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 Cheat Sheet Series, SAMM Project and Zap Attack Proxy Project

OWASP TOP 10

- The 10 most common vulnerabilities found on the web today.
- We will be reviewing the 2017 version – released in October 2017 and supersedes the 2013 version.
- 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

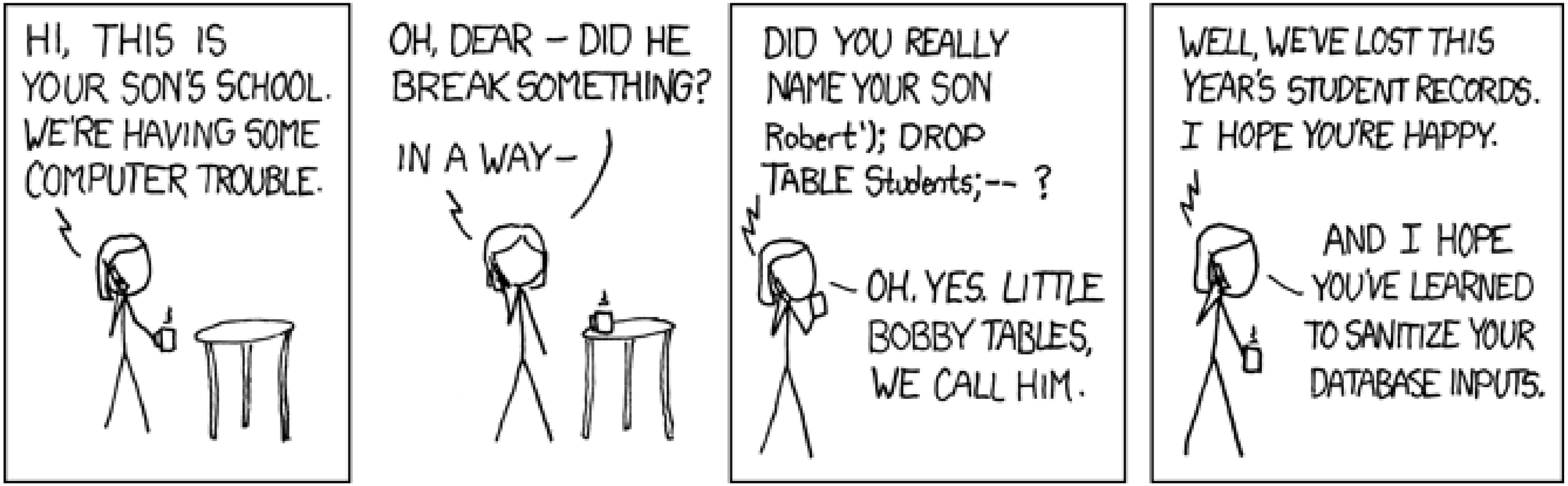
"The primary aim of the OWASP Top 10 is to educate developers, designers, architects, managers and organizations about the consequences of the most important web application security weaknesses. The Top 10 provides basic techniques to protect against these high-risk problem areas – and also provides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

2017 OWASP TOP 10

A1	Injection
A2	Broken Authentication
A3	Sensitive Data Exposure
A4	XML External Entities (XXE)
A5	Broken Access Control
A6	Security Misconfiguration
A7	Cross-Site Scripting (XSS)
A8	Insecure Deserialization
A9	Using Components With Known Vulnerabilities
A10	Insufficient Logging & Monitoring

A 1

INJECTION



INJECTION

THREE-TIME CHAMPION — 2010, 2013, 2017

THREAT AGENTS / ATTACK VECTORS	SECURITY WEAKNESS		IMPACTS
APP-SPECIFIC | EXPLOITABILITY: SEVERE	PREVALENCE: COMMON	DETECTABILITY: AVERAGE	IMPACT: SEVERE
Almost any source of data can be an injection vector, environment variables, parameters, external and internal web services, and all types of users. Injection Flaws occur when an attacker can send hostile data to an interpreter.	Injection flaws are very prevalent, particularly in legacy code. Injection vulnerabilities are often found in SQL, LDAP, XPath, or NoSQL queries. Injection flaws are easy to discover when examining code		Injection can result in data loss, corruption, or disclosure to unauthorized parties, loss of accountability, or denial of access. Injection can sometimes lead to complete host takeover. The business impact depends on the needs of the application and data.

