



## Assignment of bachelor's thesis

<b>Title:</b>	Security Analysis of GoOut
<b>Student:</b>	Kryštof Rohan
<b>Supervisor:</b>	Ing. Josef Kokeš, Ph.D.
<b>Study program:</b>	Informatics
<b>Branch / specialization:</b>	Information Security 2021
<b>Department:</b>	Department of Information Security
<b>Validity:</b>	until the end of summer semester 2024/2025

### Instructions

The goal of the thesis is to perform a security analysis of the web version of the GoOut application (<https://goout.net>).

- 1) Research the state-of-the-art of threat modelling and penetration testing of applications. Focus particularly on web applications (e.g. OWASP Top Ten).
- 2) Study the GoOut application from the point of view of an external user. Identify potential threats using a suitable methodology.
- 3) Perform a black-box penetration test of the application.
- 4) List discovered vulnerabilities, evaluate their seriousness.
- 5) Focusing on the most serious vulnerabilities, provide recommendations for a fix or mitigation (where possible).



Bachelor's thesis

# SECURITY ANALYSIS OF GOOUT

**Kryštof Rohan**

Faculty of Information Technology  
Katedra informační bezpečnosti  
Supervisor: Ing. Josef Kokeš, Ph.D.  
May 7, 2024

Czech Technical University in Prague  
Faculty of Information Technology

© 2024 Kryštof Rohan. All rights reserved.

*This thesis is school work as defined by Copyright Act of the Czech Republic. It has been submitted at Czech Technical University in Prague, Faculty of Information Technology. The thesis is protected by the Copyright Act and its usage without author's permission is prohibited (with exceptions defined by the Copyright Act).*

Citation of this thesis: Rohan Kryštof. *Security Analysis of GoOut*. Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2024.

# Contents

<b>Acknowledgments</b>	<b>v</b>
<b>Declaration</b>	<b>vi</b>
<b>Abstract</b>	<b>vii</b>
<b>List of abbreviations</b>	<b>viii</b>
<b>Introduction</b>	<b>1</b>
<b>1 Key Penetration Testing Concepts</b>	<b>2</b>
1.1 Background . . . . .	2
1.2 OWASP Top 10 . . . . .	3
1.2.1 Broken Access Control . . . . .	3
1.2.2 Cryptographic Failures . . . . .	4
1.2.3 Injection . . . . .	4
1.2.4 Insecure Design . . . . .	5
1.2.5 Security Misconfiguration . . . . .	5
1.2.6 Vulnerable and Outdated Components . . . . .	6
1.2.7 Identification and Authentication Failures . . . . .	6
1.2.8 Software and Data Integrity Failures . . . . .	7
1.2.9 Security Logging and Monitoring Failures . . . . .	7
1.2.10 Server-Side Request Forgery . . . . .	8
1.3 OWASP API Security Top 10 . . . . .	8
1.3.1 Broken Object Level Authorization . . . . .	9
1.3.2 Broken Authentication . . . . .	9
1.3.3 Broken Object Property Level Authorization . . . . .	9
1.3.4 Unrestricted Resource Consumption . . . . .	10
1.3.5 Broken Function Level Authorization . . . . .	10
1.3.6 Unrestricted Access to Sensitive Business Flows . . . . .	10
1.3.7 Server Side Request Forgery . . . . .	11
1.3.8 Security Misconfiguration . . . . .	11
1.3.9 Improper Inventory Management . . . . .	11
1.3.10 Unsafe Consumption of APIs . . . . .	11
1.4 Threat Modeling . . . . .	12
1.5 Common Vulnerability Scoring System v3.x . . . . .	12
<b>2 GoOut</b>	<b>13</b>
2.1 Mission and Vision . . . . .	13
2.2 Components and Technologies . . . . .	14
2.3 Defined Scope and Testing Notes . . . . .	17

