STM JOURNALS

International Journal of
Information Security Engineering

ISSN: 3049-1150
Volume 3, Issue 1, 2025
January–June
DOI (Journal): 10.37591/IJISE

https://journals.stmjournals.com/ijise

Review IJISE

How Beagle View of OWASP Top-10 Can Restrict the Optimal Web-App Security: The Security Ladder for Optimal Posture

Ashish Kumawat[1,*], Kiran Gunturu[2], Aditya Sharad Bhosale[3]

Abstract

Is OWASP Top-10 good enough to ensure that your web application is secure enough? When it comes to application security, many of us in the information security community, are generally overwhelmed with the OWASP Top-10. Even when we deliberate with developers, OWASP Top-10 is on top of our agenda, and surprisingly the only agenda in many discussions. The approach adopted for this paper is two-pronged: ***Secondary data*** *– Open source, literature review;* ***Primary data*** *– Proof of concept for those vulnerabilities which are not covered by OWASP Top-10. As part of our study, we have demonstrated how OWASP Top-10 can cover a variety of vulnerabilities while there can be vulnerabilities which remain uncovered in OWASP Top-10. Our findings suggest that remaining focused just on one framework may not be an optimal strategy. To overcome this bias, we have suggested a security ladder methodology which can suit a variety of organizations and can cater to an optimal web app security posture.*

Keywords: OWASP Top-10, threat modeling, web-application security, cybersecurity, security ladder

INTRODUCTION

The era we live in is marked with complex interactions of Riggs' prismatic model, Methodological individualism and misuse of the internet. It reveals a complex landscape of social behavior, governance, and technology.

Riggs' prismatic model, which describes societies that exhibit characteristics of both traditional and modern structures, can be observed within internet governance and digital interactions. In prismatic societies, roles, norms, and structures are often inconsistent or overlapping, producing administrative inefficiencies and societal challenges [1]. This model is especially relevant to digital spaces, where outdated setups coexist with emerging tech trends. Such prismatic characteristics become apparent as what ideally is expected out of certain developments gets tormented due to rapid technological advancements and existing cyber-behaviors [2].

In the case of internet abuse, when users act in self-interest and affect the online environment, methodological individualism provides insight into how individual behaviors affect larger societal effects. This exchange demonstrates how hard it is for traditional civilizations to adjust to changing digital behavior, which leaves cybersecurity

***Author for Correspondence**
Ashish Kumawat
E-mail: kumawatashish593@gmail.com

[1]Senior Security Consultant, Cybersecurity Professional, qSEAp Infotech Pvt. Ltd., Millenium Business Park, Mahape, Navi Mumbai, Maharashtra, India
[2]Associate Security Consultant, Cybersecurity Professional, qSEAp Infotech Pvt. Ltd., Millenium Business Park, Mahape, Navi Mumbai, Maharashtra, India
[3]Security Consultant, Cybersecurity Professional, qSEAp Infotech Pvt. Ltd., Millenium Business Park, Mahape, Navi Mumbai, Maharashtra, India

Received Date: January 22, 2025
Accepted Date: January 24, 2025
Published Date: February 12, 2025

Citation: Ashish Kumawat, Kiran Gunturu, Aditya Sharad Bhosale. How Beagle View of OWASP Top-10 Can Restrict the Optimal Web-App Security: The Security Ladder for Optimal Posture. International Journal of Information Security Engineering. 2025; 3(1): 19–39p.

regimes vulnerable to abuse [3]. According to this method, the individual behaviors of internet users are the cause of social effects including the frequency of cyberbullying, disinformation, and digital fraud [4]. People who abuse the internet, whether by disseminating misleading information or launching cyberattacks, frequently do so out of a sense of self-interest (Figure 1).

Before we delve into things further, let us now understand the latest trends in web application security first:

Zero Trust Architecture

The conventional notion of a reliable network perimeter is called into question by this method. Because threats can come from both internal and external sources, zero trust architecture (ZTA) stresses that all access requests, regardless of where they come from, must be verified and approved. This model requires granular access controls, continuous verification, and assumes that no user or device should be trusted by default, even if it is within the network. By enforcing policies that include multi-factor authentication (MFA), least-privilege access, and network segmentation, ZTA significantly mitigates the risk of lateral movement by attackers within a network.

Application Programming Interface Security

Application programming interfaces (APIs) have become essential for facilitating service communication as businesses move toward microservices and cloud-native apps [5]. However, because APIs are regularly the target of illegal access and data breaches, increasing reliance has also brought up new security issues [6]. Securing APIs requires implementing strong authentication and authorization mechanisms, such as OAuth 2.0 and JSON Web Tokens (JWT), to ensure that only legitimate users can access sensitive endpoints. Input validation to lessen injection threats, rate limiting to stop misuse, and routinely checking for unusual activity are more recommended practices in API security [7]. Though closely related to web-app security, we have kept API security out of scope in the present paper so as to ensure the specificity of the paper.

DevSecOps

Instead of handling security as an afterthought, DevSecOps integrates security into every phase of the software development and deployment lifecycle [8]. In order to proactively detect and address vulnerabilities prior to the deployment of applications, this strategy places a strong emphasis on ongoing cooperation between the development, operations, and security teams [9]. Automated testing, code reviews, and vulnerability assessments are made possible by DevSecOps' integration of security into the CI/CD (continuous integration/continuous deployment) pipeline. In addition to lowering the expense of patching vulnerabilities after deployment, this proactive approach encourages developers to prioritize security and cultivates a shared responsibility culture [10].

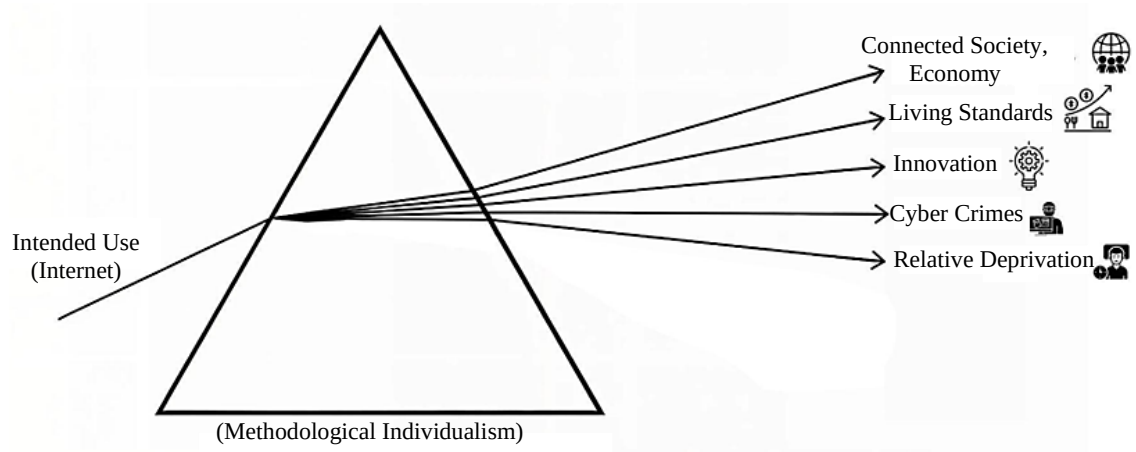

Figure 1. How methodological individualism alters the core intended objectives of internet.

Supply Chain Security

Following high-profile events like the SolarWinds assault, in which hackers used third-party dependencies to breach businesses all around the world, software supply chain security has drawn worldwide attention [11]. This kind of security places a strong emphasis on safeguarding all software elements, including tools, libraries, and dependencies that might create vulnerabilities if exploited. Implementing stringent access restrictions, conducting security checks on outside suppliers, and confirming the integrity of code dependencies are important procedures for safeguarding the software supply chain [12]. Securing the software supply chain is crucial to preventing potential backdoors and guaranteeing the dependability of the software infrastructure, especially as more businesses rely on open-source software and third-party solutions [13].

Artificial Intelligence and Cloud

Modern online application security is being profoundly shaped by artificial intelligence (AI) and cloud technologies, which are bringing cutting-edge approaches to threat detection, mitigation, and prevention. Real-time threat identification and response are made possible by AI-powered security solutions, which analyze enormous volumes of data, see trends, and spot abnormalities more quickly than conventional techniques [14]. By constantly adjusting to new threats, machine learning algorithms improve incident response, fraud detection, and vulnerability screening. AI also makes it easier to automate code reviews, behavior analysis, and testing, which saves human labor and increases the precision of security evaluations [15]. However, cloud computing has revolutionized the deployment and security of online applications. Implementing strong security measures is made easier by cloud providers' integrated security capabilities, which include web application firewalls (WAFs), encryption, and identity management. However, security solutions need to handle issues like shared responsibility models, misconfigurations, and multi-cloud complications as web applications depend more and more on cloud environments. Cloud and AI work together to enable enterprises to implement scalable and proactive security measures, increasing resistance to complex assaults and keeping up with the quickly changing web application landscape [16].