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 Specific

- 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

BROKEN AUTHENTICATION

THREAT AGENTS / ATTACK VECTORS	SECURITY WEAKNESS		IMPACTS
APP-SPECIFIC | EXPLOITABILITY: SEVERE	PREVALENCE: COMMON	DETECTABILITY: AVERAGE	IMPACT: SEVERE
Attackers have access to hundreds of millions of valid username and password combinations for credential stuffing, default administrative account lists, automated brute force, and dictionary attack tools. Session management attacks are well understood, particularly in relation to unexpired session tokens.	The prevalence of broken authentication is widespread due to the design and implementation of most identity and access controls. Attackers can detect broken authentication using manual means and exploit them using automated tools with password lists and dictionary attacks.		Attackers have to gain access to only a few accounts, or just one admin account to compromise the system. Depending on the domain of the application, this may allow money laundering, social security fraud, and identity theft, or disclose legally protected highly sensitive information.

BROKEN AUTHENTICATION

EXAMPLE ATTACK (DEMO)

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

PROTECTION AGAINST BROKEN AUTHENTICATION

- 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
- Implement and required multi-factor authentication
- Follow applicable NIST Standards (https://pages.nist.gov/800-63-3/sp800-63b.html#memsecret) for password best practices.

FIXING OUR VULNERABILITY

We use PB2KDF for our password storage

A 3

SENSITIVE DATA EXPOSURE

SENSITIVE DATA EXPOSURE

THREAT AGENTS / ATTACK VECTORS	SECURITY WEAKNESS		IMPACTS
APP-SPECIFIC | EXPLOITABILITY: SEVERE	PREVALENCE: COMMON	DETECTABILITY: AVERAGE	IMPACT: SEVERE
Rather than directly attacking crypto, attackers steal keys, execute man-in-the-middle attacks, or steal clear text data off the server, while in transit, or from the user's client, e.g. browser. A manual attack is generally required. Previously retrieved password databases could be brute forced by Graphics Processing Units (GPUs).	The most common flaw is simply not encrypting sensitive data. When crypto is employed, weak key generation and management, and weak algorithm, protocol and cipher usage is common, particularly for weak password hashing storage techniques. For data in transit, server side weaknesses are mainly easy to detect, but hard for data at rest.		Failure frequently compromises all data that should have been protected. Typically, this information includes sensitive personal information (PII) data such as health records, credentials, personal data, and credit cards, which often require protection as defined by laws or regulations such as the EU GDPR or local privacy laws.

SENSITIVE DATA EXPOSURE

EXAMPLE OF SENSITIVE DATA EXPOSURE

Logs expose passwords/sensitive data not sent over HTTPs

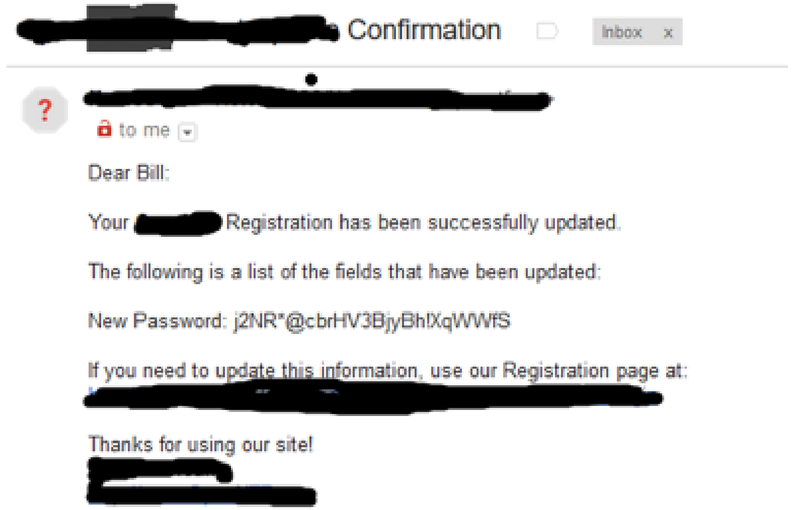
PROTECTIONS AGAINST SENSITIVE DATA EXPOSURE

- 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.
- Use

FIXED VULNERABILITY

- Removed password from being logged
- Changed route to HTTPS

REAL-LIFE EXAMPLE



A 4

XML EXTERNAL ENTITIES (XXE)

