Summary of Findings

In performing a detailed penetration testing study against team one's NISMPHP web application portal, team four identified several issues of concern. This report provides brief descriptions of each testing category and offers more details where findings were negative.

The Open Web Application Security Project (OWASP) was used to identify vulnerabilities and complicated logic flaws. While this methodology outlines a series of privacy and applications security risks, it is not a compliance methodology with General Data Protection Regulation(GPDR). We used the Common Vulnerability Scoring System (CVSS) framework to measure quantitive scores to reflect the severity of the discovered vulnerabilities. The scores were translated into a qualitative representation (such as low, medium, high, and critical) to assess and prioritise the vulnerability management process.

Below is the table showing details of the identified vulnerabilities established on category and severity of the risk. Following the table below is a detailed breakdown outlining each testing category.

SECURITY RISK	**COUNT**	**VULNERABILITY SEVERITY/SECURITY RISK**		
		HIGH	**MEDIUM**	**LOW**
A1. Injections	2		X	
A2. Broken Authentication				
A3. Cross-Site Scripting (XSS)	2		X	
A4. Sensitive Data Exposure	1	X		
A5. Insecure Deserialization				
A6. Broken Access Control				
A7. Insufficient Logging & Monitoring				
A8. Server-Side Request Forgery (SSRF)				
A9. Known Vulnerabilities	3		X	
A10. Security Misconfiguration	64			X

Table 1: Vulnerability and Security Risk

SECURITY RISK	COUNT	VULNERABILITY SEVERITY/SECURITY RISK		
		HIGH	MEDIUM	LOW
A1. Injections				
A2. Broken Authentication				
A3. Sensitive Data Exposure	1	X		
A4. Insufficient Logging & Monitoring				
A5. Insecure Deserialization				
A6. Security Misconfiguration	64			X
A7. Cross-Site Scripting (XSS)	2		X	
A8. Server-Side Request Forgery (SSRF)				
A9. Known Vulnerabilities	3		X	
A10. Broken Access Control				

Sensitive Data Exposure

The web application uses an insecure web protocol that transmits data in plain text without encryption. Confidentiality, One of the tenets of the CIA(Confidentiality, Integrity, Availability) triad is a set of rules designed to ensure that information access is limited to the authorised subjects. The impact on an individual or an entity resulting from unauthorised access to sensitive information usually determines the risk rating of the data to vary (Wesley Chai, 2021).

Communication is not secure

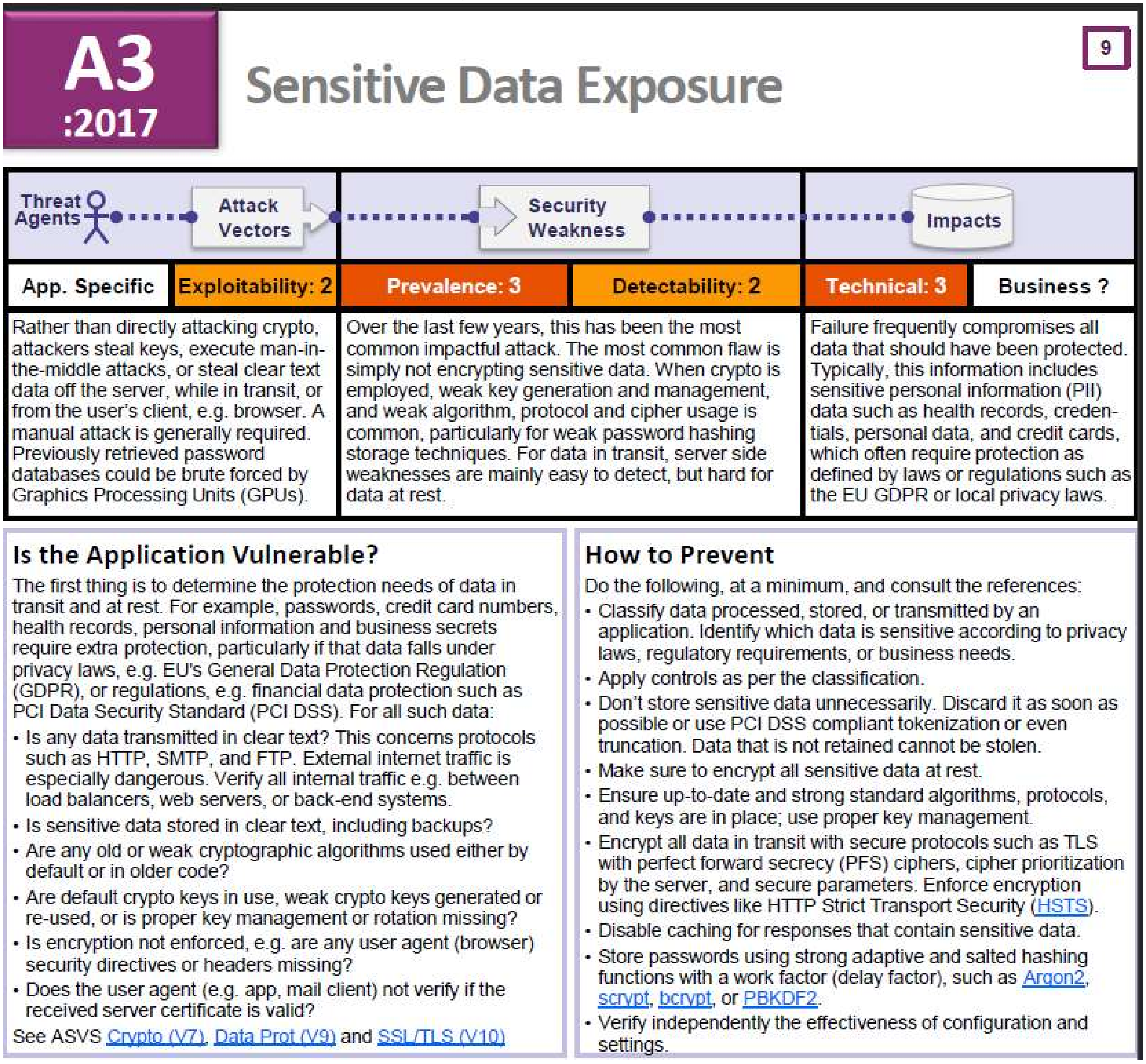
URL	Evidence
http://nismphp-env.eba-2mwmqiam.us-east-1.elasticbeanstalk.com/	Communication is made over unsecure, unencrypted HTTP.