This list can go on but these are some of the important aspects that relate directly with web-app security and were worth discussing. Let us now discuss the evolution of OWASP Top-10, its strengths, and the areas which it does not cover explicitly.

DISCUSSION

The OWASP Top 10 list has evolved significantly since its inception in the early 2000s, becoming a primary resource for web application security. This project was launched in 2003 by the Open Web Application Security Project (OWASP), aiming to address a lack of awareness and resources on web application vulnerabilities. At that time, web applications were proliferating, but security practices were largely absent, leaving applications vulnerable to attacks such as SQL injection, cross-site scripting (XSS), and session hijacking. OWASP’s goal was to create a widely accessible guide that developers, auditors, and organizations could use to identify and prioritize the most critical security issues.

2004–2007: Increasing Industry Adoption

The OWASP Top 10 gained wider recognition in 2004 with the implementation of the Payment Card Industry Data Security Standard (PCI DSS). Organizations handling payment data are expected by the industry to comply to the Top 10 security practices, which were included in this standard as compliance criteria. This integration marked a milestone, as the OWASP Top 10 became not only a guide but a benchmark for secure web practices across the industry. With updates in 2007 and 2010, OWASP adapted the Top 10 list to address emerging security issues related to the growth of e-commerce and web services, reflecting evolving threats like security misconfiguration and insecure direct object references.

2013: Shift Toward Organizational Security Practices

Categories that addressed more general concerns of safe data processing and systemic misconfigurations were included to the 2013 edition of the OWASP Top 10 in addition to particular

vulnerabilities. According to this release, developers should think about safe application design, data protection measures, and the principle of least privilege since organizational practices like security misconfiguration can create vulnerabilities in web applications. In line with the increasing demand for safe coding techniques and organization-wide security policies, these changes demonstrated a more thorough approach to security.

2017: Addressing API and Cloud Security Concerns

The vulnerabilities of increasingly sophisticated online apps, APIs, and cloud connections were highlighted in the 2017 OWASP Top 10 report. New categories like "XML External Entities (XXE)" and "Insufficient Logging and Monitoring" emphasized the need of early detection of harmful activity and visibility into application behavior. The growth of application architectures and the necessity of closer monitoring of third-party libraries and modules were highlighted by these additions, raising security issues that went beyond the program itself. As applications and infrastructures become more sophisticated, the update also highlighted the significance of integrated and ongoing security procedures to reduce risks [2].

2021: Addressing Modern Threats in Cloud and Supply Chain Security

In the 2017 OWASP Top 10 report, the dangers of more intricate web apps, APIs, and cloud interfaces were highlighted. The requirement for insight into application behavior and the capacity to identify harmful activity early on were emphasized by new categories such "Insufficient Logging and Monitoring" and "XML External Entities (XXE)." These enhancements highlighted how application architectures have changed and how third-party libraries and modules require more scrutiny, raising security issues that go beyond the program itself. The significance of integrated and ongoing security procedures to reduce risks as applications and architectures become more sophisticated was also recognized in the upgrade [2].

WHY THE OWASP TOP 10 CONTINUES TO EVOLVE

The OWASP Top 10's constant growth shows a dedication to staying up with the ever-evolving attack vectors, web technologies, and the continuing transition to cloud-native and API-driven apps. In order to move security beyond reactive patches and toward a proactive and integrated strategy, each update aims to offer security council in line with current industry trends and threat environments. Thus, the OWASP Top 10 continues to be an essential tool for directing security best practices to counter contemporary threats in a digital environment that is becoming more complicated by the day.

A common query after seeing the development of OWASP Top 10 is, "Is OWASP Top 10 the only project towards web app sec?" Since a number of projects, frameworks, and standards have evolved concurrently with the OWASP Top 10, the obvious response is "no."

SANS Top 25 Most Dangerous Software Errors

The SANS Top 25 Most Dangerous Software flaws list the most prevalent and significant coding flaws and were developed in partnership with the MITRE Corporation. The SANS Top 25 highlights particular programming faults that might result in vulnerabilities, in contrast to the OWASP Top 10, which concentrates on application-level hazards. Instances include buffer overflows and inadequate input validation, which have been crucial in application security procedures since the list's launch in 2009.

NIST Cybersecurity Framework

The NIST Cybersecurity Framework (CSF), which was created by the National Institute of Standards and Technology (NIST), offers a thorough method for controlling and lowering cybersecurity risk, particularly in critical infrastructure. It contains instructions for the five main tasks of identifying, protecting, detecting, responding, and recovering. Despite having a wider focus than the OWASP Top 10, it fits in nicely with application security frameworks and gives businesses a methodical way to protect their infrastructure, apps, and assets.

Cloud Security Alliance Top Threats to Cloud Computing

The Cloud Security Alliance (CSA) Top Threats report has grown in importance as cloud use has increased, particularly for enterprises implementing cloud-native apps. The CSA focuses on cloud-specific issues that have gained a lot of attention, such account hijacking, unsecured interfaces, and configuration errors. By addressing security threats in cloud settings, where applications are increasingly located, this list supplements the OWASP Top 10 [6].

ISO/IEC 27001 and 27002 Standards

These worldwide standards provide a basis for information security management. Despite not directly addressing application vulnerabilities, they do assist companies in developing and sustaining robust information security management systems (ISMS). These standards are essential to corporate security processes since many of the concepts in the OWASP Top 10 align with ISO 27001's requirements for risk management and safe development.

Building Security in Maturity Model

A framework for evaluating and enhancing an organization's software security maturity is provided by the Building Security in Maturity Model (BSIMM). It offers measures and standards for secure software development and focuses on actual security procedures seen in different enterprises. The BSIMM is more strategic and provides businesses with a maturity model to enhance security procedures at a high level, in contrast to the OWASP Top 10, which targets individual vulnerabilities [9].

NIST 800-53 Security and Privacy Controls for Information Systems and Organizations

NIST created this collection of rules, which offers a comprehensive manual for putting security into practice in a number of areas, including application security. NIST 800-53, which focuses on safeguarding applications at various layers, has harmonized with contemporary application security techniques and shares ideas with the OWASP Top 10. This is especially true of the 2020 version.

Application Security Verification Standard

The Application Security Verification Standard (ASVS), which was developed by OWASP, offers a more thorough framework for application security verification and is a supplement to the OWASP Top 10. Because it provides precise standards for confirming the security of online applications, going beyond OWASP's high-level dangers into concrete, quantifiable criteria for security assessments, it is especially helpful for developers and security auditors.

Center for Internet Security Controls

The Center for Internet Security (CIS) Controls provide a prioritized list of cyber protection measures. Many of the controls, such as safe configuration and continuous vulnerability assessment, concentrate especially on application security, even if they encompass a wide variety of security procedures. OWASP's emphasis on application-level vulnerabilities is complemented by the CIS Controls, which assist businesses in developing a defence-in-depth approach [4].

Having seen the ecosystem in which OWASP Top 10 gained traction, let us now focus on discussing what OWASP Top 10 covers implicitly and what it leaves explicitly:

VULNERABILITIES THAT ARE GETTING COVERED BY OWASP TOP 10 EXPLICITLY/IMPLICITLY

A01:2021 – Broken Access Control

The situation of broken access control can represent itself in a variety of other vulnerabilities. Privilege escalation happens when attackers take advantage of loopholes in access controls to obtain elevated privileges, which allows them to access resources that are forbidden or do unlawful operations. Attackers might alter inputs or file paths to gain unauthorized access to data or folders by exploiting flaws like path traversal and Insecure Direct Object References (IDOR). Systems are further exposed

by the absence of Role-Based Access Control (RBAC) and the uneven application of access restrictions, which do not limit actions according to user roles. Unauthorized operations may result from problems such as incorrect authorization of HTTP methods and manipulating requests to get around access constraints. Attackers can compromise accounts through unrestricted file uploads and inadequate session management, which includes incorrect session expiration or fixation. Furthermore, there is a risk of privilege escalation and the exposure of sensitive data when administrative functions are not sufficiently restricted.

A02:2021 – Cryptographic Failures

The cryptographic failures can lead to insecure data storage, such as failing to encrypt sensitive information like passwords or financial data. It may lead to exposure to misuse if compromised. Weak or broken cryptographic algorithms, outdated methods (e.g., MD5, SHA1), and improper key management, including hardcoded or unrotated keys, increase the risk of unauthorized access. Insecure data transport, like using HTTP instead of HTTPS, and misconfigured TLS/SSL protocols can lead to data interception through man-in-the-middle (MITM) attacks. Exposure of sensitive data often results from insufficient encryption at rest or in transit. Hardcoded secrets, outdated cryptographic libraries, and improper authentication mechanisms further weaken data security, leaving systems vulnerable to modern cryptographic attacks.