XML EXTERNAL ENTITIES (XXE)

THREAT AGENTS / ATTACK VECTORS	SECURITY WEAKNESS		IMPACTS
APP-SPECIFIC | EXPLOITABILITY: MODERATE	PREVALENCE: COMMON	DETECTABILITY: AVERAGE	IMPACT: SEVERE
Attackers can exploit vulnerable XML processors if they can upload XML or include hostile content in an XML document, exploiting vulnerable code, dependencies or integrations.	By default, many older XML processors allow specification of an external entity, a URI that is dereferenced and evaluated during XML processing. SAST (Source Code Analysis Tools) and DAST (Dynamic Application Security Testing) tools can help discover potential flaws.		These flaws can be used to extract data, execute a remote request from the server, scan internal systems, perform a denial-of-service attack, as well as execute other attacks. The business impact depends on the protection needs of all affected application and data.

XML EXTERNAL ENTITIES

EXAMPLE OF XML EXTERNAL ENTITIES

We process XML without denying external DTD.

PROTECTIONS AGAINST XML EXTERNAL ENTITIES

- Don’t use XML, use JSON.
- Update XML libraries and parsers.
- Disable XML external entity and DTD processing.
- Implement positive whitelisting of valid XML documents.

FIXED VULNERABILITY

- Disabled DTD processing.

A 5

BROKEN ACCESS CONTROL

BROKEN ACCESS CONTROL

THREAT AGENTS / ATTACK VECTORS	SECURITY WEAKNESS		IMPACTS
APP-SPECIFIC | EXPLOITABILITY: MODERATE	PREVALENCE: COMMON	DETECTABILITY: AVERAGE	IMPACT: SEVERE
Exploitation of access control is a core skill of attackers. SAST and DAST tools can detect the absence of access control but cannot verify if it is functional when it is present. Access control is detectable using manual means, or possibly through automation for the absence of access controls in certain frameworks	Access control weaknesses are common due to the lack of automated detection, and lack of effective functional testing by application developers. Manual testing is the best way to detect missing or ineffective access control, including HTTP method (GET vs PUT) controller, direct object references, etc.		The technical impact is attackers acting as users or administrators, or users using privileged functions, or creating, accessing, updating or deleting every record. The business impact depends on the protection needs of the application and data.

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 6

SECURITY MISCONFIGURATION

SECURITY MISCONFIGURATION

THREAT AGENTS / ATTACK VECTORS	SECURITY WEAKNESS		IMPACTS
APP-SPECIFIC | EXPLOITABILITY: EASY	PREVALENCE: COMMON	DETECTABILITY: AVERAGE	IMPACT: MODERATE
Attackers will often attempt to exploit unpatched flaws or access default accounts, unused pages, unprotected files and directories, etc to gain unauthorized access or knowledge of the system	Access control weaknesses are common due to the lack of automated detection, and lack of effective functional testing by application developers. Manual testing is the best way to detect missing or ineffective access control, including HTTP method (GET vs PUT) controller, direct object references, etc.		Security misconfiguration can happen at any level of an application stack, including the network services, platform, web server, application server, database, frameworks, custom code, and pre-installed virtual machines, containers, or storage. Automated scanners are useful for detecting misconfigurations, use of default accounts or configurations, unnecessary services, legacy options, etc.

SECURITY MISCONFIGURATION

EXAMPLE OF SECURITY MISCONFIGURATION

Stack traces leaking to users & Not Using HTTPS

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
- Default accounts/passwords aren’t disabled.

FIXED VULNERABILITY

Replaced with a custom error page

A 7

CROSS-SITE SCRIPTING (XSS)

CROSS-SITE SCRIPTING (XSS)

THREAT AGENTS / ATTACK VECTORS	SECURITY WEAKNESS		IMPACTS
APP-SPECIFIC | EXPLOITABILITY: EASY	PREVALENCE: COMMON	DETECTABILITY: AVERAGE	IMPACT: MODERATE TO SEVERE
Automated tools can detect and exploit all three forms of XSS, and there are freely available exploitation frameworks.	XSS is the second most prevalent issue in the OWASP Top 10, and is found in around two thirds of all applications. Automated tools can find some XSS problems automatically, particularly in mature technologies such as PHP, J2EE / JSP, and ASP.NET.		The impact of XSS is moderate for reflected and DOM XSS, and severe for stored XSS, with remote code execution on the victim's browser, such as stealing credentials, sessions, or delivering malware to the victim.

