do perform escaping discussed in 5-(i) and take the countermeasures against cross-site scripting vulnerability properly, the characters shown above, such as "`+ADw-`", will not be escaped since those characters are processed by the web application set with other character codes, such as UTF-8, EUC-JP or SHIFT_JIS, and not recognized as something that should be escaped.

To prevent this problem, you could perform another escaping for the HTML text assuming it is

[18] See Common Mistake #2 in 3.5.3.

encoded by UTF-7 as well, but assuming only UFT-7 is insufficient. There would also be some side effects that the UTF-7-based escaping may replace a legitimate character string not to be escaped in other character codes and interfere with the normal operations.

Thus, to solve this issue, it is effective to make sure to specify the charset parameter without omitting it. Set the character code that the web application intends to handle the character strings when outputting HTML pages in the Content-Type of the accompanying HTTP header[19].

■ Mitigation Measures

5-(ix)

☞ Set the HttpOnly attribute of the cookie and disable the TRACE method to prevent disclosure of cookie information.

"`HttpOnly`" is an attribute you can set on the cookie and will deny the scripts within HTML text access to the cookie. This will prevent the cookies from being stolen even if the website has cross-site scripting vulnerabilities.

To do this, set the `HttpOnly` attribute in the `Set-Cookie` HTTP header when creating a cookie like: "`Set-Cookie: [snip]; HttpOnly`".

There are a few things you should know about when adopting this countermeasure.

First, you need to disable the TRACE method on the web server. When the TRACE method is enabled, if the website has cross-site scripting vulnerability, an attacker could obtain the whole HTTP request header the browser sends using the attacking method called "Cross-Site Tracing". An HTTP request header contains cookie information[20], thus the cookie will be 'stolen' even if the `HttpOnly` attribute is set.

Secondly, the `HttpOnly` attribute is not supported by all browsers, thus it is not the solution that could benefit and protect all website visitors.[21].

Understand that this is not the solution that would eliminate all the vulnerabilities cross-site scripting vulnerability may pose and other threats besides cookie information leak still remain and that it may not work depending on the web browsers the user uses. After that, decide whether it's worth adopting to your website.

By implementing these measures, security against cross-site scripting attacks is expected to improve. For more information on cross-site scripting vulnerability, you could refer to the following documents as well.

[19] W3C Recommendation HTML 4.0.1 says that the browser must follow the priority defined below when deciding which character set to use (http://www.w3.org/TR/html401/charset.html#h-5.2.2).
1. An HTTP "charset" parameter in a "Content-Type" field
2. A META declaration with "http-equiv" set to "Content-Type" and a value set for "charset"
3. The charset attribute set on an element that designates an external resource a web application uses the POST method and sets session ID in a hidden parameter to pass it around.

Thus, it will be recommended specifying the character code in "an HTTP charset parameter in a Content-Type field".

[20] When you use the Basic authentication scheme, the user ID and password can be stolen as well.

[21] For more information on the HTTPOnly-compliant browsers, refer to the following: Browsers Supporting HTTPOnly: http://www.owasp.org/index.php/HTTPOnly#Browsers_Supporting_HTTPOnly

■ References

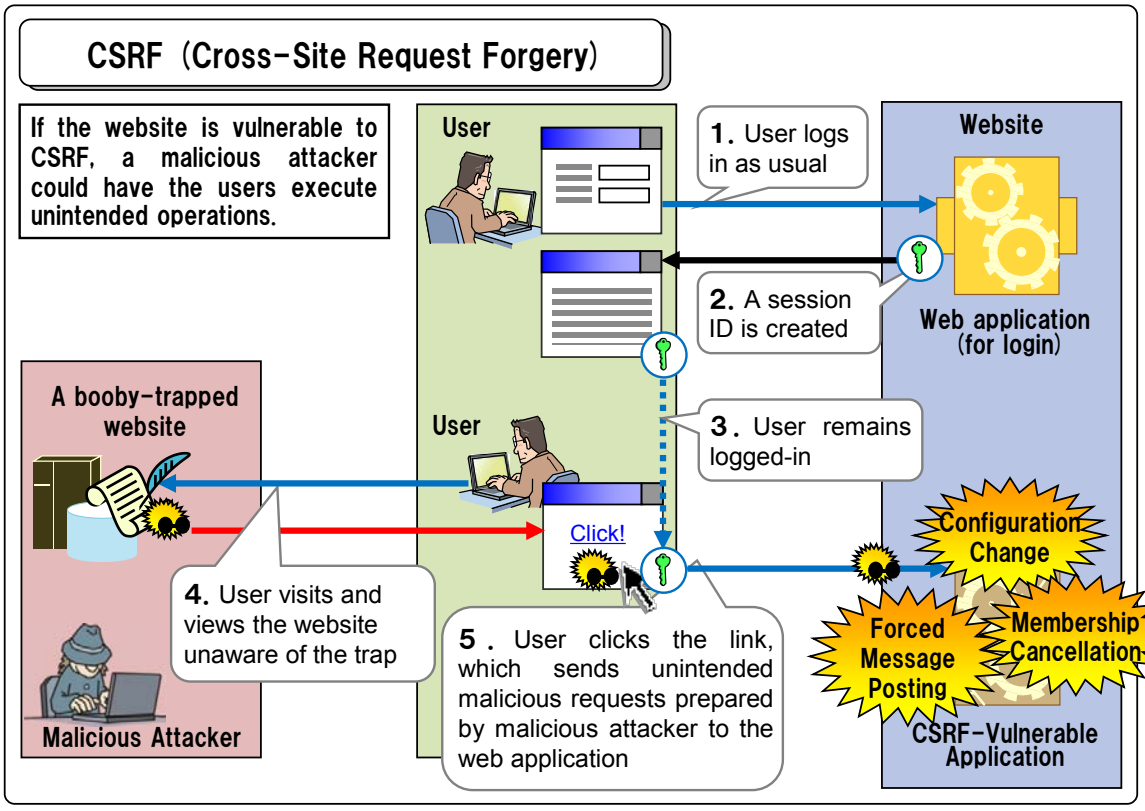
IPA：知っていますか？脆弱性（ぜいじゃくせい）2. クロスサイト・スクリプティング
http://www.ipa.go.jp/security/vuln/vuln_contents/xss.html (Japanese Only)
http://www.ipa.go.jp/security/vuln/vuln_contents/xss_flash.html (Japanese Only)

IPA：セキュア・プログラミング「エコーバック対策」
http://www.ipa.go.jp/security/awareness/vendor/programmingv2/contents/601.html
(Japanese Only)
http://www.ipa.go.jp/security/awareness/vendor/programmingv2/contents/602.html
(Japanese Only)

IPA：情報セキュリティ白書 2007 年版「ますます多様化するフィッシング詐欺」
http://www.ipa.go.jp/security/vuln/documents/2006/ISwhitepaper2007.pdf (Japanese Only)

1.6 CSRF (Cross-Site Request Forgery)

Some websites require the users to login to offer their services. Here, if a website does not have a mechanism to verify whether a request made by a logged-in user is indeed the request intended by the user, the website may accept a malicious request set up by other external parties. If the website has this vulnerability, its user could suffer from doing unintended things on the website through the trap set up by malicious attackers. This issue is called "Cross-Site Request Forgery vulnerability" and the attacking method exploiting this vulnerability is called "Cross-Site Request Forgery attack".



■ Possible Threats

This vulnerability could allow malicious attackers to[22]:

- **Access the services normally available only for the users who have properly logged in**
  - e.g. Transferring money, purchasing goods or canceling the membership unintended by the user
- **Add and modify information normally permitted only for the users who have properly logged in**
  - e.g. Application settings (passwords, administrator functions etc.), writing inappropriate entries

[22] Compared to the possible threats by improper session management (discussed in 1.4), one thing may differ that an attacker may not view information available only for the users who have properly logged in. If an attacker succeeds in an attack that could enable the further attacks, such as changing password, it could lead to information leak.

■ Websites That Need Special Attention

The websites that implement session management using the following technologies may be vulnerable to CSRF attacks.

- Session management using cookies
- Basic Authentication
- SSL Client Authentication

If the website is applicable to the above and offers sensitive services, such as making online payment, ramification would be huge and extreme caution is called for.

- **Websites offering online payment**
  - e.g. Online banking, online trading, online shopping, online auction
- **Other websites that might offer login feature**
  - e.g. Access to administrator functions, members-only bulletin board and blogs

■ Reported Vulnerabilities

Reports related to CSRF vulnerability account only a few percents of all website-related cases but it keeps coming up since about 2006. An example of the reported cases is a web management interface for the embedded system, such as network hard disks, reported with this vulnerability. The following are some of the software products with this issue reported to IPA. The vulnerabilities in these products are now fixed.

・SquirrelMail Vulnerable to Cross-Site Request Forgery
http://jvndb.jvn.jp/jvndb/JVNDB-2009-002207

・Cross-Site Request Forgery Vulnerability in Oracle iPlanet Web Server
http://jvndb.jvn.jp/jvndb/JVNDB-2010-000042

・e-Pares Vulnerable to Cross-Site Request Forgery
http://jvndb.jvn.jp/jvndb/JVNDB-2010-000022

■ Fundamental Solutions

6-(i)-a

☞ **Access the web page, in which certain operation is to be executed, via the POST method with a secret having the previous web page insert it in its hidden filed, and execute the requested operation only when the secret is correct.**

Let's see an example where the process transits like the following: "data input page → confirmation page → data registration". First, when outputting the confirmation page, set a secret in a hidden parameter. The secret can be the session ID used for session management or you can create another ID (the second session ID) at the time of login. Firstly, when creating a session ID, you should make sure

that predicting the session ID is difficult using a mechanism like cryptographically secure pseudo-random number generators. Secondly, when the registration process receives a request from the confirmation page, check the value set in the hidden parameter with the secret and proceed to registration procedure only if they match[23]. In this way, an attacker cannot launch attacks unless s/he somehow obtains the secret set in the hidden parameter.

Remember to implement this measure using the POST method[24]. If you use the GET method, a secret will be seeable for the external website through the Referer.

6-(i)-b

☞ Ask for password right before executing requested operation and proceed only when the password is correct.

By performing password authentication, the CSRF vulnerability can be eliminated[25]. This measure requires change in user interface design specification. If you cannot change the user interface design specification and the countermeasure you can take is limited to implementation change, consider the measure discussed in 6-(i)-a or 6-(i)-c.

Compared to the solution 6-(i)-a, this measure may be easier to implement in some cases. For example, If are using the Basic authentication without session management, a secret needs to be newly created to implement the measure recommended in 6-(i)-a. In that case, if it is difficult to use a safe pseudo random number generator or such, this measure can be easier to adopt.

6-(i)-c

☞ Check the referrer whether it is the expected URL and proceed only when the URL is correct.

By checking the Referer information, you could confirm whether the user's browsing path (transition) is following the steps that ought to be. If you cannot confirm, do not proceed[26]. If the Referer is absent, stop proceeding as well since an attacker could launch CSRF attacks using a technique that can clear the Referer.

Depending on the websites, however, an attacker may be able to set a tarp on the targeted website itself and in this case, this measure may not work properly. In addition, when a user bans to send the Referer in the browser or personal firewall settings, the user may not be able to use the website and experience inconvenience. When adopting this measure be sure to care about these issues.

---

[23] See Corrective Measure #1 in 3.6

[24] RFC2616, which specifies specification for HTTP/1.1, says we should use the POST method instead of the GET method when sending confidential data (15.1.3 Encoding Sensitive Information in URI's).
RFC2616：Hypertext Transfer Protocol -- HTTP/1.1 http://www.ietf.org/rfc/rfc2616.txt

[25] See Corrective Measure #2 in 3.6.

[26] See Corrective Measure #3 in 3.6.

■ Mitigation Measures

6-(ii)

☞ Notify to the prespecified email address automatically when important operations have been done.

Email is sent in a post-incident manner and thus cannot prevent CSRF attacks, but it could raise a red flag that something may be amiss when the attack actually happens. Be careful not to include sensitive information related to privacy in the body of email.

By implementing these measures, security against CSRF attacks is expected to improve. For more information on CSRF vulnerability, you could refer to the following documents as well.

■ References

IPA: 知っていますか？脆弱性（ぜいじゃくせい）3. CSRF（クロスサイト・リクエスト・フォージェリ）
http://www.ipa.go.jp/security/vuln/vuln_contents/csrf.html (Japanese Only)
http://www.ipa.go.jp/security/vuln/vuln_contents/csrf_flash.html (Japanese Only)

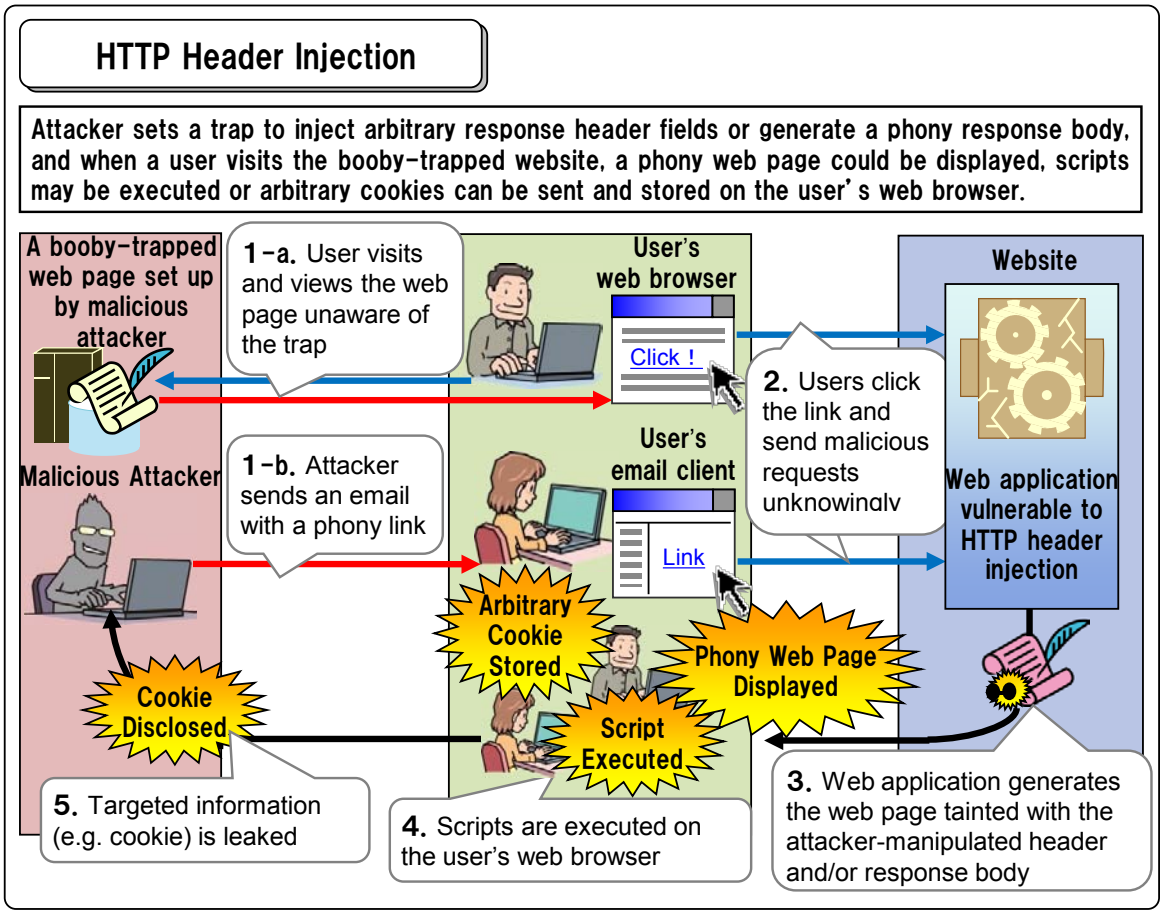
IPA: セキュア・プログラミング講座「リクエスト強要（CSRF）対策」
http://www.ipa.go.jp/security/awareness/vendor/programmingv2/contents/301.html
(Japanese Only)

産業技術総合研究所 高木浩光:「CSRF」と「Session Fixation」の諸問題について
http://www.ipa.go.jp/security/vuln/event/documents/20060228_3.pdf (Japanese Only)

IPA: 情報セキュリティ白書 2006 年度版「ウェブサイトを狙う CSRF の流行」
http://www.ipa.go.jp/security/vuln/documents/2005/ISwhitepaper2006.pdf (Japanese Only)

1.7 HTTP Header Injection

Some web applications dynamically set the value of the HTTP response header fields based on the value passed by the external parameters. For example, HTTP redirection is implemented by setting a redirected-to URL specified in the parameter to the Location header field, or a web application may set the names entered in a bulletin board to the Set-Cookie header filed. If the process of building an HTTP response header in such web applications has vulnerabilities, an attacker could add header fields, manipulate the response body and have the web application generate multiple responses. This issue is called "HTTP Header Injection vulnerability" and the attack method exploiting this vulnerability is called "HTTP Header Injection attack". In particular, the attack that leads the web application to produce multiple responses is called "HTTP Response Splitting attack".



■ Possible Threats

This vulnerability could allow malicious attackers to:

- Present the same threats posed by the cross-site scripting vulnerability

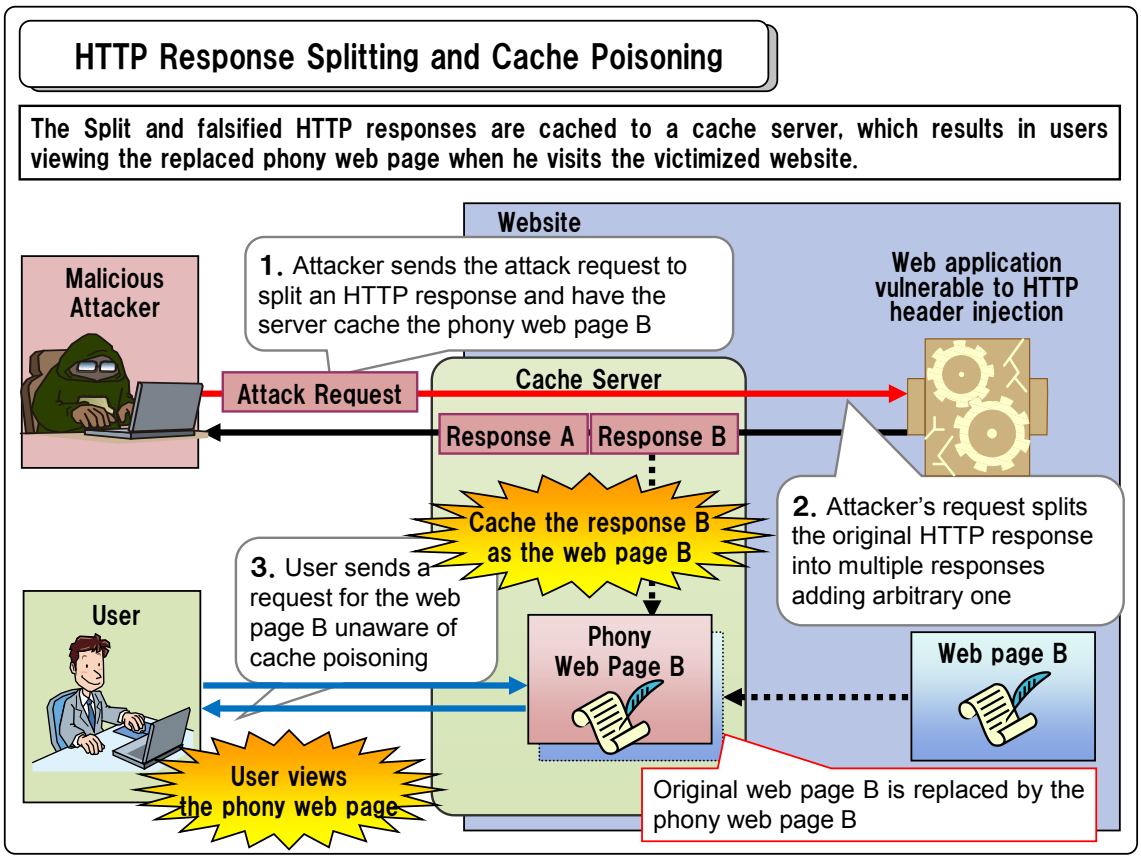
If an arbitrary response body is injected, the user's browser may result in displaying the false information or be forced to execute arbitrary scripts. Those are the same threats discussed earlier in "1.5 Cross-Site Scripting".

- Create arbitrary cookies

When an HTTP Set-Cookie header is inserted, an arbitrary cookie is created and stored in the user's browser.

- Poison web cache

HTTP response splitting forces a web server to generate multiple HTTP responses and could inflict cache poisoning[27], which results in web page falsification, by having a proxy server cache an arbitrary HTTP response and replacing the original cached web page with it. The users visiting the victimized website are to view the replaced phony web page. Compared to the cross-site scripting attack, in which only a targeted individual would fall victim just once right after the attack, the threat cache poisoning poses would affect a larger number of users and last long time.



■ Websites That Need Special Attention

Regardless of what kind of website it is or who operates it, the websites that dynamically set the value of the HTTP response header fields, such as the Location header field and the Set-Cookie header field, based on the values passed by the external parameters should be cautious about this vulnerability. The websites

[27] Watchfire Co. has published a paper on HTTP Response Splitting/Cache Poisoning: Watchfire: HTTP Response Splitting, Web Cache Poisoning Attacks, and Related Topics
http://www.watchfire.com/jp/securityzone/whitepapers.asp (link broken as of January 2010)
Some of the threats discussed in the paper would stem from the vulnerabilities in proxy servers or web servers. Check for "HTTP Request Smuggling"(*1) and "HTTP Response Smuggling"(*2)" with those products as well for they pose the similar threats.
*1 Watchfire: "HTTP Request Smuggling" http://www.watchfire.com/jp/securityzone/whitepapers.asp (link broken)
*2 SecurityFfocus: "HTTP Response Smuggling" http://www.securityfocus.com/archive/1/425593/30/0/threaded

that use cookies for session management and that have set up a reverse proxy should take extra caution.

