Why Web Security?

About OWASP

The Open Web Application Security Project (OWASP) is an open community dedicated to enabling organizations to develop, purchase, and maintain applications and APIs that can be trusted.

At OWASP, you'll find free and open:

- Application security tools and standards.
- Complete books on application security testing, secure code development, and secure code review.
- Presentations and videos.
- Cheat sheets on many common topics.
- Standard security controls and libraries.
- Local chapters worldwide.
- Cutting edge research.
- Extensive conferences worldwide.
- Mailing lists.

Learn more at: https://www.owasp.org.

All OWASP tools, documents, videos, presentations, and chapters are free and open to anyone interested in improving application security.

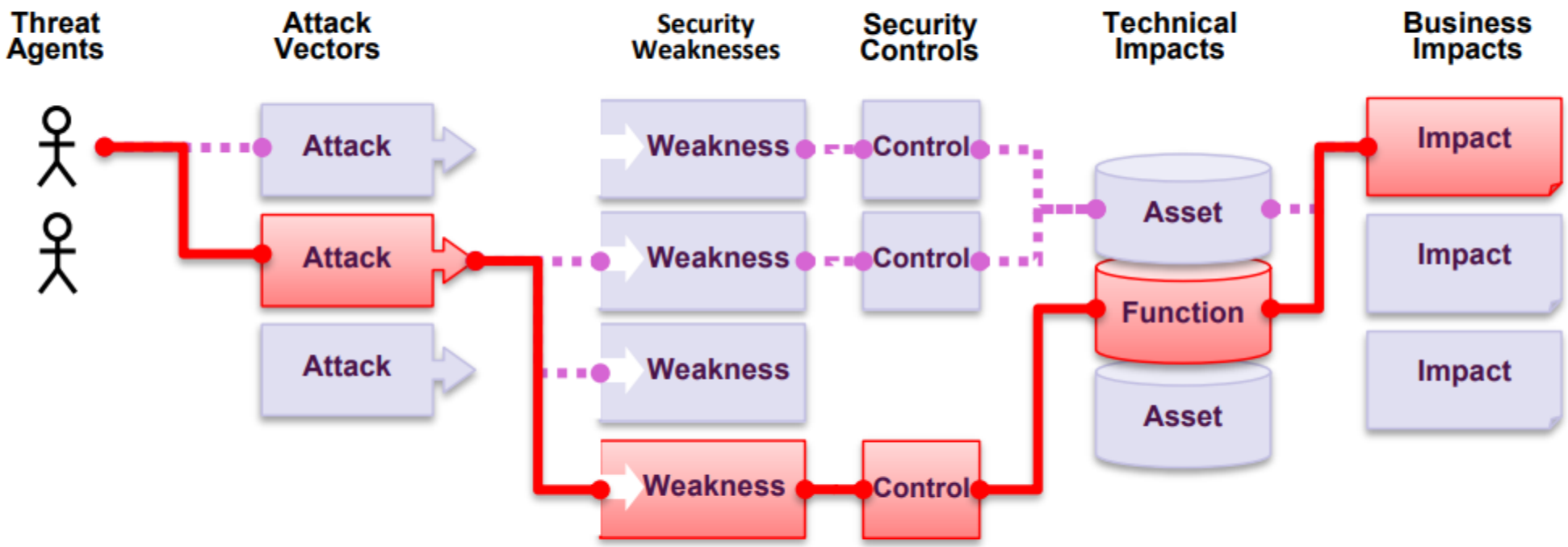
OWASP

OWASP Top 10 - 2017

The Ten Most Critical Web Application Security Risks

What Are Application Security Risks?

Attackers can potentially use many different paths through your application to do harm to your business or organization. Each of these paths represents a risk that may, or may not, be serious enough to warrant attention.



Sometimes these paths are trivial to find and exploit, and sometimes they are extremely difficult. Similarly, the harm that is caused may be of no consequence, or it may put you out of business. To determine the risk to your organization, you can evaluate the likelihood associated with each threat agent, attack vector, and security weakness and combine it with an estimate of the technical and business impact to your organization. Together, these factors determine your overall risk.

What's My Risk?

The OWASP Top 10 focuses on identifying the most serious web application security risks for a broad array of organizations. For each of these risks, we provide generic information about likelihood and technical impact using the following simple ratings scheme, which is based on the OWASP Risk Rating Methodology.

Threat Agents	Exploitability	Weakness Prevalence	Weakness Detectability	Technical Impacts	Business Impacts
Appli-cation Specific	Easy: 3	Widespread: 3	Easy: 3	Severe: 3	Business Specific
	Average: 2	Common: 2	Average: 2	Moderate: 2	
	Difficult: 1	Uncommon: 1	Difficult: 1	Minor: 1	

In this edition, we have updated the risk rating system to assist in calculating the likelihood and impact of any given risk. For more details, please see **Note About Risks**.

Each organization is unique, and so are the threat actors for that organization, their goals, and the impact of any breach. If a public interest organization uses a content management system (CMS) for public information and a health system uses that same exact CMS for sensitive health records, the threat actors and business impacts can be very different for the same software. It is critical to understand the risk to your organization based on applicable threat agents and business impacts.

Where possible, the names of the risks in the Top 10 are aligned with Common Weakness Enumeration (CWE) weaknesses to promote generally accepted naming conventions and to reduce confusion.

References

OWASP

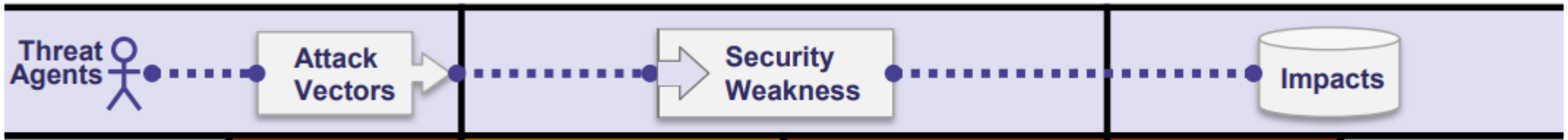
- OWASP Risk Rating Methodology
- Article on Threat/Risk Modeling

External

- ISO 31000: Risk Management Std
- ISO 27001: ISMS
- NIST Cyber Framework (US)
- ASD Strategic Mitigations (AU)
- NIST CVSS 3.0
- Microsoft Threat Modelling Tool

OWASP Top 10 - 2013		OWASP Top 10 - 2017
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]	U	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 Contr [Merged+A4]	U	A7:2017-Cross-Site Scripting (XSS)
A8 – Cross-Site Request Forgery (CSRF)	☒	A8:2017-Insecure Deserialization [NEW, Community]
A9 – Using Components with Known Vulnerabilities	→	A9:2017-Using Components with Known Vulnerabilities
A10 – Unvalidated Redirects and Forwards	☒	A10:2017-Insufficient Logging&Monitoring [NEW,Comm.]

