Security Testing for E-Commerce Applications

Alexandru-Petrișor LAZĂRA
Faculty of Electronics, Telecommunications and Information Technology,
University POLITEHNICA of Bucharest, Romania
alexandru.lazara@stud.etti.upb.ro

Abstract

Over the past decade, as the e-Commerce market has evolved into a shopping ecosystem involving multiple devices and store concepts, retailers have been continuously innovating the online shopping experience introducing convenient features like multi-device optimizations, product customization, quick and secure checkout processes, or recurrent payments to attract more customers and influence purchase decisions. The main guidelines that are followed in this paper are revolving around security testing and how it can be performed in the form of manual and automated testing, with aid from automated security tools. This paper looks at the threats e-Commerce Applications are facing in regards with cybersecurity and intends to assist preventing vulnerabilities being exploited by malicious intended users by showing the importance of performing security testing to identify weaknesses, mitigate risks and to raise awareness of the importance of strong security measures and procedures.

Index terms: e-commerce security, security testing, software vulnerability, software risk mitigation, automated security tools

1. Introduction

As e-Commerce continues to evolve, the increased use of digital technology has made it vulnerable to cyber-attacks and fraudulent activities. The implementation of online shopping was accelerated by the pandemic, which registered a major worldwide impact. As reported by the International Trade Administration, global e-Commerce revenue has increased by 19% between the period preceding and following the COVID-19 outbreak in 2020 and reported an additional 19% sales growth for 2020 to the existing 9% regular forecast sales growth rate [1].

Although the market already reached its maturity phase during pandemic and the competition is extremely high and cost intensive in most developed countries, the global e-Commerce market is anticipating to continue its growth trend, with an expected Compound Annual Growth Rate of 11.51% from 2023 to 2027, resulting in a projected market volume of $6.35 trillion and an anticipated user base that amounts to 5,288.5 million worldwide, as Statista is highlighting [2].

Following the above-mentioned trends in the e-Commerce market and given the integration of digital information and technology into the basic operations of the retail activity, the risk of cyber-attacks and fraudulent activities related to online shopping also increased. The repercussions of cybersecurity breaches can lead to operational, financial, reputational, and strategic ramifications for an organization, all of which may result in significant costs [3], yet their effects are not limited to the organization itself, but also extends to its customers which may suffer from the loss of personal information and financial resources. Consequently, ensuring the security of e-Commerce Applications has become a major concern for both businesses and customers.

2. Cybersecurity measures against cyber-attacks

Gartner defines cybersecurity as "the practice of deploying people, policies, processes, and technologies to protect organizations, their critical systems and sensitive information from digital attacks" and enlists some of the most common and notable types of cybersecurity attacks – that are also impacting the e-Commerce market – as follows: phishing and social-engineering-based attacks, internet-facing service risks (including cloud services), password-related account compromises, misuse of information, network-related and Man-in-the-Middle attacks, supply chain attacks, ransomware, and Denial-of-Service attacks (DoS) [3].

In addition, Gartner also provides a list of technical defense measures against cyber-attacks, which includes performing network and perimeter security, endpoint security, application security, data security, Identity and Access Management, and implementing a Zero-Trust Architecture [3]. One critical technical measure that is not mentioned in the list, but is equally important, is software security testing. **Security testing** is a type of software testing – an important phase in the Software Development Lifecycle (SDLC) – that asserts "whether software is vulnerable to cyber-attacks and tests the impact of malicious or unexpected inputs on its operations", providing confirmation that "systems and information are safe and reliable, and that they do not accept unauthorized inputs" [4]. By continuously performing security testing, the software is scanned for weaknesses in its implementation, design, and coding. This help organizations ensuring that software meets the necessary security requirements, by fixing any identified vulnerabilities before cybercriminals take advantage of them, thus avoiding serious security breaches. However, security testing is a specialized testing process and should be conducted by a team of certified testers that have a certain level of expertise in this field.

Furthermore, technology control isn't the only line of defense against cyber-attacks. Building employee awareness and establishing cybersecurity procedures at the organizational level are also essential measures for ensuring an effective cybersecurity strategy. The **ISO/IEC 27000** family of standards define requirements, provide direct support guidance and interpretation for the process to establish, implement, maintain, and improve an Information Security Management System, including risk management, information security controls, and incident management [5].

3. Security Testing

When an application is developed, numerous levels of software testing should be carried out by the development team to detect any potential defects – **bugs** – that may occur and result in an observed failure in the application execution [6]. Left unchecked, many of these bugs can be exploited by attackers, but as already stated, security testing can be used to verify whether Software Applications carry such vulnerabilities and to minimize their associated risks.