■ Reported Vulnerabilities

Reports related to HTTP header injection vulnerability account only a few percents of all website-related cases but it keeps coming up since we started receiving the reports. The following are some of the software products with this issue reported to IPA. The vulnerabilities in these products are now fixed.

- Active! mail 6 Vulnerable to HTTP Header Injection
  http://jvndb.jvn.jp/jvndb/JVNDB-2010-000050
- Web Mailer from CGI RESCUE Vulnerable to HTTP Header Injection
  http://jvndb.jvn.jp/jvndb/JVNDB-2009-000024
- Multiple Cybozu Products Vulnerable to HTTP Header Injection
  http://jvndb.jvn.jp/jvndb/JVNDB-2007-000814

■ Fundamental Solutions

7-(i)-a

☞ Do not print out HTTP header directly and do it through an HTTP header API provided by execution environment or programming language.

In some web application execution environments, a web application may directly print out an HTTP response header specifying the fields such as `Content-Type`. In these cases, if the application prints out the input value passed by the external parameter straight to the field, line feed characters may be set along. A line feed character is used to separate the HTTP headers so that allowing line feed insertion may become the cause of arbitrary header/body injection or response splitting. Since the header structure is quite complex as you can see in the option of non-breaking space and difficult to take care of everything manually, you would better use an HTTP header API offered by programming languages or execution environments.

Note that some execution environments are reported to have vulnerabilities that their HTTP header API does not handle line feed characters appropriately. If that is the case, apply security patches to correct the problem, or take the measure 7-(i)-b or 7-(ii) if that is not possible.

7-(i)-b

☞ If HTTP header API that offers line feed neutralization is not available for use, implement it manually.

For example, you may add a non-breaking space after unexpected line feeds, remove the string that follows the unexpected line feeds[28], or stop printing out a web page when detecting the unexpected line feeds.

[28] See Corrective Measure in 3.7.

■ Mitigation Measures

7-(ii)

☞ **Remove all line feed characters that appear in the external text input.**

Remove all line feed characters that appear in the input text passed by the external parameters. You may even want to remove all control characters instead of just line feed characters. Note that if a web application needs to accept character strings that may contain line feeds, such as input data in the `<textarea> ... </textarea>` tags, systematically removing every single line feed from all input data may hinder the web application's proper operation and thus caution is advised.

By implementing these measures, security against HTTP header injection attacks is expected to improve. For more information on HTTP header injection vulnerability, you could refer to the following documents as well.

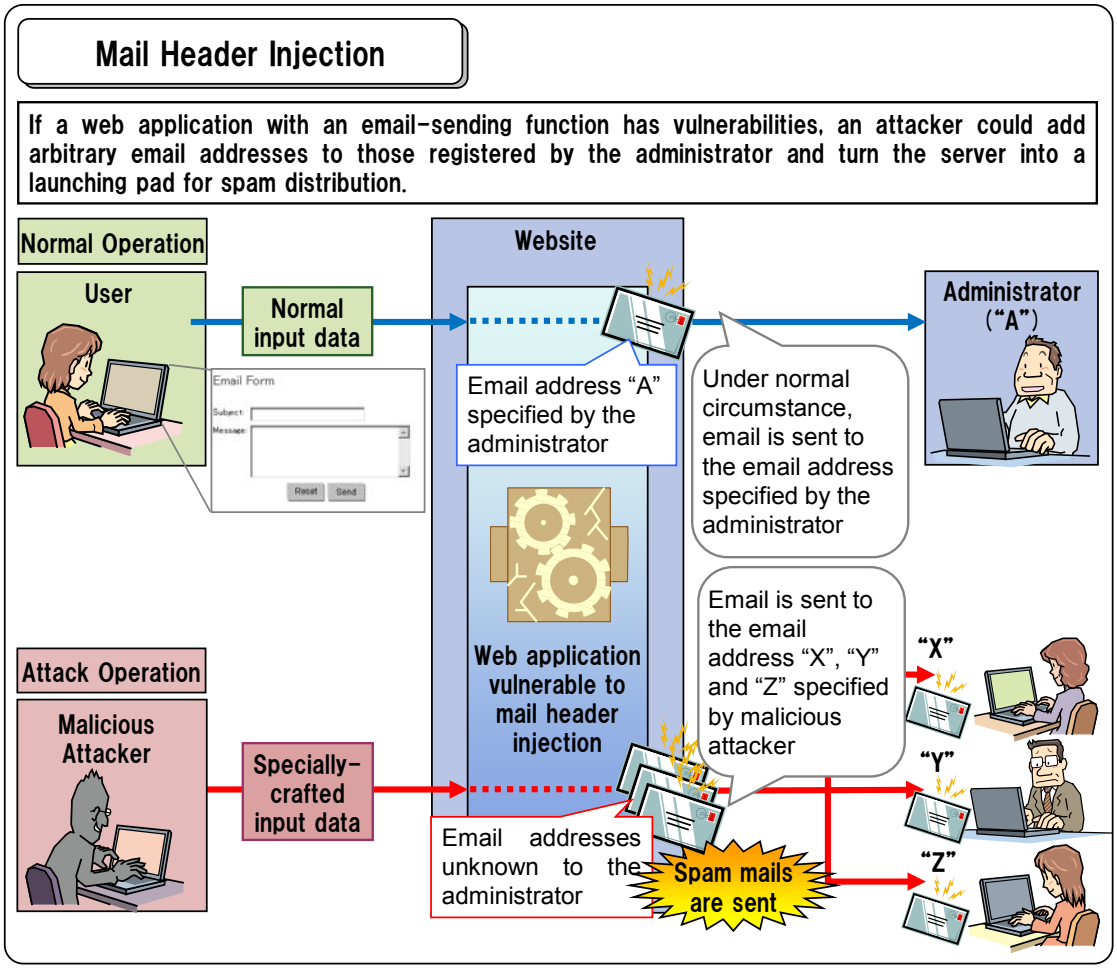
■ References

IPA: 知っていますか？脆弱性（ぜいじゃくせい）「7. HTTP ヘッダ・インジェクション」
http://www.ipa.go.jp/security/vuln/vuln_contents/hhi.html (Japanese Only)
http://www.ipa.go.jp/security/vuln/vuln_contents/hhi_flash.html (Japanese Only)

IPA: セキュア・プログラミング講座「HTTP レスポンスによるキャッシュ偽造攻撃対策」
http://www.ipa.go.jp/security/awareness/vendor/programmingv2/contents/603.html
(Japanese Only)

1.8 Mail Header Injection

Some web applications provide a function that sends emails to the particular email addresses about, for example, the merchandise the users have purchased or survey replies. In general, these email addresses are prespecified and only the web administrator can change. Depending on how it is implemented, however, an attacker may be able to set and change them to arbitrary email addresses. This vulnerability is called “Mail Header Injection” and the attacking method exploiting this vulnerability is called “Mail Header Injection attack”.



■ Possible Threats

This vulnerability could allow malicious attackers to:

- Third party mail relay

Used as a launching pad for spam distribution.

■ Websites That Need Special Attention

The websites that have a function to send the user input data to the administrator via email should be warned of third party mail relay. For example, a function like the “Contact” or “Survey” form could be susceptible.

■ Reported Vulnerabilities

Reports related to the vulnerabilities that enable Third Party Mail Relay attacks account only a few percents of all website-related vulnerabilities but it keep coming up intermittently since we started receiving the reports. The following are some of the software products with this issue reported to IPA. The vulnerabilities in these products are now fixed.

- FORM2MAIL from CGI RESCUE Allows Unauthorized Email Transmission
  http://jvndb.jvn.jp/jvndb/JVNDB-2009-000023
- MailDwarf Vulnerability Allows Unauthorized Sending of Emails
  http://jvndb.jvn.jp/jvndb/JVNDB-2007-000229

■ Fundamental Solutions

8-(i)-a

☞ Use the fixed values for the header elements and output all external input to the email body.

In a case where the value of email header elements, such as "`To`", "`Cc`", "`Bcc`" and "`Subject`", is to be set based on external input, or the data output process to the email sending function is vulnerable, if the external input is directly used as the output value, the line feed characters included in the external input will be inserted as unnecessary line breaks. If this is allowed, an attacker could exploit it to insert arbitrary email headers, alter the email body or send email to arbitrary addresses. It is recommended you do not use external parameters to set the value of the email header elements[29].

8-(i)-b

☞ If the fixed values cannot be used for the header, use an email-sending API offered by the web application's execution environment or language.

An example of where you cannot use the fixed value for the header elements is the case that you want to change the subject.

If you need to use the external input as the header values, it recommended to use an email-sending API offered by the web application's execution environment or language. However, some APIs cannot handle the line feed character appropriately or can insert multiple headers. In those cases, apply security patch or implement the necessary modification not to allow the line break on your own.

For example, to prevent the line break, you can insert a space or horizontal tab after the line feed character to have the program process the lines as one continuous line, delete the characters after the line feed character or stop generating a web page if the line break is detected.

[29] See Corrective Measure #1 in 3.8.

8-(ii)

☞ **Do not specify the email addresses in HTML.**

This may sound absurd but it did happen nevertheless and we feel we should warn you not to specify the recipient email addresses directly in the hidden parameter.

Implementation like specifying recipient email addresses directly in a parameter that is to be passed to the web application may be exploited by the third party mail relay attack by changing the parameter value.

■ Mitigation Measures

8-(iii)

☞ **Remove all line feed characters that appear in the external text input.**

Remove all line feed characters that appear in the input text passed by the external parameters[30]. You may even want to remove all control characters instead of just line feed characters. Note that if a web application performs the removal process on those that may contain line feeds, such as the mail contents, systematically removing every single line feed from all input data may hinder the web application's proper operation and thus caution is advised.

By implementing these measures, security against Mail Header Injection is expected to improve. For more information on Mail Header Injection, you could refer to the following documents as well.

■ References

IPA: 知っていますか？脆弱性（ぜいじゃくせい）「10. メール不正中継」
http://www.ipa.go.jp/security/vuln/vuln_contents/mail.html (Japanese Only)
http://www.ipa.go.jp/security/vuln/vuln_contents/mail_flash.html (Japanese Only)

IPA: セキュア・プログラミング講座「メールの第三者中継対策」
http://www.ipa.go.jp/security/awareness/vendor/programmingv2/contents/201.html
(Japanese Only)

[30] See Corrective Measure #2 in 3.8.

1.9 Lack of Authentication and Authorization

There are some inadequately designed websites in operation due to lack of the operator's security awareness. In this chapter, we will show you the vulnerabilities reported to us that stem form the lack of important functions such as "authentication" and "authorization".

1.9.1 Lack of Authentication

■ Fundamental Solutions

9-(i)

☞ When a web site needs access control, implement an authentication mechanism that requires users to enter some kind of secret information, such as password.

Normally, when the website handles sensitive information or allows only the owner/provider of each information to change or edit the information, it needs an authentication mechanism.

However, there was a reported case about a vulnerable website where a user could login to the website and access the personal information by just providing his or her email address.

Email address is open information available to others and using such information to limit access to personal information means having almost no authentication mechanism at all[31].

Design a web application to require something that people think it should be kept secret, such as password.

1.9.2 Lack of Authorization Control

■ Fundamental Solutions

9-(ii)

☞ Implement authorization as well as authentication to make sure that a login user cannot pretend to be other users and access their data.

When implementing authentication into the website and allowing only the owner/provider of each information to view or change the information, if more than one user are able to login and use the services at the same time, you may need to implement authorization with which you control who can do what.

[31] Paragraph 2 of Article 2 of the Unauthorized Computer Access Law defines "identification code" and (1) says it is "A code the content of which the access administrator concerned is required not to make known to a third party wantonly;". According to this definition, an email address may not be accepted as an identification code and in turn a login mechanism that requires only email address may not be accepted as an access control mechanism.
The Unauthorized Computer Access Law: http://www.ipa.go.jp/security/ciadr/law199908.html

A typical web application equipped with authorization mechanism issues a session ID to each user upon successful login to implement session management, and obtains the user ID from the user's session ID through session variables each time access is made. A simple web application can use such user ID as a key to allow database search or data modification. In this case, each user can access only his or her own database entries, thus it can be said that authorization control is implemented in terms of results.

Some websites, however, store the user ID in URL or POST parameters. Such implementation, where an externally manipulatable user ID is used as a key to allow database operations, leads to the vulnerability where any login user can pretend to be another user and access supposedly inaccessible data.

This problem stems from the lack of authorization control. You should implement a mechanism to verify whether a user ID requesting database access is the same as the user ID of a login user who is supposed to be issuing that request, or avoid getting the user ID from external parameters and obtain it through session variables.

In other cases, when a web application uses an order number as a key for database search or data modification, if the order number is stored in URL or POST parameters, it may lead to the vulnerability where any login user could set other user's order number in the URL or POST parameter and gain access to the order information of the other users, which should be kept private.

The cause of this problem is also the absence of authorization control. Always check whether an order number used to search the database is indeed an order number the requesting user has authorization over.

2. Approaches to Improve Website Security

This chapter discusses the approaches to improve website security. In the previous chapter, we have shown the solutions and countermeasures against vulnerabilities at the stage of software design and development. In this chapter, what we provide are solutions and countermeasures you could apply at the operational level.

2.1 Secure Web Server

To safely operate a website, the administrator should not only secure web applications but also securely guard the web server. Use the following chapters as reference, and see if your web server's settings and operation are secure.

1)

☞ **Check OS and software vulnerability information constantly and take necessary actions accordingly.**

Even though you implement an authentication mechanism to control access to the server, it can be breached if the attackers exploit the vulnerabilities in OS or applications. Vulnerabilities have been found on a daily basis. Check vulnerability information provided by OS and software vendors constantly and update software or practice necessary workarounds.

2)

☞ **Implement an authentication mechanism other than using passwords for remote server access.**

It is a common practice to allow remote access to the web server for management efficiency, but if authentication is done by just password, its security may be breached by brute force attacks. To ensure higher security, we recommend the use of a cryptographic authentication method, such as public key authentication.

3)

☞ **When using password authentication, make sure to use a sufficiently complex string.**

Make a password used to access the web server sufficiently complex. Refer to the following document for password management as well.

4)

☞ **Disable unused services and delete unnecessary accounts.**

If the services unused in operating the website are left enabled on the web server, it is likely for the administrators to neglect the proper maintenance over those services and keep using old versions full of

vulnerabilities. Likewise, if you allow unnecessary user accounts to be kept alive, they may not be carefully managed and used wrongly. Make sure to disable unused services and delete unnecessary accounts.

5)

☞ **Do not place a file you do not intend to make public under the public directory on the web server.**

☞ Basically, the files under the public directory on the web server are accessible from the outside world. Even if you do not offer the direct links to those files on the web pages, people can access them by directly specifying the path. Be careful not to place the files you do not intend others to view under the public directory on the web server.

■ References

IPA IT Security Center
http://www.ipa.go.jp/security/english/index.html

JVN（Japan Vulnerability Notes）
http://jvn.jp/en/

JVN iPedia Vulnerability Countermeasure Information Database)
http://jvndb.jvn.jp/en/

IPA：パスワードの管理と注意
http://www.ipa.go.jp/security/fy14/contents/soho/html/chap1/pass.html (Japanese Only)

IPA：セキュアな Web サーバの構築と運用に関するコンテンツ
http://www.ipa.go.jp/security/awareness/administrator/secure-web/ (Japanese Only)

2.2 Configure DNS Security

If the administrators are not careful in configuring and operating the domain names or the DNS servers they are using, malicious attackers could hijack their domain names. If domain names are hijacked, the visitors to the website can be directed to a phony website prepared by the attacker even though the visitors type in the correct URL. Domain name hijacking will affect not only website browsing but also emailing, and basically all Internet services. It may be a DNS issue but should be taken seriously for it will directly affect the website.

1)

☞ **Check on the registration status on domain names and DNS servers and take actions accordingly.**

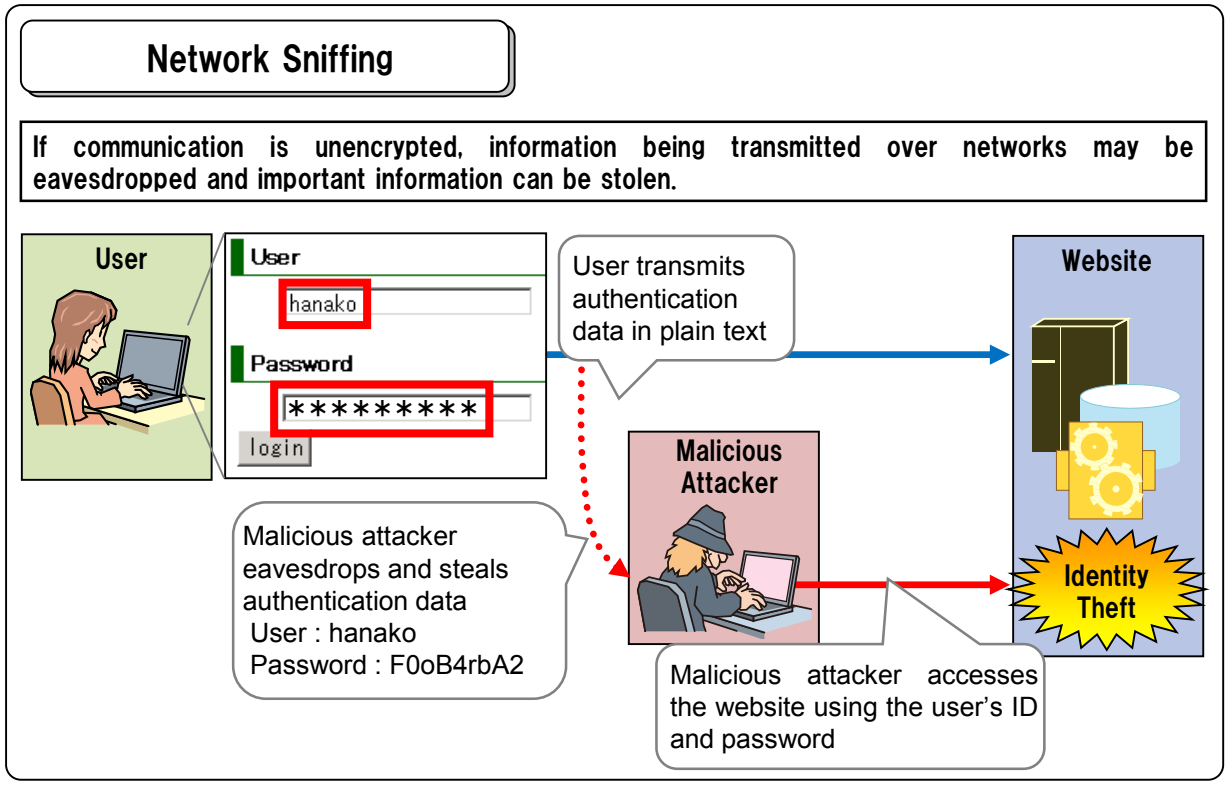
Check on the registration status on the domain names and DNS servers and take actions accordingly. If you are outsourcing the DNS server operation, you should ask the outsourcee to take necessary actions. For more information, you could refer to the following document as well.

■ References

IPA: ドメイン名の登録と DNS サーバの設定に関する注意喚起
http://www.ipa.go.jp/security/vuln/20050627_dns.html (Japanese Only)

2.3 Protect against Network Sniffing

The information to be exchanged between the website and the user may be leaked through network sniffing. If communication or data is unencrypted, captured information may be used for spoofing or other malicious purposes.



Since network sniffing takes place on the communication channel between the website and the user, it is difficult to prevent it solely by the website through configuration and operation. It is, however, possible to prevent an attacker to obtain sensitive information even the sniffing itself has succeed by encrypting the communication between the website and the visitor. Those websites which handle authentication information or personal information should consider adopting the following countermeasures against network sniffing.

1)

☞ **If the website handles important information, encrypt communication.**

A popular way to encrypt communication is to use HTTPS with SSL (Secure Socket Layer) or TLS (Transport Layer Security). If the website handles information that needs to be securely protected, such as personal information and authentication information, we recommend you encrypt the communication channel.

If you have the website on a rental server, know that some rental servers do not support HTTPS. The websites on those servers are better not to handle important information.

2)

☞ **Do not send important information via email to notify the user and show it on the HTTPS encrypted web page instead.**

There are cases where the website needs to inform the users about something important, such as personal information or password. If you are to send important information over the network, you should encrypt either communication or information itself in order to counter sniffing. When you need to notify information that requires encryption, use HTTPS and show it on the secure web page.

When you use email instead, you could encrypt the contents using S/MIME (Secure/Multipurpose Internet Mail Extensions) or PGP (Pretty Good Privacy) but they require users to set up a use environment and private key, thus may not be practical.

3)

☞ **Encrypt important information that a website operator receives via email.**

When a website is set to send sensitive user input, such as personal information, to the web operator using the email sending function of a web application, encrypt the data with S/MIME or PGP. If those technologies cannot be used, encrypt the email body with other methods.