A1 :2017 Injection



App. Specific	Exploitability: 3	Prevalence: 2	Detectability: 3	Technical: 3	Business ?
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, OS commands, XML parsers, SMTP headers, expression languages, and ORM queries. Injection flaws are easy to discover when examining code. Scanners and fuzzers can help attackers find injection flaws.		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 Overview

Attack Vectors	Security Weaknesses		Technical Impacts
Exploitability Easy	Prevalence Common	Detectability Average	Impact Severe

Understanding SQL Injection

Trusted

```
http://www.mysite.com/Widget?Id=1 or 1=1
```



```
SELECT * FROM Widget WHERE ID = 1 or 1 = 1
```

Always true

Common Defences Against Injection Attacks

Whitelist
untrusted data

- What input do we *trust?*
- Does it adhere to expected patterns?

Parameterise
SQL statements

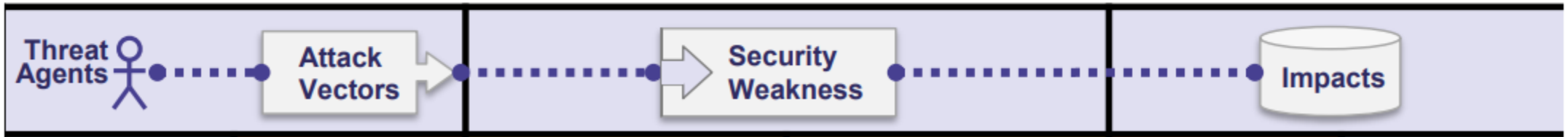
- Separate the query from the input data
- Type cast each parameter

Fine tune DB
permissions

- Segment accounts for admin and public
- Apply the "principle of least privilege"

A2 :2017

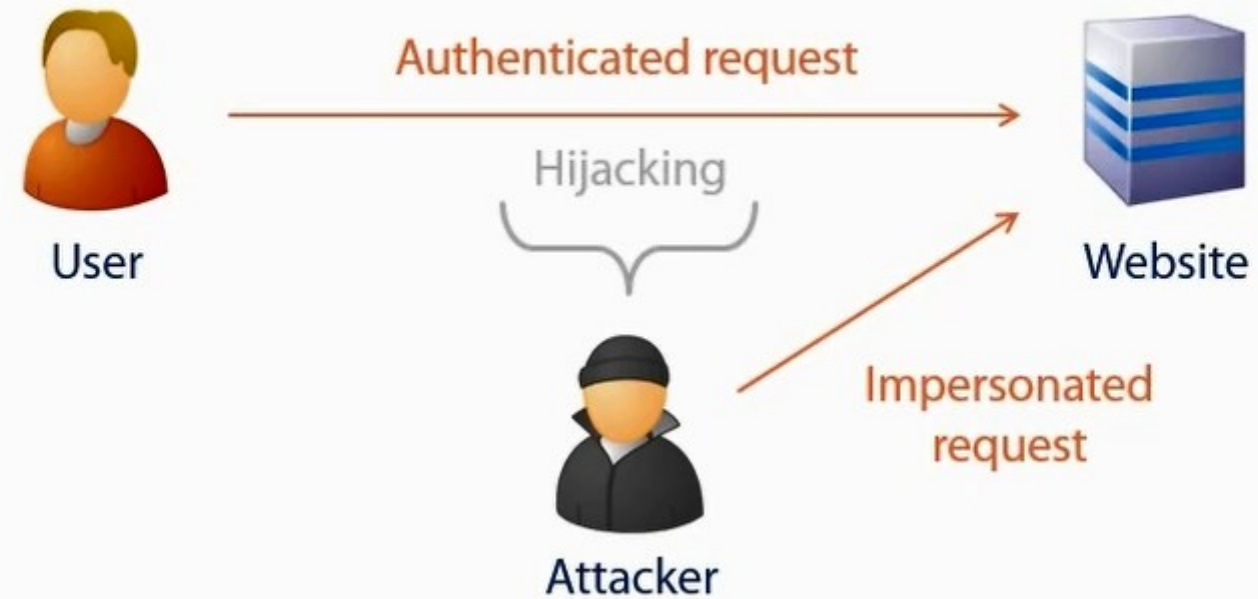
Broken Authentication



App. Specific	Exploitability: 3	Prevalence: 2	Detectability: 2	Technical: 3	Business ?
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. Session management is the bedrock of authentication and access controls, and is present in all stateful applications. 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 & Session Management Overview

Attack Vectors	Security Weaknesses		Technical Impacts
Exploitability Average	Prevalence Widespread	Detectability Average	Impact Severe



Understanding Hijacking

Is the Application Vulnerable?

Confirmation of the user's identity, authentication, and session management are critical to protect against authentication-related attacks.

There may be authentication weaknesses if the application:

- Permits automated attacks such as credential stuffing, where the attacker has a list of valid usernames and passwords.
- Permits brute force or other automated attacks.
- Permits default, weak, or well-known passwords, such as "Password1" or "admin/admin".
- Uses weak or ineffective credential recovery and forgot-password processes, such as "knowledge-based answers", which cannot be made safe.
- Uses plain text, encrypted, or weakly hashed passwords (see **A3:2017-Sensitive Data Exposure**).
- Has missing or ineffective multi-factor authentication.
- Exposes Session IDs in the URL (e.g., URL rewriting).
- Does not rotate Session IDs after successful login.
- Does not properly invalidate Session IDs. User sessions or authentication tokens (particularly single sign-on (SSO) tokens) aren't properly invalidated during logout or a period of inactivity.

How to Prevent

- Where possible, implement multi-factor authentication to prevent automated, credential stuffing, brute force, and stolen credential re-use attacks.
- Do not ship or deploy with any default credentials, particularly for admin users.
- Implement weak-password checks, such as testing new or changed passwords against a list of the top 10000 worst passwords.
- Align password length, complexity and rotation policies with NIST 800-63 B's guidelines in section 5.1.1 for Memorized Secrets or other modern, evidence based password policies.
- Ensure registration, credential recovery, and API pathways are hardened against account enumeration attacks by using the same messages for all outcomes.
- Limit or increasingly delay failed login attempts. Log all failures and alert administrators when credential stuffing, brute force, or other attacks are detected.
- Use a server-side, secure, built-in session manager that generates a new random session ID with high entropy after login. Session IDs should not be in the URL, be securely stored and invalidated after logout, idle, and absolute timeouts.

Common Defences Against Broken Authentication

Protect the cookies

- Use the HttpOnly flag
- Make sure they're flagged as "Secure"

Decrease the window of risk

- Expire sessions quickly
- Re-challenge the user on key actions

Harden the account management

- Allow (and encourage) strong passwords
- Implement login rate limiting and lockouts

Broken Authentication in the Wild – Apple Hack

While configuring iMessages on OS X Mountain Lion, Martin Levy at ShootitLive, found that they were able to take full control of someone else's Apple ID over the same Wi-Fi network, which could mean that they can have full access to the other person's iTunes and App Store accounts; they could change the verified email address and even change the security settings around.