<b>3</b>	<b>The Penetration Test</b>	<b>18</b>
3.1	Burp Suite	18
3.2	Information Gathering	19
3.3	OWASP Top 10 Testing	19
3.3.1	Broken Access Control	19
3.3.2	Cryptographic Failures	20
3.3.3	Injection	20
3.3.4	Insecure Design	21
3.3.5	Security Misconfiguration	22
3.3.6	Vulnerable and Outdated Components	23
3.3.7	Identification and Authentication Failures	24
3.3.8	Software and Data Integrity Failures	25
3.3.9	Security Logging and Monitoring Failures	25
3.3.10	Server-Side Request Forgery	26
3.4	OWASP Top 10 Testing Evaluation	26
3.5	OWASP API Security Top 10 Testing	26
3.5.1	Broken Object Level Authorization	26
3.5.2	Broken Authentication	27
3.5.3	Broken Object Property Level Authorization	27
3.5.4	Unrestricted Resource Consumption	28
3.5.5	Broken Function Level Authorization	28
3.5.6	Unrestricted Access to Sensitive Business Flows	28
3.5.7	Server Side Request Forgery	29
3.5.8	Security Misconfiguration	29
3.5.9	Improper Inventory Management	31
3.5.10	Unsafe Consumption of APIs	31
3.6	OWASP API Security Top 10 Testing Evaluation	32
3.7	Recommendations	32
3.7.1	“Locking out” the entire sale without any purchase	32
3.7.2	Accessible DEV Environment	32
3.7.3	Identification and Authentication Failures	33
3.7.4	XSS	33
<b>4</b>	<b>Conclusion</b>	<b>35</b>
<b>A</b>	<b>Evaluated vulnerability findings</b>	<b>36</b>
	<b>Contents of the attachment</b>	<b>41</b>

## List of Figures

2.1	Diagram showing the sets of rights for user type in GoOut . . . . .	14
2.2	Activity feed component of GoOut web application . . . . .	15
2.3	Hall sale-form component of GoOut web application . . . . .	16
3.1	Proof of concept of the invalid phone being saved in the database . . . . .	22
3.2	Individual organizer account permissions settings in GoOut Admin . . . . .	23
3.3	Error and warning messages logged in chat by Slack integration . . . . .	26
3.4	API Misconfiguration issue leading to an informational disclosure . . . . .	31

## List of Tables

1.1	CVSS v3.1 Ratings table conversion . . . . .	12
3.1	Penetration test findings based on OWASP Top 10 . . . . .	19
3.2	Penetration test findings based on OWASP API Security Top 10 . . . . .	27
A.1	Full list of found vulnerabilities with their CVSS vector string . . . . .	36

## List of code listings

3.1	Function responsible for sending group SMS to an external gateway . . . . .	21
3.2	Malicious HTML form exploiting the CSRF vulnerability . . . . .	25
3.3	Commented out code checking for invalid password reset hashes . . . . .	25
3.4	Payload in response from /services/user/v1/accountExists endpoint . . . . .	28
3.5	Required JSON payload structure to exploit the BFLA . . . . .	29
3.6	Payload send via HTTP POST to /services/saleform/purchase/v1/pay . . . . .	30
3.7	Thrown error contained in JSON response . . . . .	30
3.8	Example of a counting expression as the rate limiting . . . . .	33
3.9	Vulnerable code to XSS . . . . .	34
3.10	Properly escaped code preventing XSS . . . . .	34

*I would like to express my sincerest gratitude to my supervisor, Ing. Josef Kokeš, Ph.D., who has consistently served as an exemplary role model for his profound depth of knowledge, his capacity to impart valuable advice, and his commitment to sharing his expertise. In the spirit of gratitude and respect, I extend my sincerest thanks to GoOut for their constructive and forward-thinking approach to this thesis. In particular, I would like to express my gratitude to David Zettl, who provided me with the opportunity to achieve my full potential. Furthermore, I would like to express my gratitude to the Boualay family for their invaluable assistance in proofreading this thesis, which has significantly enhanced the quality of the text. In closing, I would like to express my gratitude to my family for their unwavering love and support, particularly during those moments when I challenged their beliefs in security.*



## Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular the fact that the Czech Technical University in Prague has the right to conclude a licence agreement on the utilization of this thesis as a school work pursuant of Section 60 (1) of the Act.

In Praze on May 7, 2024

## Abstract

Information security is undoubtedly a key concept in the internet. This thesis focuses on security of one particular web application, GoOut, made by a company that differencies in the ticketing field. In order to strengthen its security, a penetration test of the web application and the API endpoints is performed. The penetration test is using a black box approach with a primary focus on the OWASP resources, namely the Top 10 lists. Vulnerabilities and other potential findings are discussed and rated based on their severity. Although no critical vulnerabilities are found, there are a few other discoveries such as problems in the authentication or potential lockout of the sale. The contribution of this thesis is to help GoOut securing their web application by providing findings of the penetration test with possible remediation and also provide insight to anyone how safe and secure it is.

**Keywords** security analysis, OWASP, web application security, GoOut, API security

## Abstrakt

Informační bezpečnost zcela jednoznačně představuje klíčový koncept v celém internetu. Tato bakalářská práce se zaměřuje na zabezpečení jedné specifické webové aplikace, GoOut, která je vytvořena unikátní firmou v odvětví prodeje vstupenek. S cílem posílit zabezpečení je proveden penetrační test webové aplikace a API endpointů. Tento penetrační test využívá black box přístupu a zaměřuje se primárně na zdroje firmy OWASP, jmenovitě Top 10 listy. Zranitelnosti a další potenciální nálezy jsou diskutovány a ohodnoceny na základě jejich závažnosti. I když žádná kritická zranitelnost není nalezena, je zde několik jiných nálezů, jako například problémy s procesem autentizace nebo potenciální uzamčení prodeje. Přínosem bakalářské práce je pomoc GoOut k zabezpečení webové aplikace poskytnutím nálezů penetračního testu s možnými nápravami a zároveň umožnit komukoli náhled do její bezpečnosti.