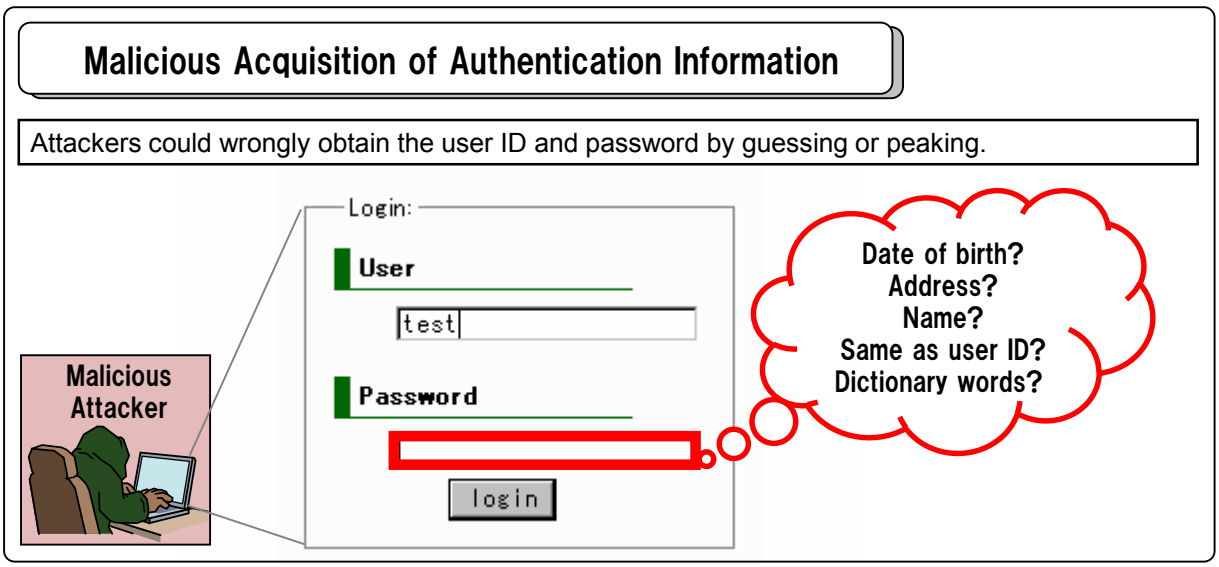
It is possible to encrypt communication between the mail servers (SMTP oversell) or between the mail server and the website operator (POP/IMAP over SSL), but it is unsafe because encryption may not be done on some intermediate channels depending on network configuration.

■ References

IPA：電子メールのセキュリティ 電子メールの安全性を高める技術の利用法
http://www.ipa.go.jp/security/fy18/reports/contents/email/email.pdf (Japanese Only)

2.4 Secure Password

Most common way of implementing user authentication is to use a user ID /password pair. If password management and processing by the website are inappropriate, the risk of a malicious attacker stealing user authentication information becomes higher.



One way to wrongly obtain the user ID or password is guessing it. This is often tried for the websites with a simple password scheme full of easy-to-guess passwords. How you display a web page may give out even more hints for attackers to work on. If the website has an authentication mechanism, be careful about the following points.

1)

☞ **Set hard-to-guess default passwords.**

When issuing a default password, use secure random numbers to eliminate regularity and, if possible, make it long and use alphabets, numbers and symbols. If password generation has regularity, an attacker could register more than once and try to work out the generation mechanism. Some users may never change their default password, thus making the default password difficult to guess is essential.

2)

☞ **Require users to enter the current password to change password.**

Make sure to require the user to enter the current password to change password.

3)

☞ **Do not give away unnecessary hints in authentication error message.**

When a user makes a mistake in providing authentication information, returning “Password doesn’t match” on the error page would imply that “the user ID is correct”. This could help an attacker finds out the user IDs and thus not recommended. Make the error message like “Invalid user ID or password” and try not to give away clues possibly used to guess authentication information unnecessarily.

4)

☞ **Mask password being entered in the password box.**

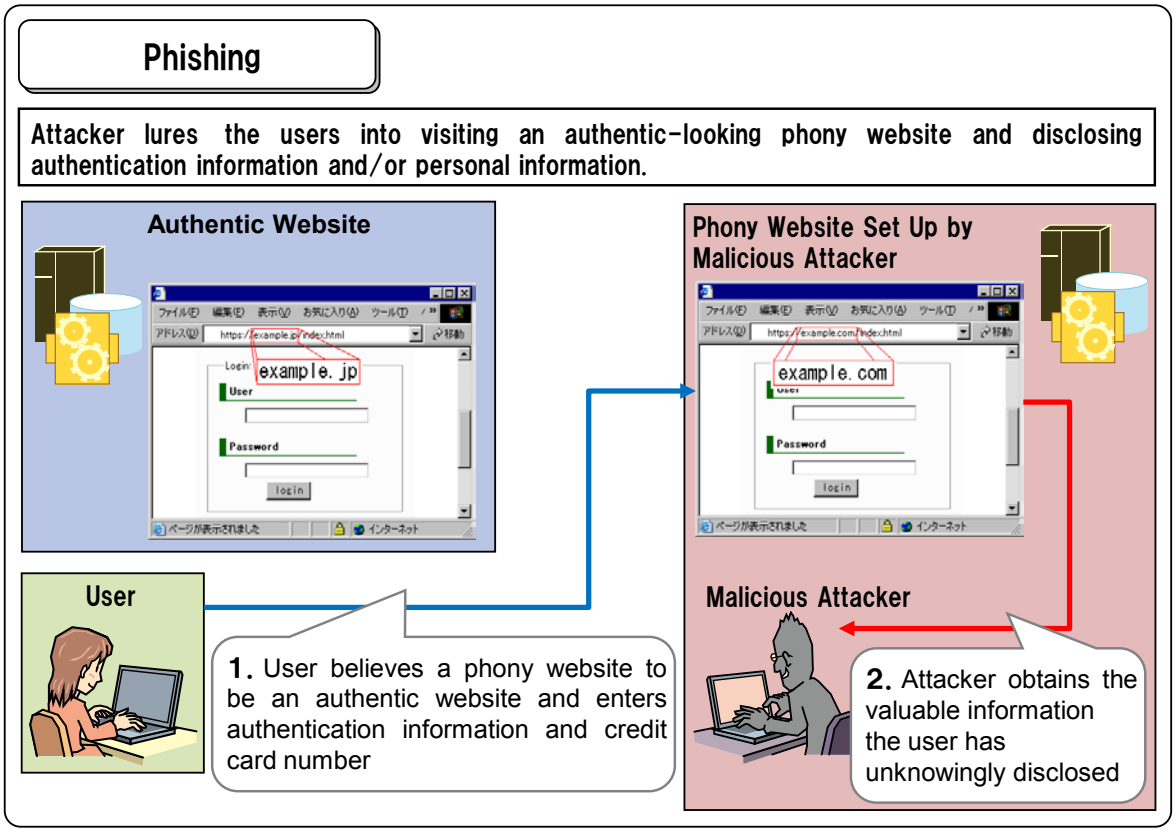
☞ Mask the password entered by the user with asterisk (*).

■ References

IPA: セキュア・プログラミング「ユーザ認証対策 パスワードフィルタ」
http://www.ipa.go.jp/security/awareness/vendor/programmingv2/contents/101.html
(Japanese Only)

2.5 Mitigate Phishing Attacks

Phishing is malicious action to steal user authentication information and financial information like credit card numbers by creating phony online banking or shopping websites and luring the Internet users into visiting them to make the visitors disclose the information. Users need to be diligent to avoiding falling victim to phishing scam, but some careless websites may hinder the users' efforts and result in fostering phishing.



To protect themselves, the users need to be able to check the website they are visiting and verify that it is the authentic one. The website operators should think of adopting the following methods to allow their users to confirm the authenticity of the website.

1)

☞ **Get an Extended Validation (EV) SSL certificate and allow users to verify who the website operator is.**

A server certificate is necessary to properly implement SSL-encrypted communication but it can be used to verify who the operator of the website is as well. The users can check the server certificate when a web page requires them to enter their password or credit card number and confirm who the operator of the website they are dealing with is.

Use a server certificate that is officially issued by CA. Do not use a self-issued certificate for it is difficult to distinguish a self-issued one from a forged one and does not offer any security.

Some CA services do not support EV SSL certificates. A non-EV SSL certificate, provided it is an official one, does enable SSL-encrypted communication, but it indicates the website operator with the server's hostname, thus is not capable of showing who the operator is.

2)

☞ **When using frames, do not generate child frame URL based on external parameter values.**

If a website uses frames and sets a child frame URL based on external parameter values, it is possible for attackers to exploit this approach for phishing. If an attacker inserts an arbitrary URL into the external parameter, the user is redirected to a phony website set up by the attacker within the parent frame of the authentic website. The domain name shown in the address bar will be the correct one and that makes it difficult for the users to see that the child frame is in fact a phony web page.

3)

☞ **When dynamically managing where to direct the user after login using the redirect function, make sure to permit only URLs within your own site's domain as a redirect-to URL.**

When a user tries to access a web page that is accessible only after login, some websites retain the URL of the web page in a parameter and after the user has logged in successfully, they will redirect the user to that web page. If the website does not limit the value of the parameter that holds a redirect-to URL, an attacker could exploit it by setting the URL of a phishing website in the parameter.

Under this phishing attack, a careful user may check if the login screen displayed is authentic, then confirming that, the user will log in. The user may not, however, keep his or her diligence and check whether or not the redirected web page is authentic. For example, if a redirected web page, which is a set-up by the attacker, looks the same as the authentic login screen and says "Login Error. Please Try Again", the user would likely conclude that s/he may have mistyped the password and reenter the authentication information without much suspicion.

Do not allow arbitrary URLs to be set in the redirect URL parameter and permit only the URLs that are within your own website's domain. In addition, make sure to implement this countermeasure into all the web pages that use the redirect function.

For more information on mitigating phishing attacks, you could refer to the following documents as well.

■ References

IPA: PKI 関連技術解説 「認証局と電子証明書」
http://www.ipa.go.jp/security/pki/031.html (Japanese Only)

IPA: セキュア・プログラミング講座「真正性の主張」
http://www.ipa.go.jp/security/awareness/vendor/programmingv2/contents/202.html
(Japanese Only)

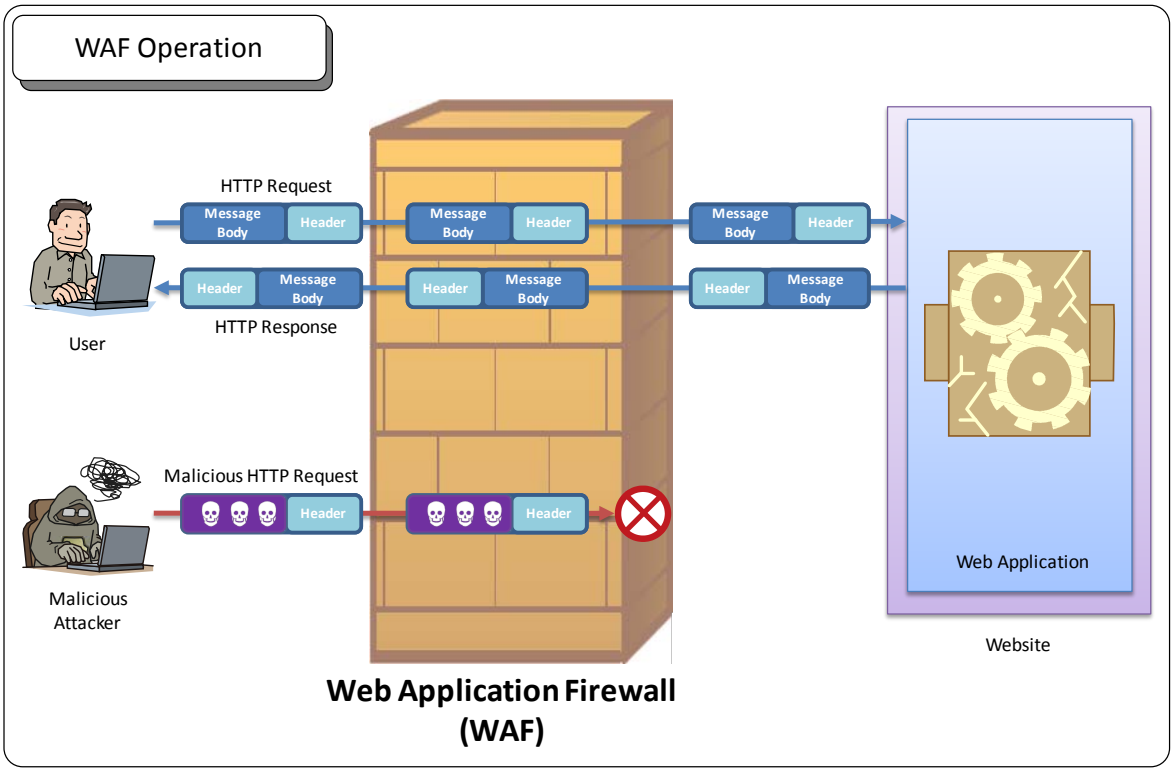
産業技術総合研究所: 安全な Web サイト利用の鉄則
http://www.rcis.aist.go.jp/special/websafety2007/ (Japanese Only)

2.6 Protect Web Applications with WAF

To secure a web application, it is important to make sure to not have vulnerabilities in the first place and to promptly remove vulnerabilities if found any. While securing the web application itself as far as possible, the administrator could add another layer of security to its operation using a WAF (Web Application Firewall) to protect the web application from cyber attacks that try to exploit web application vulnerabilities.

WAF is a security software or hardware that inspects HTTP traffic (including HTTPS[32]) between websites and users, and automatically cut off harmful traffic like cyber attacks. By using a WAF, the following benefits are expected:

- **Protect web applications from cyber attacks that try to exploit vulnerabilities.**
- **Detect cyber attacks that try to exploit vulnerabilities.**
- **Prevent multiple web applications from cyber attacks.**



WAF

Depending on the developmental status and operational situation of a web application, making good use of the WAF may be more effective than taking time and money in modifying the web application.

[32] Some WAF may not monitor and check HTTPS transmission,

■ Cases where use of WAF is more effective - where modification of application is difficult -

Web application developers should try to make an application vulnerability-free in the first place using the chapter 1 of this guideline “Web Application Security Implementation”. However, new vulnerabilities could be found after completing the development of the application. The new vulnerabilities must be taken care of promptly, but any modification of the application at that stage may be infeasible. Under such situation, WAF can be used to mitigate the effect of cyber attacks to protect the web application. For example, WAF would be useful in the following situations.

1) When it is difficult to have the application developer modify the application

☞ When a vulnerability is found in an application, sometimes it may be difficult to have the application developer directly modify the application.

When an organization decides to develop an application, it may outsource the application development to an outside company. When a vulnerability is found in the application, there might be a case where having the company that developed the application modify the application is difficult (i.e. the company is no longer in the software development business).

It is possible to have some other company modify the application, but the cost could be much higher and over budget, making the modification infeasible.

2) When a vulnerability is found in license protected software applications

☞ When a website is built with commercial products or open source software, it may be difficult to be actively involved with and make sure of modification of the application.

In recent years, web applications, like wiki and blogging applications, are available both as commercial and open source software, enabling anyone to use a web application without developing it oneself.

When a vulnerability is found in those software, it is up to the software developers whether and when to modify and provide an update version or security patch. If the support period for the software is already over, it could be possible that the vulnerability is left as it is.

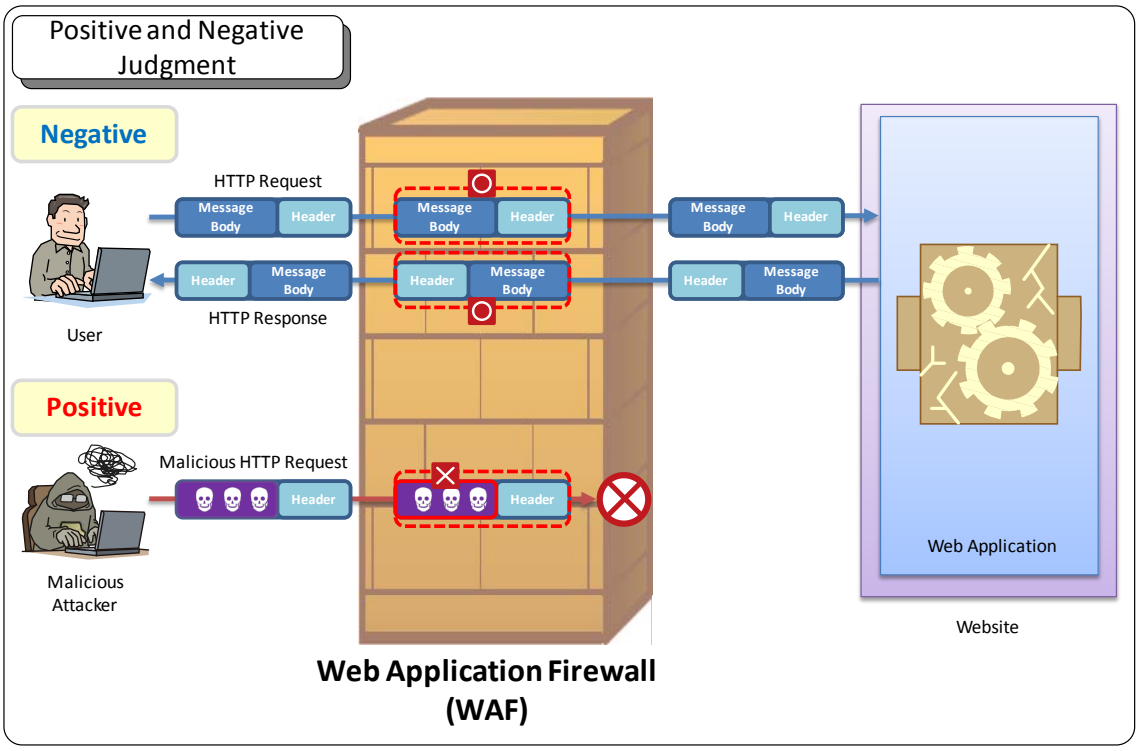
As for open source software, the user organization can confirm the vulnerability and modify the software if it has the software engineers capable to do it, but not all the user organizations may have the luxury of the in-house talent.

■ WAF filtering of HTTP traffic

WAF automatically scans HTTP traffic like HTTP requests and HTTP responses between websites and users based on the WAF rules created by the website administrator. Through its scanning, a WAF checks whether or not the traffic are "harmful" to the website and the user and filters the traffic. The WAF rules are set to catch things like the character strings that can be used in cyber attacks to exploit web application vulnerabilities and parameter types and values defined in the web application specifications[33].

When the scanned HTTP packets are deemed "rightful" (judged negative), the WAF forwards the HTTP packets to the website or the user. On the other hand, when the HTTP packets are deemed "harmful" (judged positive), the WAF proceeds to execute the preset actions, such as notifying the administrator and cutting off the transmission, without forwarding the HTTP packets.

Because WAF does this filtering mechanically, sometimes filtering errors occur.



Positive and Negative Judgment

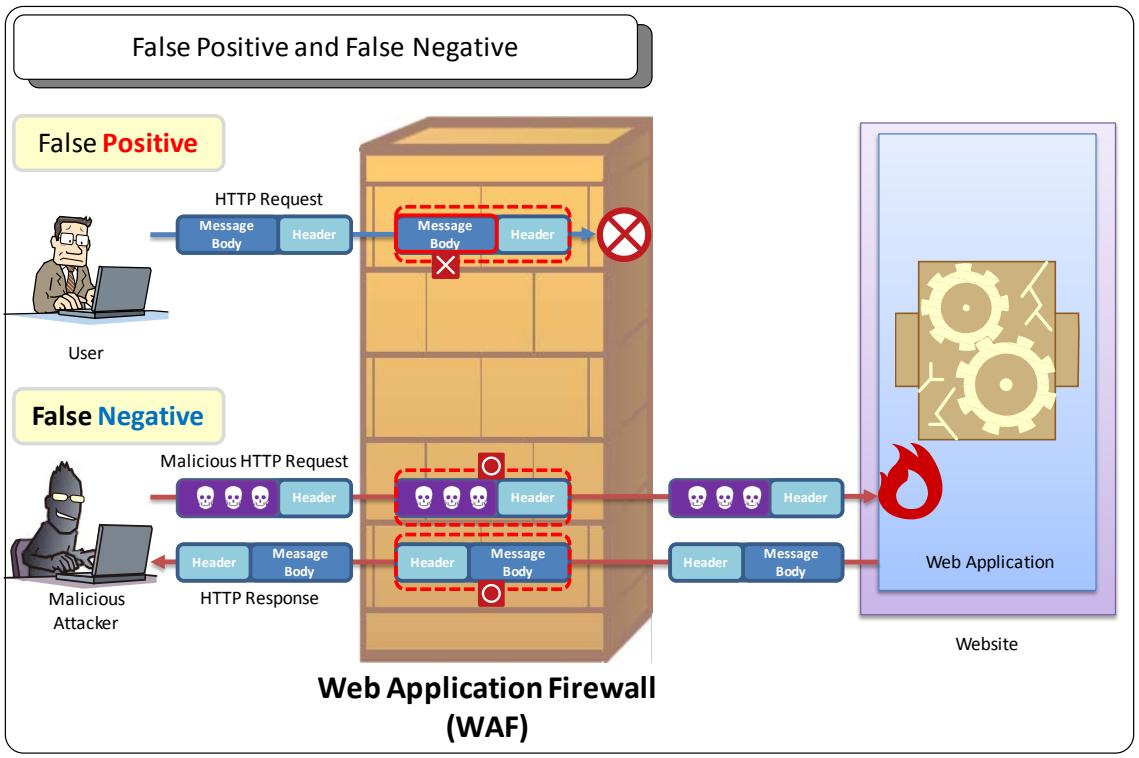
[33] For example, say, a web application has a parameter named "id". If the web application expects numeric numbers as the value for "id", then any values other than numeric numbers (e.g. a character string "example") are illegal for the "id" parameter. To protect this web application with a WAF, you can set a WAF rule that allows only numeric numbers as the value for the "id" parameter.

■ HTTP filtering errors

Depending on the contents of HTTP packets, sometimes filtering errors occur. There are two types of errors: false positive and false negative.

False positive is a rightful HTTP packet that gets mistakenly judged harmful. Similarly, false negative is a harmful HTTP packet that gets mistakenly judged rightful.

When using WAF, the website administrator should take into account the possibility of filtering errors of false positive and false negative.



