DOI: 10.2507/35th.daaam.proceedings.025

SECURE CODING GUIDELINES AND STANDARDS

Kristina Loncar, Jasmin Redzepagic* & Vedran Dakic

This Publication has to be referred as: Loncar, K[ristina]; Redzepagic, J[asmin] & Dakic, V[edran] (2024). Secure Coding Guidelines and Standards, Proceedings of the 35th DAAAM International Symposium, pp.0191-0198, B. Katalinic (Ed.), Published by DAAAM International, ISBN 978-3-902734-44-0, ISSN 1726-9679, Vienna, Austria
DOI: 10.2507/35th.daaam.proceedings.025

Abstract

This paper delves into fundamental secure coding techniques and guidelines to reduce cybersecurity risks during software development. By analysing past cyberattacks and current security weaknesses, the article underscores the importance of secure coding for maintaining the integrity of information. It details critical concepts from the Open Worldwide Application Security Project (OWASP), focusing on the OWASP Top 10 vulnerabilities list. The article also explores methods to address these vulnerabilities, including issues with access control, encryption errors, and injection flaws. The paper emphasises the necessity of adhering to secure coding protocols from organisations such as OWASP, SEI CERT, and ISO to bolster application security. Additionally, it addresses typical challenges developers encounter, such as inadequate security knowledge and limited resources. It offers strategies to cultivate an active security mindset among development teams, as recorded during the case study.

Keywords: secure coding practices; cybersecurity; OWASP; software vulnerabilities.

1. Introduction

In an era where digital transformation is integral to the success of organisations, the security of software systems has become critical. Secure coding standards and guidelines are essential for creating resilient apps to fend off hostile attacks and safeguard confidential information. These guidelines cover a wide range of recommended practices and regulations to handle the various security flaws that may occur while developing software. Through adherence to these standards, developers may effectively detect and address potential security vulnerabilities, guaranteeing that their software is not only operational but also resistant to external assaults. Secure coding techniques must be used to protect individual programs and the more extensive information technology ecosystem that contemporary businesses depend on.

Financial losses, legal repercussions, and reputational harm are just a few of the severe and far-reaching effects of ignoring secure coding procedures. Rigid adherence to secure coding standards could have prevented many undetected vulnerabilities that lead to data breaches and security incidents. Furthermore, these practices must be implemented to comply with regulatory requirements in several industries, highlighting their crucial significance in operational and legal frameworks. One might logically think that cyberattacks emerged alongside the advent of the first programming languages, computers, and networks. However, this isn't correct. Interestingly, the first instances of what could be considered cyberattacks took place in 19th-century France [1], where individuals used the telegraph system to pilfer financial market data - a concept almost inconceivable today.

Shifting to a more contemporary example, the conversation around cybersecurity was significantly shaped by an event in 1988 when Robert Morris, a Cornell University student, unintentionally unleashed a virus that rapidly spread across over six thousand MIT computers, causing millions of dollars in damages [2]. Despite his lack of malicious intent, Morris was prosecuted under the 1986 Computer Fraud and Abuse Act. Since that time, cybersecurity has become an increasingly critical and relevant field. With the evolution of the Internet and technological advancements, malicious actors have continuously exploited new vulnerabilities within software and tools. Understanding these significant attacks and ways to thwart them is essential in computer studies today.

2. OWASP top security risks

The Open Worldwide Application Security Project (OWASP) is a non-profit organisation with worldwide recognition which seeks to enhance software security. OWASP was founded in 2001 and offers abundant free and open tools, resources, and community-driven projects to assist corporations, security experts, and developers identify, comprehend, and mitigate web-based applications' most significant security risks. The group aims to increase software security awareness so that people and businesses can decide what constitutes a real danger to their software [3]. The following sub-chapters will present the most critical web application security risks.

2.1. Broken access control

A key component of information security is Access Control, which controls user access to resources inside an information system. It entails selective restriction of data, applications, and systems access to guarantee that only authorised users can carry out particular tasks [4].

If Access Control were compromised, we would take into account the following scenarios:
- Elevation of privilege or any violation of the Least Privilege principle
- Bypassing Access Control through the modification of the internal application state, HTML page, or URL
- Metadata manipulation, such as tampering with the JWT token.

The Open Web Application Security Project (OWASP) outlines several best practices for preventing and fixing issues related to broken access control. These best practices ensure that systems enforce appropriate permissions and restrict unauthorised access.