**Klíčová slova** bezpečnostní analýza, OWASP, bezpečnost webových aplikací, bezpečnost API, GoOut

## List of abbreviations

2FA	Two-factor Authentication
ACL	Access Control List
API	Application Programming Interface
B2B	Business-to-business
B2C	Business-to-consumer
BFLA	Broken Function Level Authorization
BOLA	Broken Object Level Authorization
CAPTCHA	Completely Automated Public Turing test to tell Computers and Humans Apart
CMS	Content Management System
CORS	Cross-Origin Resource Sharing
CPU	Central Processing Unit
CVE	Common Vulnerability and Exposures
CVSS	Common Vulnerability Scoring System
DEV	Development
DNS	Domain Name System
DoS	Denial of Service
GDPR	General Data Protection Regulation
HMAC	Hash-Based Message Authentication Code
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
ID	Identification
IT	Information Technology
IP	Internet Protocol
JSON	JavaScript Object Notation
JWT	JSON Web Token
NVD	National Vulnerability Database
OSINT	Open-Source Intelligence
OWASP	Open Web Application Security Project
SHA512	Secure Hash Algorithm 512-bit
SMS	Short Message Service
SSH	Secure Shell Protocol
SQL	Structured Query Language
SSO	Single Sign-On
SSRF	Server-Side Request Forgery
TLS	Transport Layer Security
TOR	The Onion Router
URL	Uniform Resource Locator
UX	User Experience
VPN	Virtual Private Network
WAF	Web Application Firewall
WSTG	Web Security Testing Guide
XSS	Cross-site Scripting

# Introduction

Information technology has become an inseparable part of our daily lives. We have slowly but surely moved our identities, finances, hobbies, and jobs online. While technology was initially intended to simplify processes, there will always be those who exploit it for their own gain. Web applications are no exception; even if they are used by thousands of users, there are still numerous ways in which they can be potentially attacked and, worse, exploited.

Cybersecurity plays a key role in the hierarchy. Bad actors use it for their unethical purposes while others use it to prevent them from doing so. This is also my main motivation: to use my knowledge to make the world a better and safer place. In particular, to secure a specific web application, GoOut, a culture guide that helps connect people with culture.

I hope that this thesis will not only be an excellent resource for other students of security but also a good indication of the journey GoOut has gone through to achieve an even more robust security for its users, including me personally.

The primary goal of the research part is to explore the possibilities of web application penetration testing, mainly resources provided by OWASP. This includes choosing and analyzing existing concepts, such as suitable methodology or a standardized vulnerability scoring system. The next goal is to dive into the GoOut web application and analyze its components and functionalities before choosing suitable tools for security testing.

In the practical part, the objective is to conduct a penetration test of GoOut in order to strengthen its security. The goal is to identify vulnerabilities, if any, and analyze their severity and possible threats related to them. Once analyzed, each has to be evaluated by its severity based on the GoOut business and technical perspective. The final objective is to provide recommendations on how to fix or mitigate any vulnerabilities found.

# Key Penetration Testing Concepts

*This chapter focuses on the current state of internet security, with a particular emphasis on the web application security. It explains why security is such a significant issue and covers two well-known publications by the nonprofit security organization OWASP that focus on common mistakes and vulnerabilities found in web applications. It also covers key concepts like threat modeling and consistent vulnerability severity scoring.*

## 1.1 Background

Humanity has made a significant progress in recent decades, adapting to new challenges and developing processes not only to overcome failures but also to prevent them once and for all. Namely, medicine could have been considered black magic just a few hundred years ago, what no one expected to become a daily routine for numerous job positions – scientific research, transportation, medical guidance, doctors, and many others nowadays. It might not be surprising that there is always an enormous amount of testing done before any medical product can be used outside of the laboratories as this type of testing is standard in today’s world. In fact, the testing process can be seen in almost every other field as well, improving the quality of products or services while also preventing potential consequences of failure.

The IT field is mostly not an exception; on the other hand, product testing does not necessarily include testing from the security perspective. As one of the fastest growing fields, it creates a potential risk with a considerable impact as IT is used everywhere in our daily lives. We use computers, software, websites, and hardware to communicate, work, transport, keep records, learn, relax, deal with business, transfer money, calculate, etc. IT security testing, which includes the penetration testing process, is often skipped or completely ignored even though people’s health and money depend on it. To make the situation even worse, it is estimated that web applications were targeted by approximately 172 cyber-attacks[1] per day during 2022.