False Positive and False Negative

■ False positive and false negative in WAF

1) False Positive

【Causes】

False positive errors in a WAF occur when the WAF rules to filter the HTTP traffic between the web application to be protected and the user are not properly defined.

【Impact】

The availability of the website is reduced due to the filtering of rightful HTTP packets.

【Examples】

The rightful HTTP packets can be cut off by a WAF if the WAF rules are not well elaborated.

Let's assume that a WAF rule is defined to detect HTML special characters "<" and ">" and drop the packet if it contains them to protect the website visitors from cyber attacks that exploit cross-site scripting vulnerability. In this case, just entering a mathematical expression including "<" and/or (depending on the rule) ">" will make the WAF cut off the transmission.

In general, WAF rules do not include such obviously standard characters mentioned above, but false positive is not 100 percent avoidable because of the way the WAF works.

2) False Negative

【Causes】

False negative errors in WAF occur in the following 2 situations:

a) The WAF rules are not defined appropriately to detect harmful HTTP packets.

b) The WAF rules are eased up to minimize the risk of false positive occurrence.

【Impact】

The web application cannot be properly protected from cyber attacks that try to exploit web application vulnerabilities.

【Examples】

WAF cannot detect harmful HTTP packets because of the behavioral difference between the WAF and the web application.

When the parameters with the same name exist in the different fields of an HTTP request, such as a query string, message body and Cookie, how to process the parameters differs depending on the programming language the web application is written, its middleware and how the web server is configured. It is reported that taking advantage of this difference[34] and sending an HTTP request where a malicious character string built to exploit the web application vulnerability is split into the same-named multiple parameters may work to avoid the WAF detection.

[34] The attacking method that exploits the software's behavioral difference in handling the same-named parameters is called HTTP Parameter Pollution (HPP).

A reference material at the end of this guideline, "HTTP Parameter Pollution[35]", shows an example of this scenario with an open source software ModSecurity. It points out that a WAF rule "`select 1, 2, 3`" cannot detect the following HTTP request as harmful[36].

```
index.aspx?page=select 1&page=2,3 from table where id=1
```

When the specifications of a protocol are not clearly defined by the RFC or equivalent documents, the developers of programming languages, middleware and web servers are left to implement those parts of the protocol based on their own interpretation. The differences in interpretation and implementation lead to the differences in the behavior of the software as well as the WAF. Taking advantage of those differences, the attackers may succeed in avoiding the WAF detection of the attacks that try to exploit web application vulnerabilities.

■ To securely employ WAF

To reduce the risk of false positive and false negative occurrence, it is useful to have a test period during which the WAF is set not to drop the positive HTTP packets that have matched the WAF rules and just monitor HTTP traffic. The administrator should use the test period to see if the rightful HTTP traffic is not mistakenly cut off by the WAF and the WAF rules are appropriately defined to protect the web applications. Confirming the adequateness of the WAF operation requires good understanding of the web applications to be protected and expertise on the HTTP and related protocols, and time to test thoroughly. Using the outside entity capable for the task is a good option, too.

For more information on protecting web application with WAF, you could refer to the following document as well.

■ References

IPA Web Application Firewall (WAF) Guide
http://www.ipa.go.jp/security/vuln/documents/waf_en.pdf

[35] Luca Carettoni, Stefano diPaola, "HTTP Parameter Pollution", OWASP AppSec Europe 2009
http://www.owasp.org/images/b/ba/AppsecEU09_CarettoniDiPaola_v0.8.pdf

[36] The latest version of ModSecurity v2.5.10 + Core Rule Set v2.0.2, as of October 2009, supports the rules to detect HPP.

2.7 Secure Mobile Websites

Security measures addressed in this book is necessary regardless of whether the website is a PC website or mobile website[37]. However, when creating mobile websites, sometimes the mobile websites require the different design from those for the PCs due to the restriction of the available functions. This section discusses the problems often found with the mobile websites and things to take care of. When creating a mobile website, take into account the issues addressed in this section and consider modifying the design of the website if necessary.

2.7.1 Issues with Session Management

Until May 2009, the mobile browsers for all models of some mobile phone service providers (hereafter called "carriers") did not support the basic HTTP functions like the cookie and Referer. Hence, the mobile websites were required to design without the use of those functions.

After May 2009, some models of those carriers began to support the cookie and Referer and some still do not.

The models that do not support the cookie are forced to store the session ID in the URL to manage sessions. As shown in the fundamental solution 4-(ii) in 1.4, storing a session ID as an URL parameter generally poses a risk of session hijacking attacks since the browser sends the URL that includes a session ID through the function that passes the Referer to the linked website. To prevent the risk, some mobile websites avoid including external links in them or, even if included, the websites are designed to first redirect to a web page whose URL does not include the session ID before allowing to access external websites. However, those measures do not solve the fundamental cause and personal information leakage incidents have kept occurring due to the case where the user publishes the URL and it is reflected by search engines.

Implementing the workarounds like the above should be avoided as much as possible. It used to be adequate to decide the implementation method at the carrier level and apply such workarounds only for the carriers that did not support the cookie, and implement the same, common cookie-based session management as that for PCs for other carriers. After May 2009, however, one carrier may offer both cookie-supported and unsupported models, and thus the implementation method should be chosen at the model level instead of the carrier level.

Under the current circumstance, creating a website using old know-hows that are unique to mobile environment may endanger the safety of the website. It is necessary to revise the old know-hows.

2.7.2 Issues with Cross-Site Scripting

Mobile phones released before 2009 mostly did not support JavaScript, but after 2009, the models that supported JavaScript, XMLHttpRequest and some other functions began to come out, getting advanced just

[37] In this section, a mobile website means a web service provided by Japanese mobile carriers (e,g, i-mode and EZweb).

like PCs.

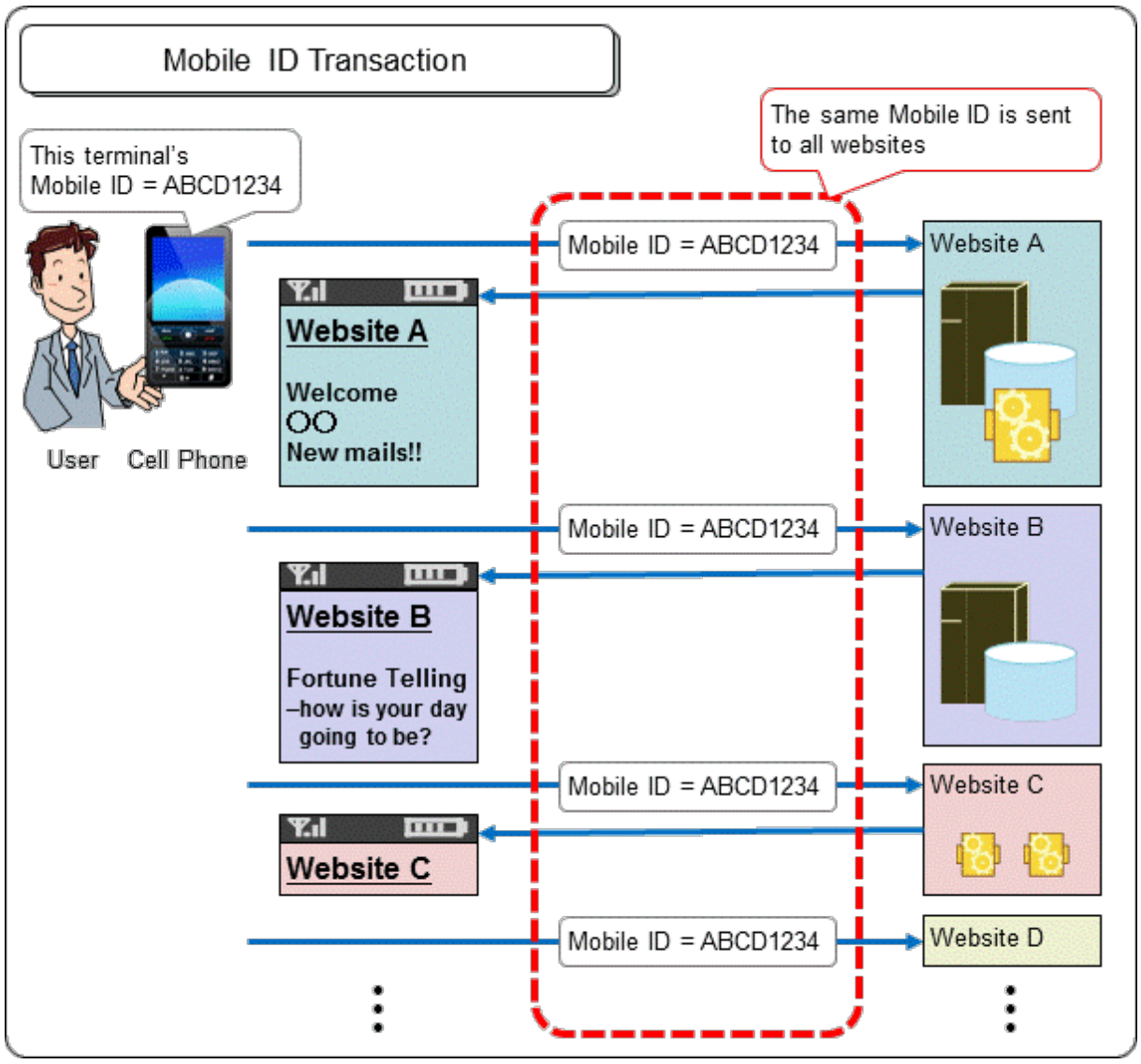
In the days when mobile browsers did not support JavaScript, it was said that implementing the countermeasures against cross-site scripting attacks was unnecessary for the mobile websites. However, not-so-small number of mobile browsers start to support JavaScript these days and implementing the countermeasures against cross-site scripting attacks is necessary just like for PCs.

2.7.3 Issues with Mobile ID

■ What is Mobile ID?

When the users access a website using their mobile phone, sometimes an identifier allocated to the mobile phone or mobile phone service user (hereafter called “mobile ID”) is sent to the website. The official name for the mobile ID differs depending on the carriers. Popular examples are “i-mode ID”, ”EZ Number”, “User ID”, “FOMA Terminal Serial Number”, “FOMA Card Serial Number” and “Terminal Serial Number”. The mobile ID has the following characteristics.

(1) The same mobile ID is sent to all websites.
(2) It is sent to all websites regardless of whether or not they are the carrier-certified official websites.
(3) It is stored in the HTTP request header (User-Agent header or an extension header unique to each carrier) and sent to the website.
(4) By changing the settings, the user can stop sending the mobile ID. The default setting is to send the mobile ID.



At first, some carriers set to send out the mobile ID only to their official websites. After March 2008, however, all carriers set to send out the mobile ID to all websites. This sped up the use of the mobile ID rapidly and the websites with mobile ID vulnerabilities have been surfaced.

■ Vulnerable Authentication using Mobile ID

Some websites authenticate the users using only the mobile ID. Such authentication method is often called "easy login". However, since the mobile ID is sent to all websites, it is a public information. Therefore, just checking if a user entered the correct mobile ID is not enough to authenticating the user. In the past, there were two premises to support the validity of the authentication using the mobile ID.

(a) Access to a mobile website is made only through a mobile browser or a mobile website can distinguish whether the access is made from a mobile browser or other medium.

(b) A user cannot modify the HTTP request header through the mobile browser.

However, these premises do not stand anymore these days.

To meet the premise (a), the websites often implement access control based on the source IP address using the list of IP addresses provided by the carriers. However, the list is not very reliable because the

carriers do not guarantee the accuracy, authenticity or recency. In addition, some carriers allow to access the mobile websites through the PC using the same source IP addresses.

To ensure the safety of the user authentication using the mobile ID, it is necessary that the mobile ID sent to the mobile website cannot be faked. In reality, due to the broken premise (a), it is known that some carriers cannot prevent the mobile ID allocated to each terminal from being faked, and depending on the implementation of the web applications, the mobile ID allocated to each mobile phone service user can be faked as well[38]. For that matter, a mobile ID is easily faked with smartphones[39].

As described, authenticating the user using the mobile ID is not that easy. We recommend implementing the same user authentication methods as PCs, such as using the cookie or password, or other safe methods provided by the carriers. The carriers may give out the information about how to use the mobile ID safely to the websites that are officially certified by the carriers (so-called "official websites"), but non-official websites cannot obtain the information. As a result, the safety of the websites may be neglected. For more information about authentication, see the next section, too.

2.7.4 Issues with Authentication Information

In this section, the issues about authentication information (the information the website uses to authenticate the user) that are often seen in the mobile websites.

■ Use Non-Secret Information as Authentication Information

Authentication information, such as password and PIN, should be secret between the user and the website.

Some websites use a non-secret information like the date of birth as the authentication information, but people other than the user him- or herself likely know the user's birth date and it cannot be used as the authentication information. The safe authentication cannot be achieved with such non-secret information[40].

[38] Session Management Vulnerabilities in Mobile Web Application
http://staff.aist.go.jp/takagi.hiromitsu/paper/scis2011-IB2-2-takagi.pdf

[39] In this book, smartphones means the mobile devices whose browser directly accesses the mobile services using the HTTP protocol instead of through the carrier's gateway.

[40] The date of birth and telephone number do not meet the definition of identification code (the code that is not to be easily shared with the third parties) defined in the Article 2 of the Unauthorized Computer Access Law. Thus, the websites solely using such information as the authentication information may be considered that they do not offer an access control protection.

■ Issues of Authentication Strength

To prevent the third parties from guessing or finding the authentication information by trial and error, the websites need to provide the environment where the users could set up and use a sufficiently sophisticated authentication information.

The mobile phone's user interface is different from PC and unsuitable for entering a long character string. For that reason, the mobile websites often end up accepting only the numbers as the authentication information. However, such authentication can be easily broken.

For example, in the case of using a 4-digit number as the authentication information, there are only 10,000 combinations and the correct combination is very likely obtained by trial and error.

The 4-digit authentication may seem safe since it is widely adopted as a form of identification by the banks for the ATM transactions or the access to the call centers, but it is valid because the number of times a user can authenticate him- or herself by try and error is restricted. If it is not restricted, the safety of this method is lost.

For the websites, restricting the number of times one can try to authenticate by trial and error is quite difficult in most cases. For example, it can be implemented by locking up the user account after the authentication attempt is continuously failed for the certain number of times, but such a simple measure cannot prevent the reverse brute force attack where a third party tries to break the authentication by trial and error changing the user IDs instead of passwords.

The authentication information is the one and only factor the web service can use and rely on. Do not limit the authentication information to the numbers and allow the users to use a sophisticated, long password that consists of alphabets, numbers and symbols.

■ Security and Convenience

A strong password that has a higher password pattern takes time to enter and is burdensome for the users. For that, when designing a website, the developer may be tempted to design one with a lower password pattern for convenience instead of security. However, considering the safety of the users, keep the strength of the password with a higher password pattern and reduce the frequency of entering the password.

There are some ways to reduce the frequency to enter the password. A popular example measure is to use the cookie to issue a session ID effective for the certain period of time[41] and consider that the user has successfully logged-in while the session ID is valid. PC websites often offer the choices to the users whether or not to use this function with the explanation such as “next time you do not have to log in “ or “it will keep the logged-in status”

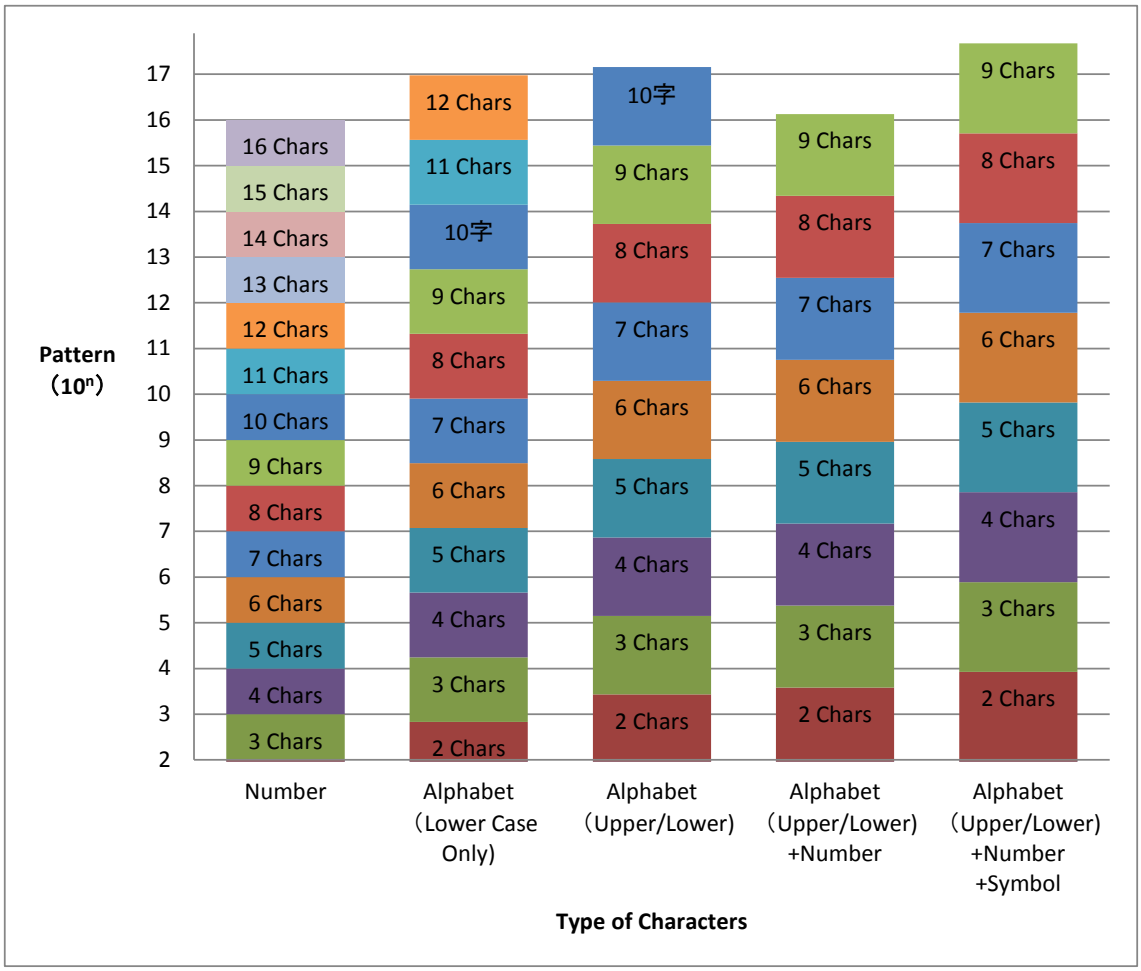
The longer the validity period of the session ID is, the lesser the entry of the password. Consider an adequate time of period depending on the services provided by the websites.

■ Types of Characters Used for Password and Password Pattern

For the PC websites, the users are often recommended to use a password that consists of alphabets, numbers and symbols in combination. Such password is strong but can be unpractical to be used for the mobile phone. For the mobile phone, a number-only password is feasible but in that case, we recommend you increase the number of digits and make sure that the password has a sufficient password patterns. The figure next page shows that the number of password patterns per popular character combinations for the password. Use the graph to keep the adequate level of password patterns.

[41] The session ID use here cannot be guessed by the third party. For more information, see Fundamental Solution 4-(i) on page 18.

For example, to achieve the same level of the password pattern as an 8-character password that consists of alphabets (do not discriminate the upper/lower case letters), numbers and symbols, 16 digits are required. Also, you can see that a 4-digit password has only the same level of the password pattern as a 2-character password that consists of alphabets (do not discriminate the upper/lower case letters), numbers and symbols.



3. Case Studies

Up to this chapter, we have shown the vulnerability measures for the web application and the approaches to improve the website security. In this chapter, we will present two cases of failure where vulnerabilities were not properly taken care of and show some examples of how to fix them.

3.1 SQL Injection

In this section, we will show sample user authentication programs, which fail to properly counter the SQL Injection vulnerability.

■ PHP and PostgreSQL

【Vulnerable Implementation】

```
$query  = "SELECT * FROM usr WHERE uid = '$uid' AND pass = '$passh';
$result = pg_query($conn, $query);
```

PHP

The above is part of source code implementing user authentication.

`$uid` in the first line is the user ID to be provided by the user. `$passh` is a hash value the web application calculates based on the password the user enters. In the first line, the web application uses these variables to compose an SQL statement and assigns them to `$query`. The `pg_query()` function[42] in the second line is a PostgreSQL function provided by PHP and executes `$query`, which is the SQL statement set in the first line. This sample program, however, lacks escaping process for the `$uid` value, which allows an attacker to launch SQL injection attacks by inserting a specially-crafted value that would turn into a malicious SQL statement.

【What's Wrong?】

Like in this case, if a web application does not perform escaping for the values passed by the external parameters, it may cause execution of unexpected SQL statements.

For example, suppose a user enters "`taro'--`" as user ID, the SQL statement to be sent to the database will be the following:

```
SELECT * FROM usr WHERE uid = 'taro'--' AND pass ='eefd5bc2...'
```

SQL

The single quote (') used in the SQL statement above is a special character, which defines a string literal by enclosing a data string within a pair of single quotes. Likewise, two consecutive hyphens (--) are a special character which tells the database to ignore everything that comes after it as comments. Which means the database will ignore [' AND pass = eefd5bc2..] when the value [taro' --] is set in `$uid`. As a result, the SQL statement to be sent to and executed by the database would become like this.

[42] pg_query: http://jp.php.net/manual/en/function.pg-query.php

```
SELECT * FROM usr WHERE uid = 'taro'--
```
SQL

What it means is that if a user account ”taro” does exist in the database, the attacker could log in without knowing taro’s corresponding password. What’s more, not only bypassing user authentication but also arbitrary database manipulation becomes possible by just changing a string to feed into `$uid`. This problem is caused by the fact that the web application does not perform escaping for the value of the elements that compose an SQL statement.