Details

Risk description:
The communication between the web browser and the server is done using the HTTP protocol, which transmits data unencrypted over the network. Thus, an attacker who manages to intercept the communication at the network level, is able to read and modify the data transmitted (including passwords, secret tokens, credit card information and other sensitive data).

Recommendation:
We recommend you to reconfigure the web server to use HTTPS - which encrypts the communication between the web browser and the server.

Figure 1 expands on the finding further to assist with the understanding concerning compliance with standards and regulations.



A3
:2017
9
Threat Agents
Attack Vectors
Security Weakness
Impacts

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.

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

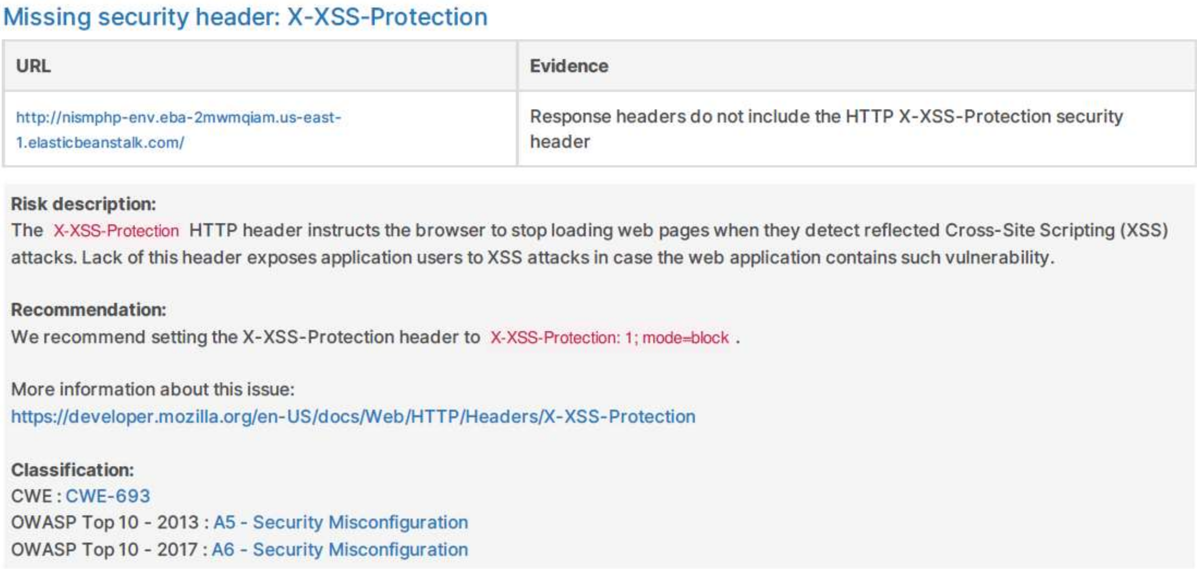
Do the following, at a minimum, and consult the references:

- 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.