Martin has described the process of how to take control of someone else's Apple ID and from the looks of it the attack seems to be similar to that of a 'Session Fixation Attack'. Once the user logs in there is some kind of ID string in the URL, we assume it to be a session ID, which Apple is probably not cross-checking with the cookie that is set on the user's system.

A3 :2017

Sensitive Data Exposure

App. Specific	Exploitability: 2	Prevalence: 3	Detectability: 2	Technical: 3	Business ?
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).		Over the last few years, this has been the most common impactful attack. 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 Overview

Attack Vectors	Security Weaknesses		Technical Impacts
Exploitability Difficult	Prevalence Uncommon	Detectability Average	Impact Severe

Understanding Sensitive Data Exposure

Is the Application Vulnerable?

The first thing is to determine the protection needs of data in transit and at rest. For example, passwords, credit card numbers, health records, personal information and business secrets require extra protection, particularly if that data falls under privacy laws, e.g. EU's General Data Protection Regulation (GDPR), or regulations, e.g. financial data protection such as PCI Data Security Standard (PCI DSS). For all such data:

- Is any data transmitted in clear text? This concerns protocols such as HTTP, SMTP, and FTP. External internet traffic is especially dangerous. Verify all internal traffic e.g. between load balancers, web servers, or back-end systems.
- Is sensitive data stored in clear text, including backups?
- Are any old or weak cryptographic algorithms used either by default or in older code?
- Are default crypto keys in use, weak crypto keys generated or re-used, or is proper key management or rotation missing?
- Is encryption not enforced, e.g. are any user agent (browser) security directives or headers missing?
- Does the user agent (e.g. app, mail client) not verify if the received server certificate is valid?

See ASVS Crypto (V7), Data Prot (V9) and SSL/TLS (V10)

How to Prevent

Do the following, at a minimum, and consult the references:

- Classify data processed, stored, or transmitted by an application. Identify which data is sensitive according to privacy laws, regulatory requirements, or business needs.
- Apply controls as per the classification.
- Don’t store sensitive data unnecessarily. Discard it as soon as possible or use PCI DSS compliant tokenization or even truncation. Data that is not retained cannot be stolen.
- Make sure to encrypt all sensitive data at rest.
- Ensure up-to-date and strong standard algorithms, protocols, and keys are in place; use proper key management.
- Encrypt all data in transit with secure protocols such as TLS with perfect forward secrecy (PFS) ciphers, cipher prioritization by the server, and secure parameters. Enforce encryption using directives like HTTP Strict Transport Security (HSTS).
- Disable caching for responses that contain sensitive data.
- Store passwords using strong adaptive and salted hashing functions with a work factor (delay factor), such as Argon2, scrypt, bcrypt, or PBKDF2.
- Verify independently the effectiveness of configuration and settings.

Common Defences Against Sensitive Data Exposure

Minimise

sensitive data collection

- You can't lose what you don't have!
- Reduce the window of storage

Apply

HTTPS everywhere

- It's too easy to "insufficiently" implement
- Start with it everywhere – it's easy!

Use

strong crypto storage

- Hashing algorithms designed for passwords
- Be very careful with key management

Sensitive Data Exposure in the Wild – Tunisia

January 30th, 2011 | 18:20 GMT - by Lucian Constantin

Tunisian Gov Is Primary Suspect in Mass Theft of Gmail, Yahoo and Facebook Logins

SHARE: +1 0 Like Share 0 Tweet 17 Adjust text size:

The Tunisian government is suspected of injecting password stealing JavaScript code into the login pages of popular websites via its Internet agency that controls the entire country's Internet gateways.

The Tunisian government is suspected of injecting password stealing JavaScript code into the login pages of popular websites via its Internet agency that controls the entire country's Internet gateways.

All private Internet service providers in Tunisia go out through the infrastructure provided and maintained by the Tunisian Internet Agency (Agence tunisienne d'Internet).

ATI is run by the Ministry of Communications and has the ability to block websites deemed inappropriate by the government. At one time, these included Flickr, YouTube, and Vimeo.

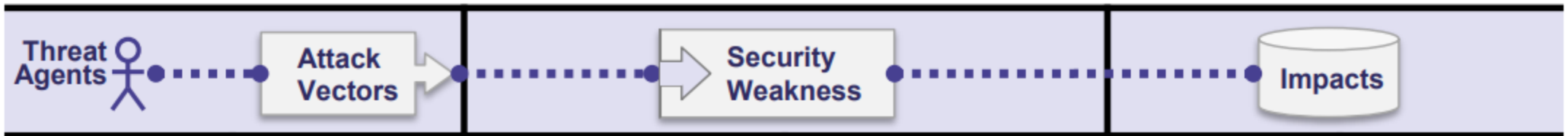
The Tech Herald reports that several security experts have analyzed the source code of Facebook, Yahoo and Gmail as seen in Tunisia and the conclusion is unanimous - there's something surreptitious going on.

The rogue code is customized for each of the websites and its purpose is to hijack login credentials when they are inputted into login forms.

The data is encrypted with a weak algorithm and submitted via GET request to a non-existent URL. For example, Gmail logins are sent to an URL of the form http://www.google.com/wo0dh3ad?q=[five random digits][encrypted username][encrypted