The `pg_query()` function is capable of executing multiple SQL queries. If this function is vulnerable to SQL injection, an attacker could insert more queries in addition to the original one, which will heighten the threats. The below is an example illustrating this issue.

```
// Set two SQL queries in $query
$query = "SELECT item FROM shop WHERE id = 1;
          SELECT item FROM shop WHERE id = 2;"
$result = pg_query($conn, $query);
```
PHP

【Corrective Measure #1】

Use the prepared statements

Use the `pg_prepare()` function[43] or the `pg_execute()` function[44] instead of the `pg_query()` function.

```
$result = pg_prepare($conn, "query", 'SELECT * FROM usr WHERE uid= $1 AND pass=$2);
$result = pg_execute($conn, "query", array($uid, $passh));
```
PHP

The `pg_prepare()` function and the `pg_execute()` function are PostgreSQL functions provided in PHP 5.1.0 and later and supported only by PostgreSQL 7.4 and later.

The `pg_prepare()` function generates a prepared statement. Its third argument is an SQL statement where the variables are referred to using the placeholders (bind variables) `$1`, `$2`... without actual value.

The `pg_execute()` function executes the prepared statement the `pg_prepare()` function has created. When the placeholders are used in a prepared statement, the `pg_execute()` function converts each element of the third argument (`$uid` and `$passh` in this case) into a string and set them in the corresponding placeholders (called “binding”) and executes the completed SQL statement. The use of placeholders saves you from explicitly performing escaping.

[43] pg_prepare: http://jp.php.net/manual/en/function.pg-prepare.php
[44] pg_execute: http://jp.php.net/manual/en/function.pg-execute.php

【Corrective Measure #2】

Use the function equipped with a placeholder capability

Use the `pg_query_params()` function[45] instead of the `pg_query()` function.

```php
$result = pg_query_params($conn, 'SELECT * FROM usr WHERE uid = $1
                                  AND pass = $2', array($uid, $passh));
```

The `pg_query_params()` function is a PostgreSQL function provided in PHP 5.1.0 and later[46] and supported only by PostgreSQL 7.4 and later.

The `pg_query_params()` function does not create a prepared statement but is equipped with a placeholder capability. It takes an SQL statement in which placeholders (`$1`, `$2`, ...) are used as the second argument and the actual values for the placeholders as the third argument. The use of placeholders saves you from explicitly perform escaping.

【Corrective Measure #3】

Use an escape function

Use the `pg_escape_string()` function[47] and perform escaping for all elements in an SQL statement to be executed by the `pg_query()` function.

```php
$query = "SELECT * FROM usr WHERE uid = '".pg_escape_string($uid)."'
                            AND pass = '".pg_escape_string($passh)."'";
$result = pg_query($conn, $query);
```

The `pg_escape_string()` function is a PostgreSQL function provided in PHP 4.2.0 and later and supported only by PostgreSQL 7.2 and later. It will escape the special characters designated in PostgreSQL.

You can write an escape function yourself but it will be difficult to cover all the special characters that have a unique meaning in PostgreSQL, thus not recommended. Use `pg_escape_string()` and it will perform necessary escaping automatically.

In the code above, `$passh` goes through escaping process as well. `$passh` is a hash value calculated from the password and unlikely to be exploited in SQL injection attempts. Nevertheless, we recommend performing escaping for these internally processed elements like `$passh` as well. This will save you from checking all elements whether or not you should perform escaping for them. With a complex program, doing that may be impractical and even create a cause of vulnerability by missing what you should have performed escaping for. We recommend that you uniformly perform escaping for all the elements that make up an SQL statement.

[45] pg_query_params: http://jp.php.net/manual/en/function.pg-query-params.php
[46] Support for PHP4 has been discontinued since December 31, 2007. All PHP4 users are encouraged to upgrade to PHP5.
PHP4 end of life announcement: http://www.php.net/index.php#2007-07-13-1
[47] pg_escape_string: http://jp.php.net/manual/en/function.pg-escape-string.php

■ PHP and MySQL

【Vulnerable Implementation】

```
$query = "SELECT * FROM usr WHERE uid = '$uid' AND pass = '$passh'";
$result = mysql_query($query);
```

PHP

This is part of source code implementing user authentication.

Same as the problematic implementation shown above in 1-1) PHP and PostgreSQL, this sample program also lacks escaping process for the input value for `$uid`, which allows an attacker to launch SQL injection attacks by inserting a specially-crafted value that would turn into a malicious SQL statement.

【Corrective Measure #1】

Use the prepared statements

Instead of the `mysql_query()` function, use the mysqli extension[48], such as `mysqli_prepare()`[49], `mysqli_stmt_bind_param()`[50] and `mysqli_stmt_execute()`[51].

```
// Create a prepared statement
$stmt = mysqli_prepare($conn, "SELECT * FROM usr WHERE uid= ? AND pass = ?");
// Bind $uid and $passh to the SQL statement (corresponding placeholders)
mysqli_stmt_bind_param($stmt, "ss", $uid, $passh);
// Execute the SQL statement
mysqli_stmt_execute($stmt);
```

PHP

The `mysqli_prepare()`, `mysqli_stmt_bind_param()` and `mysqli_stmt_execute()` function are MySQL functions provided in PHP mysqli extension and supported only by MySQL 4.1.3 and later.

The `mysqli_prepare()` function generates a prepared statement. The second argument is the prepared statement where an SQL statement is expressed using the placeholder "`?`" without actual value.

The `mysqli_stmt_bind_param()` function binds the actual data value (bind value) to the placeholders within the prepared statement created by the `mysqli_prepare()` function. The third and later arguments (`$uid` and `$passh` in this case) are the bind values. The second argument "`ss`" indicates the type of the bind values (`s` for string). Since both elements, `$uid` and `$passh`, are the "string" type, two ss are set.

The `mysqli_stmt_execute()` function executes the completed prepared statement. The use of these functions saves you from explicitly performing escaping.

[48] Mysqli extension: http://jp2.php.net/manual/en/ref.mysqli.php
[49] mysqli_prepare: http://jp.php.net/manual/en/function.mysqli-prepare.php
[50] mysqli_stmt_bind_param: http://jp.php.net/manual/en/mysqli-stmt.bind-param.php
[51] mysqli_stmt_execute: http://jp.php.net/manual/en/function.mysqli-stmt-execute.php

【Corrective Measure #2】

Use an escape function

Use the `mysql_real_escape_string()` function[52] to perform escaping for all elements that make up an SQL statement to be executed by the `mysql_query()` function.

```
$query = "SELECT * FROM usr WHERE uid = '".
          mysql_real_escape_string($uid)."' AND pass = '".
          mysql_real_escape_string($passh)."'";
$result = mysql_query($query);
```

PHP

The `mysql_real_escape_string()` function is a MySQL function provided in PHP 4.3.0 and later. It will escape the MySQL special characters.

You can write an escape function yourself but it will be difficult to make sure to cover everything and thus not recommenced.

Likewise, to make sure you escape everything you need to, we recommend that you uniformly perform escaping for all the elements that make up an SQL statement including internally processed ones, such as `$passh`.

■ Perl (with DBI)

【Vulnerable Implementation】

```
$query = "SELECT * FROM usr WHERE uid = '$uid' AND pass = '$passh'";
$sth =$dbh->prepare($query);
$sth->execute();
```

Perl

This is part of source code implementing user authentication. This example uses a standard database interface module called DBI[53] commonly used with Perl.

To access the database, it uses database handles (e.g. the prepare method) or statement handles (e.g. the execute method). This sample program, however, lacks escaping process for the input value for `$uid` and allows an attacker to launch SQL injection attacks by inserting a specially-crafted value that turns into a malicious SQL statement.

【What's Wrong?】

This sample demonstrates a common but dangerous coding error when using Perl DBI.

The `prepare()` method in the DBI module generates a prepared statement and does support placeholders. Likewise, the `execute()` method executes the prepared statement created by the `prepare()` method and it is also capable of binding if the prepared statement contains the placeholders.

What's wrong in this sample program is that it does not use the placeholders nor perform escaping even though a composed SQL statement contains exploitable variables, which makes this application vulnerable to SQL injection attacks.

[52] mysql_real_escape_string: http://jp.php.net/mysql_real_escape_string
[53] DBI: http://dbi.perl.org/about/

【Corrective Measure #1】

Use prepared statements with placeholders

```
$sth =$dbh->prepare("SELECT * FROM usr WHERE uid = ? AND pass = ?");
$sth->execute($uid, $passh);
```

Perl

When composing an SQL statement in the `prepare()` method of the DBI module, use the placeholder "`?`" in the place of variables. Then, specify the bind values to be set to the placeholders in the `execute()` method.

【Corrective Measure #2】

Use an escape function

Use the `quote()` method in the DBI module and perform escaping for the variables.

```
$sth = $dbh->prepare("SELECT * FROM usr
                      WHERE uid =".$dbh->quote($uid)." AND
                      pass =".$dbh->quote($passh));
$sth->execute();
```

Perl

The `quote()` method will take in a string specified in its argument, escape the special characters in the string and return the output after enclosing it with double quotes.

What it recognizes as the special characters differs from database engine to database engine and it is an issue that must be dealt with when performing escaping. DBI provides a set of drivers, called DBD (DataBase Drivers) to adapt to various database engines. The `quote()` method in DBI lets DBD handle the database engine differences and offers the user transparency to this issue.

3.2 OS Command Injection

In this section, we will present a sample mail sending program that is vulnerable to OS command injection.

■ A Perl program that invokes the sendmail command

【Vulnerable Implementation】

```
$from =~ s/"|;|'|<|>|\|| //ig;
open(MAIL, "|/usr/sbin/sendmail -t -i -f $from");
```

Perl

The above is part of a program that sends an email with the email address that the user has input in the web form as the sender.

The input email address is stored in the variable `$from`. The first line removes the special shell characters `"`, `;`, `'`, `<`, `>`, `|` and space from the content of `$from`. The second line invokes the OS's sendmail command to start a mail sending process and passes the content of `$from` to a command line option.

Despite the sanitization in the first line, this implementation is still vulnerable to OS command injection.

【What's Wrong?】

In this implementation, if the value of `$from` is someone@example.jp, the following command is executed as it is meant to be.

```
/usr/sbin/sendmail -t -i -f someone@example.jp
```

sh

However, if the value of `$from` is maliciously crafted and `` `touch[0x09]/tmp/foo` `` (where `[0x09]` means horizontal tabulation) is entered, the following command will be executed and an OS command injection attack could be successfully done.

```
/usr/sbin/sendmail -t -i -f `touch[0x09]/tmp/foo`
```

sh

The back quote (`) is a shell meta-character that executes any sell command put between the back quotes and returns the command output to the command line. In the sample program, the double quote and single quote are sanitized but the back quote is left untouched. This neglect resulted in allowing an attacker to execute arbitrary command.

In addition, removing the space in the first line of the sample program may give a false sense of assurance that an attacker cannot freely specify command line options even if s/he could execute arbitrary command. However, using the horizontal tabulation `[0x09]` like the above enables the attacker to specify arbitrary command line options as well. Here, the horizontal tabulation works as a separating character just like the space.

What character has what sell functionality differs depending on the kind of shells. Do not sanitize the characters with wild guess or the sanitization will likely end up incomplete.

【Corrective Measure #1】

Use libraries

By stopping invoking OS commands, the underlying cause of the OS command injection vulnerability will be removed. See if the functionality presently enabled by invoking OS commands can be done using the existing libraries.

```
use Mail::Sendmail;
%mail = (From => $from, …);
sendmail(%mail);
```

Perl

The task of the sample program is to send an email. With a mail sending library `MAIL::Sendmail`, you can still execute the task while fundamentally removing OS command injection vulnerability.

【Corrective Measure #2】

Not to put the parameter value in the command line

If substitutable libraries are unavailable and you cannot stop using the commands, there is still a chance you can remove the OS command injection vulnerability by changing the way to invoke the command.

```
$from =~ s/\r|\n//ig;
open(MAIL, '|/usr/sbin/sendmail -t -i');
…
print MAIL "From: $from\n";
```

Perl

In the sample program, specifying the sender's email address using the command line option was what it led to the vulnerability. However, it is not mandatory to specify the sender's email address through the command line option and it could be specified in the email header through the command's standard input. This way, the value of `$from` is not used in the command line and therefore remove the OS command injection vulnerability.

Note that if you modify the program like the above, you may make the program vulnerable to email header injection now. Make sure that the value of `$from` does not contain the link break characters. See the corrective measure #2 in 3.8 as well.

【Corrective Measure #3】

Invoke the commands without shell access

If substitutable libraries are unavailable and you cannot stop using the commands, there is still a chance you can remove the OS command injection vulnerability by invoking the command without shell access.

```
open(MAIL, '|-') || exec '/usr/sbin/sendmail', '-t', '-i', '-f', '$from';
```

Perl

In Perl, you can invoke the command directly without shell access. The code above executes the same functionality as the second line of the sample program. Even if the value of `$from` contains the special shell characters, the OS command injection vulnerability is harmless because the shell commands are not to be executed.

3.3 Unchecked Path Parameters

In this section, we present a sample file display program with unchecked path parameter vulnerability.

■ A PHP program that opens and displays the file content to the screen

【Vulnerable Implementation】

```
$file_name = $_GET['file_name'];
if(!file_exists($file_name)) {
  $file_name = 'nofile.png';
}
$fp = fopen($file_name,'rb');
fpassthru($fp);
```

PHP

The above is part of a program that opens and displays the contents of the specified file to the screen. The `$file_name` variable in the first line is replaced by the name of a file specified in the `file_name` parameter in a URL. If the file indeed exists, the `fopen()` function in the fifth line opens it and the `fpassthru()` function in the sixth line outputs its contents to the screen. If the file does not exist, the contents of the `nofile.png` file is outputted. The fundamental premise here is that only the files stored in the server's public directory are to be specified in a URL.

This implementation simply ignores the possibility where a file name set in a URL may be an absolute path or contain `../`, which makes the program vulnerable to directory traversal.

【What's Wrong?】

In this implementation, if the `file_name` parameter in a URL is set with `/etc/passwd`, the contents of `/etc/passwd` will be outputted to the screen.

By predefining an accessible directory like the following, you could prevent an absolute path from being set for a file path parameter. However, setting a relative path that refers to the upper directories in a URL, like `../../../etc/passwd`, still allows the contents of `/etc/passwd` to be disclosed on the screen.

```
$file_name = $_GET['file_name'];
$dir = '/home/www/image/';  //predefine the directory
$file_path = $dir . $file_name;
if(!file_exists($file_path)) {
  $file_path = $dir . 'nofile.png';
}
$fp = fopen($file_path,'rb');
fpassthru($fp);
```

PHP

【Corrective Measure】

Extract only the file name from the path parameter

By extracting just the file name from the path parameter using the functionalities that come with OS or programming languages, the underlying cause of the path parameter directory traversal vulnerability will

be removed.

```
$dir = '/home/www/image/';
…
$file_name = $_GET['file_name'];
…
if(!file_exists($dir . basename($file_name))) {
  $file_name = 'nofile.png';
}
$fp = fopen($dir . basename($file_name),'rb');
fpassthru($fp);
```

PHP

`basename()` is a function that extracts only a file name (excl. directories) from the path parameter value. By using the `basename()` function, only the file name is extracted for use even if an absolute path or relative path using `../` is specified, and the path parameter directory traversal vulnerability will be removed.

3.4 Improper Session Management

In this section, we present a sample session ID generation program with session management vulnerability.

■ Session ID Generation with Perl

【Vulnerable Implementation】

```
sub getNewSessionId {
  my $sessid = getLastSessionId ('/tmp/.sessionid');
  $sessid++;
  updateLastSessionId ('/tmp/.sessionid', $sessid);
  return $sessid;
}
```

Perl

The above is part of a program that generates a session ID. This program calls the `getNewSessionId()` function and generates a session ID. The `getNewSessionId()` function returns a session ID as incrementing the number stored in the `/tmp/.sessionid` file.

With this implementation, the session ID can be easily guessed, making it vulnerable to session hijacking.

【What's Wrong?】

In this implementation, the session ID is a number and issued in sequence from 1, and then 2, 3, 4, and so on. The program stores the latest session ID in the `/tmp/.sessionid` file. When an attacker accesses the website, a new session ID is issued to the attacker as well. For example, if the session ID issued for the attacker is "3022", it is quite likely that the session ID "3021" is also valid at that time. By accessing the website using the session ID "3021", the attacker could hijack the other user's session that are allocated with the session ID "3021".

To prevent such session hijacking attacks, the session ID should be generated using a pseudo random number generators.

【Common Mistake #1】

Generate a session ID based on the information easily guessable by the third party

```
sub getNewSessionId {
  my $sessid = time() . '_' . $$;
  ...
  return $sessid;
}
```

Perl

This program uses a value that is made up by combining the UNIX timestamp[54] and the process ID as the session ID. Here, a new process is created upon access to the website through CGI.

[54] The number of seconds since midnight, January 1, 1970. Also called "the Epoch".

When the `getNewSessionId()` function is called, the program concatenates the UNIX timestamp (the `time()` function), an underscore (`_`), and the process ID (the variable "`$$`"), and returns the resulting string as a session ID. For example, a session ID of "1295247752_27554" will be returned.

This program is vulnerable to session hijacking attacks that exploit the easiness of guessing the session IDs.

Let's suppose that 10 sessions were created in one minute after the time the attacker established his or her session. The attacker can figure out the process ID used in his or her session to connect the web application from the session ID issued for the attacker. In general, a process ID is issued in sequence for new processes. If the process ID issued for the attacker's session was "27554", the attacker can guess that the process ID for other users' session would be "27555", "27556" … "27564". Next, if the attacker's UNIX timestamp was "1295247752", the UNIX timestamp for the next one minute after the attacker's session establishment had to be a value between "1295247753" and "1295247812". The number of possible session ID combinations based on the guess of the process ID and the UNIX timestamp is 600. By going through these 600 possible session IDs, it is possible that the attacker succeeds in the session hijacking attack.

【Common Mistake #2】

Generate a session ID based on the information easily obtainable by the third party

```
use Digest::SHA qw(sha256_hex);
...
sub getNewSessionId {
  my $sessid = '';
  $sessid = $sessid . $ENV{'REMOTE_ADDR'};
  $sessid = $sessid . $ENV{'REMOTE_PORT'};
  $sessid = $sessid . time();
  $sessid = Digest::SHA::sha256_hex($sessid);
  ...
  return $sessid;
}
```

Perl

This program calculates a hash of the value that is made up by combining the user's source IP address, the source port number and the UNIX timestamp, and uses it as a session ID.

When the `getNewSessionId()` function is called, the `getNewSessionId()` function concatenates the user's source IP address (`$ENV{'REMOTE_ADDR'}`), the source port number (`$ENV{'REMOTE_PORT'}`) and the UNIX timestamp (the `time()` function), and generates a string. Then, the `getNewSessionId()` function calculates a SHA-256 hash of the resulting string and returns the hash as a session ID. For example, a session ID of "093a2031a79cb4904b1466ee7ad5faaa3afe7b787db66712f407326b213cc2a4" will be returned.

The use of a hash may give an impression that it is a safe method. But if the mechanism to generate a session ID is disclosed to the third party[55], the third party may manage to guess the session IDs.

By luring the user to a malicious website, the attacker can obtain the user's source IP address[56]. On the other hand, the source port number will be unobtainable information for the attacker. Yet, the scope of the

[55] For example, this program is an open-source software or the source code is leaked to the public.
[56] Depending on the network path to access the website, an IP address may not be specified.

possible source port number is between 1024 to 65535, and depending on the user's network environment, the scope can be narrowed down to around 20,000.

For the network environment where the number of the possible source port number is limited to 20,000, suppose that an user establishes a session with this web application and the attacker obtains the user's source IP address within 10 seconds before and after the user's session establishment, the number of the possible session ID combination is 200,000 (20,000 possible source port numbers times 10 possible UNIX timestamps). By going through these 200,000 possible session IDs, it is possible that the attacker succeeds in the session hijacking attack.

3.5 Cross-Site Scripting

In this chapter, we show sample programs in which the cross-site scripting vulnerability is not properly taken care of.

The cross-site scripting vulnerability is difficult to eradicate because of its nature. In many cases, however, the problem stems from the fact such that the developers did not implement basic countermeasures to begin with nor did so in a mistaken way. We will divide the sample cases into three categories and explain each one.

1. Countermeasures unimplemented
2. Insufficient countermeasures
3. Misguided countermeasures

3.5.1 Countermeasures Unimplemented

■ No escaping is done

【Vulnerable Implementation】

By extracting just the file name from the path parameter using the functionalities that come with OS or programming languages, the underlying cause of the path parameter vulnerability will be removed.


```
use CGI qw/:standard/;
$keyword = param('keyword');
...
print ... <input name="keyword" type="text" value="$keyword">
      ... The search results for ”$keyword”
```

Perl

This is part of source code implementing outputting of the search results.

A string entered into the search form, “IPA” is sent to the web application and set to `$keyword`. This web application embeds the `$keyword` value in multiple places, such as in the form or title when it outputs a search result page. However, it does not perform escaping for the `$keyword` value before outputting it. This will be a cause that allows an attacker to insert arbitrary scripts.

【What's Wrong?】

How a web application should output strings differs depending on whether it outputs a string as text or HTML tags. In this sample case, `$keyword` is a search keyword and supposed to be outputted as text. Therefore, the special characters, such as &, <, >, " and '[57], that may be included in `$keyword` need to be escaped.

Neglecting this process results in a defect of incorrect page display due to the control characters included in `$keyword`. The cross-site scripting attack exploits this defect.