Here are some of the key recommendations:
- Least Privilege - Ensure that users are only granted the minimum access levels necessary for their roles. Regularly review and update permissions to ensure they are appropriate.
- Deny by Default - Implement an access control policy that denies all access by default, only allowing access to resources explicitly granted. This minimises the chance of unauthorised access due to misconfigurations or overlooked permissions [5].
- Access Control Mechanisms - Use role-based access control (RBAC) to enforce user roles and states in determining resource access. Where applicable, consider attribute-based access control (ABAC) for more granular control based on user attributes and resource tags.
- Secure Server-Side Enforcement—Ensure that access controls are enforced server-side and not reliant on the client-side. Client-side checks should only be used for user experience enhancements and must be duplicated on the server.
- Audit Logs and Monitoring—Implement comprehensive logging of all access control decisions and regularly monitor access patterns for anomalies. This helps detect and respond to unauthorised access attempts.
- Static and Dynamic Testing - Regularly test access controls statically and dynamically. Automated tools scan for common access control vulnerabilities and conduct periodic security audits. We can create tools that trace the impacts of code changes on the security of a given software and evaluate it [6]
- Segregation of Duties - Separate responsibilities and duties to reduce the risk of fraud or data tampering. This involves defining roles to limit the power of any single individual where sensitive transactions or data access are involved.
- Use of Secure Direct Object References - Ensure that when direct references to internal implementation objects such as files, directories, or database keys are exposed to users, they cannot manipulate them to gain unauthorised access to other objects.
- Token Management - Securely manage authentication tokens, session identifiers, and other credentials. Ensure they are not exposed in URLs, logs, or referrer headers.
- Regular Updates and Patch Management - Keep all systems updated with the latest security patches and updates. This helps mitigate vulnerabilities that could be exploited to bypass access controls.

By implementing these strategies, organisations can strengthen their security posture and reduce the risks of broken access control.

2.2. Cryptographic failures

Vulnerabilities resulting from poor use of cryptography to secure sensitive data are known as cryptographic failures in the OWASP Top Ten. This group of security hazards deals with how cryptographic safeguards might break, exposing or allowing unwanted access to confidential data. The brief overview of OWASP's best practices for preventing and resolving cryptographic failures [7] are:

- Insecure Algorithms and Practices: Cryptographic failures often stem from outdated or insecure algorithms, cypher modes, and hash functions. Additionally, problems may arise from poorly generated initialisation vectors or mismanaged cryptographic keys.
- Lack of Enforcement: Sometimes, encryption protocols are not adequately enforced due to missing directives, which leads to vulnerabilities in securing data.

To safeguard the confidentiality and integrity of data, OWASP advises:

- Robust Encryption: Encrypt all sensitive data, both at rest and in transit, with robust and vetted algorithms, cypher modes, and hash functions. It's also crucial to phase out outdated protocols like FTP and SMTP when transmitting sensitive information. Furthermore, new principles in cryptography might be used to enhance further encryption security - principles like quantum cryptography [8] or the application of fractal geometry in the field of cryptographic security [9]
- Manage Data Lifecycle: Proactively manage the storage and transmission of sensitive data. Disable data caching to prevent unintended data leaks and promptly delete unnecessary data to minimise risk exposure
- Secure Randomness: Ensure that cryptographic processes utilise true randomness. Avoid predictable seeding in cryptographic functions to prevent attackers from exploiting weak randomness in encryption, which could lead to data breaches.

These guidelines emphasise the importance of cryptographic solid practices and the need for a systematic approach to managing and securing data throughout its lifecycle, enhancing overall security measures against potential cryptographic failures.

2.3. Code Injection

Code injections occur when malicious code is introduced and executed in an application without proper security [10]. These vulnerabilities allow attackers to input code into areas where it is not intended to be, exploiting the system. Different types of code injections include:

- SQL Injections: Attackers insert SQL queries into inputs that directly interact with databases, allowing unauthorised access or operations
- Cross-Site Scripting (XSS): Malicious scripts are embedded into web pages through HTML script tags, affecting users who access these pages
- Command Injections: Commands are injected into an application, which then executes these commands at the operating system level
- Lightweight Directory Access Protocol (LDAP) Injections: Input parameters in LDAP queries are manipulated, altering directory services and information.