Figure 1: OWASP A3 Sensitive Data Exposure (Adapted from (Dehalwar *et al.*, 2018))

Cross-Site Scripting (XSS)

Failing of restricting the sources that are allowed to input data into the web application leads to malicious data from the victim’s web browser included with dynamic content of the browser delivered to the web application. Thus, a cross-site scripting attack is successful. The usage of such attacks at other times results in defaced websites (Rosencrance, 2018).



Missing security header: X-XSS-Protection

URL	Evidence
http://nismphp-env.eba-2mwmqiam.us-east-1.elasticbeanstalk.com/	Response headers do not include the HTTP X-XSS-Protection security header

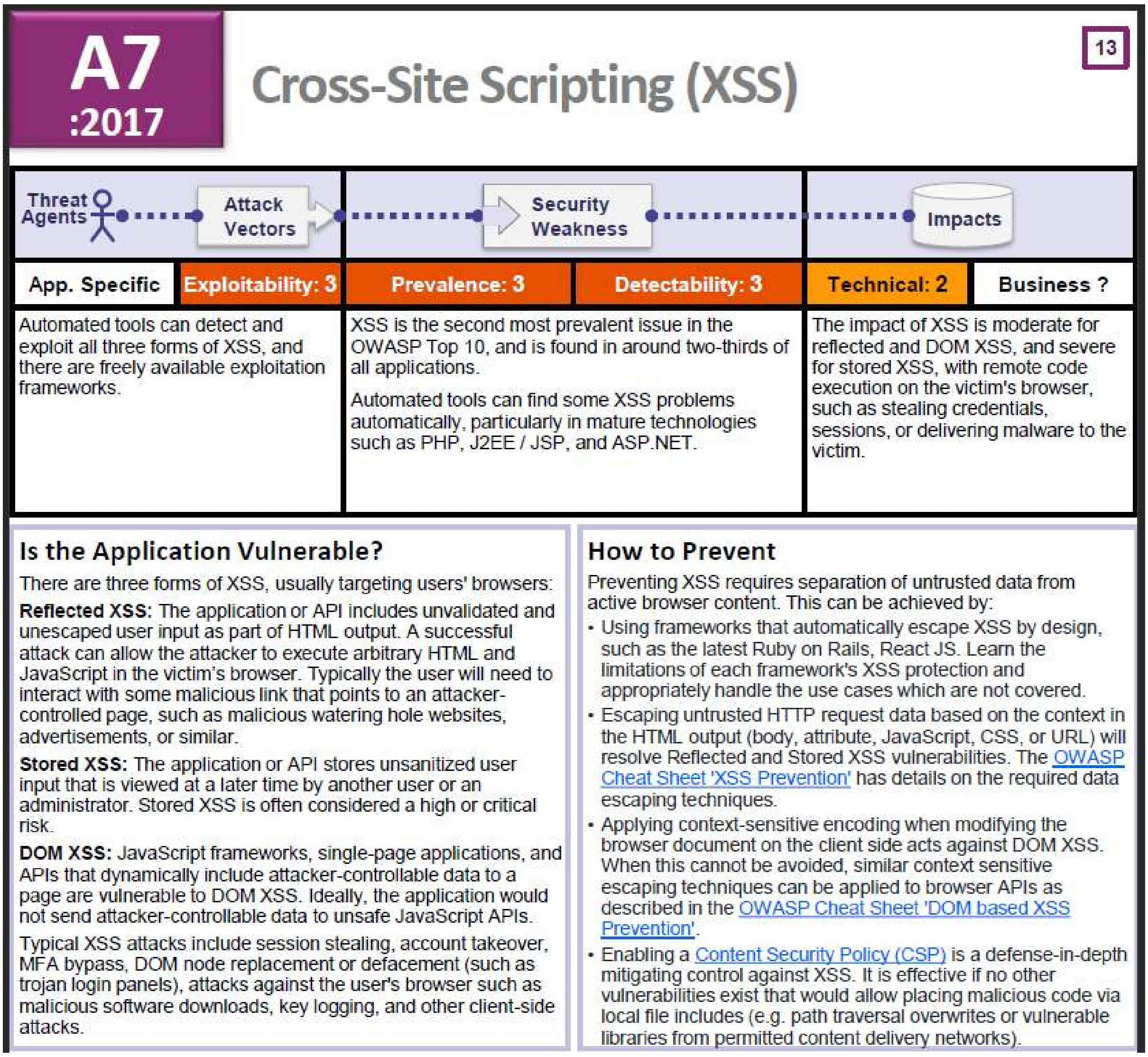
Risk description:
The X-XSS-Protection HTTP header instructs the browser to stop loading web pages when they detect reflected Cross-Site Scripting (XSS) attacks. Lack of this header exposes application users to XSS attacks in case the web application contains such vulnerability.