【Corrective Measure #1】

Use an escape function

Use the `escapeHTML()` function in the CGI module.

```
use CGI qw/:standard/;
$keyword = param('keyword');
...
print "<input ... value=\"".escapeHTML($keyword)."\"...";
print "The Search results for ".escapeHTML($keyword)."...";
```

Perl

The `escapeHTML()` function is a Perl function provided in a Perl module CGI. The CGI module is part of the standard Perl 5 distribution.

The `escapeHTML()` function takes in a string specified in its argument, escapes all HTML special characters in the string and returns the result. The table below shows the special characters the `escapeHTML()` function escapes and their corresponding escape sequence[58].

Special Character	Escape Sequence
&	&amp;
<	&lt;
>	&gt;
"	&quot;
'	&#39;

[57] The commonly used quotation mark in the tags is " (double quote) but ' (single quote) is widely used as well , thus we address both of them here.

[58] CGI.pm finely defines what characters are to be escaped depending on character codes. For example, with ISO-8859-1 and WINDOWS-1252, 0x8B (Single Left-Pointing Angle Quotation Mark) and 0x9b (Single Right-Pointing Angle Quotation Mark) will be escaped, too.

【Corrective Measure #2】

Use a self-made escape function

```
print "<input ... value=\"".&myEscapeHTML($keyword)."\"...";
print "The search results for ".&myEscapeHTML($keyword)."...";
...
# Self-made escape function: myEscapeHTML
sub myEscapeHTML($){
        my $str = $_[0];
        $str =~ s/&/&amp;/g;
        $str =~ s/</&lt;/g;
        $str =~ s/>/&gt;/g;
        $str =~ s/"/&quot;/g;
        $str =~ s/'/&#39;/g;
}
```

Perl

■ No character code is set

【Vulnerable Implementation】

Response from web application

```
HTTP/1.1 200 OK
...
Content-Type: text/html
```

(1) No character code is set in HTTP response header

```
<HTML>
<HEAD>
<META http-equiv="Content-Type" content="text/html">
```

(2) No character code is set in HTML META declaration

HTTP Response

This is part of the HTTP response from a web application.

The value of "`Content-Type`" field is used to tell the browser the media type of the entity-body sent to it. The response shown above, however, does not provide that information. In such a case, the browser decides what character code to use on its own based on its preimplemented rule. For example, the browser will guess what to use from the entity-body, and this behavior could be exploited by attackers in cross-site scripting attacks.

【What's Wrong?】

This sample program has no defense against the cross-site scripting attacks that exploit the browser's character-code handling behavior. To solve this problem, you need to make sure to set a character code in the "`Content-Type`" field of HTTP response header.

For more information, see 5-(viii) in 1.5.3 "Measures common to all web applications".

【Corrective Measure】

Set a character code in the "Content-Type" field of HTTP response header

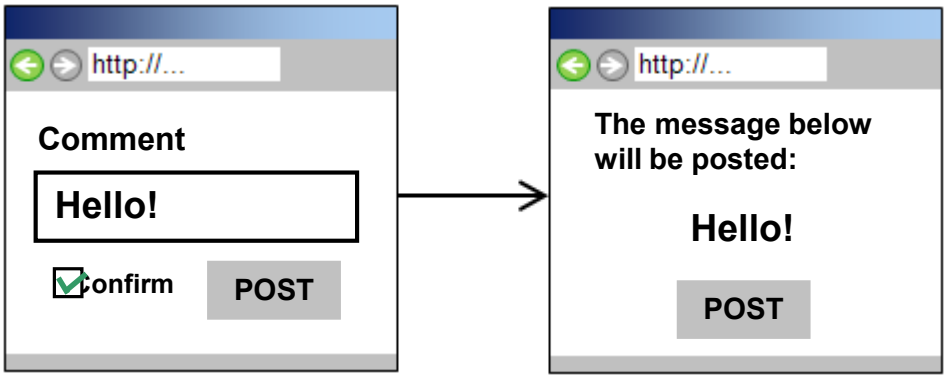
```
HTTP/1.1 200 OK
...
Content-Type: text/html; charset=UTF-8
```

HTTP Response

3.5.2 Insufficient Countermeasures

■ Perform escaping for the text input at the timing of data entry

【Vulnerable Implementation】



This sample program has no defense against the cross-site scripting attacks that exploit the browser's character-code handling behavior. To solve this problem, you need to make sure to set a character code in the "Content-Type" field of HTTP response header.

Message Posting Form

```
<textarea name="comment" ...
<input name="agree" type="checkbox" value="yes">...
<input name="uid" type="hidden" value="12345678">...
```

HTML

Confirmation Screen

```
$comment = escapeHTML(param('comment'));
$agree   = param('agree');
$uid     = param('uid');
...
print "The message below will be posted:<BR>".$comment."...
print "<input ... hidden ... =\"".$uid ...
```

Perl

This is part of source code outputting a confirmation page using the HTML source code of a message posting form and the data entered in the form.

The message posting form has three elements: a comment field in which a user can enter comments, a check box and the user ID which is not to be displayed on the web page.

These three values entered in the message posting form are passed to the web application and only two

of them, the comments (`$comment`) and the user ID (`$uid`), are to be outputted onto the confirmation page. While the web application performs escaping for the comments when they are entered, it does not do so on the user ID. This is an example where the developers implement some but insufficient measures due to not properly knowing what elements should go through escaping process.

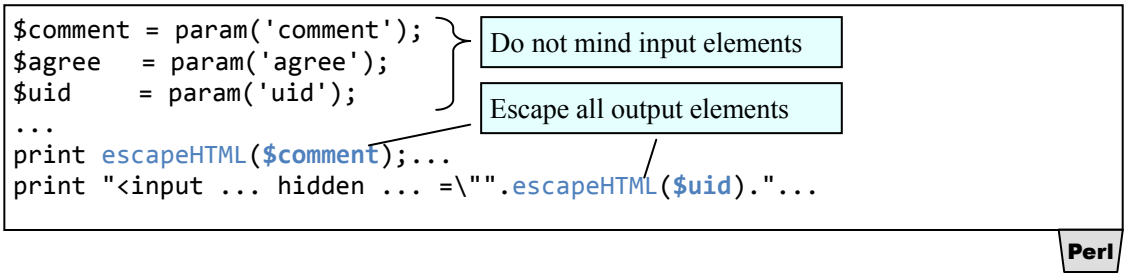
【What's Wrong?】

One of the common misunderstandings over escaping is that the text enterable fields are the only elements that need to go through escaping process.

Attacks are not limited to text enterable fields, such as a comment field in a message posting form. Focusing on particular elements, such as text enterable fields, leads to missing other elements. Likewise, performing escaping for the values when they are entered right away, albeit it could be a good display of countering attacks at the earliest possible opportunity, could lead to missing what really needs to be done. The elements that should go through escaping process to prevent cross-site scripting attacks are the "output elements". If you perform escaping for the "input elements", you will miss the cases where an arithmetic operation right before outputting the web page is used to generate malicious HTML tags and scripts. It also makes it costly to determine from the source code whether escaping is done for indeed all the necessary elements.

【Corrective Measure】

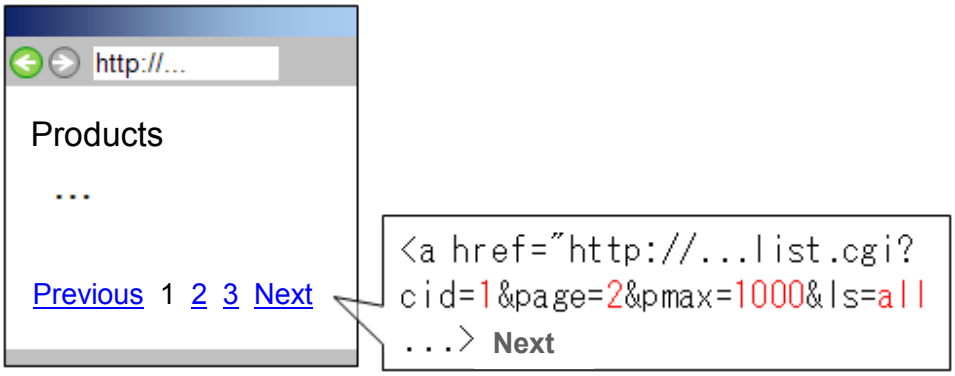
Perform escaping focusing on "output elements"



Do not mind the input elements. Rather, focus on the output elements and perform escaping for them.

【Common Mistake #1】

Did not perform escaping for the elements that would make up links (URLs)



In the picture above, "`cid`", "`page`", "`pmax`" and "`ls`" are used as the parameters to make up a URL of the links, such as "Previous" and "Next". These elements outputted in the tags should go through escaping process too, but this operation is often forgotten.

【Common Mistake #2】

Did not perform escaping for the URL to display on the "404 Not Found" error page

In the picture above, the web application outputs the URL originally requested by the user on the error page for the HTTP status code 404. This URL information should go through escaping process too, but many web applications miss it.

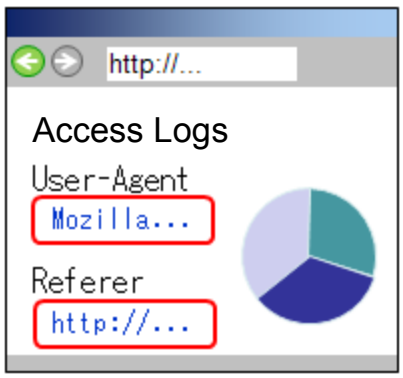
For example, cross-site scripting attacks will succeed by luring the user to a booby-trapped URL like shown below.

```
http://example.com/<script>...</script>
```

URL

【Common Mistake #3】

Did not perform escaping for access log information to be outputted



This could be a web application that outputs the statistics information about a web page based on web server access logs. For example, it may show the pages the users have requested, the User-Agent and Referer information. In many cases, a web application likely does not perform escaping when it uses the server's internal data.

For example, a malicious attacker could send a request where scripts are set in the elements, such as the User-Agent and Referer, and have them logged in to the access log.

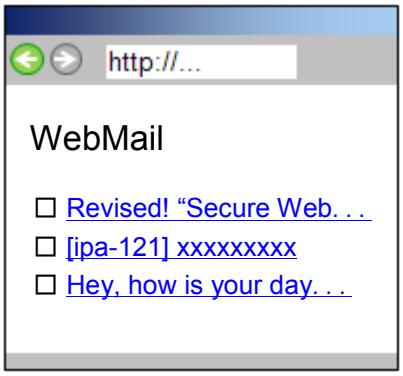
```
GET /example.html / HTTP/1.1
Host: example.com
User-Agent: Mozilla/5.0...<script>...
Referer: http://example.net/<script>...
```

HTTP Request

Provided that, if escaping is not done, the users who view the access log page will keep accessing the script-embedded web page almost permanently.

【Common Mistake #4】

Did not perform escaping for web mail information to be outputted



This could be a web application that outputs email information on the web page. For example, it may show the sender, subject and contents of email. In many cases, web applications neglect escaping when they use the server’s internal data.

For example, a malicious attacker could send an email crafted like below.

```
To: jiro@example.com
From: taro@example.com
Subject: Revised!”Secure Web Site...<script>...
Body: IPA hopes you have been...<script>...
```

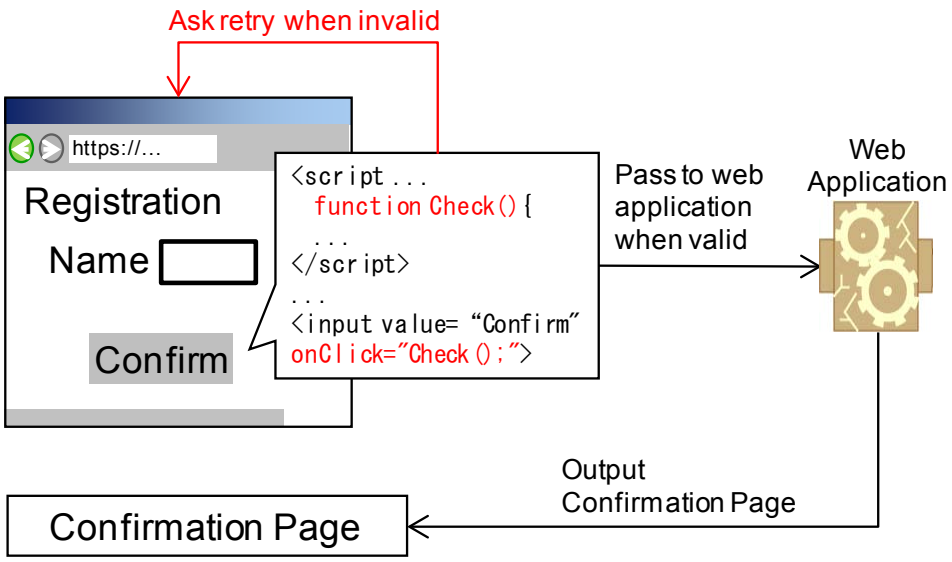
Email

Provided that, if escaping is not done, the users who view the webmail page will keep accessing the script-embedded web page almost permanently.

3.5.3 Misguided Countermeasures

■ Implement checking function for user input

【Vulnerable Implementation】



This sample program embeds a user input check mechanism into the input form page using JavaScript. Through this checking function, only permitted values are to be passed over to the outputting process of the web application. This tends to make us think no unintended characters would be outputted on the confirmation page, but in fact this input checking does not work well against cross-site scripting vulnerability.

【What's Wrong?】

This sample is implementing the right measure in the wrong place. Client-side input validation is effective for reducing user input error but not for countering cross-site scripting vulnerability. In most cross-site scripting attacks, malicious requests are sent directly to the vulnerable web application through the trap (hyperlinks in email or on web page) set up by malicious attackers, and thus bypass client-side input validation like shown above.

In addition, input validation could not be a fundamental solution for it is almost impossible to cover everything you should not accept to counter cross-site scripting vulnerability. Read Chapter 1-5 "Cross-Site Scripting" and take appropriate action.

■ Take blacklist approach only

【Vulnerable Implementation】

```perl
if ($a =~ /(script|expression|...)/i){  # check input
    error_html();    # if detecting dangerous value, return error message
    exit;
} else {
    ...
    print $a;       # if no dangerous value is detected, then proceed
```

Perl

This sample program takes blacklist approach for implementing input validation. The blacklist here defines potentially dangerous characters often used in cross-site scripting as unacceptable values. For example, when an input value `$a` includes suspicious strings, such as "`script`", the web application cancels processing and returns an error.

At the first look, it may look like this works well to nullify cross-site scripting attacks, but this implementation has a vulnerability an attacker could exploit to bypass the input validation function.

【What's Wrong?】

Bypass input checking using control characters

Input validation does not offer a fundamental solution to cross-site scripting vulnerability.

For example, if the following string is entered to `$a`, input validation against a string "`script`" will be bypassed.

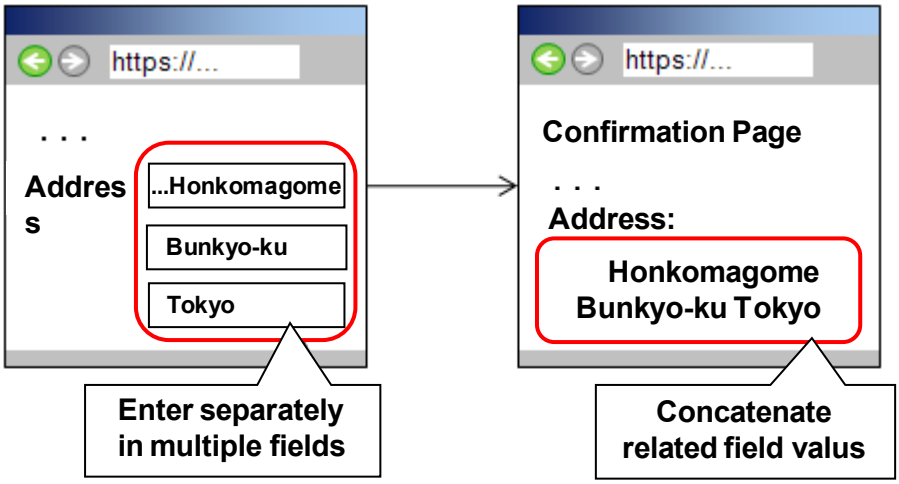
```
<s%00cript>alert(0)</s%00cript>
```

TXT

After bypassing input validation, "`%00`" in `$a` will be decoded and outputted to the web page as a NULL character. Web browsers often ignore the NULL character, and as a result, the value of `$a` will be interpreted as a script. That says a simple pattern-matching input validation function cannot prevent scripts from being inserted. There are more control characters attackers could use to bypass input checking besides the NULL character.

【Common Mistake #1】

Bypass input checking using input concatenation



This problem occurs with a web application that takes blacklist approach for implementing input validation. The input form offers multiple fields for the address information where a user enters it in some units, such as prefecture and municipality. The input validation function checks whether an input value contains the unacceptable strings defined by the blacklist, such as "`script`", and if positive, have the application stop processing further. After input checking, those multiple field values are concatenated to form address information like the following.

```perl
if ($addr1 =~ /script/i){       # check input (same for $add2 and $add3)
    error_html();               # if detecting dangerous value,
    exit;                       # return error message
} else {
    ...
    print $addr1.$addr2.$addr3;  # concatenate checked values
```

Perl

Let's see what will happen when the input data shown below are entered.

Variable	Value
$addr1	<scr
$addr2	ipt>alert(1)</s
$addr3	cript>

None of these parameters on its own contains a string "script", thus each parameter value will be accepted. After passing input checking, these three values are concatenated and form the following string.

```txt
<script>alert(1)</script>
```

TXT

This issue still remains even if you have solved the previous problem of bypassing input validation using the control characters.

Input checking cannot in nature deal with the cases where scripts are to be formed as a result of arithmetic operations after checking is performed. We recommend that you complement input checking with some other fundamental solutions.

【Common Mistake #2】

Bypass input checking abusing match-then-delete reaction

```perl
$a =~ s/(script|expression|...)//gi;
...
print $a;
```

Perl

This sample program implements input validation in a way that a web application deletes strings that match the unacceptable values defined in the blacklist. For example, if $a contains a string "script", the input checking function removes the string and outputs the rest.

Suppose that the following string is entered to $a.

```txt
<script>alert(1)</script>
```

TXT

The application will perform deletion based on the blacklist and output the result shown in the following box. The script tags are nullified and the attack fails.

```txt
<>alert(1)</>
```

TXT

Let's see what will happen with this one.

```
<sscriptcript>alert(1)</sscriptcript>
```
TXT

The application performs deletion and outputs the following result, which turns out to be a script.

```
<script>alert(1)</script>
```
HTML

As you see, simply deleting dangerous strings is not recommended for that very operation may be exploited to help the formation of scripts. When removing the dangerous strings, we recommend replacing them with harmless strings instead of deleting them.

For more information, see 1.5.2 Mitigation Measure 5-(vii).

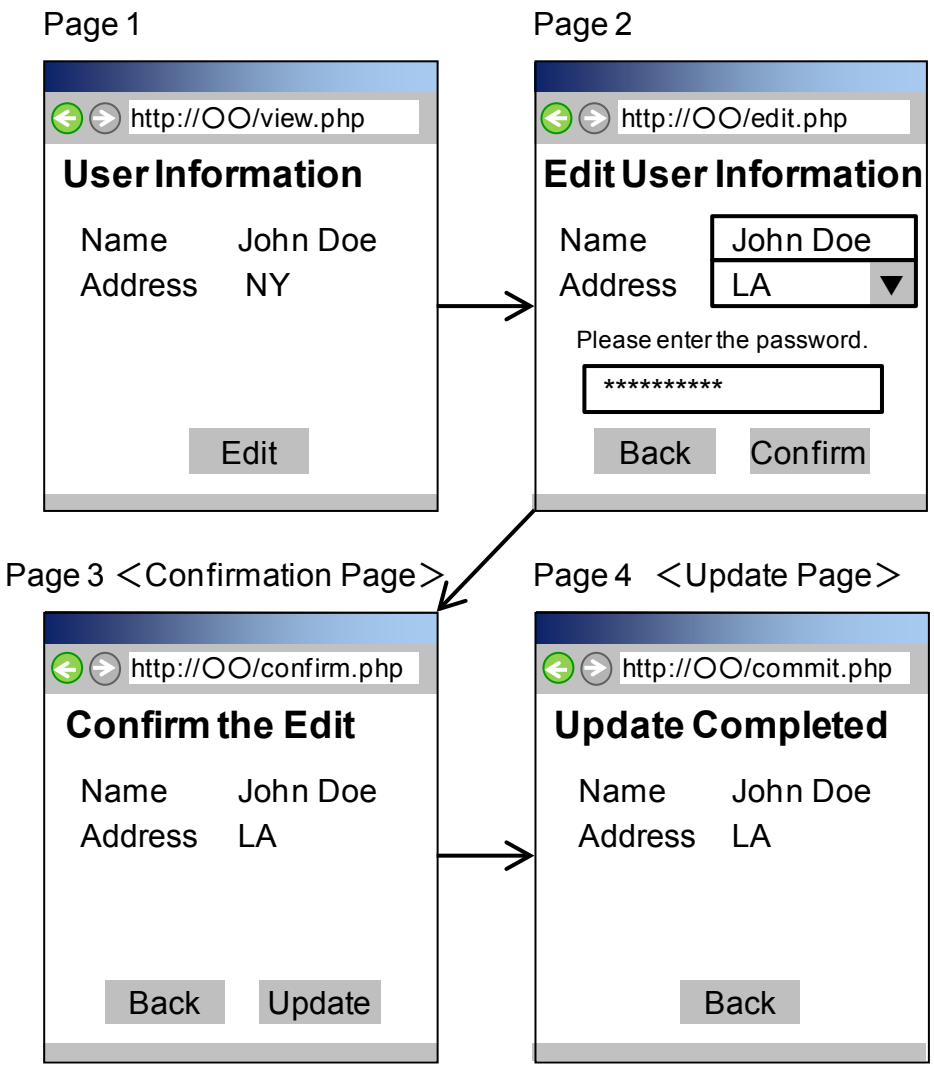
3.6 CSRF (Cross-Site Request Forgery)

In this section, we present a sample user registration program that is vulnerable to CSRF (Cross-Site Request Forgery).