A03:2021 – Injection

Implicit or explicit injection attacks, which are addressed in OWASP Top 10, use untrusted data inputs to change commands or queries. Attackers can run harmful SQL code thanks to SQL injection, jeopardizing the integrity of databases. In a similar vein, server breach results from the execution of arbitrary system-level commands made possible via Command and OS Command Injection. By targeting directory services and XML structures, LDAP, XML, and XPath Injection circumvent authentication or expose private information. Unauthorized access is made possible by NoSQL Injection, which takes advantage of databases such as MongoDB's inadequate input validation. By manipulating serialized data, object injection frequently results in remote code execution. Through the use of carriage return and line feed (CRLF) characters, CRLF injection alters HTTP responses, which may result in cache poisoning, HTTP response splitting, or cross-site scripting (XSS).

A04:2021 – Insecure Design

Insecure design encompasses various vulnerabilities stemming from inadequate security practices during application development. A lack of threat modeling in the design phase leads to unidentified risks and potential exploits. Insufficient security controls, such as missing encryption, authentication, or authorization mechanisms, weaken the system's overall defense. Unverified inputs or outputs, when not properly validated, expose applications to injection attacks, XSS, and similar vulnerabilities. The absence of secure coding practices, including improper error handling and weak authentication, further amplifies risks. Business logic errors, stemming from flawed application workflows, also contribute to insecure design, compromising the application's functionality and security.

A05:2021 – Security Misconfiguration

Inadequate system or application configurations that lead to vulnerabilities are the source of security misconfigurations. Systems are vulnerable to brute force attacks when default or weak credentials, including factory-set usernames and passwords, are used. The attack surface is increased by superfluous features, services, or open ports, which give attackers additional points of access. Sensitive information is vulnerable to unwanted access when files, databases, or servers have incorrect permissions. Attackers can obtain important hints from misconfigured error messages that expose application or system information. Security is further compromised by unpatched software with known flaws. Furthermore, allowing debugging or thorough logging in production settings runs the danger of giving hostile actors access to private data and possible vulnerabilities.

A06:2021 – Vulnerable and Outdated Components

When apps depend on out-of-date or unsupported libraries and frameworks with known vulnerabilities, they expose users to serious security concerns. Systems are more vulnerable to vulnerabilities via unpatched or insecure software components when third-party dependencies are not audited and updated. These risks are increased when third-party software with known security flaws is used, which could result in application compromise. Patches, dependency management, and regular updates are necessary.

A07:2021 – Identification and Authentication Failures

When systems fail to appropriately safeguard user credentials or authentication processes, identification and authentication problems occur. Credential stuffing is the practice of gaining access to user accounts through automated attacks utilizing credentials that have been stolen. In order to get around account lockout regulations, password spraying targets numerous accounts with similar passwords. Brute force attacks attempt to guess session keys or passwords in a methodical manner until they succeed. Attackers can take over a session by forcing users to work within a preset session ID, which is known as session fixation. Passwords are additionally vulnerable to compromise when they are stored in a weak manner, such as without the appropriate hashing or encryption. To reduce these threats, strong authentication procedures and safe credential storage must be put in place.

A08:2021 – Software and Data Integrity Failures

Software and data integrity failures arise from the inability to ensure the trustworthiness of code, updates, or transmitted data. Insecure code integrity allows attackers to modify code or software updates, leading to compromised functionality. Vulnerabilities in managing software updates, such as using unverified or compromised versions, further expose systems to threats. Additionally, insecure data transmission, where data integrity is not verified, permits attackers to intercept or tamper with sensitive information. Ensuring secure code validation, update mechanisms, and data transmission processes is critical to maintaining software and data integrity.

A09:2021 – Security Logging and Monitoring Failures

Systems that lack the tools to identify, address, or evaluate security issues experience security logging and monitoring failures. It is more difficult to detect attacks when event logging is not there since it is unable to record important security events such as unsuccessful login attempts or illegal access. Malicious activities go undetected due to inadequate monitoring and a failure to identify suspicious activity. Systems are additionally exposed to unsolved threats by inadequate incident response and warning procedures. Furthermore, forensic analysis is hampered by inadequate logging data, which makes it difficult to conduct efficient breach investigations. For complete security, it is imperative to put in place reliable logging, monitoring, and response systems.

A10:2021 Server-Side Request Forgery

Vulnerabilities where a server handles unauthorized requests to unexpected locations are exploited via Server-Side Request Forgery (SSRF). When attackers employ SSRF to get access to internal services or data, internal resource exposure takes place. Additionally, it can get beyond security measures like firewalls and proxies, enabling illegal communication. In extreme situations, SSRF gives attackers access over the server by taking advantage of weak internal services to enable remote code execution. Furthermore, SSRF can cause data leaks, which raises the possibility of data breaches by disclosing private corporate information. Strict access restrictions and strong validation are necessary to reduce the danger of SSRF.

VULNERABILITIES THAT ARE NOT COVERED BY OWASP TOP 10

Denial of Service Attack

A denial of service (DoS) attack is a malicious attack to interrupt the regular operations of a server, service or network by sending a large number of illegitimate requests, causing it to crash and become unresponsive.

There are different variations of DoS attacks, such as

- *Volume-based attacks:* flooding the target with a significant amount of traffic.
- *Protocol attacks:* exploit weaknesses in network protocols to consume server resources and a lot more.

Some of the attacks are

1. *ReDoS (Regular Expression Denial of Service):* Exploits vulnerable regex patterns that cause the server to consume excessive CPU or memory when processing crafted inputs.
2. *Long String DoS Attack:* Sending a very long length string which involves operations like hashing that overloads the server during such operations (Figure 2).

To demonstrate it, we captured a login request and replaced the password with a very long length string that we can verify with the content length.

The response for the crafted request was a 502 Bad Gateway which proven that the server could not handle that request. It was because of converting that long length password to hash which requires more computing power (Figure 3).

By sending a lot of such requests the server was made unresponsive. However, as per owasp.org, DDos was mentioned among A11:2021 – Next Steps. Let us explore another important vulnerability not getting explicitly covered by OWASP Top 10.

Race Condition

Race condition is a vulnerability that occurs when the server or application processes the requests concurrently without proper safeguards. This allows unexpected results to be exploited by malicious entities.

Race conditions often occur in parallel processes. Moreover, the application itself can be vulnerable if it leverages multi-threaded processing.

The race condition vulnerability can lead to several types of cyber-attacks including but not limited to privilege escalation, business logic issues, data corruption, DoS, Information leakage, and rate limiting bypass (Table 1).

To demonstrate how race condition vulnerability can be exploited, we consider Portswigger's Web Security Academy.

It demonstrates an e-commerce website which sells different products. A wallet called Store Credit was associated with each account. Users could buy things in the e-commerce website using their store credit.

Table 1. Some real-life CVEs (common vulnerabilities and exposures) of race condition vulnerabilities.

Meltdown (CVE-2017-5754)
TIBCO (CVE-2018-18808)
Metinfo (CVE-2018-18808)
Wind River (CVE-2019-12263)
Juniper (CVE-2020-1667)
Windows OLE (CVE-2023-29325)
GpuService (CVE-2023-40131)