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
- Use frameworks that support automatically escaping XSS by design (Asp.NET MVC, React, Angular,etc)

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 8

INSECURE DESERIALIZATION

INSECURE DESERIALIZATION

THREAT AGENTS / ATTACK VECTORS	SECURITY WEAKNESS		IMPACTS
APP-SPECIFIC | EXPLOITABILITY: POOR	PREVALENCE: RARE	DETECTABILITY: AVERAGE	IMPACT: MODERATE
Exploitation of deserialization is somewhat difficult, as off the shelf exploits rarely work without changes or tweaks to the underlying exploit code.	This issue is included in the Top 10 based on an industry survey and not on quantifiable data. Some tools can discover deserialization flaws, but human assistance is frequently needed to validate the problem. It is expected that prevalence data for deserialization flaws will increase as tooling is developed to help identify and address it.		While some known vulnerabilities lead to only minor impacts, some of the largest breaches to date have relied on exploiting known vulnerabilities in components. Depending on the assets you are protecting, perhaps this risk should be at the top of the list.

INSECURE DESERIALIZATION

EXAMPLE OF INSECURE DESERIALIZATION ATTACK

Our API is using JSON.NET with TypeNameHandling.All turned on, allowing arbitrary code execution (https://www.alphabot.com/security/blog/2017/net/How-to-configure-Json.NET-to-create-a-vulnerable-web-API.html)

PROTECTIONS AGAINST DESERIALIZATION ATTACK

- Stick to a well known framework (JSON.NET, DataContractJSONSerializer) that by default don't allow type serialization.
- Implementing integrity checks such as a HMAC signature to verify payloads haven't been altered in transit.
- If you must accept multiple types, use a strict whitelist of allowed types for deserialization.
- Only accept data from trusted sources.
- Run deserialization in low access threads (i.e. use the default IIS user, don't run IIS as a SystemUser)

FIXED VULNERABILITY

A 9

USING COMPONENTS WITH KNOWN VULNERABILITIES

USING COMPONENTS WITH KNOWN VULNERABILITIES

THREAT AGENTS / ATTACK VECTORS	SECURITY WEAKNESS		IMPACTS
APP-SPECIFIC | EXPLOITABILITY: AVERAGE	PREVALENCE: COMMON	DETECTABILITY: AVERAGE	IMPACT: SEVERE
While it is easy to find already-written exploits for many known vulnerabilities, other vulnerabilities require concentrated effort to develop a custom exploit.	Prevalence of this issue is very widespread. Component-heavy development patterns can lead to development teams not even understanding which components they use in their application or API, much less keeping them up to date.		The impact of deserialization flaws cannot be overstated. These flaws can lead to remote code execution attacks, one of the most serious attacks possible. The business impact depends on the protection needs of the application and data.

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

INSUFFICIENT LOGGING AND MONITORING

INSUFFICIENT LOGGING AND MONITORING

THREAT AGENTS / ATTACK VECTORS	SECURITY WEAKNESS		IMPACTS
APP-SPECIFIC | EXPLOITABILITY: AVERAGE	PREVALENCE: COMMON	DETECTABILITY: POOR	IMPACT: MODERATE
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.

INSUFFICIENT LOGGING AND MONITORING

EXAMPLE OF INSUFFICIENT LOGGING AND MONITORING

We don’t log authentication and authorization attempts and privileged attempts.

PROTECTIONS AGAINST INSUFFICIENT LOGGING AND MONITORING

- Use a well known logging and auditing framework for authentication and authorization attempts
- Make sure logs are in an easily readable format.
- Ensure high value actions are stored in an unalterable format for audit purposes.
- Setup monitoring and alerting for suspicious activities.

FIXED VULNERABILITY

- Add auditing using a combination of Log4Net and Audit.net

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

OWASP Cheat Sheet Series:
https://www.owasp.org/index.php/OWASP_Cheat_Sheet_Series

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/#

JSON Attacks in .NET/JSON:

https://www.blackhat.com/docs/us-17/thursday/us-17-Munoz-Friday-The-13th-JSON-Attacks-wp.pdf

THANK YOU.

VML