To effectively counter these injection threats, several defensive measures are recommended:

- Input Validation: Implement rigorous validation of all user inputs to detect and neutralise potentially malicious data. Use whitelisting approaches to only allow inputs that meet specific criteria.
- Use of Special Character Escaping: Escaping special characters in inputs can prevent interpreters from executing unwanted commands or queries.
- Client-side Script Management: Set the HttpOnly flag in cookies to prevent access to cookie data via client-side scripts, reducing the risk of XSS attacks.
- Parameterized Queries: Always use parameterised queries or prepared statements when accessing databases. This technique ensures that input data is treated as data rather than executable code, which helps prevent SQL injections.
- Content Security Policy (CSP): Implement a robust Content Security Policy to restrict the sources of executable scripts and mitigate XSS risks by specifying which scripts are allowed to run and from where.
- Regular Security Audits: Conduct regular security audits and use tools designed to detect injection vulnerabilities in code before production deployment.

Organisations can significantly reduce the risk of injection vulnerabilities by adopting these strategies, protecting their systems and user data from malicious activities.

2.4. Insecure design

Insecure design refers to vulnerabilities originating from fundamental system architecture flaws. It's crucial to understand the distinction between design and implementation: while a system with a robust design might function adequately even if not implemented perfectly, allowing room for future enhancements, a system with inherent design flaws cannot be rectified by flawless implementation, as it lacks essential security controls from the start [11]. Examples of such insecure designs include [12]:

- Storage of Non-Encrypted Sensitive Information: Unencrypted sensitive data exposes it to potential breaches and unauthorised access.
- Logs and Alerts Containing Exploitable Information: Logs that include sensitive data can provide attackers with the information they need to exploit the system further.
- Inadequate Isolation of Roles and Privileges: Poor segregation of duties can lead to excessive access rights, increasing the risk of insider threats or accidental misuse.
- Unrestricted File Uploads: Allowing files to be uploaded without restrictions or proper security checks can lead to the uploading of malicious files, potentially compromising the system.

To mitigate these risks, it is advisable to implement several strategic measures:
- Adopt a Security-by-Design Approach: Integrate security at the earliest design stages and throughout development. This includes using threat modelling to identify and address potential security issues before they become embedded in the system.
- Principle of Least Privilege: Ensure access controls are strictly enforced, granting users the minimum access necessary for their job functions.
- Encrypt Sensitive Data: Always encrypt sensitive data at rest and in transit to protect it from unauthorised access.
- Secure Logging Practices: Logs should not contain sensitive information. Implement proper log management and monitoring to prevent unauthorised access and manipulation.
- Robust File Handling Mechanisms: Implement strict validation and sanitisation processes for file uploads to prevent malicious files from being uploaded to the system.
- Regular Testing: Continuously perform unit and integration tests to identify and resolve vulnerabilities promptly. This proactive approach helps ensure security flaws are caught and remediated before the software reaches production.
- Feature Reduction: Remove any features that are not being used to reduce the attack surface. A streamlined codebase minimises potential entry points for attackers and simplifies security management.
- Adopt Microservices Architecture: Implement microservices to promote effective compartmentalisation of different application functions. This structure can isolate breaches to a single service without compromising the entire system.

Addressing these aspects of insecure design can significantly enhance a system's security posture, reducing vulnerabilities and strengthening its defences against potential attacks.

2.5. Security misconfiguration

Software engineers frequently use frameworks to develop applications [13], which reach their full potential only when their complex configuration files are correctly established and managed. While default configurations offer convenience, they are typically insecure and pose an easy attack target. Examples of security misconfiguration might include [14]:

- Inadequate Security Settings: Security settings on application servers, frameworks, libraries, and databases are not configured to secure values, or they remain unencrypted.
- Outdated Software: Security patches and updates are often disabled by default, making unpatched, outdated software vulnerable to exploits.
- Enabled Unnecessary Features: Unneeded ports, services, default accounts, and permissions are left enabled, increasing the attack surface unnecessarily.

To reduce the likelihood of configuration mismanagement, consider the following measures:
- Reduce Platform Complexity: Streamline platform functionality by removing or uninstalling unused tools and frameworks, which reduces potential vulnerabilities.
- Implement Automated Security Processes: Set up automated systems to regularly check and confirm the appropriateness of configuration settings across all environments, ensuring the latest security updates are consistently applied.
- Tighten Cloud Storage Settings: If using cloud storage, turn off unnecessary permissions and directory browsing to protect against unauthorised access and data breaches.