Recommendation:
We recommend setting the X-XSS-Protection header to X-XSS-Protection: 1; mode=block .

More information about this issue:
https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-XSS-Protection

Classification:
CWE : CWE-693
OWASP Top 10 - 2013 : A5 - Security Misconfiguration
OWASP Top 10 - 2017 : A6 - Security Misconfiguration

Figure 2 illustrates and details methods that malicious actors can exploit Cross-Site Scripting vulnerabilities.



A7 :2017

Cross-Site Scripting (XSS)

13

Threat Agents → Attack Vectors → Security Weakness → Impacts

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.	

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

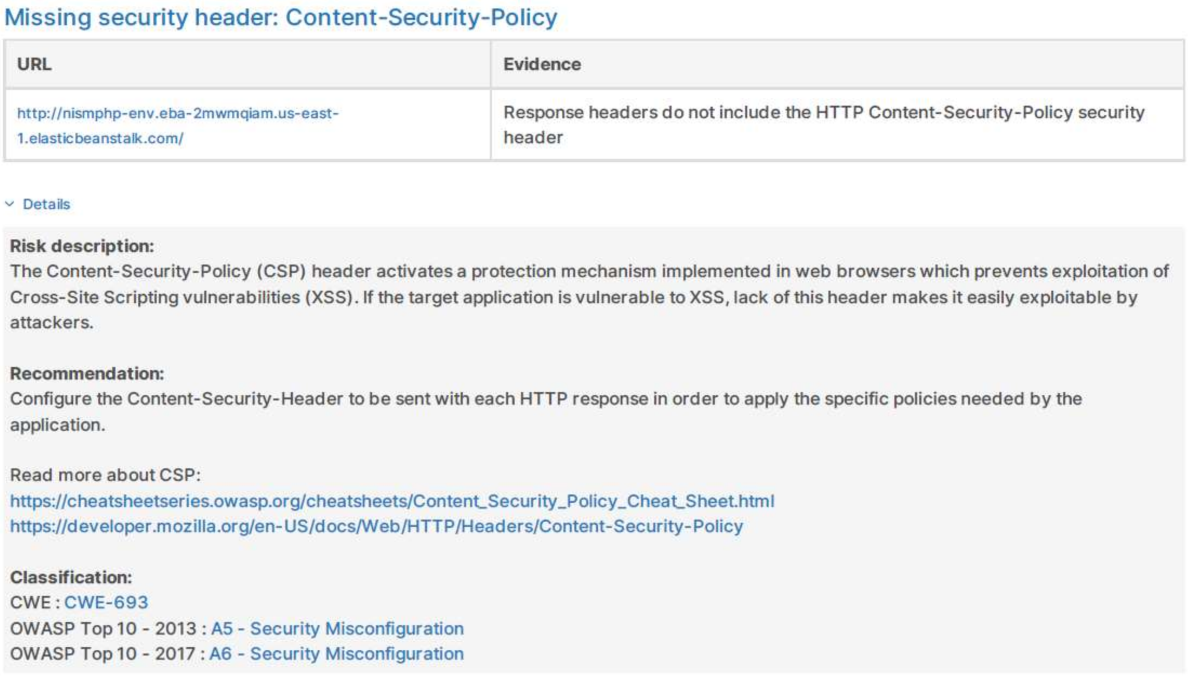
Preventing XSS requires separation of untrusted data from active browser content. This can be achieved by:

- 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).

Figure 2: OWASP A7 Cross-Site Scripting (XSS) (Adapted from (Dehalwar *et al.*, 2018))

Security Misconfiguration

This can include a default account, unpatched or unmaintained server code, references to old versions of services, and so on. Attackers can exploit any security misconfiguration to gain access, elevate privileges, or violate the confidentiality or integrity of the data.