■ A PHP program that registers user information

【Vulnerable Implementation】

The following figures shows an example of typical web page flow when updating the user information registered for a members-only website. Here, the user is changing his address from New York to Los Angeles.



In this website's structure, the user first confirms his user information currently registered on the Page 1 (`view.php`) and clicks the "Edit" button to proceed to Page 2 (`edit.php`) if he wants to change his profile. There, he edits the information, enters the password and moves to Page 3 (`confirm.php`). On Page 3, the user is required to confirm the changes he has made and click the "Update" button, which executes the changes and puts the process forward to Page 4 (`commit.php`) where the result of editing is presented.

On Page 2, the password is required to authenticate the user and only if the password is correct, then

the user can proceed to Page 3.

Let's assume that the HTML source code for Page 3 is written like the following.

```
<form action="commit.php" method="post">
  <input type="hidden" name="new_name" value="John Doe">
  <input type="hidden" name="new_address" value="LA">
  <input type="submit" name="back" value="Back">
  <input type="submit" name="commit" value="Update">
</form>
```

HTML

When the user clicks the "Update" button on Page 3, the following code in `commit.php`, the source code for Page 4, executes the update. The `$_SESSION['authenticated']` variable stores the information whether the user has logged in as a true or false value.

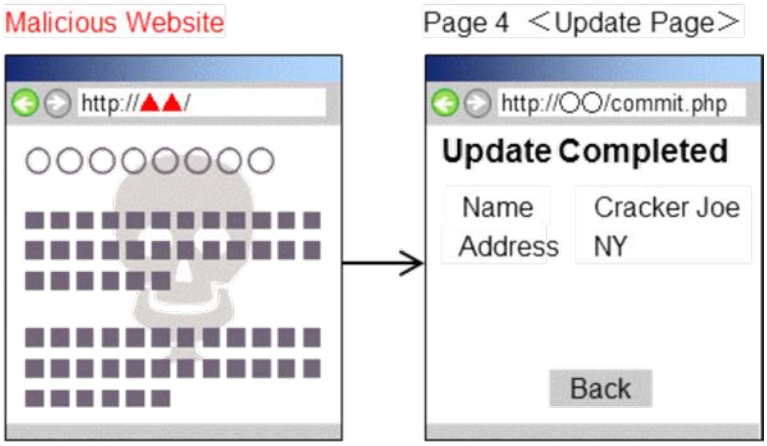
```
session_start();
if( ! $_SESSION['authenticated'] ) { exit(); }
update_userinfo($_SESSION['uid'],$_POST['new_name'], $_POST['new_address']);
```

PHP

The second line checks if the user has logged in, but it does not check whether the profile change request has indeed been made by the logged-in user, which makes the whole process vulnerable to cross-site request forgery.

【What's Wrong?】

If the user is lured to a malicious website while logged in to this members-only website, an attacker could forge and redirect a user request to Page 4 and execute the profile change without and against the user's will.



The following is an example of an HTML source code embedded in malicious websites. It is similar to the source code for Page 3, but the lines written in red are different. In this case, a CSRF attack is executed just by accessing the malicious website.

```HTML
<form action="http://○○/commit.php" method="post" name="f1">
  <input type="hidden" name="new_name" value="Cracker Joe">
  <input type="hidden" name="new_address" value="NY">
</form>
<script>document.forms['f1'].submit();</script>
```

The commit.php program in the sample program cannot discern the difference between a rightful request made by the user and a request forged by an attacker and executes the change request.

When implementing the feature where logged-in users change the settings or post stuff, be aware of cross-site request forgery and implement necessary measures.

【Corrective Measure #1】

Embed a secret information in the confirmation page and check it in the update page

By embedding a secret in the confirmation page and later confirming it in the update page, the underlying cause of the CSRF vulnerability will be removed.

In the sample code, Page 3 corresponds to the confirmation page and Page 4 corresponds to the update page. With the following sample modified program, the PHP session ID is used as a secret.

First, embed the session ID to Page 3 (shown in red).

```HTML
<form action="commit.php" method="post">
  <input type="hidden" name="new_name" value="John Doe">
  <input type="hidden" name="new_address" value="LA">
  <input type="hidden" name="sid" value="6a0752gpmhignmnq9f5iah8h71">
  <input type="submit" name="back" value="Back">
  <input type="submit" name="commit" value="Update">
</form>
```

Next, the `commit.php` program for Page 4 checks the value of the secret. Make sure to check it before executing the changes. If the secret is not correct, stop the process. The modified `commit.php` program would be like the following (added the line written in red).

```PHP
session_start();
if( ! $_SESSION['authenticated'] ) { exit(); }
if( $_POST['sid'] != session_id() ) { exit(); }
update_userinfo($_SESSION['uid'],$_POST['new_name'], $_POST['new_address']);
```

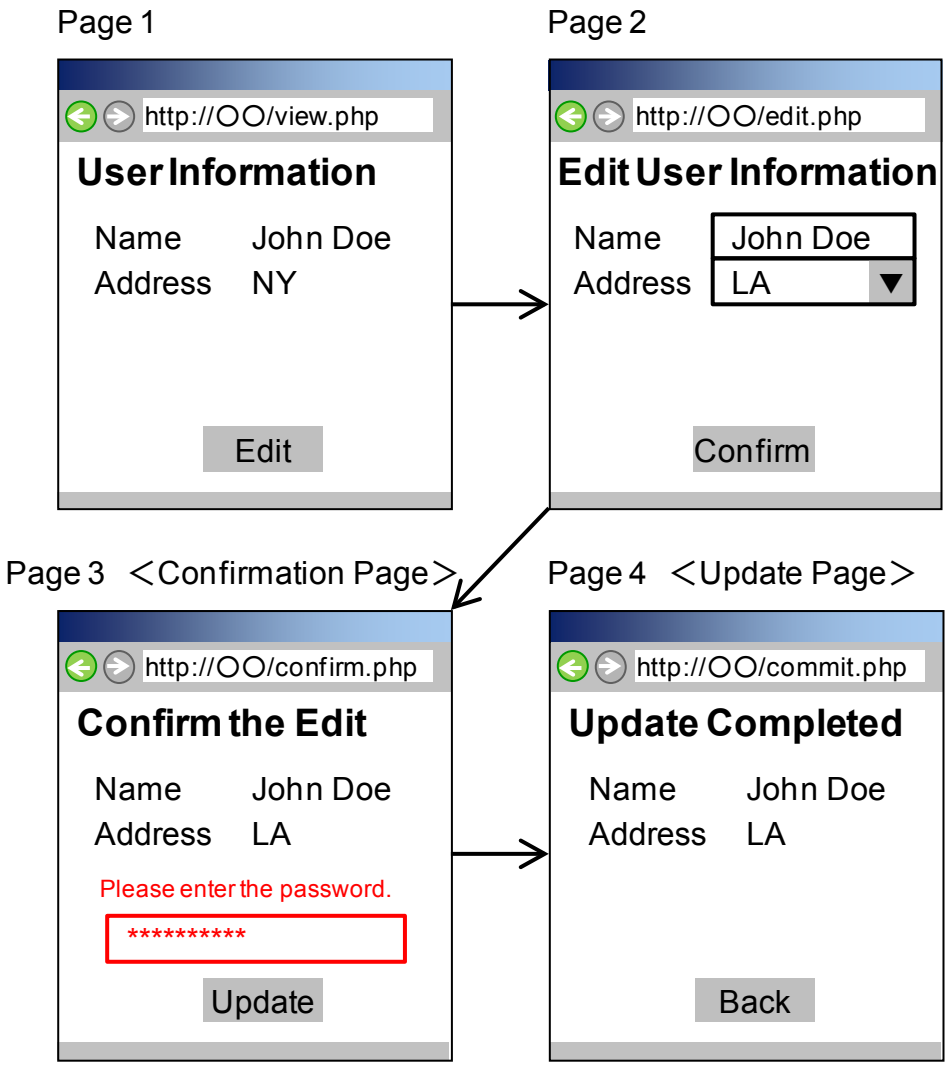
This measure requires that you can generate a secret in the way that the third parties cannot figure out and store it safely, and that you can use the POST method to pass the secret to the update page. If you cannot satisfy all three of them, then other measures should be considered.

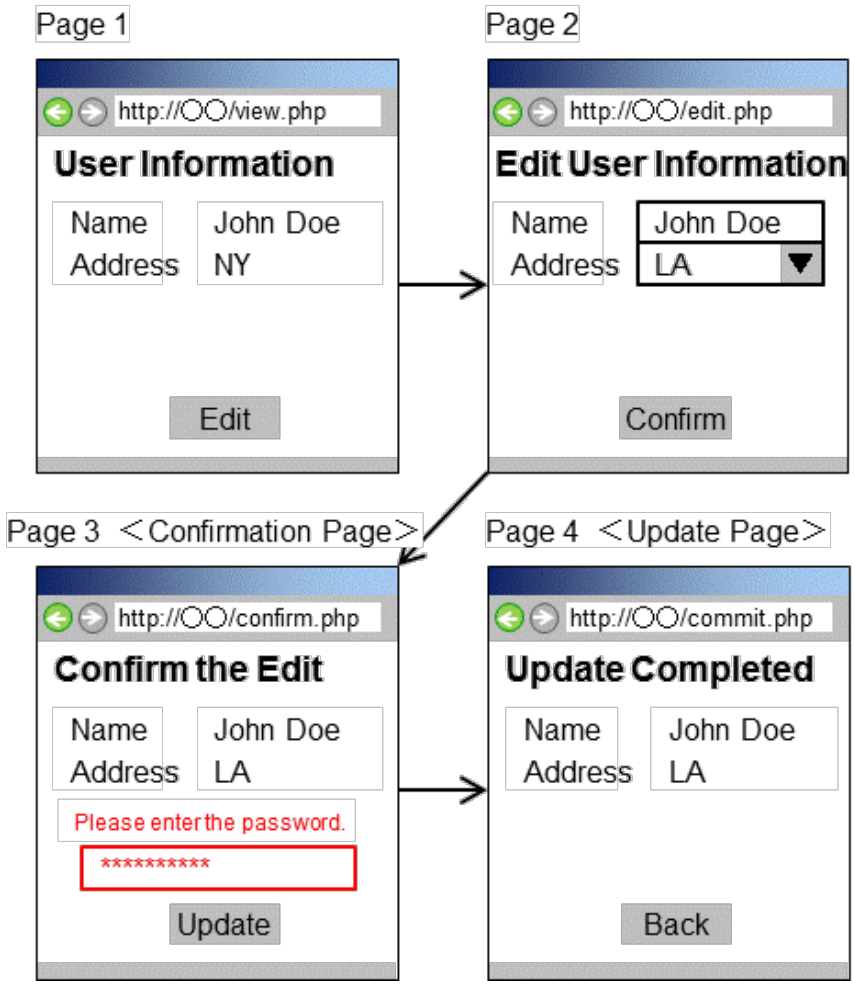
【Corrective Measure #2】

Check the password in the update page

By requiring the password on the confirmation page and confirming the password in the update page right after that, the underlying cause of the CSRF vulnerability will be removed.

In the sample program, the password is required in Page 2 and checked in Page 3. By changing it to require the password in Page 3 and check it in Page 4, the CSRF vulnerability will be removed.





This measure requires the modification of the user interface. If it is not feasible, then another measure should be considered.

【Corrective Measure #3】

Check the Referer in the update page

By checking the Referer in the update page, the underlying cause of the CSRF will be removed. Here, the `commit.php` program in the sample code was modified like the following (added the line written in red).

```php
session_start();
if( ! _SESSION['authenticated'] ) { exit(); }
if( $_SERVER['HTTP_REFERER'] != 'http://oo/confirm.php' ) { exit(); }
update_userinfo($_SESSION['uid'],$_POST['new_name'], $_POST['new_address']);
```

As a side effect, note that this measure cannot execute the change requests if the user's browser is set not to send the Referer field or the user is accessing the website through the proxy server that removes the Referer from the HTTP requests made by the user's browser.

3.7 HTTP Header Injection

In this section, we present a sample redirector program that is vulnerable to HTTP header injection.

■ A Perl program that executes URL redirection

【Vulnerable Implementation】

```
$cgi = new CGI;
$num = $cgi->param('num');
print "Location: http://example.jp/index.cgi?num=$num\n\n";
```

Perl

The above is part of a program that redirects a website visitor to a predefined URL using the Location header. The sample program above first inputs the value of the num parameter into the $num variable (the second line). The program then creates the Location header based on the value of $num and outputs an HTTP response. The program assumes that only numerical numbers are entered as the value of the num parameter.

This implementation simply ignores the possibility where a value that includes the line break characters may be specified for the num parameter and allows an attacker to create unexpected HTTP responses.

【What's Wrong?】

In this implementation, if a visitor accesses the URL where its num parameter is set with 3%0D%0ASet-Cookie:SID=evil, malicious arbitrary cookie will be issued. Moreover, the visitor may be redirected to a fraudulent web page depending on the way the parameter value is crafted.

```
HTTP/1.x 302 Found
Date: Sat, 07 Mar 2009 01:49:48 GMT
Server: Apache/2.2.3 (Unix)
Set-Cookie: SID=evil
Location: http://example.jp/index.cgi?num=3
Content-Length: 292
Connection: close
Content-Type: text/html; charset=iso-8859-1
```

HTTP Response

【Corrective Measure】

Forbid the line break characters as the parameter values used to create the header

By properly restricting the use of the line break characters, the underlying cause of the HTTP header injection vulnerability will be removed.

```
### The function that returns the first line of a multi-line string
# Parameter: String. Ignore anything other than the first parameter
# Return value: A string before the line breaker (\r, \n, \r\n)
sub first_line {
  $str = shift;
  return ($str =~ /^([^\r\n]*)/)[0];
}
```

Perl

The above is a function that returns the first line (without the line break character at the end of the line) of the string passed by the parameter. By having the parameter value entered externally go through this function, the program can ensure the output value is appropriate as the value for the HTTP response header field and therefore remove the vulnerability.

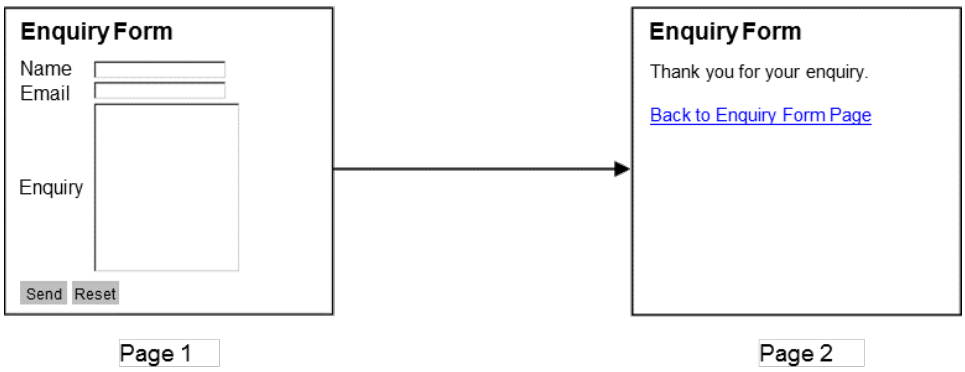
The HTTP header field does allow a multi-line string as its value but the function above is written in the way that does not support it. Note that the use of this function is inappropriate for the web application with which a multi-line string is expected for the HTTP header filed value since the second line and the lines after that will be discarded by it.

3.8 Mail Header Injection

In this section, we present a sample enquiry form program that is vulnerable to mail header injection.

■ An email sending function with Perl

【Vulnerable Implementation】




```perl
open (MAIL, "| /usr/sbin/sendmail -t -i");
print MAIL << "EOF";
To: info\@example.com
From: $email
Subject: Enquiry ($name)
Content-Type: text/plain; charset="ISO-2022-JP"

$inquiry
EOF
close (MAIL);
```

Perl

The above is part of a program that sends the user's input into the enquiry form as an email to the website administrator[59]. When the user enters the value to the name, email address and enquiry field on the Page 1 and clicks the send button, the program calls the sendmail command provided by OS and sends an email to the website administrator's email address `info@example.com`. When the program sends the email, the user input for each header is stored into the `$name`, `$email` and `$inquiry` variable

[59] When using a character set other than US-ASCII in the mail header, it must be encoded following RFC2047.

and then creates the mail headers and body using those variables. After completing sending the email, the program outputs the Page 2.

This implementation is vulnerable to mail header injection since it outputs the user input value directly into the mail header.

【What's Wong?】

In this implementation, the program passes the mail headers and body to the sendmail's standard input to send email. The sendmail command determines the destination email addresses based on the input for the `To`, `Cc` and `Bcc` header. If the user input on the Page 1 is "`anzen`" (the name field), "`anzen@example.net`" (the source IP address field) and "`Hello, World`" (the body field), the program sends the email shown below to `info@example.com`.

```
To: info@example.com
From: anzen@example.net
Subject: Enquiry(anzen)
Content-Type: text/plain; charset="ISO-2022-JP"

Hello, World
```

Mail

However, if a user feeds the input value including a line feed character and mail headers to the name field or the email address field, the user can sends the email to arbitrary destination addresses. For example, if the user input for the mail address field is "`anzen@example.net%0d%0aBcc%3a%20user@example.org`", the passing data from the program to the sendmail command will be the following. The sendmail command will send the email to `user@example.org` in addition to `info@example.com` based on the user input passed from the program.

```
To: info@example.com
From: anzen@example.net
Bcc: user@example.org
Subject: Enquiry(anzen)
Content-Type: text/plain; charset="ISO-2022-JP"

Hello, World
```

TXT

【Corrective Measure #1】

Do not output the user input value into the mail header

By not outputting the user input into the mail headers, the underlying cause of the mail header injection vulnerability will be removed.

```
open (MAIL, "| /usr/sbin/sendmail -t -i");
print MAIL << "EOF";
To: info\@example.com
From: webform\@exmaple.com
Subject: Enquiry
Content-Type: text/plain; charset="ISO-2022-JP"

=========================================
Name: $name
Email Address: $email
=========================================
$inquiry
EOF
```

Perl

With this corrective measure, the input value stored in the `$name` and `$email` variable is not outputted into the mail headers but into the mail body. The value for the `From` and `Subject` header is fixed to `webform@exmaple.com` and `Enquiry`, respectively. If line feed character is included in the value for the `$name` or `$email` valuable, the layout of the body will be disrupted a little but this measure can prevent arbitrary mail headers from being inserted[60].

【Corrective Measure #2】

Remove line feed character from the value for the mail header variables

Removing the line feed characters from the values for the mail header variables will mitigate the risk of mail header injection vulnerability.

```
$name  =~ s/\r|\n//g;
$email =~ s/\r|\n//g;
```

Perl

In this corrective measure, the line feed characters (`\r` and `\n`) are removed from the value of the `$name` and `$email` variable using regular expressions.

[60] If the program is written to reply to the user automatically upon receiving the enquiry, the program can still be exploited to send out spam mails even if this corrective measure is implemented.

Postface

Security practices to secure web applications and websites we have presented in this book will help you mitigate the threats the website operators are facing. Once security implementation and internal checking are done, it is beneficial to have a third-party entity perform penetration test or code review to assure secure implementation. It is recommended a website undergo an external vulnerability testing depending on the importance the website has on you or your organization's success.

We hope this book will help you secure your website.