Security testing for e-Commerce Applications refer to the process of rigorously searching and identifying vulnerabilities in the shopping application, starting from the fundamental interactions present in all different types of implementations, such as user authentication and authorization, continuing with more specialized elements like promotions, product configurations, shopping carts, payment gateways, or the order management system, which are usually targeted by malicious users. Performing security testing can help highlight weaknesses in the implementations of such elements, as those that permit unauthorized users to force and bypass certain authorization restrictions to modify relevant product information, including descriptions, prices, or quantities, through injection methods as SQLi (also known as SQL injection) – an attack that "consists of insertion or injection of a SQL query via the input data from the client to the application" to gain access to read or modify sensitive data from a database and even execute administration operations on it [7].

Session hijacking is another form of attack that grants the attacker unauthorized access, which consists in "the exploitation of the Web session control mechanism, which is normally managed for a session token". Because TCP (Transmission Control Protocol) connections are used intensively in HTTP (Hypertext Transfer Protocol) communication, the Web Server needs a method to recognize every user's connection. The most advantageous method "depends on a token that the Web Server sends to the client browser after a successful client authentication". There are different ways to compromise the session token, however the most common ones are predictable session token, session Sniffing, Man-in-the-Middle and Man-in-the-Browser attacks, and client-side attacks (such as XSS – Cross Site Scripting, or the use of malicious JavaScript Codes) [8].

To detect and address vulnerabilities in this kind of software, both manual and automated security testing needs to be used. In **manual testing**, the software engineer mainly "assumes the role of a user executing the System Under Test (SUT) to verify its behavior and find any observable defects", while **automated testing** is using "scripts that execute without human intervention to test the SUT's behavior". However, security engineers can fully benefit from automated security testing only if they acquire expertise in using testing techniques together with available testing tools such as **vulnerability scanners and analyzers** [9]. Brian Hambling's ISTQB Foundation Guide describes security testing tools as instruments used "to test the functions that detect security threats and to evaluate the security characteristics of software" and to assess the capacity of the SUT to handle computer viruses, protect data confidentiality and integrity, prevent unauthorized access, carry out authentication checks of users, remain available under a DoS attack, or check non-repudiation attributes of digital signatures [6]. Even if vulnerability scanners and analyzers alone are powerful tools, an even more accurate risk rating can be obtained by combining the mentioned testing types with the services provided by a **penetration tester**, that plays the role of an attacker to find and exploit vulnerabilities. While "penetration testing occurs at the end of the SDLC, the results of the penetration test can provide feedback for tests even in its earlier phases". The penetration test report should provide clear details about the discovered vulnerabilities, including how to fix them, and how a specific vulnerability can be exploited by an attacker. A security team needs to help the development interpret the penetration test report and provide guidance in addressing the found vulnerabilities [10].

It is important to note that testing benefits most from implementing **CI/CD (Continuous Integration and Continuous Delivery)** pipelines. This ensures that testing is an ongoing process and that any issues are identified and resolved quickly, reducing the risk of security breaches and other defects that can negatively impact the software development process.

Regardless of the type of testing, it is crucial to avoid any negative impact on the production environment. Working on a **test environment** that is an exact replica of the production environment, with personal identifiable information anonymized, is an endorsed practice to avoid this risk.

3.1. Manual Security Testing

Staying at the forefront of any application, manual security testing must be carried out for every new iteration of the SDLC, for every new added feature, as well as for every change that is impacting the existing features, as part of the regression testing. To improve this process and increase vulnerability discovery, when a new weakness is discovered during testing, it is essential to create a new test case in the designated testing suite. This will help ensuring that the bug can be addressed and resolved in future iterations of the SDLC. It is also important to document all actions taken during the security testing process, including any changes made to the system, and to ensure that the system is returned to its initial state after the testing is completed.

The payment gateway is one of the most critical and complex element of an e-Commerce application. In this case, manual security testing should be performed to ensure that users cannot make unauthorized electronic payments or that their sensitive payment information is being handled securely to prevent it from being intercepted and used by attackers. As a vendor, it is usually a good

practice to use a payment gateway provider instead of developing an internal one, since they can afford investing more time and resources in security measures like tokenization, "the process of protecting sensitive data by replacing it with an algorithmically generated number called a token"; or by using a payment processing encryption method that uses "keys which are encrypted and decrypted to keep sensitive customer information safe" [11].