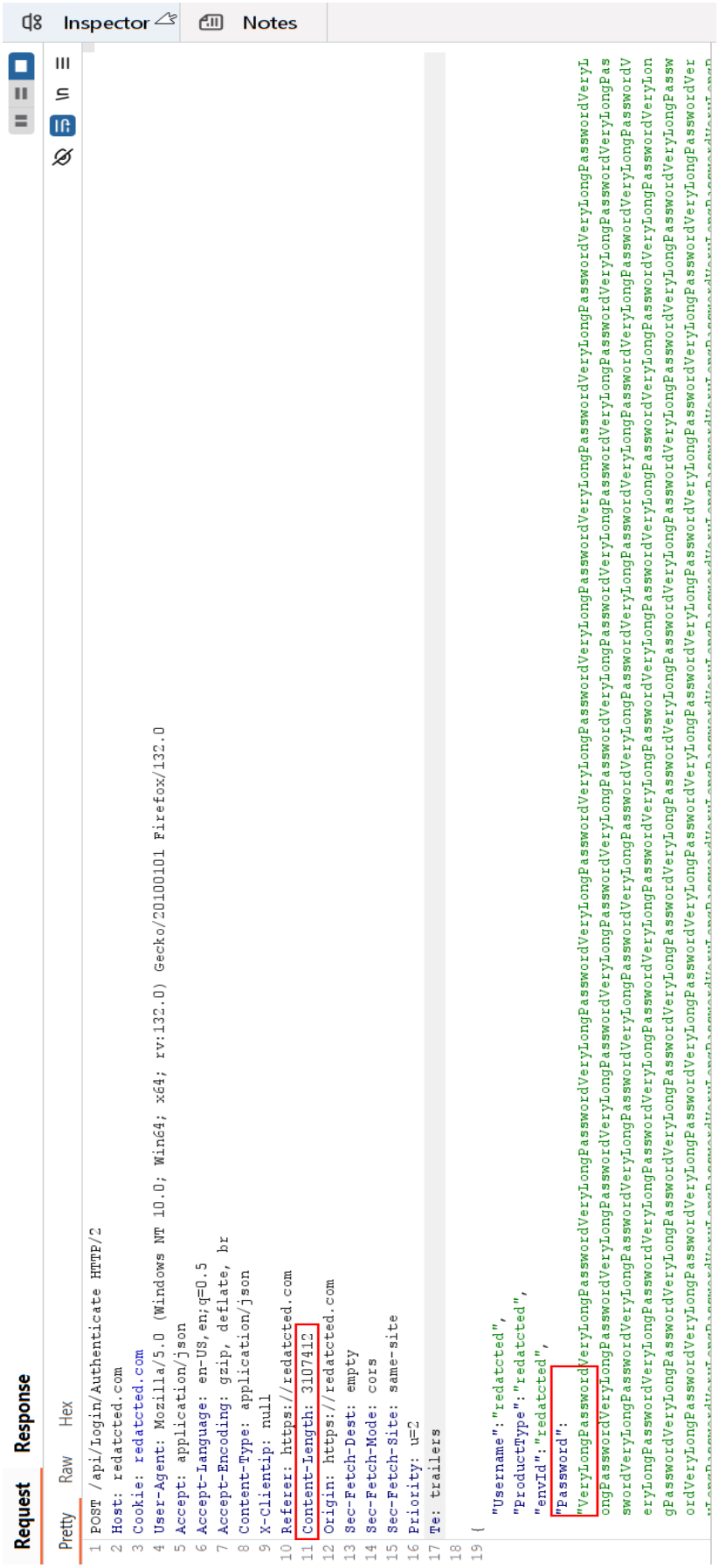

Figure 2. long string password.

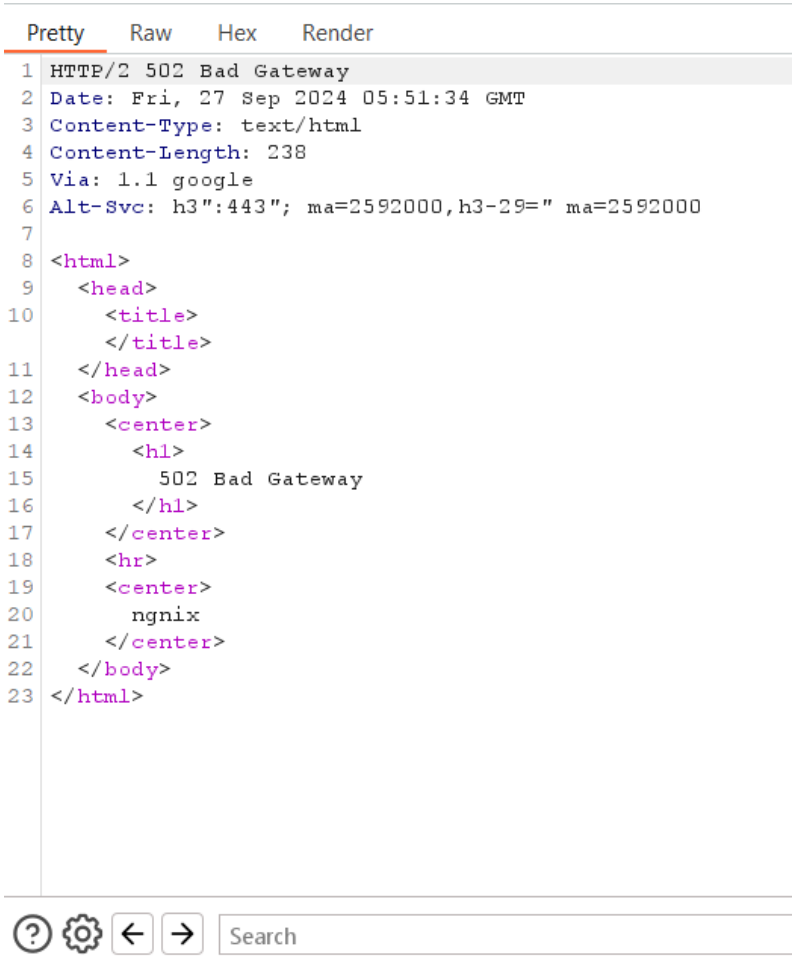

Figure 3. 502 Bad Gateway.

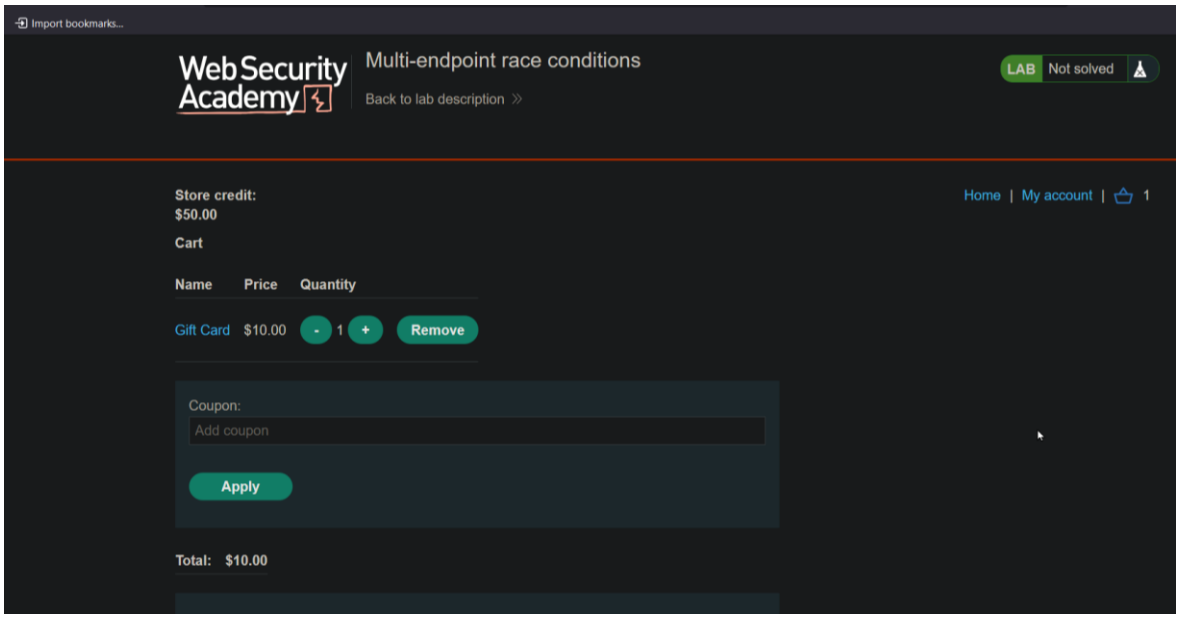

Figure 4. Adding a higher valued product to cart.

The race condition vulnerability allows attackers to get products at unintended price. Starting with it, we added a product to cart worth $10 which was less than our Store Credit (Figure 4).

After analyzing the application behavior, it was identified that the first request would always take more time than subsequent requests, when we sent a group (single connection request). Figures 5, 6, and 7 highlight the request time taken for three requests: (1) initial, (2) add product to cart, and (3) checkout request.

Then instead of the previous gift card product (which was less than our store credit), we added a different product in the cart which was more than our store credit. With our previous analysis of the application behavior, we sent three requests parallelly at once (Figure 8). Owing to this, a race condition could occur that allowed to buy products at unintended price.

After multiple tries by sending all three requests parallelly at once, we could observe that the application processed our newly added product despite of not having the enough store credit (Figure 9).

RESULTS

Based on the above discussion, we could conclusively establish that while OWASP Top 10 is a comprehensive project, it does not (and practically no single project/framework/standard can) cover a number of vulnerabilities related to web application security. But our overemphasis on just one framework or Top-10 may lead to a situation of Beagle view (narrow sightedness). This Beagle phenomenon may not allow us to look beyond and thus neglecting other vulnerabilities.

Therefore, to overcome such weakness, we suggest *a security ladder* approach for web application security.

This security ladder is inspired from the seminal work of Arnstein SR in her 2019 article, "A Ladder of Citizen Participation" [1]. Arnstein's *Ladder of Citizen Participation* is a model that outlines various levels of public involvement in decision-making processes, emphasizing the degree of power citizens hold at each level. The model categorizes eight levels of participation, divided into three main groups: *Nonparticipation*, *Tokenism*, and *Citizen Power*. This ladder illustrates a spectrum from low to high citizen agency in governance, challenging policymakers to recognize and address power imbalances in community engagement efforts (Figure 10).

BASIC WEB APP SECURITY (THE FOUNDATION)

A fundamental approach to web application security consists of SAST, DAST, Security by Design, and developer training. While Security by Design incorporates security principles into the architecture from the beginning, guaranteeing resilience against attacks, developer training enables teams to produce secure code, lowering the possibility of introducing faults. By combining preventative, investigative, and remedial methods, this synergy builds a multi-layered defense. This paradigm successfully protects applications against changing threats by promoting a culture of secure coding and integrating with standards like as OWASP. This proactive strategy minimizes vulnerabilities before to deployment, lowers risks, and is consistent with the "Shift-Left Security" concept.

Evolved Web App Security

The three levels of testing—Black Box, Grey Box, and White Box—along with application audits, user awareness, and training, create a well-rounded strategy for enhancing web application security.