Missing security header: Content-Security-Policy

URL	Evidence
http://nismphp-env.eba-2mwmqiam.us-east-1.elasticbeanstalk.com/	Response headers do not include the HTTP Content-Security-Policy security header

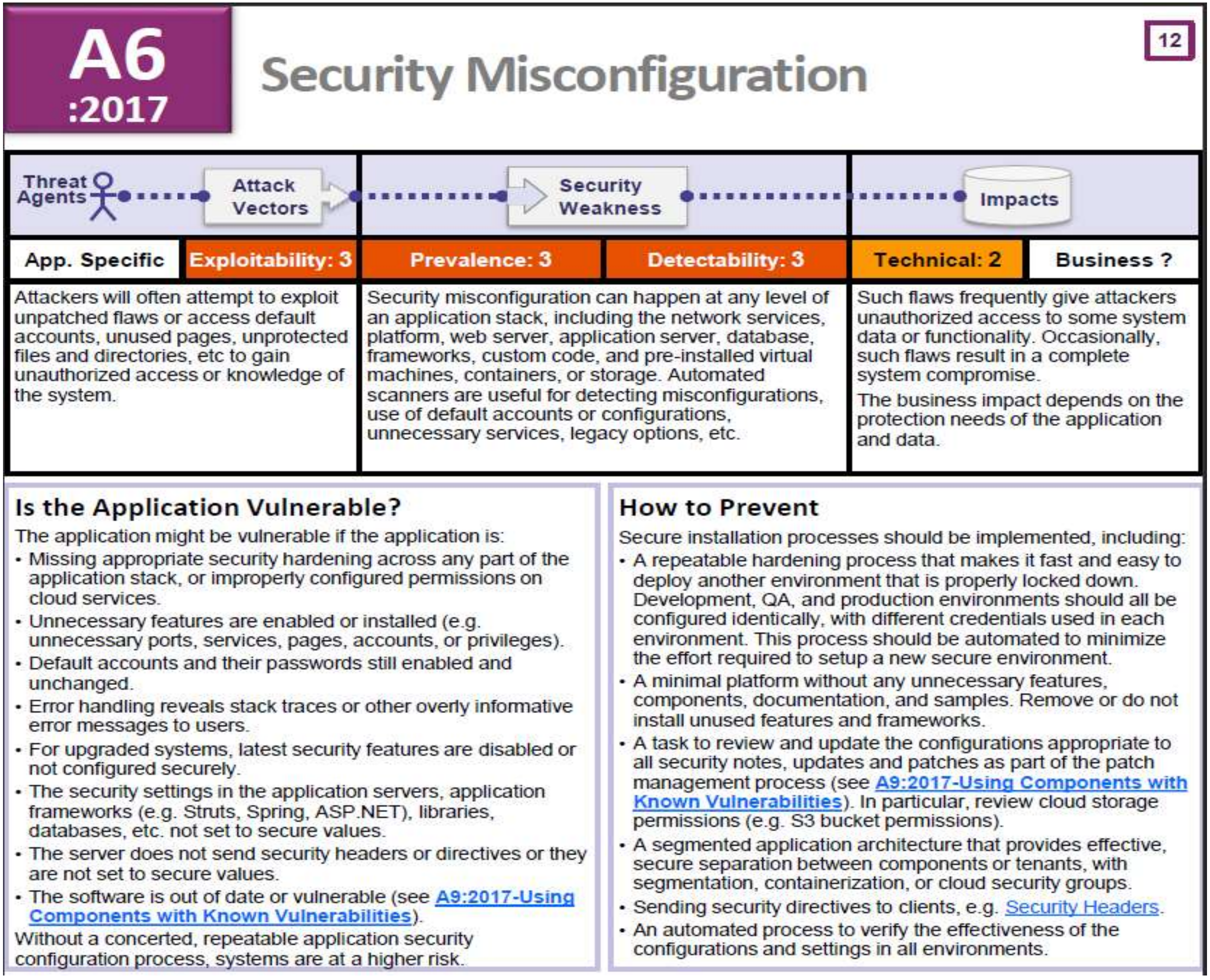
Risk description:
The Content-Security-Policy (CSP) header activates a protection mechanism implemented in web browsers which prevents exploitation of Cross-Site Scripting vulnerabilities (XSS). If the target application is vulnerable to XSS, lack of this header makes it easily exploitable by attackers.

Recommendation:
Configure the Content-Security-Header to be sent with each HTTP response in order to apply the specific policies needed by the application.

Read more about CSP:
https://cheatsheetseries.owasp.org/cheatsheets/Content_Security_Policy_Cheat_Sheet.html
https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Security-Policy

Classification:
CWE : CWE-693
OWASP Top 10 - 2013 : A5 - Security Misconfiguration
OWASP Top 10 - 2017 : A6 - Security Misconfiguration

Figure 3 shows some of the causes of security misconfiguration.



A6 :2017

Security Misconfiguration

12

Threat Agents → Attack Vectors → Security Weakness → Impacts

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.	