Penetration testing[2] aims to reveal vulnerabilities and weak parts of tested components mimicking real-world attacks using the actual tools and methods the attackers would use. Penetration testing can be used in many scenarios, differing in scope, approach, and time. Scope defines which components should be tested and which ones should be left untouched. The testing approach also plays a crucial role. An attacker can access the tested components either from the wide internet without any previous knowledge of their functionalities, related components, or used resources (known as black box testing) or with everything revealed, connected, and explained, such as source code, flow charts of the connected components, and documentation of the

functionality (known as white box testing). Eventually, gray box testing, an approach combining both previously mentioned approaches, might also be used. Finally, time defines the length of the whole penetration process — in hours, days, or weeks.

For example, a bank may have developed a new blog for its customers that requires testing before publishing. A two-day penetration test is planned to use a black box testing approach, allowing only the mentioned blog to be tested while restricting the attacker from testing other internal bank systems or network communications. The outcome of the penetration test will be a report, not only containing findings but also recommended tips to mitigate them. Both the development and management teams should read the report to resolve everything found in the application. Once resolved, the bank can also use the report from the marketing perspective as a proof of its system's hardening.

As with any other process, penetration testing also has its methods to do it sufficiently. These methods, again, depend on the approach, length, and scope of the testing. Namely, a company called OWASP focuses on researching potential attack vectors, attack methods used in the wild, as well as the frequencies of vulnerabilities found in the area of web application security. Based on this research, they create resources for organizations and penetration testers worldwide. [3]

## 1.2 OWASP Top 10

One of their incredibly recognized publications is OWASP Top 10. A list of the top ten critical security risks of web applications based on broad research, updated once every few years to always stay up to date with the developing IT field. Unlike OWASP Web Security Testing Guide (WSTG), which is often used as a framework by penetration testers, OWASP's Top 10 should not stand as the only methodology provided from the security perspective. Whereas WSTG covers almost every aspect of web application testing, the Top 10 highlights only what should never be overlooked. However, if nothing else were tested, the Top 10 definitely would have helped to harden the web application's security. [4]

To gain a comprehensive understanding of the top ten security risks, the following sections discuss each item of the Top 10 in depth with relevant examples.

### 1.2.1 Broken Access Control (A01:2021)

In everyday life??, it is common to encounter no trespassing signs or any other restrictive instructions. The question is: what prevents bad actors from doing the opposite of what they are told? Sometimes, nothing, as these restrictions are often intended to keep us safe from potential harm. Furthermore, if those restrictions were placed, for example, to prevent a customer from accessing the cash desk, bad actors would simply not care as long as there is no physical barrier, pin locking, or cameras. The very same applies to internet security. Of course, no one should be allowed to access the admin dashboard or the customer's personal information except the privileged person. On the other hand, can any written rule indeed prevent the bad actor from accessing the information? Without proper security measures in place, it is just a matter of time.

Broken Access Control moved up from the fifth position on the last list to the first place on the most recent list, due to its frequent oversight in practice. The following questions should always be asked:

1. Is the principle of least privilege<sup>1</sup> in place?
2. Is access denied by default with explicitly set exceptions, or in other words, is an allowlist in place?

---

<sup>1</sup>Least privilege stands for restricting everyone only to their required resources and nothing else.

3. Does the application security not depend on locally stored user data or any data which can be modified in any way from the user perspective?

Yet the last point might sometimes be necessary for the intended functionality. Therefore, an adequate solution is required, for example, signing data with asymmetric cryptography to ensure integrity.

A very simple, yet unfortunately very common, example of broken access control vulnerability is missing authentication or authorization checks on a particular page. Developers may have restricted access to the main dashboard, such as in the admin panel, yet they still need to restrict access to any other sub-pages connected to it. The most concerning aspect is that this attack can be executed successfully without requiring advanced tools or in-depth knowledge. An attacker can easily edit the URL in their browser and access content they were never intended to view. [5]

## 1.2.2 Cryptographic Failures (A02:2021)

It is well known that web applications contain an enormous amount of information. On the other hand, not everyone realizes how much of this information is actually sensitive. Term sensitive can be straightforwardly applied to credit card details or user credentials. However, there are several regulations, such as data protection and privacy laws like the EU's General Data Protection Regulation (GDPR), that also define what is sensitive and how these data should be handled or stored.