Optimal (Matured) Web App Security

An ideal web application security architecture is produced by addressing third-party threats, using Zero Trust Architecture (ZTA), and integrating security throughout the lifecycle. By ensuring that external suppliers, libraries, and integrations adhere to security requirements, third-party risk management stops dependencies from introducing vulnerabilities. This is enhanced by ZTA, which minimizes implicit trust and enforces stringent access restrictions to guarantee that all requests—internal or external—are validated. This strategy lessens the attack surface and lessens the dangers of insider threats and compromised third parties.

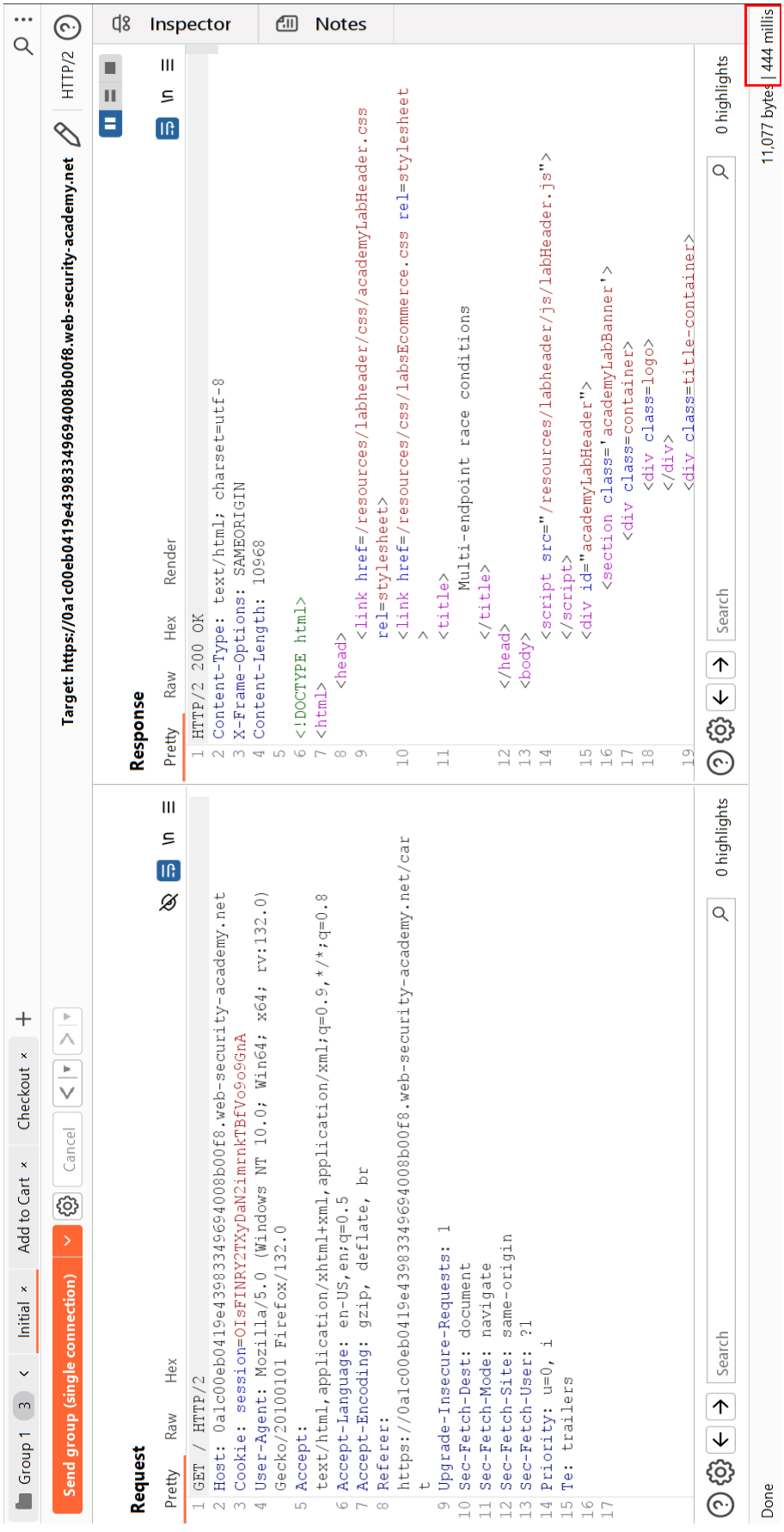

Figure 5. Request time taken for initial request.

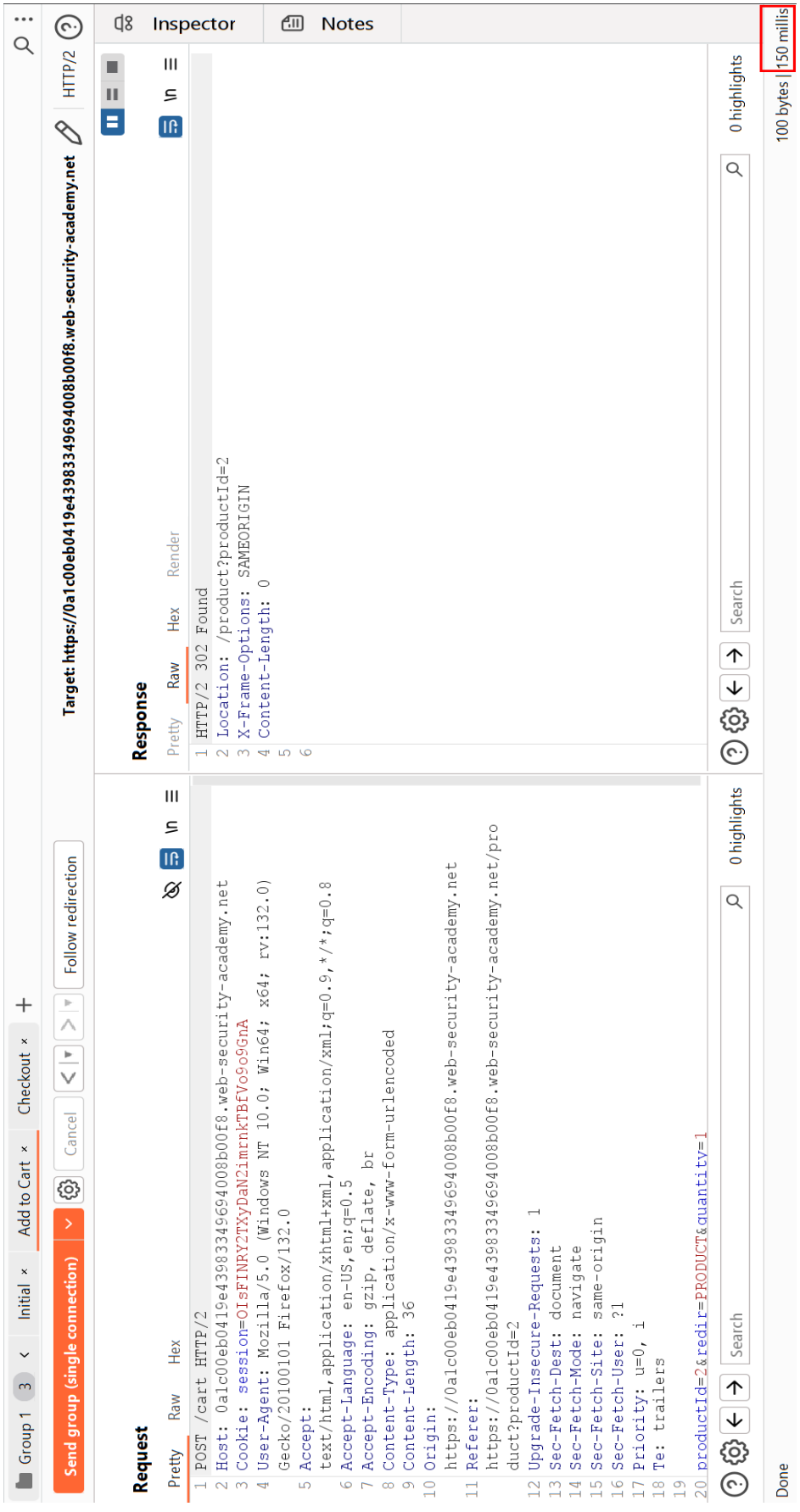

Figure 6. Request time taken for addition of product to cart.