A5 :2017

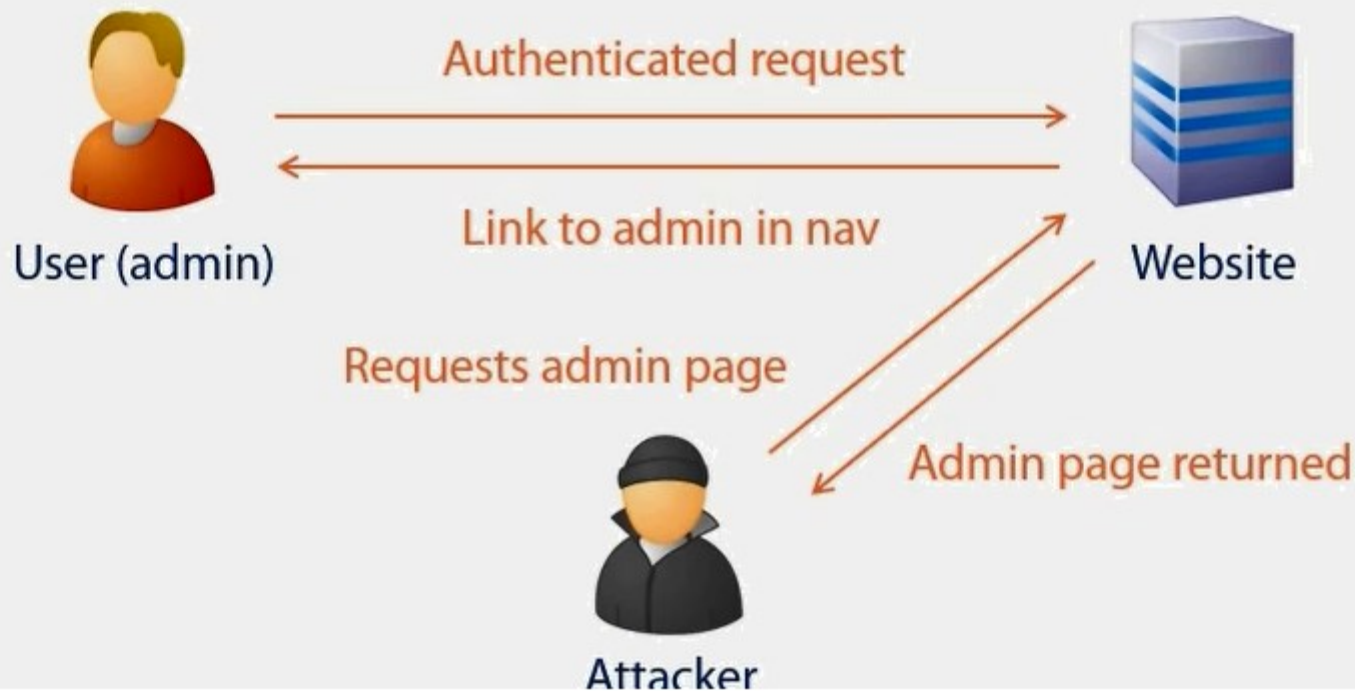
Broken Access Control



App. Specific	Exploitability: 2	Prevalence: 2	Detectability: 2	Technical: 3	Business ?
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. Access control detection is not typically amenable to automated static or dynamic testing. Manual testing is the best way to detect missing or ineffective access control, including HTTP method (GET vs PUT, etc), 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.	

Missing Function Level Access Control

Attack Vectors	Security Weaknesses		Technical Impacts
Exploitability Easy	Prevalence Common	Detectability Average	Impact Moderate



Understanding Missing Function Level Access Control

- Does the UI show navigation to unauthorised functions?
- Are server side authentication or authorisation checks missing?
- Are server side checks done that solely rely on information provided by the attacker?
- Are system or diagnostic resources accessible without proper authorisation?
- Will “forced browsing” disclose unsecured resources?

Common Defences Against Missing Function Level Access Control

Define a clear authorisation model

- Define centrally and consistently
- Use roles and apply membership

Check for forced browsing

- Check for default framework resources
- Automated scanners are excellent for this

Always test unprivileged roles

- Capture and replay privileged requests
- Include POST requests and async calls

Is the Application Vulnerable?

Access control enforces policy such that users cannot act outside of their intended permissions. Failures typically lead to unauthorized information disclosure, modification or destruction of all data, or performing a business function outside of the limits of the user. Common access control vulnerabilities include:

- Bypassing access control checks by modifying the URL, internal application state, or the HTML page, or simply using a custom API attack tool.
- Allowing the primary key to be changed to another users record, permitting viewing or editing someone else's account.
- Elevation of privilege. Acting as a user without being logged in, or acting as an admin when logged in as a user.
- Metadata manipulation, such as replaying or tampering with a JSON Web Token (JWT) access control token or a cookie or hidden field manipulated to elevate privileges, or abusing JWT invalidation
- CORS misconfiguration allows unauthorized API access.
- Force browsing to authenticated pages as an unauthenticated user or to privileged pages as a standard user. Accessing API with missing access controls for POST, PUT and DELETE.

How to Prevent

Access control is only effective if enforced in trusted server-side code or server-less API, where the attacker cannot modify the access control check or metadata.

- With the exception of public resources, deny by default.
- Implement access control mechanisms once and re-use them throughout the application, including minimizing CORS usage.
- Model access controls should enforce record ownership, rather than accepting that the user can create, read, update, or delete any record.
- Unique application business limit requirements should be enforced by domain models.
- Disable web server directory listing and ensure file metadata (e.g. .git) and backup files are not present within web roots.
- Log access control failures, alert admins when appropriate (e.g. repeated failures).
- Rate limit API and controller access to minimize the harm from automated attack tooling.
- JWT tokens should be invalidated on the server after logout.

Developers and QA staff should include functional access control unit and integration tests.

Missing Access Controls in the Wild – Westfield

Westfield iPhone app in privacy fiasco

CHRIS GRIFFITH | THE AUSTRALIAN | SEPTEMBER 15, 2011 2:51PM 18 SAVE

WESTFIELD has temporarily pulled Find My Car from its iPhone app after a security analyst showed he could monitor all cars parked in its Bondi Junction shopping centre.

The retail giant's action follows **a blog by software architect Troy Hunt** who found URLs containing the number plates of all cars at Westfield's Bondi Junction centre were publicly accessible – no hacking was required.

The app lets a shopper enter their number plate and, after choosing a photo of their car from four displayed vehicles, seeks to guide the shopper back to their parking bay.

Sydney-based Hunt was able to develop software that could inform him of when all cars arrived and left the shopping centre, and exactly where they were parked.

Westfield's Find My Car feature made parking data publicly available. *Source:* Supplied

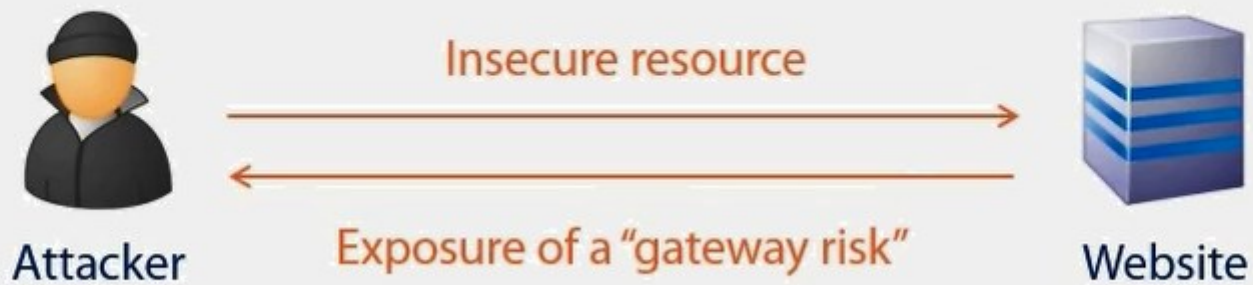
A6 :2017

Security Misconfiguration

App. Specific	Exploitability: 3	Prevalence: 3	Detectability: 3	Technical: 2	Business ?
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.		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.		Such flaws frequently give attackers unauthorized access to some system data or functionality. Occasionally, such flaws result in a complete system compromise. The business impact depends on the protection needs of the application and data.	

Security Misconfiguration Overview

Attack Vectors	Security Weaknesses		Technical Impacts
Exploitability Easy	Prevalence Common	Detectability Easy	Impact Moderate



Understanding Security Misconfiguration

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

Is the Application Vulnerable?