A visitor[6] has no idea which data they interact with are sensitive, except their personal data. This is also why social engineering is so common and successful. A common scenario is when an unsuspecting customer is asked to copy their browser request and provide it to a bad actor. There is always some social engineering beforehand to persuade the victim to do so. While it may not seem harmful, the victim could also be copying their sensitive session hash, which could result in account theft or worse.

In order to ensure the confidentiality of any provided data, cryptographic algorithms are being used. However, it is important to note that the use of cryptographic algorithms does not necessarily mean that transmitted data is indeed secure. Issues might arise essentially anywhere, with the type of algorithms or methods, their usage, or even the implementation in the code.

Starting with their type, usage of any old or weak cryptographic algorithms is problematic. Databases are often breached just to reveal how badly were the credentials handled and stored. Usage of deprecated cryptographic padding methods or deprecated hash functions[7] such as MD5 may still be seen. Even if strong and verified methods are being used, a single weak spot in the whole cryptography scheme would have a significant impact on the overall security. Not enforcing the usage of the cryptography, reusing initialization vectors or completely ignoring them, not validating provided certificates, relying on randomness that does not meet cryptographic requirements, or, in some cases, coding the whole cryptography algorithm manually will still result in a high risk of unsecure implementation. [8]

## 1.2.3 Injection (A03:2021)

Web applications are mostly not just hand-crafted static pages with nothing but information for visitors. For example, even online blogs that are actually based on providing information to visitors often allow some direct input from them, like a comment section below the blog post. Any application that allows user input exposes itself to a potential security risk. Not only is user behavior unpredictable, even though some existing behavior traits are known[9], but it is also sometimes malicious.

The mentioned comment section on the blog web application is probably intended as a space for visitors to share their opinions. What if there was a way to craft a malicious comment

containing a payload in its content that would steal any logged-in visitor's credentials upon just opening the blog post? Or what if a basic search bar used on an online e-shop could leak every single stored information from the server database with, once again, a specially crafted query? Both of these terrifying examples are actual instances of injection attacks in the wild. Unfortunately, those specially crafted payloads are often as simple as writing two lines of code.

User input may be inevitable; in that case, it is vital to validate and sanitize it properly by the application. Validation is often done on the front-end, which is an excellent feedback for any visitor struggling with application usage, like when investigating whether they were already registered or not. However, more is needed to guarantee that only validated data will come through. Those front-end validations or restrictions are done only locally, and as the attacker has full control over their local system, nothing prevents them from also skipping or disabling this security mechanism. To properly prevent possible injection attacks from succeeding is to sanitize and validate user input also on the back-end part before any further processing. However, the only way to properly mitigate the threat is to let go of user input; the fewer user input flows that exist, the better. [10]

#### 1.2.4 Insecure Design (A04:2021)

Having the best possible implementation of something will never solve the core problem if it was insecurely designed and needed security measures were never created or thought of in the first place. Imagine the described problem with an injection from the last chapter, where users were given the ability to write blog comments. The way their filled-in data will be handled and processed can be changed or secured anytime as it is basically a part of the implementation. In contrast, an insecure design example of the blog application would be when the accounts scheme allowed the existence only of the administrators, with no additional roles or hierarchy included at all. In that scenario, a visitor wanting to comment on a blog post would have to be either an administrator or not be able to comment at all. What about a user who the administrators wanted to become an article editor, yet not the full administrator? Of course, there are few workarounds, yet those are not optimal from both the security and sustainability points of view. The takeaway is to properly and securely design the product before, during, and after the implementation using the development cycle process.

Secure design should protect the confidentiality, integrity, and availability of data flows, business logic, and assets. Therefore, evaluating possible threats via threat modeling should be a part of the secure development cycle. In order to achieve an effective output[11], at least three people should be part of the threat modeling. One representative from the security team, one from the business team, and one from the technical team. Thanks to effective threat modeling, possible flaws in the product could be revealed and prevented before it is too late. [12]

#### 1.2.5 Security Misconfiguration (A05:2021)

Web applications cannot exist standing alone; in fact, they require many layers of technology. A dedicated computer, server, has to be hosting the whole instance. This instance is probably handled by virtualization, namely running in a container. Then there is a software used for the server logic and routing, consisting of many technologies one depending on another. Furthermore the web application itself might use many frameworks like Czech server-sided framework PHP Nette. A misconfiguration can exist in any possible layer, and as the layers are not limited in count, the potential for vulnerabilities always exists. [13]

The server hosting the target web application might also be running other web applications or even completely different services working on different open ports. Unlike the target web application itself, those services might be very simple to compromise. There could be a zero-day vulnerability or an already widely-known vulnerability, as the service itself could simply be outdated. Attacking services running on the same server could lead to privilege escalation,



therefore overtaking everything on the server, including the originally targeted web application. Leaking stack traces or any debug error logs to the public can both expose sensitive information and provide enough data for attackers to weaponize. Using default insecure settings or having a default account in the database with publicly known credentials also falls into this category. A network security company, LogicMonitor, suffered a security breach just a few months ago by providing its customers with default passwords, which were never forced to be changed. [14]