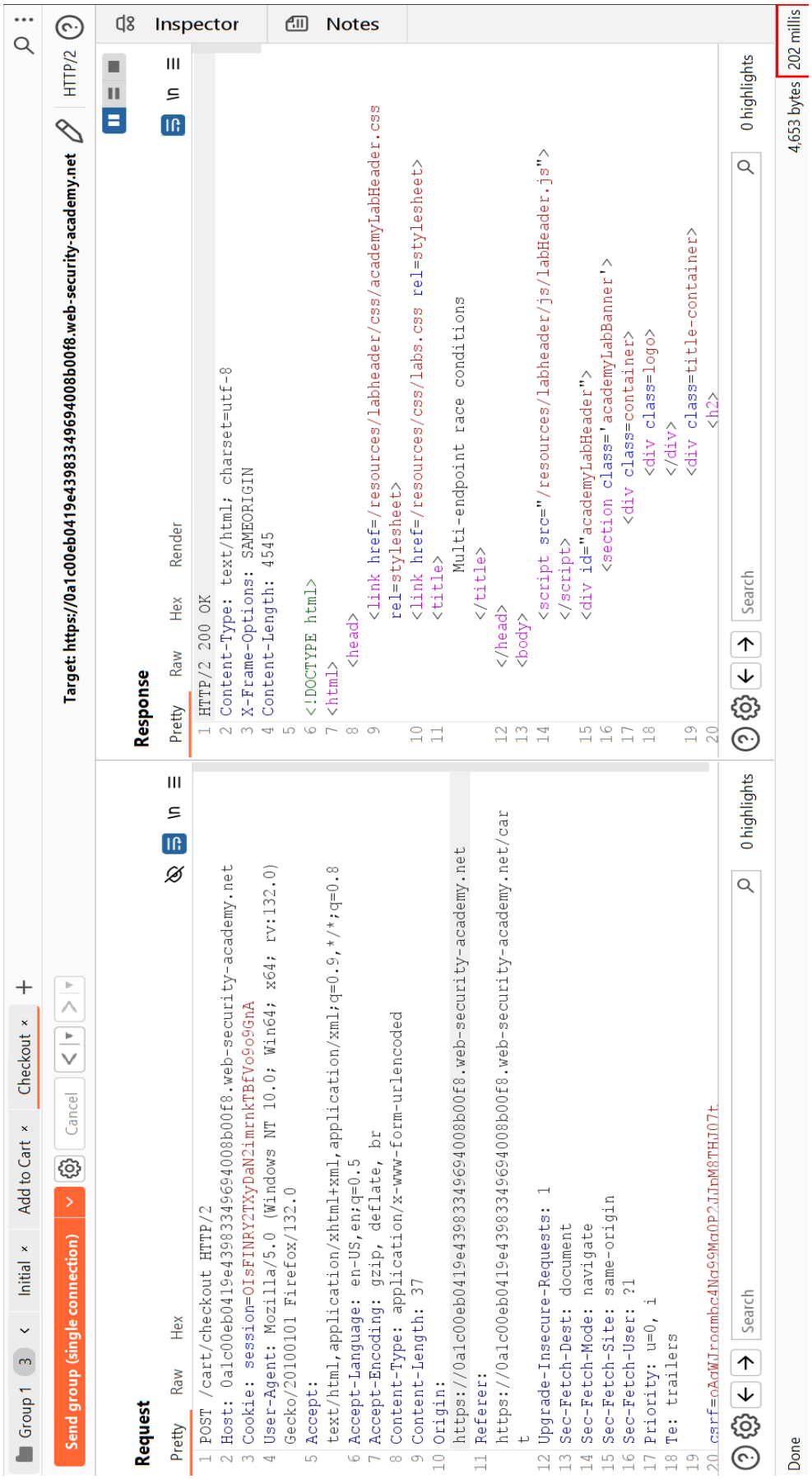

Figure 7. Time taken for checkout request.

International Journal of Information Security Engineering
Volume 3, Issue 1
ISSN: 3049-1150

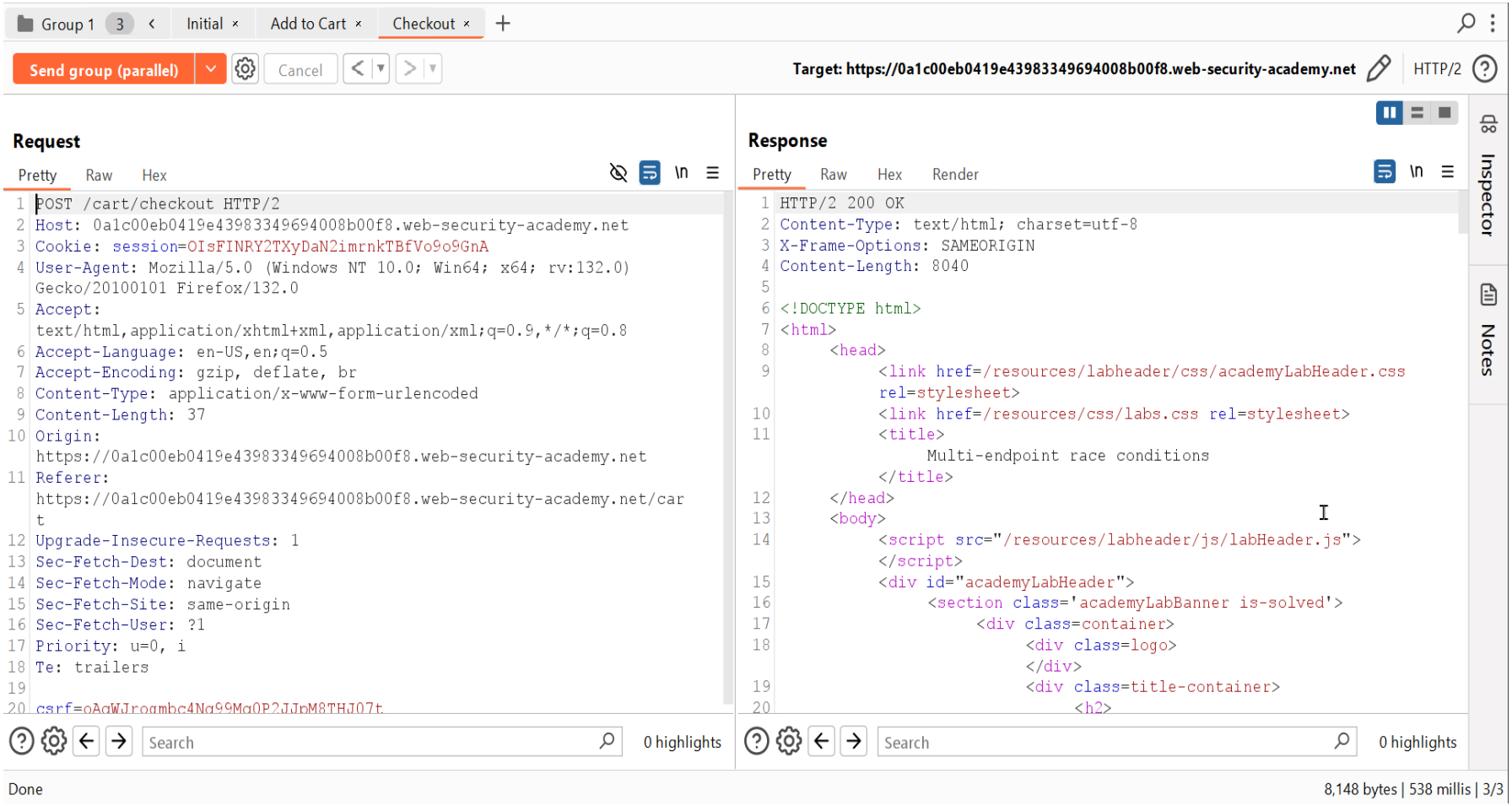

Figure 8. Parallel requests.