The application might be vulnerable if the application is:

- Missing appropriate security hardening across any part of the application stack, or improperly configured permissions on cloud services.
- Unnecessary features are enabled or installed (e.g. unnecessary ports, services, pages, accounts, or privileges).
- Default accounts and their passwords still enabled and unchanged.
- Error handling reveals stack traces or other overly informative error messages to users.
- For upgraded systems, latest security features are disabled or not configured securely.
- The security settings in the application servers, application frameworks (e.g. Struts, Spring, ASP.NET), libraries, databases, etc. not set to secure values.
- The server does not send security headers or directives or they are not set to secure values.
- The software is out of date or vulnerable (see **A9:2017-Using Components with Known Vulnerabilities**).

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

How to Prevent

Secure installation processes should be implemented, including:

- A repeatable hardening process that makes it fast and easy to deploy another environment that is properly locked down. Development, QA, and production environments should all be configured identically, with different credentials used in each environment. This process should be automated to minimize the effort required to setup a new secure environment.
- A minimal platform without any unnecessary features, components, documentation, and samples. Remove or do not install unused features and frameworks.
- A task to review and update the configurations appropriate to all security notes, updates and patches as part of the patch management process (see **A9:2017-Using Components with Known Vulnerabilities**). In particular, review cloud storage permissions (e.g. S3 bucket permissions).
- A segmented application architecture that provides effective, secure separation between components or tenants, with segmentation, containerization, or cloud security groups.
- Sending security directives to clients, e.g. Security Headers.
- An automated process to verify the effectiveness of the configurations and settings in all environments.

Common Defences Against Security Misconfiguration

Always harden the install

- Turn off features that aren't needed
- Apply the "principle of least privilege"