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

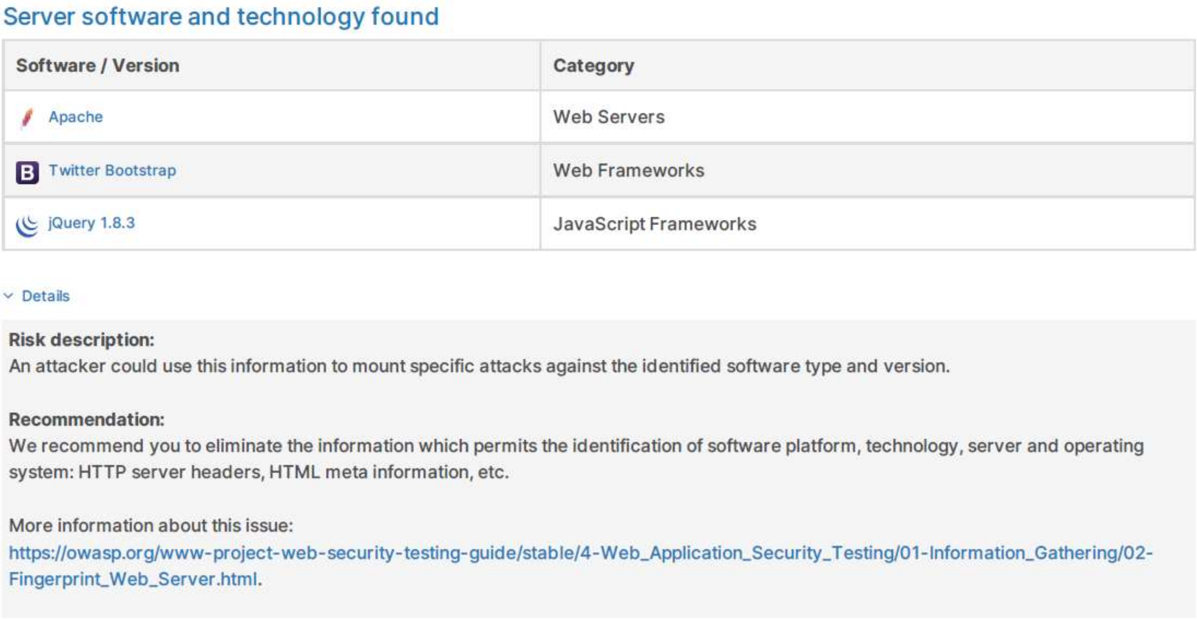
Secure installation processes should be implemented, including:

- 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.

Figure 3: OWASP A6 Security Misconfiguration (Adapted from (Dehalwar *et al.*, 2018))

Using Components with Known Vulnerabilities

(ISC) 2, (2006) states that components, such as libraries, frameworks and other software modules, almost run with full privileges. If a vulnerable component is exploited, such an attack can facilitate severe data loss or server takeover. Applications vising components with known vulnerabilities may undermine application defences and enable a range of possible attacks and impacts.



