OWASP TOP 10 VULNERABILITIES & ASP.NET

BILL DINGER
Solutions Architect | @adazlian
bill.dinger@vml.com
https://github.com/BillDinger/OwaspTop10Examples

WHAT IS OWASP?

- **Open Web Application Security Project** – a 501(c)(3) focused on developing and improving the security of software
- Formed in 2001, its core purpose is to “Be the thriving global community that drives visibility and evolution in the safety and security of the world’s software"
- Provides numerous resources besides the OWASP Top 10, including personal favorites such as the password storage cheat sheet, SAMM Project and Zap Attack Proxy Project

OWASP TOP 10

- The 10 most common vulnerabilities found on the web today, last updated in 2013; the 2017 version is a release candidate.
- We will be reviewing the 2017 release specification. It's expected to be formalized in July or August 2017.
- OWASP rates the risks based on a strict methodology that takes into account how easy it is to exploit, impact, detect, etc.

THREAT AGENTS	ATTACK VECTORS	WEAKNESS PREVALENCE	WEAKNESS DETECTABILITY	TECHNICAL IMPACTS	BUSINESS IMPACTS
APP SPECIFIC	EASY	WIDESPREAD	EASY	SEVERE	APPLICATION OR BUSINESS-SPECIFIC
	AVERAGE	COMMON	AVERAGE	MODERATE	
	DIFFICULT	UNCOMMON	DIFFICULT	MINOR	

PRIMARY AIM

"The primary aim of the OWASP Top 10 is to educate developers, designers, architects, managers and organizations about the consequences of the most important web application security weaknesses. The Top 10 provides basic techniques to protect against these high-risk problem areas – and also provides guidance on where to go from here."

2017 OWASP TOP 10

A1	Injection
A2	Broken Authentication and Session Management
A3	Cross-site Scripting (XSS)
A4	Broken Access Control
A5	Security Misconfiguration
A6	Sensitive Data Exposure
A7	Insufficient Attack Protection
A8	Cross-site Request Forgery
A9	Using Components With Known Vulnerabilities
A10	Underprotected APIs

A 1

INJECTION

INJECTION

THREE-TIME CHAMPION — 2010, 2013, 2017

THREAT AGENTS	ATTACK VECTORS	SECURITY WEAKNESS		TECHNICAL IMPACTS	BUSINESS IMPACTS
APP-SPECIFIC	EXPLOITABILITY: EASY	PREVALENCE: COMMON	DETECTABILITY: AVERAGE	IMPACT: SEVERE	APPLICATION OR 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?

INJECTION

EXAMPLE ATTACK (DEMO)

SQL Injection Demo – our login allows arbitrary SQL injection.

PROTECTION AGAINST INJECTION ATTACKS

General (LDAP, XPath, NoSQL, etc.)

- Use a parameterized API
- Escape special characters using a library designed for this purpose
- White list input validation (for example, use regular expressions to validate data)

SQL

- Parameterized queries
- Stored procedures
- Escape all user-supplied input

FIXING OUR VULNERABILITY

Added input validation, parameterized query and entity framework to our login

A 2

BROKEN AUTHENTICATION OR SESSION MANAGEMENT

BROKEN AUTHENTICATION OR SESSION MANAGEMENT

THREAT AGENTS	ATTACK VECTORS	SECURITY WEAKNESS		TECHNICAL IMPACTS	BUSINESS IMPACTS
APP-SPECIFIC	EXPLOITABILITY: AVERAGE	PREVALENCE: COMMON	DETECTABILITY: AVERAGE	IMPACT: SEVERE	APPLICATION OR 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 log out, 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.

BROKEN AUTHENTICATION OR SESSION MANAGEMENT

EXAMPLE ATTACK (DEMO)

When registering a user, we store the password using symmetric encryption

PROTECTION AGAINST INJECTION ATTACKS

- Use an authentication and session system that meets OWASP's Application Security Verification System (ASVS)
- Use a well-known implementation of authentication and authorization
- Use OWASP password storage cheat sheets, forgot password cheat sheets and session management cheat sheets to validate your implementations

FIXING OUR VULNERABILITY

We use PB2KDF for our password storage

A 3

CROSS-SITE SCRIPTING (XSS)

CROSS-SITE SCRIPTING (XSS)

THREAT AGENTS	ATTACK VECTORS	SECURITY WEAKNESS		TECHNICAL IMPACTS	BUSINESS IMPACTS
APP-SPECIFIC	EXPLOITABILITY: AVERAGE	PREVALENCE: VERY WIDESPREAD	DETECTABILITY: AVERAGE	IMPACT: MODERATE	APPLICATION OR 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 when 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 websites, 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.