Handling information securely is mandatory not only in the interactions with the payment gateway, but in all the aspects that are implying sensitive data. The level of access to data and deciding which data can be provided to users or third-party agents should be highly dependent of context. For instance, to maintain their privacy, when sending notification alerts to users, only essential information should be included. Another example consists in sharing client information to the company responsible with delivering customer's order. In this case, the e-Commerce platform must confirm that only the necessary information like delivery address and contact information is shared, and not something sensible like user's payment details or order content.

Other important aspects to be taken into consideration, as security reports are revealing, are the vulnerabilities within APIs (Application Programming Interfaces) implementation and request responses. Thus, during security testing it is considered good practice to assert whether the best practices in API authentication and authorization are being followed. Between these can be enlisted the usage of transport layer security, usage of access control measures by enabling configuration of different permissions for different API keys, as well as constructing a good management system to protect them [12]. Another good practice that needs to be followed in the security testing process is verifying that the responses received from the API requests (such as error messages, internal application errors or configurations) are not disclosing any confidential and system information.

3.2. Automated Security Testing

In comparison with manual testing, automated security testing is a faster, more effective, and dependable approach to assert the vulnerabilities and risks of a SUT, since it can perform tests more efficiently and generate reports containing information about the identified security weaknesses, as well as recommendations for corrective or preventive measures. Moreover, while this method doesn't require a certified security engineer to conduct the testing, the automated security tests that will be performed still needs to be developed by a security team of software developers (in form of unit and integration tests) and test engineers (in form of system tests) that has the required expertise.

Traditionally, security testing "have been conducted after code has been completed", however with aid from security tools, tests can also be performed "in earlier phases of the SDLC, such as develop and commit, and can run as the code is written, with feedback delivered directly in the Integrated Development Environment (IDE)". This behavior implies that security tools integrated within a technology stack are more accessible to software developers rather than to security testing engineers. As a result, the primary purpose of this kind of tools is to provide a good indication of what the found issues are and how to fix them. In this way, security tool findings can be provided directly to the developers, to enable rapid feedback on any issues, even if in some cases the security team must conduct an additional security analysis [10].

Static Application Security Testing (SAST), or Static Analysis of applications, is a security testing technique that examines an application's source code without executing it, and is used to analyze lexical, grammar, and semantic features, as well as data flow and model checking to detect hidden bugs. The main advantage of this technique consists in its high detection speed, as a Static Analysis tool can quickly check the target code. However, such tools carry "a high false rate in practice, due to the lack of an easy-to-use vulnerability detection model", making it challenging to identify satisfactory results [13] [14]. Between the tools available for performing SAST, we can enlist names as SonarQube or Roslynator. **SonarQube** is an open-source platform developed by SonarSource. It offers the capability to assess the health of an application and to identify newly

introduced issues, provides reports on duplicated code, coding standards, unit tests, code coverage and code complexity, bugs, and security recommendations. It helps analyze the quality on more than 30 programming languages, frameworks, and Infrastructure as Code (IaC) Platforms [15]. **Roslynator** is another powerful open-source SAST tool which consists in a collection of more than 500 analyzers, refactorings and fixes for C#, and is powered by Roslyn, an open-source implementation of both the C# and Visual Basic compilers with an API surface. It provides a wide range of features for improving code quality, including code style analysis, code formatting, and code refactoring [16] [17].

In contrast, **Dynamic Application Security Testing (DAST)**, or Dynamic Analysis of applications, requires code execution to observe its behavior. "By monitoring the running states and analyzing the runtime knowledge, Dynamic Analysis tools can detect program bugs precisely". The advantage of this method is its high accuracy, but there still exist the shortcomings of slow speed, low efficiency, high requirements on the technical level of testers and application architecture, poor scalability, and the difficulty to carry out large-scale testing [13] [14]. In industry, there is a series of DAST analyzers as the ones provided by **GitLab**, for scanning websites: DAST proxy-based analyzer and DAST browser-based analyzer; and for scanning APIs: DAST API analyzer. Based on differences between different scan results on the source and target branches, these analyzers can produce a DAST report that GitLab utilizes to identify vulnerabilities in applications [18].

At commit time, automated testing known as SAST, is checking for "known insecure patterns in the source code before being added to the code repository or merged to the main branch". Security tests run at build time are known as **Software Composition Analysis (SCA)**, and are checking for vulnerabilities in libraries, or in used container images. Security tests that run at deploy time, which "allows automated testing on a running application", are the ones referenced as DAST. For development on a new codebase, it can be useful to incorporate security testing from the start, as automated tests can be configured to fail a code build if the tests do not pass [10].