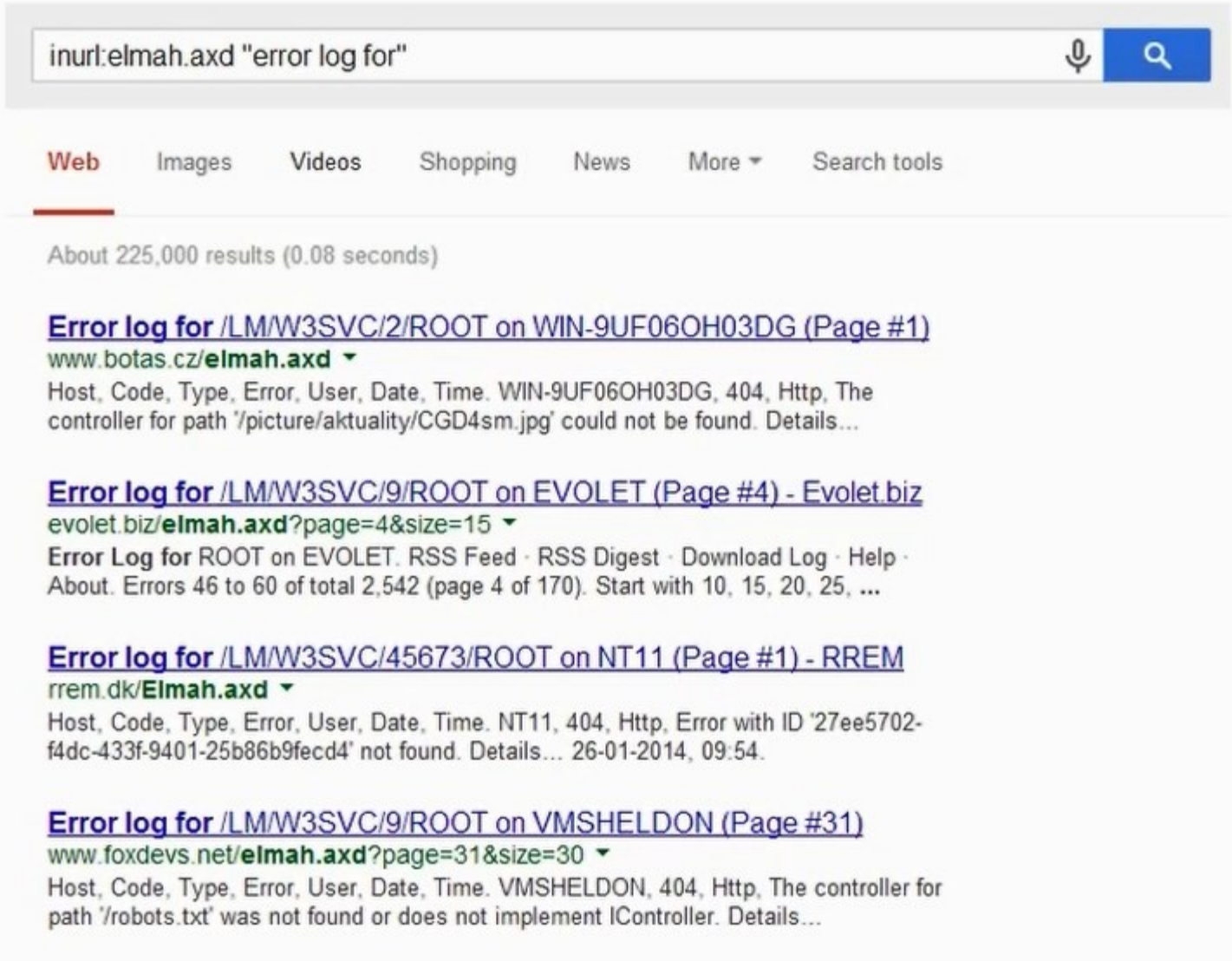
Tune the app security config

- Ensure it's production-ready
- Defaults are often not right

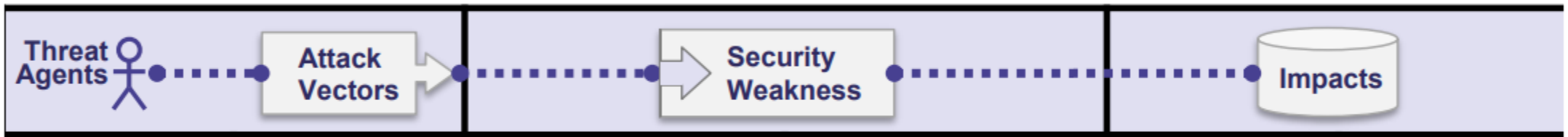
Ensure packages are up to date

- Be conscious of 3rd party tool risks
- Have a strategy to monitor and update

Security Misconfiguration in the Wild – ELMAH



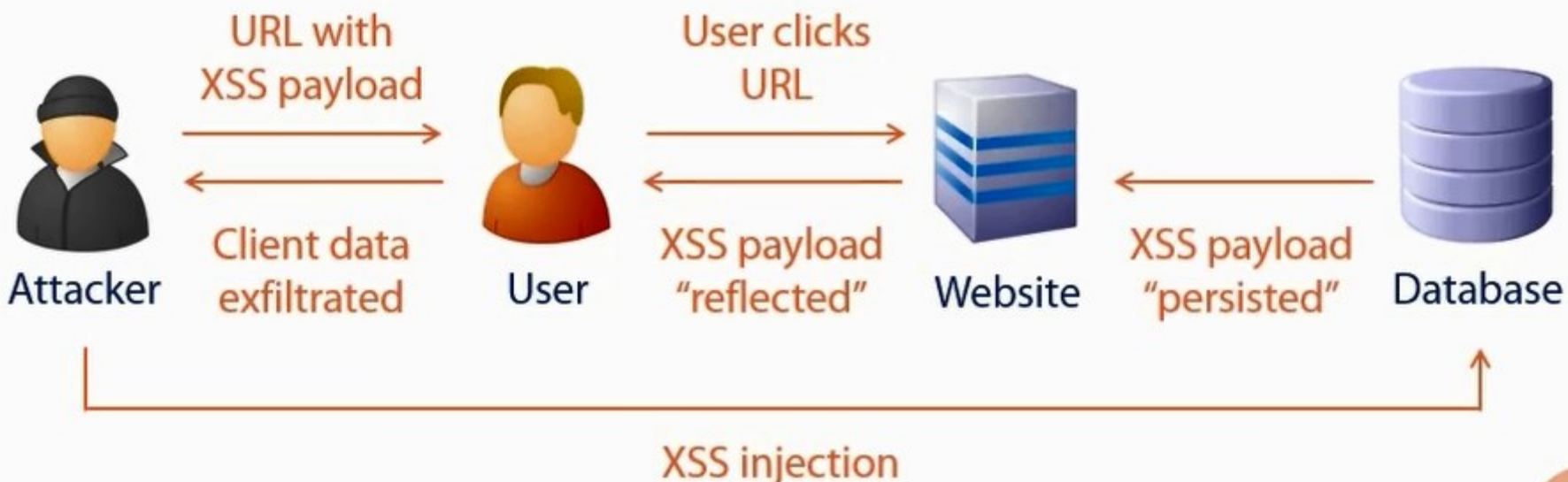
A7 :2017 Cross-Site Scripting (XSS)



App. Specific	Exploitability: 3	Prevalence: 3	Detectability: 3	Technical: 2	Business ?
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.	

XSS Overview

Attack Vectors	Security Weaknesses		Technical Impacts
Exploitability Average	Prevalence Very Widespread	Detectability Easy	Impact Moderate



Understanding XSS


```
<script>alert(document.cookies)</script>
```

Is the Application Vulnerable?

There are three forms of XSS, usually targeting users' browsers:

Reflected XSS: The application or API includes unvalidated and unescaped user input as part of HTML output. A successful attack can allow the attacker to execute arbitrary HTML and JavaScript in the victim's browser. Typically the user will need to interact with some malicious link that points to an attacker-controlled page, such as malicious watering hole websites, advertisements, or similar.

Stored XSS: The application or API stores unsanitized user input that is viewed at a later time by another user or an administrator. Stored XSS is often considered a high or critical risk.

DOM XSS: JavaScript frameworks, single-page applications, and APIs that dynamically include attacker-controllable data to a page are vulnerable to DOM XSS. Ideally, the application would not send attacker-controllable data to unsafe JavaScript APIs.

Typical XSS attacks include session stealing, account takeover, MFA bypass, DOM node replacement or defacement (such as trojan login panels), attacks against the user's browser such as malicious software downloads, key logging, and other client-side attacks.

How to Prevent

Preventing XSS requires separation of untrusted data from active browser content. This can be achieved by:

- Using frameworks that automatically escape XSS by design, such as the latest Ruby on Rails, React JS. Learn the limitations of each framework's XSS protection and appropriately handle the use cases which are not covered.
- Escaping untrusted HTTP request data based on the context in the HTML output (body, attribute, JavaScript, CSS, or URL) will resolve Reflected and Stored XSS vulnerabilities. The OWASP Cheat Sheet 'XSS Prevention' has details on the required data escaping techniques.
- Applying context-sensitive encoding when modifying the browser document on the client side acts against DOM XSS. When this cannot be avoided, similar context sensitive escaping techniques can be applied to browser APIs as described in the OWASP Cheat Sheet 'DOM based XSS Prevention'.
- Enabling a Content Security Policy (CSP) is a defense-in-depth mitigating control against XSS. It is effective if no other vulnerabilities exist that would allow placing malicious code via local file includes (e.g. path traversal overwrites or vulnerable libraries from permitted content delivery networks).

Common Defences Against XSS Attacks

Whitelist
untrusted data

- What input do we *trust?*
- Does it adhere to expected patterns?

Always
encode output

- Never simply reflect untrusted data
- This applies to data in your own DB too

Encode for
context

- HTML / attributes / JavaScript / CSS
- Wrong encoding in wrong context is useless

XSS in the Wild – Samy's MySpace Hack

Samy Kamkar

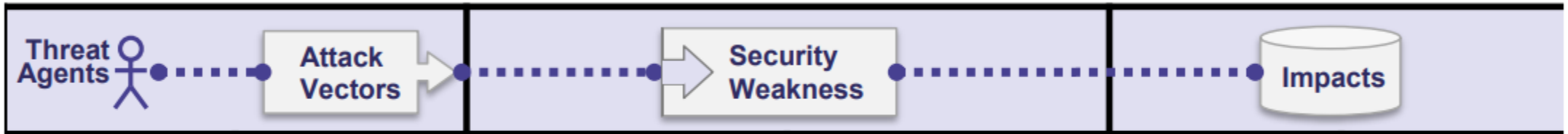
From Wikipedia, the free encyclopedia

Samy Kamkar (born December 10, 1985)[1] is a privacy and security researcher, computer hacker, whistleblower and entrepreneur. At the age of 17, he co-founded Fonality, a unified communications company, which raised over $24 million in private funding.[2] He is possibly best known for creating the Evercookie, which appeared in a top-secret NSA document[3] and the front page of the New York Times,[4] and the MySpace worm Samy (XSS), over which he was subsequently raided by the United States Secret Service for creating and releasing.[5] He is also known for his work with The Wall Street Journal and his discovery of the illicit mobile phone tracking where the Apple iPhone, Google Android and Microsoft Windows Phone mobile devices transmit GPS and Wi-Fi information to their parent companies. His mobile research led to a series of class-action lawsuits against the companies and a privacy hearing on Capitol

In 2005, Kamkar released the Samy worm, the first self-propagating cross-site scripting worm, onto MySpace.[7] The worm carried a payload that would display the string "but most of all, Samy is my hero" on a victim's profile and cause the victim to unknowingly send a friend request to Kamkar. When a user viewed that profile, they would have the payload planted on their page. Within just 20 hours[8] of its October 4, 2005 release, over one million users had run the payload,[9] making Samy the fastest spreading virus of all time.[10] The MySpace team temporarily shut down MySpace to fix the problem that allowed the worm to operate.