A straightforward solution would be to do the very opposite of what was mentioned in the previous paragraph. The primary strategy is to minimize the attack vectors by disabling or completely removing unused components while hardening all those remaining indispensable or necessarily existing ones. Prevent usage of any default credentials, and if necessary, always make sure to demand their change right after usage. Disable any debug outputs, like Tracy in PHP Nette, or stack traces otherwise used by developers in the production and restrict the development version, including production's and development's log files, from public access. [15]

### 1.2.6 Vulnerable and Outdated Components (A06:2021)

As already mentioned, web applications usually require many other dependencies or dependent components, such as libraries, for their functionality. Requiring any other component also means relying on its security, and with any other component used, the attack vector for the application will become wider. This is a common problem for a worldwide known content management system (CMS) called WordPress, used by 43.2% out of all websites[16]. Chances of being infected with any malware during 2022 while having more than twenty plugins installed are 147.66x higher than when compared with a brand new instance of WordPress without any plugin in use, based on SiteLock's Website Security publication. [1]

Sometimes, versions of depending components in use are not even known by the developers, and those versions are then even often outdated. Missing crucial patch updates for existing vulnerabilities, possibly already exposed in the wild, makes the web application an easy target. For example, as of 25. 2. 2024[17] the outdated versions of PHP 7 are still being used by over 50 % of web applications which use PHP as its main back-end core. Not to mention that 15 % of web applications use even older versions of PHP.

Developers working on web applications might change and vulnerabilities may arise at any time. Therefore, a frequent, continuous inventory of versions of the used components, both client-sided and server-sided, should be done. This way, it is simple to properly keep track of all possible gateways to potential security incidents. Online resources like Common Vulnerability and Exposures (CVE) listings or National Vulnerability Database (NVD) should be checked to ensure any vulnerabilities of the used components are identified that may become threatening in the future. A great example of components with already known vulnerabilities was published by Sonatype[18], based on their research over 23 % of recent log4j downloads were versions still affected by the log4shell vulnerability of 2021.

Even with a continuous inventory of used components, avoiding possible zero-day attacks is impossible. Therefore, limiting dependencies on components will also help to shorten the potential attack vector. Not every component will be updated forever and may eventually one day suddenly result in a backdoor containing code injected into the web application. Automated tools usually search for vulnerable and outdated components, so not even the most minor bug could stay unnoticed. [19]

### 1.2.7 Identification and Authentication Failures (A07:2021)

Authentication, verifying the identity of a user, process, or device, often as a prerequisite to allowing access to resources in an information system, is usually a key element in web applications.

Except for static informational web applications, most of them require at least some sort of administration panel or a private page which is behind the mentioned authentication.

Unfortunately, there are many types of attacks related to the authentication itself. Password guessing might probably be the simplest one to execute for attackers. It is resistant to any security policy except those that could, cause much more trouble to users than attackers. An example of that is account lockout which can be exploited in such a way as denial of service (DoS) to affect the component availability, preventing targeted registered users from logging in. Luckily, password guessing can be mitigated by disallowing default or weak passwords by enforcing strict password policies like testing against the top-used or worst passwords while verifying policies both on the client-side and server-side. Unlike password guessing, brute force password attacks can be mitigated or entirely prevented by implementing the CAPTCHA mechanism and rate limiting any login attempts. Yet password spraying, abuse of stolen credentials, and credential stuffing attacks could bypass these protections, so multi-factor authentication implementation and adequate failed attempts logs are always recommended as well.

With authentication itself being secured, there are still many other possible pitfalls. Both user sessions and authentication tokens have to be properly invalidated during logout or given a period of inactivity mainly on the server-side but also on the client-side. This information is sensitive, and the web application should never expose it in an unencrypted form, such as in the GET parameters. Exposure of any details related to the given account during registration, login, or credentials recovery processes, such as its existence in the database, should also be avoided. [20]

### 1.2.8 Software and Data Integrity Failures (A08:2021)

Software and data integrity failures, a contrast to vulnerable and outdated components, happen when an assumption is made that the third-party code is safe and secure. Nonetheless, even if those components were indeed secure and up-to-date, nothing stops attackers from possibly spoofing, injecting malicious payload, or in any other way affecting the third-party code imported into the web application.

Automatic downloads or updates are often enabled. Without proper integrity verification, like the usage of digital signatures, it is only a matter of time before attackers exploit this vulnerability and unnoticeably inject anything into the targeted web applications. Automatic updating should also be restricted only to verified trusted sources and repositories.