By diligently managing and securing configuration settings, organisations can significantly reduce the risks associated with security misconfigurations. Adopting these best practices secures applications and enhances overall system resilience against cyber threats. We must always be aware that even a minor and straightforward vulnerability can lead to extensive issues and substantial harm [15].

Addressing vulnerabilities in system architecture, configurations, and operational practices is essential for safeguarding sensitive data. Ignoring these security gaps can have disastrous consequences, enabling malicious actors to steal, modify, or delete critical information. Such breaches could allow attackers to forge identities and impersonate key figures, causing irreparable damage to an organisation's reputation and financial standing. Moreover, these vulnerabilities often remain undetected, particularly when they involve spying on unsecured communication channels or deliberately slowing operations by tampering with configuration files.

The long-term effects of these breaches can go beyond immediate operational and financial issues, potentially leading to regulatory compliance breaches. Such violations could result in substantial fines and a severe loss of client trust, which can be incredibly challenging to restore. In the most severe cases, inadequately designed and secured systems may suffer irreparable damage, leading to complete organisational failure.

One other method that we could employ is indeed using Trusted Execution Environments. The two most prominent technologies are Intel's SGX (Software Guard Extensions) and AMD SEV (Secure Encrypted Virtualization)[16]. It's a two-way relationship - secure coding practices become crucial when integrating hardware security technologies like Intel Software Guard Extensions (SGX) and AMD Secure Encrypted Virtualization (SEV). Intel SGX provides developers with the tools to create secure enclaves that shield code and data from external threats, even if there's a breach in the operating system. Proper secure coding techniques ensure that the data managed within these enclaves is secure, preserving its integrity. Conversely, AMD SEV secures data at the hypervisor level, encrypting virtual machines' memory, which is essential in environments where cloud resources are shared. Secure coding practices enhance the effectiveness of AMD SEV by ensuring that data is protected not only when it is stored but also while it is being transmitted or processed.

Thus, organisations must adopt stringent security measures recommended by frameworks like OWASP. Implementing strategies such as consistent security reviews, integrating security during the design phase, and enforcing rigorous access controls can significantly reduce these risks. By proactively tackling these vulnerabilities, organisations can shield themselves from the direct and collateral damages of cyberattacks, ensuring their long-term resilience and reliability in the digital landscape.

3. Secure Coding Standards

Coding standards consist of guidelines or best practices that help organisations minimise security vulnerabilities and programming errors, enhancing the security posture of their applications. These standards are universally relevant and provide a crucial defence against prevalent cyber threats for applications of any size. Notably, there is a significant educational gap in secure coding practices in academia; for example, among the top 24 undergraduate computer science programs in the US, only one requires a cybersecurity course [17]. This highlights the critical need for formalised secure coding standards in educational curriculums.

The absence of secure coding education underscores the importance of coding standards. These standards bridge the skills gap among developers, facilitating the efficient delivery of secure code with minimal disruptions. Implementing recognised coding standards optimises development time and offers cost savings by allowing developers to focus on enhancing other application features. Moreover, these standards bring a structured discipline to the codebase, helping developers navigate and strengthen code security and organisations comply with legislative and regulatory frameworks.

One of the most essential standards in this domain is the "OWASP Top 10", which identifies the most severe web application security risks. The Open Web Application Security Project (OWASP) also provides the Application Security Verification Standard (ASVS), which outlines methodologies for building, testing, and maintaining secure applications [18]. OWASP further supports developers through tools like the Zed Attack Proxy (ZAP), a free, open-source tool to discover vulnerabilities in web applications.

Beyond OWASP, the Software Engineering Institute (SEI) offers the SEI CERT Coding Standards, which address various programming languages, including C, C++, Java, Android, and iOS [19]. These standards offer detailed guidelines, examples, and explanations to help developers avoid errors such as undefined behaviours and vulnerabilities. OWASP and SEI CERT utilise the Common Weakness Enumeration (CWE) standard, which has been maintained by The MITRE Corporation since 2006. This standard provides a common language for describing software and hardware vulnerabilities [20], facilitating more transparent and more effective communication within the cybersecurity field.