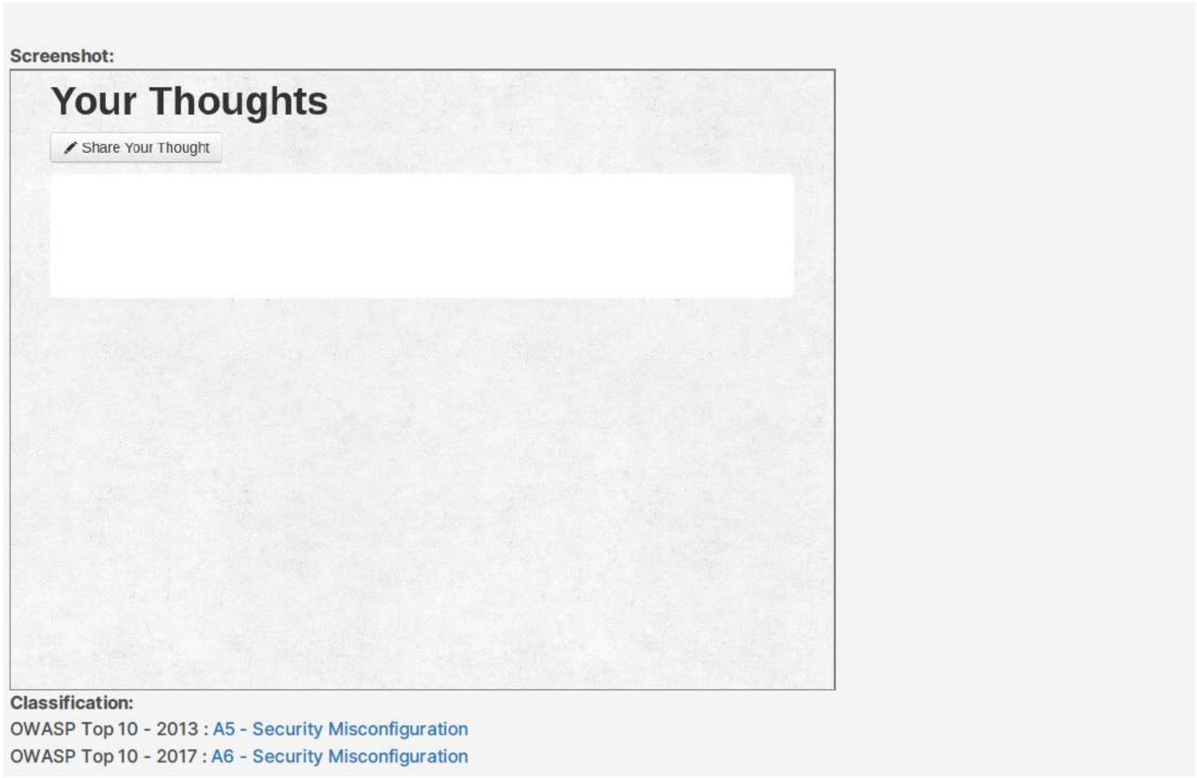
Server software and technology found

Software / Version	Category
Apache	Web Servers
Twitter Bootstrap	Web Frameworks
jQuery 1.8.3	JavaScript Frameworks

Risk description:
An attacker could use this information to mount specific attacks against the identified software type and version.

Recommendation:
We recommend you to eliminate the information which permits the identification of software platform, technology, server and operating system: HTTP server headers, HTML meta information, etc.

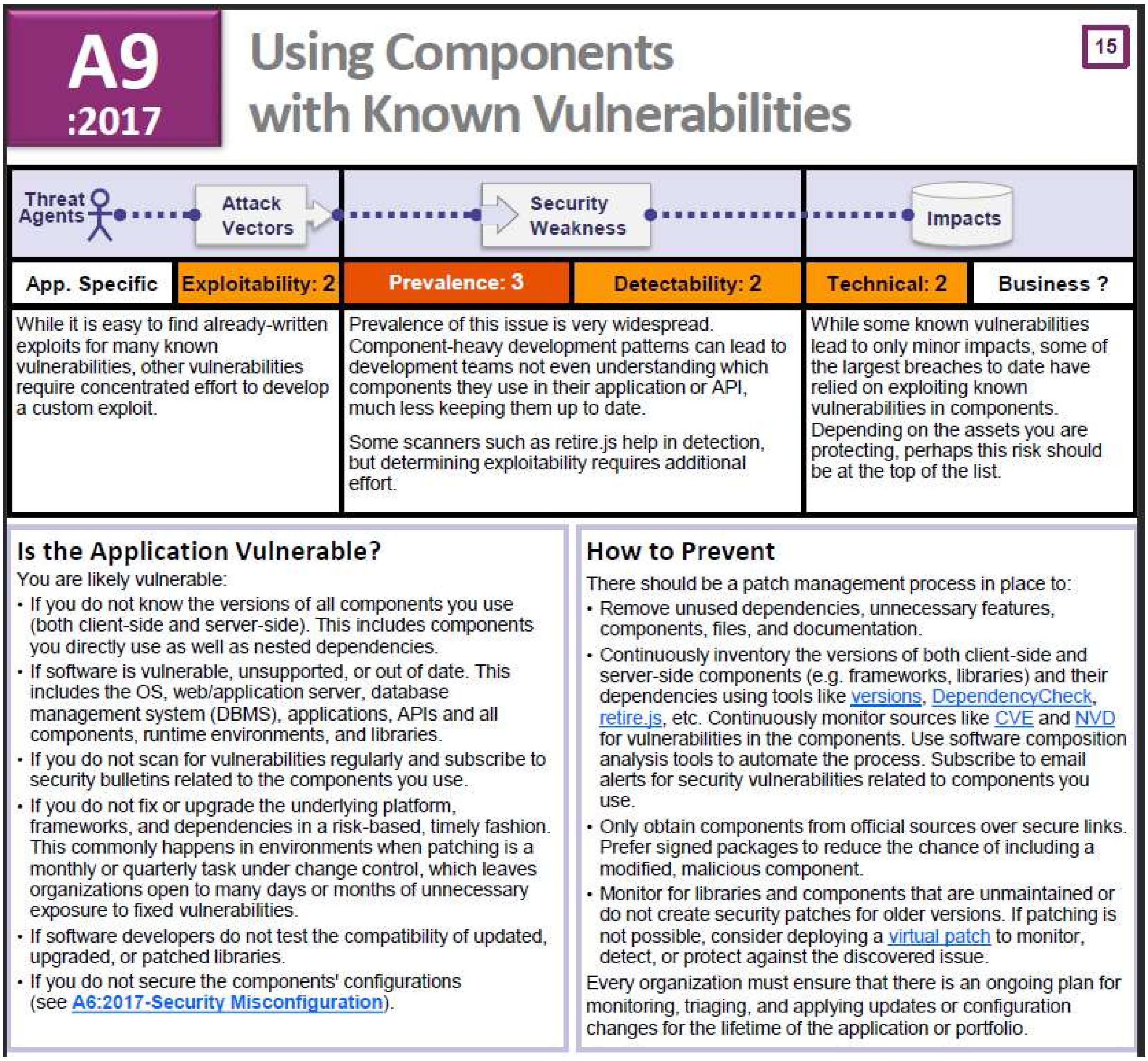
More information about this issue:
https://owasp.org/www-project-web-security-testing-guide/stable/4-Web_Application_Security_Testing/01-Information_Gathering/02-Fingerprint_Web_Server.html.