In 2006, Kamkar was raided by the United States Secret Service and Electronic Crimes Task Force, expanded from the USA PATRIOT Act, for releasing the worm.[5] Kamkar pled guilty to a felony charge of computer hacking in Los Angeles Superior Court, and was prohibited from using a computer for three years. Since 2008, Kamkar has been doing independent computer security and privacy research and consulting.[11]

A9 :2017 Using Components with Known Vulnerabilities



App. Specific	Exploitability: 2	Prevalence: 3	Detectability: 2	Technical: 2	Business ?
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. Some scanners such as retire.js help in detection, but determining exploitability requires additional effort.		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.	

Using Components with Known Vulnerabilities Overview

Attack Vectors	Security Weaknesses		Technical Impacts
Exploitability Average	Prevalence Widespread	Detectability Difficult	Impact Moderate

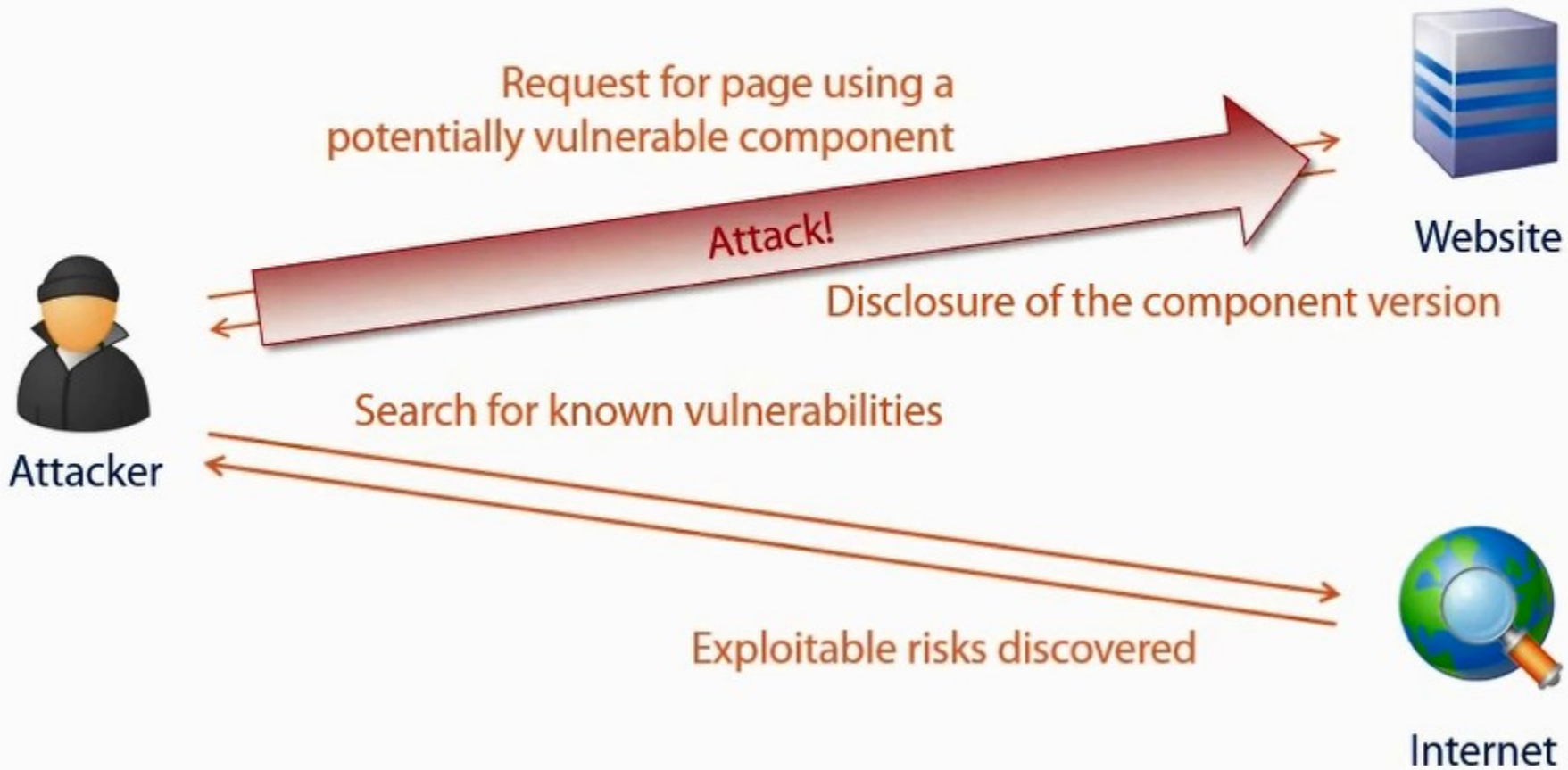
Attacker

Exploitation of vulnerable component

Website

- Circumvent access controls
- Local file inclusion
- SQL injection, XSS or CSRF
- Vulnerable to brute force login

Understanding Components with Known Vulnerabilities



Is the Application Vulnerable?

You are likely vulnerable:

- If you do not know the versions of all components you use (both client-side and server-side). This includes components you directly use as well as nested dependencies.
- If software is vulnerable, unsupported, or out of date. This includes the OS, web/application server, database management system (DBMS), applications, APIs and all components, runtime environments, and libraries.
- If you do not scan for vulnerabilities regularly and subscribe to security bulletins related to the components you use.
- If you do not fix or upgrade the underlying platform, frameworks, and dependencies in a risk-based, timely fashion. This commonly happens in environments when patching is a monthly or quarterly task under change control, which leaves organizations open to many days or months of unnecessary exposure to fixed vulnerabilities.
- If software developers do not test the compatibility of updated, upgraded, or patched libraries.
- If you do not secure the components' configurations (see **A6:2017-Security Misconfiguration**).

How to Prevent

There should be a patch management process in place to:

- Remove unused dependencies, unnecessary features, components, files, and documentation.
- Continuously inventory the versions of both client-side and server-side components (e.g. frameworks, libraries) and their dependencies using tools like versions, DependencyCheck, retire.js, etc. Continuously monitor sources like CVE and NVD for vulnerabilities in the components. Use software composition analysis tools to automate the process. Subscribe to email alerts for security vulnerabilities related to components you use.
- Only obtain components from official sources over secure links. Prefer signed packages to reduce the chance of including a modified, malicious component.
- Monitor for libraries and components that are unmaintained or do not create security patches for older versions. If patching is not possible, consider deploying a virtual patch to monitor, detect, or protect against the discovered issue.

Every organization must ensure that there is an ongoing plan for monitoring, triaging, and applying updates or configuration changes for the lifetime of the application or portfolio.