A notable example of a software supply chain attack, SolarWinds[21], occurred during 2019-2020 and brought enough attention to the previously not often used type of attack. SolarWinds' software, Orion, for monitoring performance and statistics, is used within many companies. In fact, SolarWinds' customers were the target of this notorious supply chain attack as attackers wanted to gain a persistent access to internal networks through their malicious code injected into an otherwise fully secure software. [22]

### 1.2.9 Security Logging and Monitoring Failures (A09:2021)

Attacks are definitely inevitable, and even though many threats can be mitigated, the attack vector shortened, or some known attack types explicitly prevented, there will never ever be a definite list of all possible attack types. New vulnerabilities might be found; new payloads might be invented. Strong security is one thing, but defense against the previously unknown attack is way more complicated.

No matter what, monitoring and security logging should always be a fundamental part of the defense. A straightforward scenario that could happen to any company, as it does not even require any in-depth security knowledge, is a recently fired colleague downloading and breaching customer details. Or like when ex-Google engineer, Linwei Ding, who was arrested on March 6 2024[23], for a stealing AI technology secrets and selling it to Chinese companies. Luckily, thanks

to successfully logging Linwei's access, it was possible to uncover his malicious actions. The question is, would his or anyone else's accesses be properly logged in any other web application?

Logging is not meant to provide an ability to blame a person responsible for changing the color of the navigation menu on the homepage. It is intended to have the ability to look back in time and understand what was going on at any given moment. Was the application compromised? How did the attacker get inside? What did they do in the application? What exactly could the attacker access? Do they have persistent access, or are they already prevented from accessing again? This brings up the following question: what is the log format? Can there be any malicious payload injected by the attacker? Are those logs persistent so the attackers cannot simply remove or tamper with them in any way once they, for example, escalate their privileges?

Monitoring adds another layer of awareness as, unlike the logs, it provides the opportunity to react to anomalies and security incidents in real time. Therefore, monitoring and regular evaluation of logs is necessary for everyone to possibly notice if the web application was even breached. [24]

### 1.2.10 Server-Side Request Forgery (A10:2021)

The resources the web applications require do not necessarily have to be stored locally. Communication with other services or servers is often needed, and in order to prevent bad actors from accessing what they are not supposed to access, proper security measures should be in place.

Internal services, databases, or endpoints are not accessible from the external network. Access to those should and is usually restricted by access control lists (ACL) and firewall rules. The problem arises when a node from the local area network, which could have been considered confidential, like the application server, can be in a way controlled by a remote attacker even without remote access control. Today's web application features may allow users to supply URLs for resources to fetch. Unlike previous attack types, server-side request forgery (SSRF) may occur when a user, in a way, defines to the server what and where to fetch, just like when controlling a puppet. What if the attacker asks the server to fetch locally stored environmental variables files, configurations, or any other possibly sensitive data? The same applies to the aforementioned internal-only services, which could be running in the same local area network, just on a different server.

User data must be, as always, carefully sanitized. If a feature of allowing visitors to fetch resources and data based on the provided URL or path is indeed required, a strict allowlist consisting of only the intended files and paths to be fetched should be in place. Usage of the opposite, a blocklist, is a bad practice as it could be circumvented at any time by automated tools or manually crafted payloads. While preventing possible impact, logging of those requests should also be implemented. [25]

## 1.3 OWASP API Security Top 10

In modern web applications, APIs[26] are playing a more and more important roles. They are used as customer-facing, partner-facing, or even as internal only. Therefore, a specially dedicated list OWASP API Security Top 10 was released, in 2023, to raise awareness of this often overlooked issue.

Even with the existence of OWASP Top 10, which shares a lot in common, API Security Top 10 is still recommended to check because unique security risks and vulnerabilities related only to API do exist. Sometimes, web applications are built initially stand-alone, and later mobile applications or other services requiring API endpoints are built upon them. As for that, inconsistencies, deficiencies in the implementation or just inadequately applied policies might occur, inevitably leading up to potential vulnerabilities. [27]

As with the previous list, the following sections also discuss each item of the API Security Top 10 in depth with relevant examples.

### 1.3.1 Broken Object Level Authorization (API1:2023)

Web applications also usually use databases for data storage and manipulation. In order to efficiently access one specific table's row in the database, a unique identification is necessary.

A blog, which allows visitors to create their own accounts, has to store them somehow. If registered account was given a unique ID like a number, there might also be some other additional functionalities manipulating this ID. For example, change password endpoint could require a parameter of account ID in order to reset only the desired account's password and no one else's. If there were no object-level authorization checks, to see whether that logged-in user has sufficient permissions to perform the requested action on the provided object, changing this ID in the request to anything else would result in changing anyone's password to the one given by an attacker. The attacker could then make a simple assumption that the admin ID could be the very first one ever created in the database, therefore equal to number one.