The most popular automated vulnerability discovery technique for an application is currently represented by **Fuzzing, or Fuzz testing**. This entails in generating enormous amounts of random inputs, both expected and unexpected, and tries "to detect exceptions by feeding the generated inputs to the target applications and monitoring the execution states". The application may subsequently execute successfully, raise an exception, or crash. Compared with SAST or DAST, Fuzzing is easier to deploy, has higher extensibility, applicability, and accuracy in actual testing execution, larger scalability and can be carried out with or without the source code. Thanks to this, despite having many disadvantages including the low efficiency and code coverage, Fuzzing has become the most effective and efficient state-of-the-art vulnerability discovery technique [13] [14]. One testing tools that can be used for Fuzzing is **APIFuzzer**, a public HTTP API Testing Framework that reads the API description and, fuzzes the fields to validate if the application can cope with the fuzzed parameters [19].

4. Conclusions

In conclusion, with the increase in cyberattacks, it became critical to implement multiple security measures at both organizational and technology control levels, to protect sensitive data from breaches. Adding multiple layers of security through Defense in Depth is offering stronger protection against potential threats.

However, even with such measures in place, performing security testing remains crucial to find and fix weaknesses as early as possible in the SDLC process, since early testing can prevent costly defects and improve the overall quality of the product. Moreover, supplementing manual testing with automated testing, using different testing tools, and combining internal testing with testing from

external sources, ensure that most of the possible testing scenarios are covered, mitigating risk in software by decreasing the number of potential undetected weaknesses and vulnerabilities.

References

[1]. "Impact of COVID Pandemic on eCommerce", International Trade Administration 10 2021. Available: https://www.trade.gov/impact-covid-pandemic-ecommerce.

[2]. "eCommerce - Worldwide", Statista – The Statistics Portal for Market Data, Market Research and Market Studies, 02 2023. Available: https://www.statista.com/outlook/dmo/ecommerce/worldwide.

[3]. "What Is Cybersecurity?" Gartner, 18 11 2021. Available: https://www.gartner.com/en/topics/cybersecurity.

[4]. "Security Testing: Types, Tools, and Best Practices" Bright Security, 22 05 2022. Available: https://brightsec.com/blog/security-testing/.

[5]. "ISO/IEC 27000:2018(en) Information technology – Security techniques – Information security management systems" International Organization for Standardization, 2018. Available: https://www.iso.org/obp/ui/#iso:std:iso-iec:27000:ed-5:v1:en.

[6]. B. Hambling, P. Morgan, A. Samaroo, G. Thompson and P. Williams, "Software Testing, An ISTQB–ISEB Foundation Guide, Second Edition", British Informatics Society Limited, 2010.

[7]. "SQL Injection", The Open Worldwide Application Security Project. Available: https://owasp.org/www-community/attacks/SQL_Injection.

[8]. "Session hijacking attack", The Open Worldwide Application Security Project. Available: https://owasp.org/www-community/attacks/Session_hijacking_attack.

[9]. V. Garousi and F. Elberzhager, "Test Automation: Not Just for Test Execution" in IEEE Software, vol. 34, no. 2, pp. 90-96, 28 03 2017, doi: 10.1109/MS.2017.34.

[10]. "OWASP Security Culture", The Open Worldwide Application Security Project. Available: https://owasp.org/www-project-security-culture/v10/7-Security_Testing/.

[11]. "Tokenization, Encryption, and Secure Payment Processing", TrueMerchant. Available: https://truemerchant.com/tokenization-encryption-and-secure-payment-processing/.

[12]. S. Scott and G. Neray, "Best practices for REST API security: Authentication and authorization", The Overflow, 06 10 2021. Available: https://stackoverflow.blog/2021/10/06/best-practices-for-authentication-and-authorization-for-rest-apis/.

[13]. J. Li, B. Zhao, and C. Zhang, "Fuzzing: a survey" in Cybersecurity 1, 05 06 2018. Available: https://doi.org/10.1186/s42400-018-0002-y.

[14]. P. Raghu and J. Agrah, "Practical Security Testing of Electronic Commerce Web Applications" in International Journal of Advanced Networking and Applications, 01 08 2021, doi: 10.35444/IJANA.2021.13109.

[15]. Github, SonarSource / sonarqube. Available: https://github.com/SonarSource/sonarqube.

[16]. Github, JosefPihrt / Roslynator. Available: https://github.com/JosefPihrt/Roslynator.

[17]. Github, dotnet / roslyn. Available: https://github.com/dotnet/roslyn.

[18]. Gitlab Docs, "Dynamic Application Security Testing (DAST)". Available: https://docs.gitlab.com/ee/user/application_security/dast/.

[19]. Github, KissPeter / APIFuzzer. Available: https://github.com/KissPeter/APIFuzzer.