Screenshot:

Classification:
OWASP Top 10 - 2013 : A5 - Security Misconfiguration
OWASP Top 10 - 2017 : A6 - Security Misconfiguration

Figure 4 illustrates the need for developers to establish a secure coding framework.



A9 :2017 Using Components with Known Vulnerabilities

15

Threat Agents
Attack Vectors
Security Weakness
Impacts

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.	

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.

Figure 4: OWASP A9 Using Components with Known Vulnerabilities. (Adapted from (Dehalwar *et al.*, 2018))

Figure 5 shows the distribution of these categories by amount of security reports, mean bulletins, bug bounties, exploits, altogether:

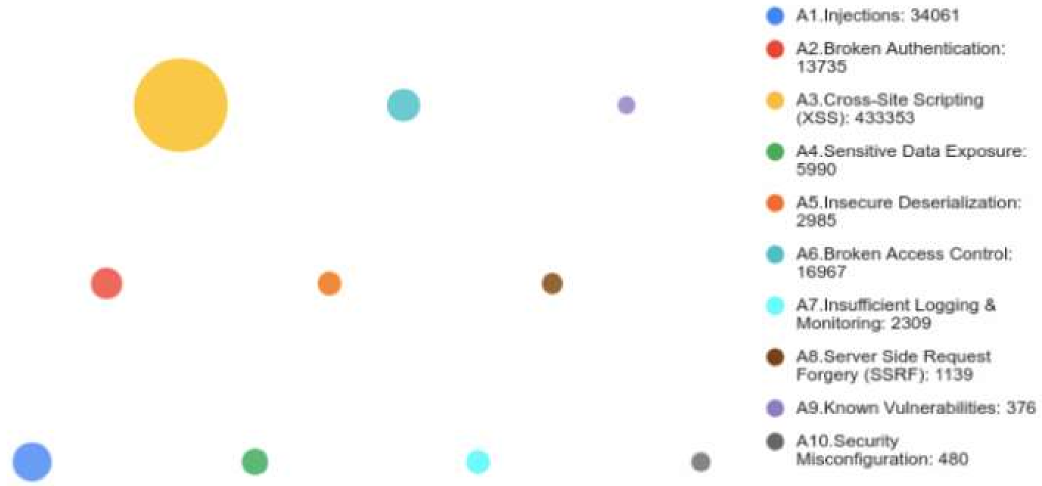


Figure 5: Distribution of OWAS categories. (Adapted from (Wallarm, 2021))

References :

(ISC) 2 (2006) *What is OWASP Top Ten? - Definition from WhatIs.com*. Available at: https://searchsoftwarequality.techtarget.com/definition/OWASP-Top-Ten [Accessed: 15 July 2021].

Dehalwar, V. *et al.* (2018) 'Review of web-based information security threats in smart grid', *2017 7th International Conference on Power Systems, ICPS 2017*, pp. 849–853. doi: 10.1109/ICPES.2017.8387407.

Rosencrance, L. (2018) *What is cross-site scripting (XSS)? - Definition from WhatIs.com*. Available at: https://searchsecurity.techtarget.com/definition/cross-site-scripting [Accessed: 15 July 2021].

Wallarm, I. (2021) *Statistics-Based OWASP Top 10 2021 Proposal - DZone Security*. Available at: https://dzone.com/articles/statistics-based-owasp-top-10-2021-proposal [Accessed: 15 July 2021].

Wesley Chai (2021) *What is the CIA Triad? Definition, Explanation and Examples*. Available at: https://whatis.techtarget.com/definition/Confidentiality-integrity-and-availability-CIA [Accessed: 28 March 2021].