References

Ministry of Internal Affairs and Communications, Japan: Communications Usage Trend Survey
http://www.soumu.go.jp/johotsusintokei/statistics/statistics05.html (Japanese Only)

IPA: 脆弱性関連情報の届出
http://www.ipa.go.jp/security/vuln/report/index.html (Japanese Only)

IPA: 知っていますか？脆弱性（ぜいじゃくせい）
http://www.ipa.go.jp/security/vuln/vuln_contents/ (Japanese Only)

IPA: セキュア・プログラミング講座（新版）
http://www.ipa.go.jp/security/awareness/vendor/programmingv2/web.html (Japanese Only)

IPA: 脆弱性関連情報に関する届出状況
http://www.ipa.go.jp/security/vuln/report/press.html (Japanese Only)

IPA: セキュア・プログラミング講座 「より良い Web アプリケーション設計のヒント」
http://www.ipa.go.jp/security/awareness/vendor/programmingv2/contents/003.html (Japanese Only)

IPA: セキュア・プログラミング講座 「SQL 注入：#1 実装における対策」
http://www.ipa.go.jp/security/awareness/vendor/programmingv2/contents/502.html (Japanese Only)

IPA: セキュア・プログラミング講座 「SQL 注入：#2 設定における対策」
http://www.ipa.go.jp/security/awareness/vendor/programmingv2/contents/503.html (Japanese Only)

IPA: Information Security White 2009 Paper Part 2
http://www.ipa.go.jp/security/vuln/documents/10threats2009_en.pdf

IPA: 情報セキュリティ白書 2008 第 2 部
http://www.ipa.go.jp/security/vuln/20080527_10threats.html (Japanese Only)

IPA: セキュア・プログラミング講座 「コマンド注入攻撃対策」
http://www.ipa.go.jp/security/awareness/vendor/programmingv2/contents/501.html (Japanese Only)

IPA: セキュア・プログラミング講座 「プログラムからのファイル流出対策」
http://www.ipa.go.jp/security/awareness/vendor/programmingv2/contents/402.html (Japanese Only)

Westpoint: Multiple Browser Cookie Injection Vulnerabilities
http://www.westpoint.ltd.uk/advisories/wp-04-0001.txt

ACROS Security: Session Fixation Vulnerability in Web-based Applications
http://www.acrossecurity.com/papers/session_fixation.pdf

IPA: 経路のセキュリティと同時にセキュアなセッション管理を
http://www.ipa.go.jp/security/ciadr/20030808cookie-secure.html (Japanese Only)

IPA: セキュア・プログラミング講座 「セッション乗っ取り」
http://www.ipa.go.jp/security/awareness/vendor/programmingv2/contents/302.html (Japanese only)
http://www.ipa.go.jp/security/awareness/vendor/programmingv2/contents/303.html (Japanese Only)
http://www.ipa.go.jp/security/awareness/vendor/programmingv2/contents/304.html (Japanese Only)
http://www.ipa.go.jp/security/awareness/vendor/programmingv2/contents/305.html (Japanese Only)
http://www.ipa.go.jp/security/awareness/vendor/programmingv2/contents/306.html (Japanese Only)

IPA: セッション管理
http://www.ipa.go.jp/security/awareness/administrator/secure-web/chap6/6_session-1.html (Japanese Only)

IPA: セッション管理の留意点
http://www.ipa.go.jp/security/awareness/administrator/secure-web/chap6/6_session-2.html (Japanese Only)

産業総合技術研究所 高木浩光:「CSRF」と「Session Fixation」の諸問題について
http://www.ipa.go.jp/security/vuln/event/documents/20060228_3.pdf (Japanese Only)

W3C: HTML 4.01 Specification
http://www.w3.org/TR/html401/

Microsoft: Mitigating Cross-site Scripting With HTTP-only Cookies
http://msdn2.microsoft.com/en-us/library/ms533046.aspx

Bugzilla@Mozilla： MSIE-extension: HttpOnly cookie attribute for cross-site scripting vulnerability prevention
https://bugzilla.mozilla.org/show_bug.cgi?id=178993

WhiteHat Security：Cross-Site Tracing
http://www.cgisecurity.com/whitehat-mirror/WH-WhitePaper_XST_ebook.pdf

IPA：セキュア・プログラミング講座 「エコーバック対策」
http://www.ipa.go.jp/security/awareness/vendor/programmingv2/contents/601.html (Japanese Only)
http://www.ipa.go.jp/security/awareness/vendor/programmingv2/contents/602.html (Japanese Only)

RFC2616： Hypertext Transfer Protocol -- HTTP/1.1
http://www.ietf.org/rfc/rfc2616.txt

IPA：セキュア・プログラミング講座 「リクエスト強要（CSRF）対策」
http://www.ipa.go.jp/security/awareness/vendor/programmingv2/contents/301.html (Japanese Only)

Amit Klein：HTTP Response Smuggling
http://www.securityfocus.com/archive/1/425593/30/0/threaded

IPA：セキュア・プログラミング講座 「HTTP レスポンスによるキャッシュ偽造攻撃対策」
http://www.ipa.go.jp/security/awareness/vendor/programmingv2/contents/603.html (Japanese Only)

IPA：セキュア・プログラミング講座 「メールの第三者中継対策」
http://www.ipa.go.jp/security/awareness/vendor/programmingv2/contents/201.html (Japanese Only)

The Unauthorized Computer Access Law
http://www.npa.go.jp/cyber/english/legislation/ucalaw.html

JVN（Japan Vulnerability Notes）
http://jvn.jp/en/index.html

JVN iPedia Vulnerability Countermeasure Information Database
http://jvndb.jvn.jp/index_en.html

IPA：パスワードの管理と注意
http://www.ipa.go.jp/security/fy14/contents/soho/html/chap1/pass.html (Japanese Only)

IPA：セキュア・プログラミング講座 「セキュアな Web サーバの構築と運用に関するコンテンツ」
http://www.ipa.go.jp/security/awareness/administrator/secure-web/ (Japanese Only)

IPA：ドメイン名の登録と DNS サーバの設定に関する注意喚起
http://www.ipa.go.jp/security/vuln/20050627_dns.html (Japanese Only)

IPA：電子メールのセキュリティ
http://www.ipa.go.jp/security/fy18/reports/contents/email/email.pdf (Japanese Only)

IPA：セキュア・プログラミング講座 「ユーザ認証対策 パスワードフィルタ」
http://www.ipa.go.jp/security/awareness/vendor/programmingv2/contents/101.html (Japanese Only)

IPA：PKI 関連技術解説 「認証局と電子証明書」
http://www.ipa.go.jp/security/pki/031.html (Japanese Only)

IPA：セキュア・プログラミング講座 「真正性の主張」
http://www.ipa.go.jp/security/awareness/vendor/programmingv2/contents/202.html (Japanese Only)

産業技術研究所：安全な Web サイト利用の鉄則
http://www.rcis.aist.go.jp/special/websafety2007/ (Japanese Only)

Terminology

Web Application
A software system that runs on a website. Usually written in Java, ASP, PHP and Perl, and allows to produce and offer dynamic web pages to users.

Escaping
A process to neutralize the characters that have a special meaning, which signifies an alternative interpretation and triggers a special processing depending on languages and environments, generally by replacing them with another characters and makes them non-special.

Encoding
A process of putting a sequence of characters (letters, numbers, symbols etc) into a specialized format based on a predefined rule. For example, the characters that cannot appear in a URL, such as Japanese letters, must be “encoded” into % and hexadecimal numbers based on RFC2396.

Line Break Codes
Control codes that signify line break in text. In general, CR (Carriage Return), LF (Line Feed), or a combination of these two is used. In the ASCII Code Table, CR and LF are defined “0x0D” and “0x0A”, respectively.

Shell
The program that interprets user input then runs and controls other programs. cmd.exe is an example for Windows shell and bash and csh for UNIX/LINUX shell.

Vulnerability
A security weakness that computer software, such as web applications, could have. It could be a cause of a web application losing its normal functions or performance due to unauthorized access or computer virus exploiting the security weakness. It could also mean a situation in which web application security is no longer maintained, as seen in the cases where personal information is not protected by appropriate access control because of improper website operation.

Session Management
A mechanism with which a website tracks a user’s activities across web pages (requests) to identify the user and keep the state of his or her operations.

Directory Traversal
An attacking method that exploits relative paths to move around and access arbitrary files in a target system. The name originates from its ability to freely traverse directories in the system. Also known as path traversal.

Decoding
A process of converting an encoded format back into the original sequence of characters.

Blacklist
A filtering mechanism opposite to whitelist. Deny those strings that are predefined on the list and allow everything else. It cannot cope with unknown attacks, which cannot be defined on the list beforehand.

Whitelist
A filtering mechanism opposite to blacklist. Allow those strings on the list and deny everything else. It can well cope with unknown attacks and safer than blacklist, but may be difficult to implement in some cases.

Cookie
A mechanism to exchange information, such as user data and access data, between a web server and browser.

SQL
A programming language designed for the manipulation and management of data in relational database (RDB). Divided into two major categories: DDL (Data Definition Language) for defining the database objects, such as structure and scheme, and DML（Data Manipulation Language） for manipulating and controlling access to data, such as SELECT, UPDATE and GRANT.

Checklist

The checklist provided here is a list of countermeasures against the web application vulnerabilities discussed in this book. When you perform security checking for your website, make use of this checklist and write down whether or not you need to implement countermeasures and if you have implemented countermeasures or not, to make sure that you have secured your website.

■ How to use the checklist

Check one that describes your situation best.

□ Done

Select this one when countermeasures have been already implemented.

□ Not Yet

Select this one when you realize you need to implement countermeasures but for some reason have not done yet.

□ No Need

Select this one when the vulnerabilities do not apply to your website or you judge your website is well protected in other ways and does not require these countermeasures.

■ Note

- It depends on web applications whether they require all, part of or none of the countermeasures discussed in this book. Please remember that the countermeasures discussed in this book are some examples of many possible solutions out there. Read the explanations carefully and understand the effects the countermeasure you are going to act on may have on your system before implementing it.

- Some countermeasures say “implement either one of them” or “implement this alternative when said countermeasure is not implementable (e.g.: the fundamental solutions to SQL Injection vulnerability 1-(i)-a and 1-(i)-b). We have put these countermeasures together into one check item. Check the “Done” box when you have implemented either one of the countermeasures listed and mark which one you have adopted.

- The fundamental solutions aim to enable a web application not to have vulnerability to begin with and thus most recommended. In the checklist, the fundamental solutions are bolded and colored so that you can see which ones are the fundamental solutions.

No	Types ofVulnerability	Type of Measure	Checkbox	Measure	Refer To
1	SQL Injection	**Fundamental**	* ☐ Done ☐ Not Done ☐ N/A	☐ **Build all SQL statements using placeholders.**	1-(i)-a
				☐ **When building an SQL statement through concatenation, use a special API offered by the database engine to perform escaping and make up the literals in the SQL statement correctly.**	1-(i)-b
		Fundamental	☐ Done ☐ Not Done ☐ N/A	**Do not write SQL statement directly in the parameter to be passed to the web application.**	1-(ii)
		Mitigation	☐ Done ☐ Not Done ☐ N/A	Limit information to display in error message on the web browser.	1-(iii)
		Mitigation	☐ Done ☐ Not Done ☐ N/A	Grant minimum privileges to database accounts.	1-(iv)
2	OS Command Injection	**Fundamental**	☐ Done ☐ Not Done ☐ N/A	☐ **Avoid using functions which could call shell commands.**	2-(i)
		Mitigation	☐ Done ☐ Not Done ☐ N/A	☐ When using functions which could call shell commands, check all variables that make up the shell parameters and make sure to execute only those that are granted to be executed.	2-(ii)
3	Unchecked Path Name Parameter / Directory Traversal	**Fundamental**	* ☐ Done ☐ Not Done ☐ N/A	☐ **Do not specify name of files stored on the web server directly using external parameter.**	3-(i)-a
				☐ **Use a fixed directory to handle filenames and nullify directory names in filenames.**	3-(i)-b
		Mitigation	☐ Done ☐ Not Done ☐ N/A	Manage file access permission properly.	3-(ii)
		Mitigation	☐ Done ☐ Not Done ☐ N/A	Check filenames.	3-(iii)
4	ImproperSession Management	**Fundamental**	☐ Done ☐ Not Done ☐ N/A	**Make session ID hard to guess.**	4-(i)
		Fundamental	☐ Done ☐ Not Done ☐ N/A	**Do not use URL parameters to store session ID.**	4-(ii)
		Fundamental	☐ Done ☐ Not Done ☐ N/A	**Set the secure attribute of the cookie when using HTTPS.**	4-(iii)
		Fundamental	* ☐ Done ☐ Not Done ☐ N/A	☐ **Start a new session after successful login.**	4-(iv)-a
				☐ **Issue a secret after login and authenticate the user with it whenever the user moves around the web site.**	4-(iv)-b
		Mitigation	☐ Done ☐ Not Done ☐ N/A	Use random session ID.	4-(v)
		Mitigation	☐ Done ☐ Not Done ☐ N/A	Set the cookie's expiration date with care when storing session ID in cookie.	4-(vi)

* Check if either one of the measures has been implemented

No	Types ofVulnerability		Type of Measure	Checkbox	Measure	Refer To
5	Cross-Site Scripting	Measures for Web Applications That Do Not Permit HTML Text Input	**Fundamental**	☐ Done ☐ Not Done ☐ N/A	**Perform Escaping for everything to be outputted to the web page.**	5-(i)
			Fundamental	☐ Done ☐ Not Done ☐ N/A	**When outputting URLs in HTML, permit only those that start with certain patterns, such as "http://" and "https://".**	5-(ii)
			Fundamental	☐ Done ☐ Not Done ☐ N/A	**Do not dynamically create the content of the <script>...</script> tag.**	5-(iii)
			Fundamental	☐ Done ☐ Not Done ☐ N/A	**Do not allow to import stylesheets from arbitrary websites.**	5-(iv)
			Mitigation	☐ Done ☐ Not Done ☐ N/A	Check input values.	5-(v)
		Measures for Web Applications That Permit HTML text Input	**Fundamental**	☐ Done ☐ Not Done ☐ N/A	**Create a parse tree from the HTML text input and extract only the necessary elements that do not contain scripts.**	5-(vi)
			Mitigation	☐ Done ☐ Not Done ☐ N/A	Nullify script strings in HTML text input.	5-(vii)
		Measures common to all web applications	**Fundamental**	☐ Done ☐ Not Done ☐ N/A	**Set the charset parameter of the HTTP Content-Type header.**	5-(viii)
			Mitigation	☐ Done ☐ Not Done ☐ N/A	Set the HttpOnly attribute of the cookie and disable the TRACE method to prevent disclosure of cookie information.	5-(ix)
6	CSRF (Cross-Site Request Forgery)		**Fundamental**	* ☐ Done ☐ Not Done ☐ N/A	☐ **Access the web page, in which certain operation is to be executed, via the POST method with a secret having the previous web page insert it in its hidden filed, and execute the requested operation only when the secret is correct.**	6-(i)-a
					☐ **Ask for password right before executing requested operation and proceed only when the password is correct.**	6-(i)-b
					☐ **Check the Referer whether it is the expected URL and proceed only when the URL is correct.**	6-(i)-c
			Mitigation	☐ Done ☐ Not Done ☐ N/A	Notify to the prespecified email address automatically when important operations have been done.	6-(ii)
7	HTTP Header Injection		**Fundamental**	* ☐ Done ☐ Not Done ☐ N/A	☐ **Do not print out HTTP header directly and do it through an HTTP header API provided by execution environment or programming language.**	7-(i)-a
					☐ **If HTTP header API that offers line feed neutralization is not available for use, implement it manually.**	7-(i)-b
			Mitigation	☐ Done ☐ Not Done ☐ N/A	Remove all line feed characters that appear in the external text input.	7-(ii)
8	Mail Header Injection		**Fundamental**	* □ Done □ Not Done □ N/A	☐ **Use the fixed values for the header elements and output all external input to the email body.**	8-(i)-a
					☐ **If 8-(i) is not implemented, the fixed values cannot be used for the header, use an email-sending API offered by the web application's execution environment or language.**	8-(i)-b
			Fundamental	☐ Done ☐ Not Done ☐ N/A	**Do not specify the email addresses in HTML.**	8-(ii)
			Mitigation	☐ Done ☐ Not Done ☐ N/A	Remove all line feed characters that appear in the external text input.	8-(iii)

* Check if either one of the measures has been implemented

No	Types ofVulnerability	Type of Measure	Checkbox	Measure	Refer To
9	Lack of Auhtentication and Authorization	**Fundamental**	☐ Done ☐ Not Done ☐ N/A	**When a web site needs access control, implement an authentication mechanism that requires users to enter some kind of secret information, such as password.**	9-(i)
		Fundamental	☐ Done ☐ Not Done ☐ N/A	**Implement authorization as well as authentication to make sure that a login user cannot pretend to be other users and access their data.**	9-(ii)

* Check if either one of the measures has been implemented

CWE Mapping Table

CWE (Common Weakness Enumeration) is a system to identify the types of vulnerabilities that come in a wide variety. CWE gives a hierarchically structured list of vulnerability types and allocates a CWE identifier (CWE-ID) to each type. The use of CWE will enable software developers and security experts to:

- Have a common language to discuss vulnerability in software architecture, design and code.
- Use as a standard measuring rule for security tools, such as vulnerability scanning tool, to enhance software security.
- Use as a common foundation to understand, mitigate and prevent vulnerability.

The table in the following page shows the mapping between the vulnerabilities addressed in this book and CWE. When implementing countermeasures individually based on CWE or checking the completeness of the countermeasures implemented, use the table as reference.

■ References

IPA: CWE (Common Weakness Enumeration) Overview
http://www.ipa.go.jp/security/english/vuln/CWE_en.html

No	"How to Secure Your Website" Types of Vulnerability	CWE Version 1.5 (Japanese)	CWE Version 1.11 (English)
1	SQL Injection	SQL Injection (CWE-89)	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') (CWE-89)
2	OS Command Injection	OS Command Injection (CWE-78)	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection') (CWE-78)
3	Unchecked Path Parameters / Directory Traversal	Path Traversal (CWE-22)	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal') (CWE-22)
4	Improper Session Management		Use of Insufficiently Random Values (CWE-330)
			Insufficiently Protected Credentials (CWE-522)
			Sensitive Cookie in HTTPS Session Without 'Secure' Attribute (CWE-614)
			Session Fixation (CWE-384)
5	Cross-Site Scripting	Cross-Site Scripting (XSS) (CWE-79)	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') (CWE-79)
6	CSRF (Cross-Site Request Forgery)	Cross-Site Request Forgery (CWE-352)	Cross-Site Request Forgery (CSRF) (CWE-352)
7	HTTP Header Injection		Improper Neutralization of CRLF Sequences in HTTP Headers (CWE-113)
8	Third Party Mail Relay		Improper Neutralization of CRLF Sequences (CWE-93)
9	Lack of Authentication and Authorization	Permissions, Privileges, and Access Controls (CWE-264)	Permissions, Privileges, and Access Controls (CWE-264)
		Improper Authentication (CWE-287)	Improper Authentication (CWE-287)

[Produced and Copyrighted by] IPA: Information-technology Promotion Agency, Japan

[Editor] Hideaki Kobayashi

[Author] Yukinobu Nagayasu Naoto Katsumi Hiroshi Tokumaru
Hiromitsu Takagi National Institute of Advanced Industrial Science and Technology (AIST)

[Advisor] Koji Yoshioka NEC System Technologies, Ltd.
Tetsushi Tanigawa NEC Corporation
Tadashi Yamagishi Hitachi Ltd.
Masashi Fujiwara Hitachi, Ltd.
Tadashi Kusama Fujitsu Limited
Noriko Totsuka Fujitsu Limited
Kosuke Ito LAC: Little eArth Corporation Co., Ltd.
Kazunao Wakai LAC: Little eArth Corporation Co., Ltd.
Shingo Otani LAC: Little eArth Corporation Co., Ltd.
Michio Sonoda Masashi Omori Hiroyuki Itabashi
Motokuni Soma Shunsuke Taniguchi Yasuo Miyakawa
Takeshi Hasegawa

*Affiliation omitted for the personnel of IPA

How to Secure Your Website

Approaches to Improve Web Application and Website Security

[Publication] Jan. 31, 2006 First Edition, First Printing
May 11, 2006 First Edition, Second Printing
Nov. 1, 2006 Second Edition, First Printing
Mar. 1, 2007 Second Edition, Second Printing
Sep. 10, 2007 Second Edition, Third Printing
Mar. 6, 2008 Third Edition, First Printing
Aug.1, 2008 Third Edition, Second Printing
Jan.20, 2010 Forth Edition, First Printing
Aug.5, 2010 Forth Edition, Second Printing
Apr.6, 2011 Fifth Edition, First Printing

[Produced and Copyrighted by] IT Security Center, Information-technology Promotion Agency, Japan

[Collaborated with] Research Center for Information Security, National Institute of Advanced Industrial Science and Technology

How to Report Information Security Issues to IPA

Designated by the Ministry of Economy, Trade and Industry, IPA IT Security Center collects information on the discovery of computer viruses and vulnerabilities, and the security incidents of virus infection and unauthorized access.

Make a report via web form or email. For more detail, please visit the web site:

URL: http://www.ipa.go.jp/security/todoke/ (Japanese only)

Computer Viruses

When you discover computer viruses or notice that your computers have been infected by viruses, please report to IPA.

Unauthorized Access

When you detect unauthorized access to your computers via network (e.g. the Internet, LANs, WANs and PC communications), please report to IPA.

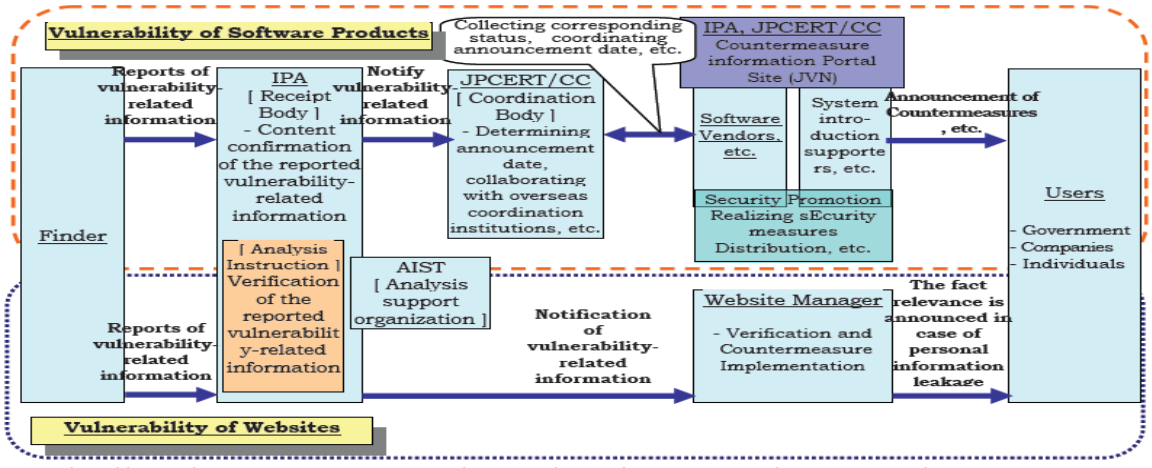
Software Vulnerability and Related Information

When you discover vulnerabilities in client software (e.g. OS and browser), server software (e.g. web server) and software embedded into hardware (e.g. printer and IC card) , please report to IPA.

Web Application Vulnerability and Related Information

When you discover vulnerabilities in systems that provide their customized services to the public, such as websites, please report to IPA.

Framework for Handling Vulnerability-Related Information
～ Information Security Early Warning Partnership ～



JPCERT/CC: Japan Computer Emergency Response Team Coordination Center, AIST: National Institute of Advanced Industrial Science and technology

INFORMATION-TECHNOLOGY PROMOTION AGENCY, JAPAN
2-28-8 Honkomagome, Bunkyo-ku, Tokyo 113-6591 JAPAN
http://www.ipa.go.jp/index-e.html

IT SECRITY CENTER
Tel: +81-3-5978-7527 FAX: +81-3-5978-7518
http://www.ipa.go.jp/security/english/