The International Organisation for Standardization (ISO) is another key player, having developed over 24,000 standards across different sectors. The ISO 27000 series, particularly ISO 27034, focuses on information security and offers extensive guidelines on essential security measures such as authentication, encryption, and data validation to enhance application security. While not a direct coding standard, NIST SP 800-53 provides comprehensive security and privacy controls for federal information systems. Part of the more extensive NIST SP 800 series, these guidelines are available freely and can be adapted by organisations according to their specific security needs, unlike the certifiable ISO standards.

Adopting these different coding standards and security frameworks is crucial for developing secure, robust, and compliant software. By adhering to these guidelines, organisations can protect themselves against the dynamic landscape of cyber threats while ensuring their software meets global security standards.

4. Examination of real-world issues with applying security standards

A study was performed on how OWASP best practices are applied in real-time situations to understand better the difficulties and consequences of implementing these security standards. By examining real-world security breach scenarios and evaluating OWASP adherence, the study aims to identify frequent issues and offer suggestions for enhancing the adoption of these practices. Two basic scenarios were studied:

- Insecure Data Transmission in E-Commerce Platforms,
- Poor Authentication Mechanisms in Financial Applications Background

4.1. Scenario 1: Insecure Data Transmission

In the scenario, the e-commerce platform experienced a data breach where attackers intercepted unencrypted data transmitted between the website and its users. The investigation revealed that the platform was not enforcing HTTPS for all communications, contrary to OWASP's recommendation for secure transmission. This lapse in security allowed attackers to exploit the vulnerability and gain unauthorised access to sensitive customer information.

The OWASP Top Ten highlights the importance of encrypting data in transit over HTTPS to guarantee its integrity and secrecy. Despite recommendations, the e-commerce platform's HTTPS migration was challenging. Two of these difficulties were managing outdated systems that were difficult to update and a general lack of knowledge about the dangers of sending unencrypted data. In addition, the vulnerability was made worse by the platform's incorrect SSL/TLS setups and expired certificates, which exposed data to possible interception.

4.2. Scenario 2: Poor Authentication Mechanisms

Robust authentication procedures are necessary to confirm user identities and stop illegal access to private data. Effective authentication protects against cyber threats by limiting authorised users' access to protected resources. Even if attackers can get passwords through phishing or other methods, compromising user accounts becomes much more challenging when robust authentication measures, such as multi-factor authentication (MFA), are used.

The scenario in the study included a breach in which attackers gained unauthorised access to user accounts by taking advantage of weak authentication methods on a financial application. Because the application did not use multi-factor authentication (MFA) and only used password-based authentication, it was especially susceptible to brute force and credential-stuffing attacks. The attackers eventually compromised many user accounts and gained access to sensitive financial information by using automated tools to systematically test various password combinations and reusing credentials that they had acquired from previous breaches. This hack highlights how crucial it is to implement robust authentication procedures to safeguard user accounts and private information.

5. Analysis of the study

Because financial systems handle large amounts of valuable data, they are often the focus of cyberattacks. Robust authentication procedures are necessary to confirm user identities and stop unwanted access. The analysis of the real-world scenario focused on how people interacted with the security incident, looking at how human factors affected the breach and pinpointing the issues that were found. These problems—explained in more detail below—showcase how important human behaviour, consciousness, and judgment are regarding cybersecurity mishaps.

5.1. The people problem

In many development teams, security is often relegated to secondary importance, perceived as less exciting or immediately rewarding than developing new application features. This attitude can lead to a reactive security posture [21] spurred by miscommunication between developers and upper management. Even in teams that recognise the importance of security, non-technical stakeholders can pressure teams to prioritise short-term gains over long-term security, encouraging a culture resistant to adopting stringent security practices.

5.2. Inconsistency

Security practices must be integrated at every phase of the software development lifecycle—design, development, testing, deployment, and maintenance. Establishing a Secure Software Development Lifecycle (SSDLC) or implementing automated security scans can help enforce consistent and proactive security measures across all projects. Without such systems, security implementation can become sporadic and reactive, undermining the effectiveness of any security protocols.

5.3. Insufficient knowledge

It is difficult to apply security best practices since software engineers frequently lack a solid foundation in the "subject" when they first enter the industry. The situation is exacerbated when the required tools or frameworks are too complex or unwieldy for practical use. This knowledge gap can hinder the effective adoption of necessary security measures, leaving applications vulnerable.

5.4. Lack of resources

Beyond knowledge gaps, a sheer lack of resources can impede the enforcement of security measures. Financial constraints may prevent teams from acquiring the best tools for security implementation. Time constraints are another critical factor; teams may find themselves pressed for time, unable to adequately address security concerns within project timelines, or dealing with large legacy systems where implementing standards would require an impractical overhaul.