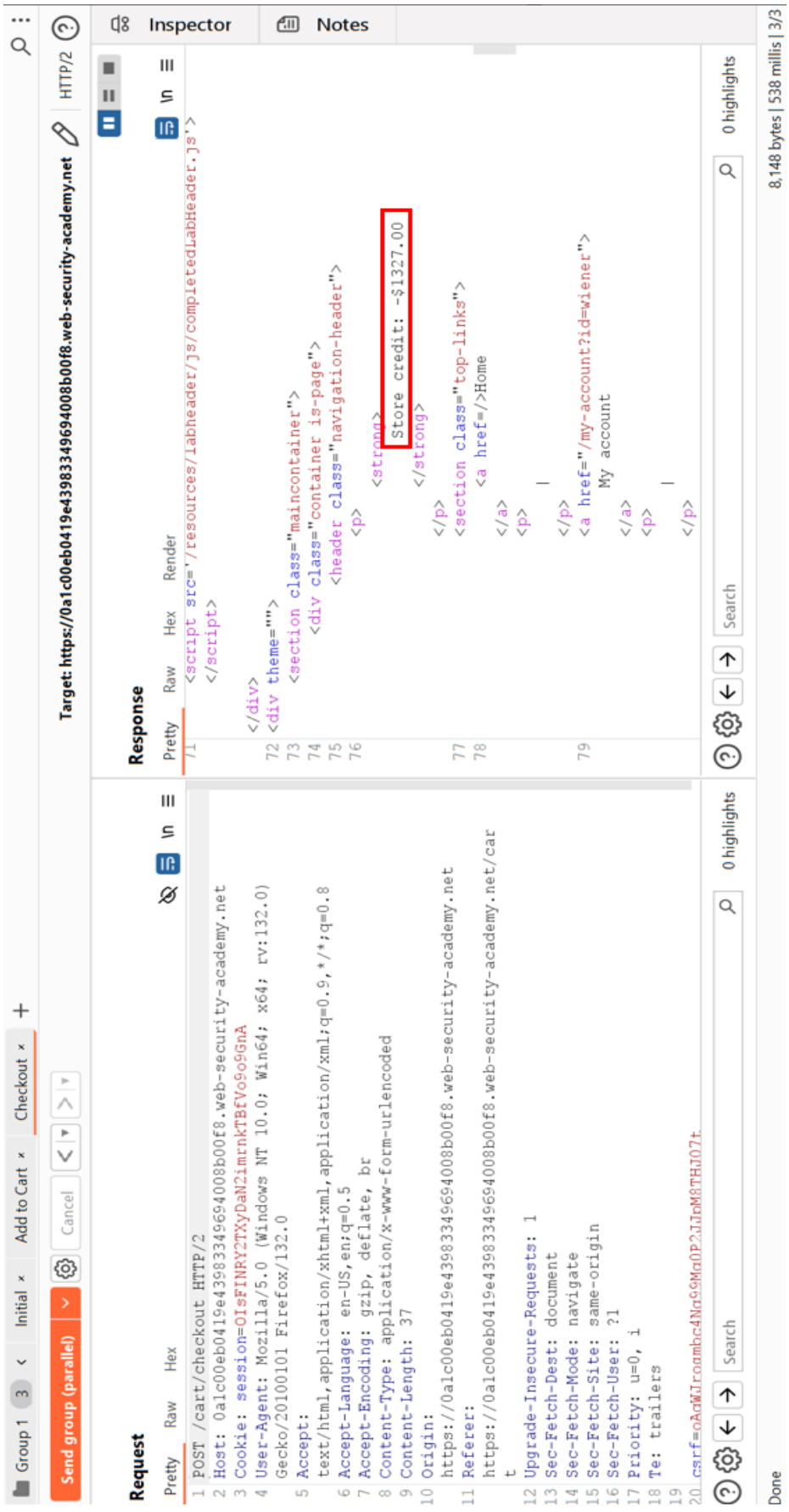

Figure 9. Buying of product despite not having credit value.

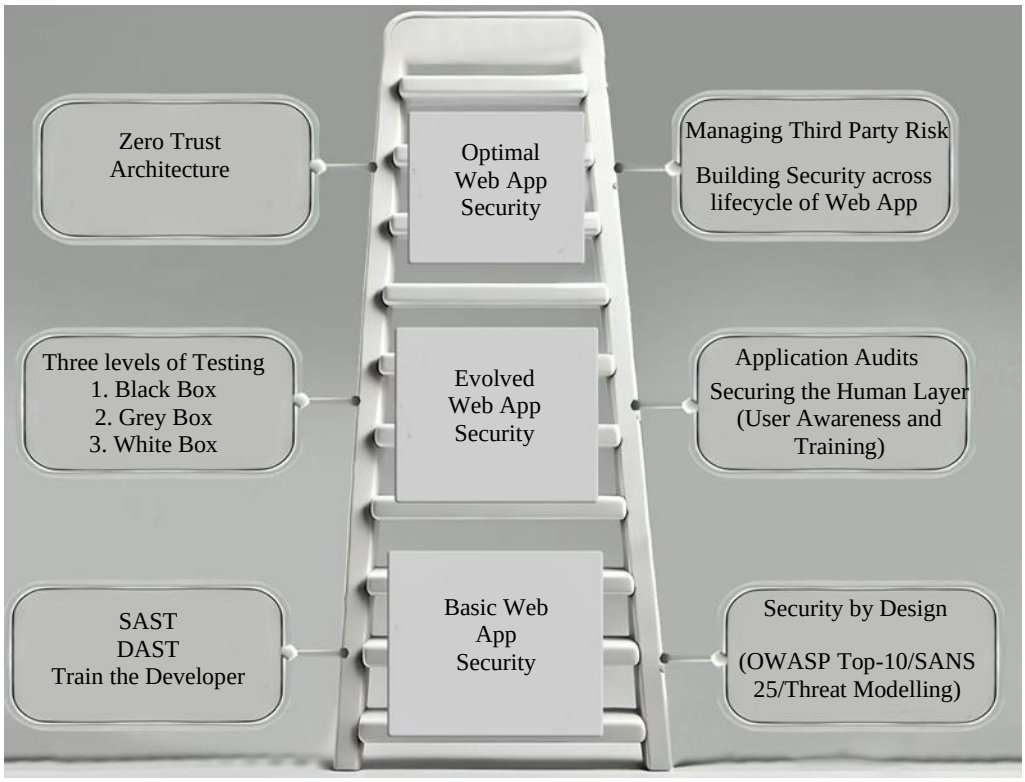

Figure 10. "The Security ladder".

Note: The experts can always argue that x should be at lower level or higher level but the purpose of this ladder is to simply argue that a variety of efforts combined together are increasing the security levels of the web application. It is also implicit that many of the mentioned steps can itself be undertaken as a part of the other step.

Let us discuss the individual components of three stages of web application security in some detail:

Security by Design is a fundamental approach that builds security into the architecture from the start, rather than adding it later. Incorporating and integrating standards like the OWASP Top 10, SANS Top 25, and threat modelling helps ensure that key vulnerabilities are proactively addressed [12].

Likewise, threat modeling identifies, categorizes, and evaluates potential threats early in the development process. By analyzing possible attack vectors, it helps designers and developers understand how data and user interactions might be exploited, allowing them to embed countermeasures to prevent such threats.

The way present global bodies are taking up the cybersecurity on global policy agendas demonstrate clearly that there would not be any dearth of frameworks/standards to take key lessons from. In many cases, that are inspired from each other (Table 2).

The Security Mindset Among the Makers: (Train the Developer)

To create robust apps, developers must receive secure coding training. In order to reduce vulnerabilities, secure coding training equips developers to use essential security procedures such input validation, encryption, and error handling. Many vulnerabilities are caused by frequent, avoidable mistakes and safe coding training assists developers in resolving these problems prior to deployment. Developers are better prepared to produce secure applications that satisfy regulatory requirements by adhering to organized training that is in line with industry standards, such as those highlighted by SANS.

Table 2. Good frameworks/standards from which we can incorporate our design principles and the security criteria.

Framework	Focus Areas	How It Helps
OWASP Secure Coding Practices (SCP)	Input validation, output encoding, authentication, access control, cryptographic practices.	Helps developers avoid security pitfalls by addressing common mistakes and integrating security early on.
NIST Secure Software Development Framework (SSDF)	Secure architecture, threat modelling, risk assessment, secure coding, vulnerability management.	Emphasizes secure design from the beginning, helping developers identify flaws early.
Microsoft STRIDE Threat Modelling	Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, Elevation of Privilege.	Encourages threat modelling during the design phase, ensuring attack vectors are identified and mitigated.
ISO/IEC 27034 Application Security	Security controls throughout the application lifecycle, from design to deployment and maintenance.	Helps implement secure practices consistently across the software development lifecycle.
CIS Controls for Secure Application Development	Secure configurations, vulnerability management, data protection, authentication, error handling.	Provides clear guidance on distinguishing secure from insecure practices, particularly for critical security areas.

Statis Application Security Testing and Dynamic application Security Testing

Static Application Security Testing (SAST) and Dynamic Application Security Testing (DAST) are two critical methodologies in application security testing, each with distinct purposes and approaches.

SAST is a white-box testing technique that analyzes an application's source code, binary, or bytecode for vulnerabilities without actually running the code. By scanning code while it is at rest, SAST finds security vulnerabilities like buffer overflows, SQL injection, and XSS right in the code structure. SAST's ability to identify vulnerabilities early in the software development lifecycle (SDLC) lowers the cost of addressing problems later on. As developers create code, SAST tools may provide them instant feedback and work well with development environments [3].

DAST, on the other hand, is a black-box testing technique that runs the program to look at it from the outside. By simulating assaults on an active application, it finds vulnerabilities in real-world scenarios, including configuration problems, runtime defects, and poor authentication methods. DAST is especially helpful for finding vulnerabilities that SAST would overlook, including errors in business logic or incorrect server setups, because it tests an application in a live environment.

LEVELS OF PENTESTING

Pentesting can be categorized along white-box, black-box, and grey-box testing. Each level has a different scope, information access, and methodology to assess an application or network’s security.

White-Box Testing

White-box testing gives the tester complete access to internal data, such as IP addresses, network architecture, source code, and configuration specifics. "Glass-box" or "clear-box" testing are other names for this stage. Simulating an insider attack or an attacker who has learned a great deal about the system is the goal of white-box testing.

Black-Box Testing

Black-box testing is carried out without any prior understanding of the network architecture or system. By trying to get past security measures without any inside knowledge, the tester mimics an external assault. This level is the most similar to how an attacker might approach an application in the real world.

Grey-Box Testing

White-box and black-box testing techniques are combined in grey-box testing. A portion of the system, such as login credentials, architectural information, or API documentation, is known to the tester. A scenario where an attacker has little insider knowledge, such as a disgruntled employee or someone with restricted but privileged access, is what grey-box testing attempts to simulate.