Common Defences Against Components with Known Vulnerabilities

Identify components and versions

- Components are often used haphazardly
- Keep track of components and versions

Components should be monitored

- Keep abreast of project updates
- Monitor CVEs impacting the components

Keep components updated

- Use the framework's package management
- Regularly monitor new releases

Using Components with Known Vulnerabilities in the Wild – WordPress Brute Force

Massive WordPress Attack Targets Weak Admin Passwords

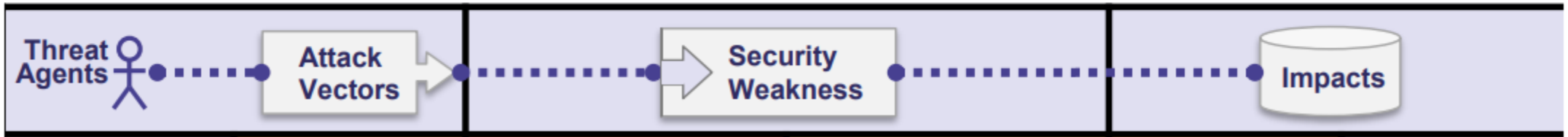
By Scott Gilbertson

If you're using the popular open source blogging tool WordPress to power your website, you may be vulnerable to a new web-based attack.

If your WordPress admin pages suddenly become sluggish, unreachable or you're unable to log in there's a good chance your site is being attacked.

According to CloudFlare CEO Matthew Prince, the attack is using brute force against WordPress' admin pages using the old default username "admin" and then trying thousands of passwords. There's nothing new about that approach, but what makes this attack different, and particularly potent, is that the attackers have some 90,000 unique IP addresses at their disposal.

A10 :2017 Insufficient Logging & Monitoring



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.	

Is the Application Vulnerable?

Insufficient logging, detection, monitoring and active response occurs any time:

- Auditable events, such as logins, failed logins, and high-value transactions are not logged.
- Warnings and errors generate no, inadequate, or unclear log messages.
- Logs of applications and APIs are not monitored for suspicious activity.
- Logs are only stored locally.
- Appropriate alerting thresholds and response escalation processes are not in place or effective.
- Penetration testing and scans by DAST tools (such as OWASP ZAP) do not trigger alerts.
- The application is unable to detect, escalate, or alert for active attacks in real time or near real time.

You are vulnerable to information leakage if you make logging and alerting events visible to a user or an attacker (see **A3:2017-Sensitive Information Exposure**).

How to Prevent

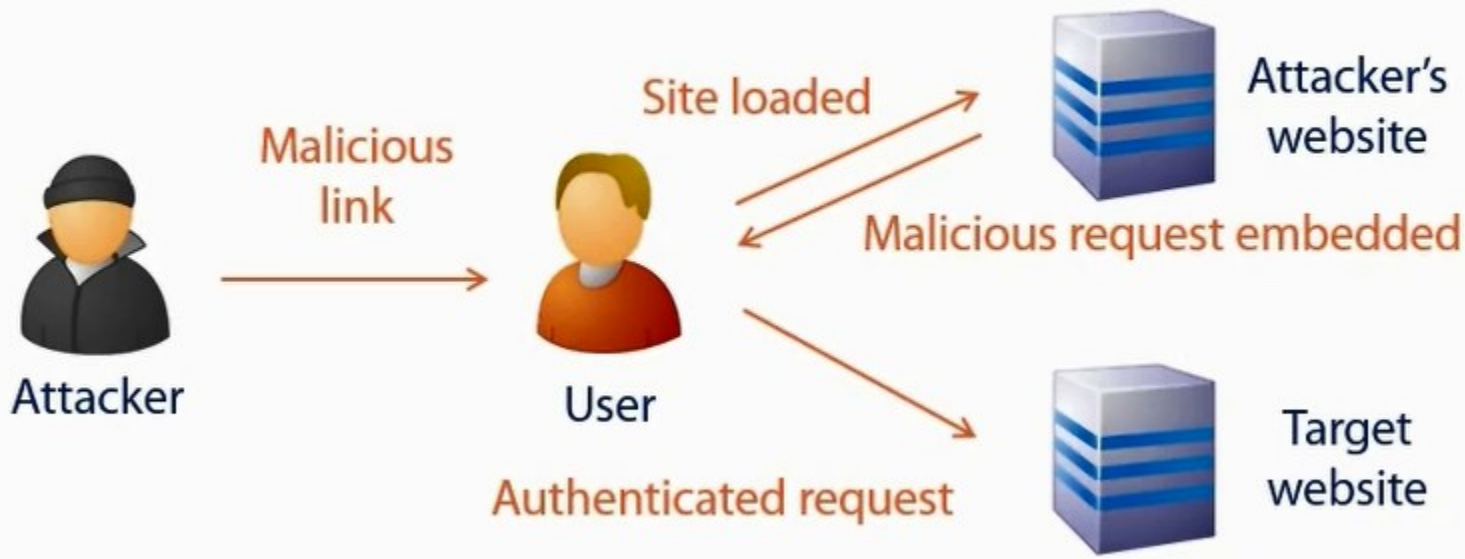
As per the risk of the data stored or processed by the application:

- Ensure all login, access control failures, and server-side input validation failures can be logged with sufficient user context to identify suspicious or malicious accounts, and held for sufficien time to allow delayed forensic analysis.
- Ensure that logs are generated in a format that can be easily consumed by a centralized log management solutions.
- Ensure high-value transactions have an audit trail with integrity controls to prevent tampering or deletion, such as append-only database tables or similar.
- Establish effective monitoring and alerting such that suspicious activities are detected and responded to in a timely fashion.
- Establish or adopt an incident response and recovery plan, such as NIST 800-61 rev 2 or later.

There are commercial and open source application protection frameworks such as OWASP AppSensor, web application firewalls such as ModSecurity with the OWASP ModSecurity Core Rule Set, and log correlation software with custom dashboards and alerting.

CSRF Overview

Attack Vectors	Security Weaknesses		Technical Impacts
Exploitability Average	Prevalence Common	Detectability Easy	Impact Moderate



Understanding CSRF

User

Authenticated request to transfer money online

Website

```
HTTP POST https://mybank.com/transfer

Auth-cookie: 87e090ace8…

TargetAccNum: 8781648823
Amount: 1000000.00
```

Attacker forges request

Common Defences Against CSRF

Employ anti-forgery tokens

- CSRF is exploited by predictable patterns
- Tokens add randomness to the request

Validate the referrer

- Valid requests don't originate externally
- The referrer is in each request header

Other defences

- Native browser defences
- Fraud detection patterns

CSRF in the Wild – Brazilian Modems

How millions of DSL modems were hacked in Brazil, to pay for Rio prostitutes

by Graham Cluley on October 1, 2012 | 15 Comments

FILED UNDER: Featured, Malware, Phishing, Vulnerability

So, you think you're doing a pretty good job in terms of computer security on your home PC? You've kept your computer fully patched against the latest

more than 4.5 million home DSL routers in Brazil were found to have been silently hacked by cybercriminals last year.

Now, how about your router?

My suspicion is that the typical computer user doesn't give a second thought about whether their router could be harbouring a security threat, imagining that the devices don't need to be treated with suspicion.

But if you think that, you're quite wrong.

Fabio Assolini, a researcher for Kaspersky Labs, gave a fascinating presentation at the Virus Bulletin conference in Dallas last week, describing how more than 4.5 million home DSL routers in Brazil were found to have been silently hacked by cybercriminals last year.