Addressing these challenges requires a multifaceted approach, including better education and training in secure coding practices, improved communication between technical and non-technical team members, and adequate resource allocation to security tasks. Organisations must also foster an environment where security is considered integral to the development process, not an optional add-on. By overcoming these obstacles, the tech industry can enhance its defence against the growing threat of cyberattacks, ensuring a safer digital environment for all users.

6. Future work

Future research in secure coding practices and standards offers numerous promising avenues that can significantly enhance cybersecurity measures within the software development community. One crucial area of focus is integrating secure coding education into the academic curriculum at all educational levels. It is essential to educate computer science students who will develop mobile apps to know how to create secure cypher applications [22]. Investigating effective methods to embed these standards from early education to university programs could help cultivate a generation of well-versed developers in cybersecurity principles, thus bridging the current educational gap. Awareness of the many security vulnerabilities and accounting for and handling all error states are crucial steps to writing secure code [23].

Developing automated security tools incorporating artificial intelligence and machine learning could also revolutionise how security threats are predicted, detected, and mitigated. Such advancements would enhance the capabilities of security tools like OWASP ZAP and alleviate the security management burden on developers, allowing them to concentrate more on innovative aspects of software development. Static and dynamic code of malware reverse engineering can also help here, and knowing the architecture of the attacked application might be of great help [24]. Exploring human factors influencing secure coding practices also presents a valuable research direction. This includes studying the psychological and organisational elements that impact the adoption and implementation of security measures. Insights gained could lead to more effective team structures and development processes that inherently promote better security practices. Another promising future research area involves advancing defensive programming techniques. Developing new frameworks and programming languages that integrate security as a core component could establish new benchmarks in software development, making software more secure from the outset. Furthermore, as technologies like IoT, blockchain, and quantum computing evolve, there is a critical need for specialised security standards that address the unique challenges posed by these technologies.

Future research could also explore the synthesis of security standards across various technology domains, such as web development, mobile applications, and cloud computing. This would help create a cohesive security strategy that minimises the inconsistencies in security implementations across different platforms. Additionally, longitudinal studies on the effectiveness of secure coding practices would provide valuable insights into their real-world efficacy and long-term benefits, enabling the refinement of current practices and validating their sustained impact on cybersecurity.

7. Conclusion

Cybersecurity is a critical issue in the technology sector, highlighting the need for rigorous security measures in all aspects of software development. While the industry recognises its importance, its integration into the computer science curriculum in schools and universities remains limited. Organisations like OWASP are at the forefront of addressing this gap by continuously identifying and evaluating common security threats and developing strategies to mitigate them. These efforts are supported by various security standards that help organisations implement secure practices efficiently and cost-effectively. The study of problems encountered when implementing OWASP best practices illustrates web application security's complex nature and importance. While OWASP offers thorough recommendations for guarding against typical security threats, implementation frequently encounters significant difficulties. However, adopting these best practices ultimately rests with the organisations themselves. It is imperative that the broader tech community, including educational institutions and businesses, understand the significance of cybersecurity and actively participate in creating a more secure digital landscape. As technology continues to evolve, so does the sophistication of cyber threats, making it crucial for us to stay ahead through proactive security practices.

By embracing these security measures and standards, we can combat potential threats and build a more secure future for cyberspace, protecting sensitive data and maintaining users' trust worldwide. This ongoing commitment to cybersecurity is essential for fostering a resilient digital environment that can withstand the challenges of tomorrow.

8. References

[1] https://arcticwolf.com/resources/blog/decade-of-cybercrime/, (2022). A Brief History of Cybercrime, Accessed on: 2023-12-06

[2] https://www.fbi.gov/history/famous-cases/morris-worm, (no date). Morris Worm, Accessed on: 2023-12-06

[3] https://owasp.org/Top10/A00_2021_Introduction/#how-the-categories-are-structured, (2021). OWASP Top 10 - 2021, Accessed on: 2023-12-06

[4] https://owasp.org/Top10/A01_2021-Broken_Access_Control/, (2021). A01:2021 – Broken Access Control, Accessed on: 2023-12-06

[5] https://owasp.org/www-project-proactive-controls/v3/en/c7-enforce-access-controls, (no date). C7: Enforce Access Controls, Accessed on: 2023-12-06