SECURING THE HUMAN LAYER (USER AWARENESS AND TRAINING)

Human error remains one of the leading causes of security breaches in today's cybersecurity landscape. Organizations must invest in comprehensive training programs that educate employees about potential threats, such as phishing, social engineering, and safe browsing practices.

Employees that get regular security awareness training are better able to identify and address security concerns. According to studies, companies that regularly provide security training greatly lower the likelihood that phishing attempts would be successful. Organizations that implemented security awareness programs saw 55% fewer successful phishing assaults than those that did not, according to a Ponemon Institute analysis [11].

Because cyber risks are always changing, it is crucial to implement ongoing training and awareness initiatives. To handle emerging dangers and strengthen current expertise, organizations should conduct refresher courses and update their training materials on a regular basis. According to a Verizon analysis, a mature security posture must include continual education and awareness in order for enterprises to adjust to shifting threat environments.

The Power of Application Audits

Critical evaluations of software applications' security, usability, and compliance are called application audits. The objectives of these audits are to find weaknesses, make sure industry standards are followed, and confirm that apps function as intended. Security assessment, compliance verification, substantive audits, functional evaluation, and a continuous improvement roadmap are a few examples. Frequently, the reporting and documentation provide sufficient evidence for the interested parties to guarantee that the application is safe from the inside out.

Building Security Across the Life-cycle of the Web-App

This approach mostly coincides with "security by design," and is often synonymous with it. It integrates security practices at every phase of the SDLC. Organizations must give equal weight to identifying security and functional needs in the early stages of web application development. To guarantee that security concerns are in line with overarching corporate goals, this entails creating extensive risk assessments, compliance requirements, and comprehensive security policies. Involving different stakeholders at this point promotes cooperation and guarantees that security is incorporated into the project from the beginning, which eventually results in a more secure and effective development process.

To build a robust application architecture, architects should apply fundamental security concepts like defense in depth and the principle of least privilege throughout the design stage. At this point, threat modeling exercises are essential because they aid in the early detection of possible weaknesses and design faults, which may greatly improve security posture.

Security During the Maintenance Phase

An application's maintenance phase is essential to maintaining continuous security and resistance to new threats. Patch management and routine updates are crucial procedures to reduce vulnerabilities that an attacker may exploit. To check the efficacy of security measures and find any new threats, regular penetration tests and vulnerability assessments should also be carried out. Staff personnel must also get security awareness training at this period in order to reinforce acceptable practices and guarantee that everyone is alert to possible risks.

Secure Disposal of Application

When an application reaches the end of its lifecycle, secure disposal is critical to prevent unauthorized access to sensitive data and ensure compliance with regulatory requirements. This process involves more than just uninstalling the software; it requires thorough data sanitization practices, such as data wiping, to eliminate residual data that could be recovered by malicious actors. Organizations should follow established guidelines for secure disposal, including those from NIST and ISO, which provide frameworks for effective data destruction methods. Additionally, it is essential to review and revoke any access permissions associated with the application, ensuring that no residual accounts remain that could be exploited.

When migrating data from a discarded application to a new system, organizations must prioritize data integrity, security, and compliance throughout the process. To ensure that redundant or outdated data is not transmitted, the first stage is a thorough data evaluation to identify which data is required for migration. Sensitive data should therefore be protected during the relocation process by using secure data transmission techniques like encryption. In order to guarantee consistency and compatibility and reduce the possibility of data loss or corruption, it is also crucial to map data fields between the old and new applications. To ensure that the data has been accurately transferred and that the new application operates as intended, extensive validation and testing should be carried out following the migration. Last but not least, companies should record the migration procedure for future use and compliance, making sure that every action follows industry best practices.

Managing Third-Party Risks

Today, we as organizations increasingly rely on external vendors for essential services. There is no dearth of real-life examples of it where organizations have suffered the significant consequences that can arise from poor management of these relationships.

Organizations should start controlling third-party risks by conducting a thorough vendor risk assessment, which looks at the security protocols and regulatory compliance of possible vendors. It is essential to have explicit contractual agreements outlining security expectations and accountability. Companies such as Facebook, for example, have improved their vendor management by putting in place stringent evaluations of third-party apps in order to safeguard user information. Effective risk mitigation and compliance maintenance by suppliers are guaranteed by ongoing monitoring conducted through frequent audits and evaluations.

As demonstrated by the 2021 Colonial Pipeline ransomware attack, incident response strategies should also incorporate third-party providers to enable coordinated action during security crises. In order to successfully handle the incident, the organization collaborated with outside cybersecurity companies, demonstrating the need of cooperation and readiness.

Zero Trust Architecture Across All Interfaces

This “Never Trust, Always Verify” and “Assume Breach” approach is particularly relevant in today's environment where organizations face increasing threats from cyber adversaries, and where traditional perimeter-based security is no longer sufficient [8].

Continuous monitoring and logging of all activity across the interfaces is one of the most important aspects of putting ZTA into practice. Organizations may identify unusual behavior and react quickly to any security problems by gathering and analyzing data from several sources. In order to detect compromised accounts or illegal access attempts, this monitoring should also contain context-rich data about user behavior, device health, and application interactions. Furthermore, ZTA may be improved by utilizing technologies like identity and access management (IAM) and micro-segmentation, which minimize the attack surface by guaranteeing that access privileges are precise and customized to the unique requirements of users and applications [16].

CONCLUSION

Ultimately, by integrating security into the architecture from the outset, we can establish a robust foundation that will not only reduce vulnerabilities but also enhance the application's long-term resilience. This approach minimizes risks associated with breaches, downtime, and costly post-deployment fixes. With secure design principles, the organization can anticipate and mitigate potential attacks, helping to avoid data breaches that could damage brand reputation, incur regulatory penalties, and erode customer trust. Moreover, secure architecture reduces the time and cost of maintenance, allowing development teams to focus on innovation rather than reactive fixes. With cyberattacks increasingly sophisticated, prioritizing security as a core business strategy is no longer optional; it is essential for staying competitive, compliant, and reliable. This proactive approach ultimately leads to an application that isn't just secure, but agile and future-ready, positioning the organization as a leader in cybersecurity and customer trust.

Competing Interests Declaration

The authors declare that they have no competing interests in relation to this work.

REFERENCES

1. Arnstein SR. A ladder of citizen participation. J Am Plann Assoc. 2019; 85 (1): 24–34.
2. Mendoza A, Gu G. Mobile application web API reconnaissance: web-to-mobile inconsistencies & vulnerabilities. In: 2018 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, May 20–24, 2018. pp. 756–769.
3. Cappelli DM, Moore AP, Trzeciak RF. The CERT Guide to Insider Threats: How to Prevent, Detect, and Respond to Information Technology Crimes (Theft, Sabotage, Fraud). Boston, MA, USA: Addison-Wesley; 2012.
4. Groš S. A critical view on CIS controls. In: 2021 16th International Conference on Telecommunications (ConTEL), Zagreb, Croatia, June 30–July 2, 2021. pp. 122–128.
5. Bashir T. Zero trust architecture: enhancing cybersecurity in enterprise networks. J Computer Sci Technol Stud. 2024; 6 (4): 54–59.
6. Cloud Security Alliance. Home | CSA. [Online]. Cloudsecurityalliance.org. 2024. Available at https://cloudsecurityalliance.org/
7. Pizzorno A. On the individualistic theory of social order. In: Bourdieu P, Coleman JS, editors. Social Theory for a Changing Society. New York, NY, USA: Routledge; 2019. pp. 209–244.
8. Gartner. Delivering Actionable, Objective Insight to Executives and Their Teams. [Online]. Gartner. 2025. Available at https://www.gartner.com/en
9. McGraw G, Chess B, Migues S. Building Security in Maturity Model. San Mateo, CA, USA: Fortify & Cigital; 2009.
10. Souppaya M, Scarfone K, Dodson D. Secure software development framework (SSDF) version 1.1. NIST Special Publication. 2022; 800: 218.
11. Ponemon Institute. Home. [Online]. Ponemon Institute. 2020. Available at https://www.ponemon.org/
12. Upadhyay D, Ware NR. Evolving trends in web application vulnerabilities: a comparative study of OWASP Top 10 2017 and OWASP Top 10 2021. Int J Eng Technol Manage Sci. 2023; 7 (6): 262–269.
13. Nedeljković N, Vugdelija N, Kojić N. Use of "OWASP Top 10" in web application security. In: Fourth International Scientific Conference on Recent Advances in Information Technology, Tourism, Economics, Management and Agriculture, Online/Virtual, October 8, 2020. p. 25.
14. Riggs FW, MacKean DD. Administration in Developing Countries: The Theory of Prismatic Society. Boston, MA, USA: Houghton Mifflin; 1964.
15. Rose S, Borchert O, Mitchell S, Connelly S. Zero trust architecture. August 2020. Available at https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-207.pdf
16. CISO Magazine. CISO MAG | Cyber Security Magazine. [Online]. 2022. Available at https://cisomag.com/magazine/