CROSS-SITE SCRIPTING (XSS)

EXAMPLE OF XSS ATTACK

Reflected XSS example – a malicious email link or social link that injects the script into the webpage

Stored XSS example – our blog comment box lets us inject JavaScript

PROTECTIONS AGAINST XSS

- To escape untrusted data, see OWASP XSS Prevention Cheat Sheet
- CSP (Content Security Policy) browser security headers
- Never insert untrusted data to JavaScript APIs that can generate active content

FIXED VULNERABILITY

- Escape data coming into our forum to prevent reflected
- Social link fixed by using textContent instead of innerHtml

XSS MATRIX

WHERE UNTRUSTED DATA IS USED

DATA PERSISTENCE

XSS	SERVER	CLIENT
STORED	STORED SERVER XSS	STORED CLIENT XSS
REFLECTED	REFLECTED SERVER XSS	REFLECTED CLIENT XSS

DOM-Based XSS is a subset of Client XSS (where the data source is from the DOM only)
Stored versus Reflected only affects the likelihood of successful attack, not the nature of the vulnerability or the most effective defense

A 4

BROKEN ACCESS CONTROL

BROKEN ACCESS CONTROL

THREAT AGENTS	ATTACK VECTORS	SECURITY WEAKNESS		TECHNICAL IMPACTS	BUSINESS IMPACTS
APP-SPECIFIC	EXPLOITABILITY: EASY	PREVALENCE: WIDESPREAD	DETECTABILITY: EASY	IMPACT: MODERATE	APPLICATION OR 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.

BROKEN ACCESS CONTROL

EXAMPLE OF BROKEN ACCESS CONTROL

Post to our Add Blog Entries doesn’t require access control

PROTECTIONS AGAINST BROKEN ACCESS CONTROL

- Check Access on every action
- Use per user/per session object references (i.e., turn GUID1 into 1, GUID2 into 2)

FIXED VULNERABILITY

Added access control to our Add Blog Entry method

A 5

SECURITY MISCONFIGURATION

SECURITY MISCONFIGURATION

THREAT AGENTS	ATTACK VECTORS	SECURITY WEAKNESS		TECHNICAL IMPACTS	BUSINESS IMPACTS
APP-SPECIFIC	EXPLOITABILITY: EASY	PREVALENCE: COMMON	DETECTABILITY: EASY	IMPACT: MODERATE	APPLICATION OR 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.

SECURITY MISCONFIGURATION

EXAMPLE OF SECURITY MISCONFIGURATION

Stack traces leaking to users

PROTECTIONS AGAINST SECURITY MISCONFIGURATION

- PATCH YOUR SYSTEMS
- Desired state configuration, using systems such as Docker, Puppet, Chef, Vagrant, etc.
- Following guidelines on configuration management and server hardening for platforms
- Separate out application roles (database, web server)
- Reduce attack surface by removing features

FIXED VULNERABILITY

Replaced with a custom error page

A 6

SENSITIVE DATA EXPOSURE

SENSITIVE DATA EXPOSURE

THREAT AGENTS	ATTACK VECTORS	SECURITY WEAKNESS		TECHNICAL IMPACTS	BUSINESS IMPACTS
APP-SPECIFIC	EXPLOITABILITY: DIFFICULT	PREVALENCE: UNCOMMON	DETECTABILITY: AVERAGE	IMPACT: SEVERE	APPLICATION OR 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 are 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 because 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.

SENSITIVE DATA EXPOSURE

EXAMPLE OF SENSITIVE DATA EXPOSURE

Logs expose passwords/sensitive data not sent over HTTPs

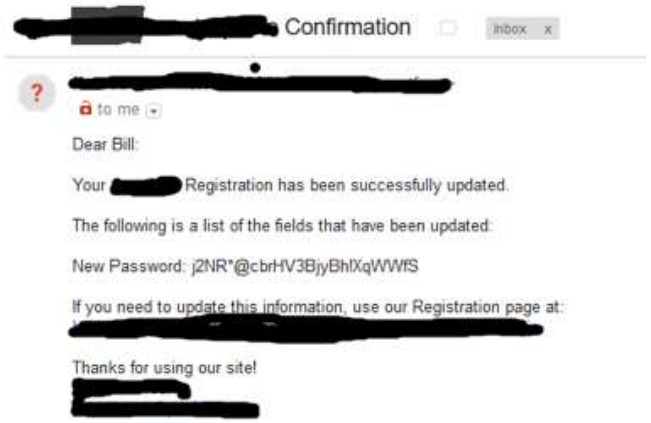
PROTECTIONS AGAINST SENSITIVE DATA EXPOSURE

- Encrypt all sensitive data at rest and in transit. Consider both internal and external attacks.
- Don’t store any sensitive data unless absolutely required.
- Ensure strong key management is used, and use FIPS 140 validated cryptographic methods.
- Hash and salt passwords using an algorithm specifically designed for them.