[6] Othmane, L. B., Jamil, A.-M., Abdelkhalek, M. (2019). Identification of the Impacts of Code Changes on the Security of Software, Proceedings of the 2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC), IEEE, ISSN 0730-3157, DOI 10.1109/COMPSAC.2019.10268

[7] https://owasp.org/Top10/A02_2021-Cryptographic_Failures/, (2021). A02:2021 – Cryptographic Failures, Accessed on: 2023-12-06

[8] Moric, Z.; Pinjuh, J. & Kurelic, S. (2021). Securing communications in chatter application. Proceedings of DAAAM, at Vienna, Austria, ISSN 1726-9679, DOI 10.2507/32nd.daaam.proceedings.013

[9] Blahova, M.; Mikulicova, M. & Hromada, M. (2020). Utilization of Fractal Geometry Possibilities for Information Systems Security. Proceedings of DAAAM, at Vienna, Austria, ISSN 1726-9679, DOI 10.2507/31st.daaam.proceedings.085

[10] https://brightsec.com/blog/code-injection/, (2022). Moradov, O.; Code Injection in Brief: Types, Examples, and Mitigation, Accessed on: 2023-12-06

[11] https://owasp.org/Top10/A04_2021-Insecure_Design/, (2021). A04:2021 – Insecure Design, Accessed on: 2023-12-07

[12] https://crashtest-security.com/insecure-design-vulnerability/, (2022). Sengupta, S.; Insecure Design, Accessed on: 2023-12-08

[13] https://www.aquasec.com/cloud-native-academy/supply-chain-security/security-misconfigurations/, (2023). Security Misconfiguration: Types, Examples & Prevention Tips, Accessed on: 2023-12-09

[14] https://owasp.org/Top10/A05_2021-Security_Misconfiguration/, (2021). A05:2021 – Security Misconfiguration, Accessed on: 2023-12-08

[15] Stanciu, A.-M.; Ciocarlie, H. (2023). Analyzing Code Security: Approaches and Tools for Effective Review and Analysis, Proceedings of the 2023 International Conference on Electrical, Computer and Energy Technologies, IEEE, ISBN:979-8-3503-2782-3, DOI 10.1109/icecet58911.2023.10389326

[16] Vukasovic, M.; Miladinovic, D.; Milakovic, A.; Vuletic, P.; Stanisavljevic, Z. (2022). Programming Applications Suitable for Secure Multiparty Computation Based on Trusted Execution Environments, Proceedings of the 2022 30th Telecommunications Forum (TELFOR), IEEE, DOI 10.1109/TELFOR56187.2022.9983726

[17] https://hbr.org/2019/08/every-computer-science-degree-should-require-a-course-in-cybersecurity, (2019). Cable, J.; Every Computer Science Degree Should Require a Course in Cybersecurity, Accessed on: 2023-12-13

[18] https://owasp.org/www-project-application-security-verification-standard/, (2022). OWASP Application Security Verification Standard, Accessed on: 2023-12-13

[19] https://www.linkedin.com/advice/0/what-most-effective-secure-coding-standards#cert-secure-coding-standards, (2023). What are the most effective secure coding standards for system development?, Accessed on: 2023-12-13

[20] https://cwe.mitre.org/about/index.html, (no date). About CWE, Accessed on: 2023-12-13

[21] https://snyk.io/learn/secure-coding-standards/, (no date). The Three Pillars for Implementing Secure Coding Standards, Accessed on: 2023-12-13

[22] Xu, J.; Yuan, X. (2015). Developing a course module for teaching cryptography programming on Android, Proceedings of the 2015 IEEE Frontiers in Education Conference (FIE), IEEE, DOI 10.1109/FIE.2015.7344086

[23] Yu, H.; Jones, N.; Bullock, G.; Yuan, X. (2011). Teaching secure software engineering: Writing secure code, Proceedings of the 2011 7th Central and Eastern European Software Engineering Conference (CEE-SECR), IEEE, ISBN:978-1-4799-8454-1, DOI 10.1109/CEE-SECR.2011.6188473

[24] Moric, Z.; Branstett, L.; Petrunic, R. (2022). Static-Analysis Techniques of Malware Reverse Engineering, Proceedings of DAAAM, Vienna, Austria, ISSN 1726-9679, DOI 10.2507/333rd.daaam.proceedings.024