FIXED VULNERABILITY

- Removed password from being logged
- Changed route to HTTPS

REAL-LIFE EXAMPLE



A 7

INSUFFICIENT ATTACK PROTECTION

INSUFFICIENT ATTACK PROTECTION

THREAT AGENTS	ATTACK VECTORS	SECURITY WEAKNESS		TECHNICAL IMPACTS	BUSINESS IMPACTS
APP-SPECIFIC	EXPLOITABILITY: EASY	PREVALENCE: COMMON	DETECTABILITY: AVERAGE	IMPACT: MODERATE	APPLICATION OR BUSINESS-SPECIFIC
Consider that 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 users 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 percent. 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.

INSUFFICIENT ATTACK PROTECTION

EXAMPLE OF SENSITIVE DATA EXPOSURE

A controller method that accepts null or malformed input and just rejects it

PROTECTIONS AGAINST INSUFFICIENT ATTACK PROTECTION

- WAF (web application firewall)
- Intrusion detection systems
- IP blacklists, multi-factor authentication, CAPTCHA, etc.

FIXED VULNERABILITY

- After malformed data is detected by our methods, we start blacklisting IP addresses
- Add WAF in Azure management portal

A 8

CROSS-SITE REQUEST FORGERY (CSRF)

CROSS-SITE REQUEST FORGERY (CSRF)

THREAT AGENTS	ATTACK VECTORS	SECURITY WEAKNESS		TECHNICAL IMPACTS	BUSINESS IMPACTS
APP-SPECIFIC	EXPLOITABILITY: AVERAGE	PREVALENCE: UNCOMMON	DETECTABILITY: EASY	IMPACT: MODERATE	APPLICATION OR 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 that 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.

CROSS-SITE REQUEST FORGERY (CSRF)

EXAMPLE OF CROSS-SITE REQUEST FORGERY

Webpage that submits a post changing the password by clicking on an image link

PROTECTIONS AGAINST CSRF

- .NET has built-in CSRF protection, as do many other frameworks such as AngularJS
- The most effective is synchronizer token pattern

XSRF/CSRF Prevention in ASP.NET MVC and Web Pages

FIXED VULNERABILITY

Added [csrf] to method

A 9

USING COMPONENTS WITH KNOWN VULNERABILITIES

USING COMPONENTS WITH KNOWN VULNERABILITIES

THREAT AGENTS	ATTACK VECTORS	SECURITY WEAKNESS		TECHNICAL IMPACTS	BUSINESS IMPACTS
APP-SPECIFIC	EXPLOITABILITY: AVERAGE	PREVALENCE: COMMON	DETECTABILITY: AVERAGE	IMPACT: MODERATE	APPLICATION OR 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, let alone 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.

USING COMPONENTS WITH KNOWN VULNERABILITIES

EXAMPLE OF COMPONENT WITH KNOWN VULNERABILITY

.NET has had 106 CVEs (and counting)

Security Vulnerabilities

PROTECTIONS AGAINST USING A COMPONENT WITH KNOWN VULNERABILITIES

- Keep packages up to date
- Use OWASP Dependency Check for your packages
- Monitor CVEs

A 1 0

UNDERPROTECTED APIS

UNDERPROTECTED APIs

THREAT AGENTS	ATTACK VECTORS	SECURITY WEAKNESS		TECHNICAL IMPACTS	BUSINESS IMPACTS
APP-SPECIFIC	EXPLOITABILITY: AVERAGE	PREVALENCE: COMMON	DETECTABILITY: DIFFICULT	IMPACT: MODERATE	APPLICATION OR 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.

UNDERPROTECTED APIS

EXAMPLE OF UNDERPROTECTED APIs

Our login method has no rate limiting

PROTECTIONS AGAINST UNDERPROTECTED APIS

- Somewhat of a meta-protection: implement CSRF protections, rate limiting, WAFs, security
- Assume all your APIs will be and can be discovered by hackers
- Validate all input for injection attacks

FIXED VULNERABILITIES

Added rate limiter

RESOURCES

.NET Security Cheat Sheet:
https://www.owasp.org/index.php/.NET_Security_Cheat_Sheet

OWASP Top 10 Project:
https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project#tab=OWASP_Top_10_for_2017_Release_Candidate

Security in the .NET framework:
https://msdn.microsoft.com/en-us/library/fkytk30f(v=vs.110).aspx

Troy Hunt's OWASP Top 10 for .NET developers:
https://www.troyhunt.com/owasp-top-10-for-net-developers-part-1/#

Demo source code:
https://github.com/BillDinger/OWASPTop10Examples

THANK YOU.