Broken Object Level Authorization (BOLA) may result in data loss or data disclosure. Manipulating with predictable numbers as user identifications is not recommended. A better approach would be to use hash values of registered usernames or emails, as those should be unique and unpredictable enough. [28]

### 1.3.2 Broken Authentication (API2:2023)

Broken Authentication is related to the Identification and Authentication Failures described in section 1.2.7. In API, a different authentication mechanisms might be used. Often, stand-alone tokens like JSON web tokens (JWT) are used. Tokens still require a proper validation for their authenticity, possibly expiration dates, and in the case of the mentioned JWT, even prevention of accepting those without a signature algorithm. Without a signature included, it is not possible to check whether they were modified by the attackers.

Yet, it is essential to keep track of all possible API authentication flows, including forgotten passwords. Proper rate limiting and logging should be in place, and sensitive operations like email changes should require additional password verification. [29]

### 1.3.3 Broken Object Property Level Authorization (API3:2023)

Web applications may store several properties per object type. Registered user accounts may be stored with the ID, email, IP address, password hash, timestamp of last login, but also with some personal information like birthdate, full name, or phone number. If a blog web application allowed visitors to view registered users' accounts with the only intention of displaying their latest activity on the blog, like when was the last time they left a comment on a post, only relevant properties from the user object should be included in the API response. Even if the front-end processed only the public information like the date of the last activity and username from the user object, other sensitive properties were still delivered to the attacker's device in the server's response. Allowing them to access personal information or any other possibly hidden properties of the user object.

Unfortunately, this vulnerability might not only cause informational disclosure. In particular, in scenarios where object properties are meant to be altered, user-provided types of properties might not be validated. Therefore, attackers could send a payload to the username change API endpoint, in which they would also include a property of `isAdmin` (supposing this property exists,

either by simply guessing or based on information gathered upon previous testing) equal to true. This could result in privilege escalation or account takeovers. [30]

### 1.3.4 Unrestricted Resource Consumption (API4:2023)

API endpoints vary in their functionality and resource requirements. A simple endpoint returning only the existence of a given username in the database will not bother anyone with its resource consumption. However, this does not apply to an API endpoint generating thumbnails based on the provided image in many resolutions. A different, yet related, scenario is an endpoint integrated with an external SMS gate, which does not overwhelm from the computing resources perspective but rather from the cost resources perspective. This endpoint could have been used as a 2FA when signing in, costing a few cents per request. Attackers could exploit this by making numerous 2FA requests from their created dummy accounts to cost the company extra money.

Each endpoint should have its own limits based on the intended functionality. For those endpoints representing potential cost resources, rate limiting based on business needs is required, while resources consuming endpoints need limitations of memory, CPU cores, file descriptors, and process usage. Furthermore, incoming parameters or payloads created by the visitor or potential attacker still require proper limitations and sanitization. [31]

### 1.3.5 Broken Function Level Authorization (API5:2023)

In contrast to BOLA from section 1.3.1, broken function level authorization (BFLA) is not related to the object's properties but rather directly to the API endpoint itself. The change password endpoint might be expected to be used by both the users and the administrators but not by non-signed-in visitors. The same applies to any admin-only related endpoint. In fact, modern web applications require many roles and specific permissions. WordPress[32] also allows usage of the roles concept and comes with six predefined roles: super admin, administrator, editor, author, contributor, and subscriber.

A complex structure of roles could contain flaws, resulting in regular users having access to higher privileged endpoints. This can also occur when the endpoint is not exposed by itself yet is still an active part of the administration. Nothing prevents attackers from guessing the URL path to that endpoint, and due to the lack of authentication and authorization checks, data disclosure, data loss, or data corruption might occur. [33]

### 1.3.6 Unrestricted Access to Sensitive Business Flows (API6:2023)

Business flows play a key role in every web application. The application provides either a product, a service, or both. Attacks targeting business flows do not necessarily harm the web application itself. For example, a cinema web application provides the service of reserving and purchasing seats for screening. Attackers could block out the whole hall using VPN and proxies to reserve each seat one by one. A different scenario of attack with the intention to spoof the trustworthiness could be the creation of a fake company with a listing of counterfeit products on business-to-business (B2B) or business-to-consumer (B2C) markets like AliExpress and boosting its confidentiality by ordering its own products from fake accounts to leave very positive feedback. Once enough trustworthiness is spoofed, real customers could be tricked into ordering expensive items just to get scammed.

The main problem of business flow attacks is that there is no uniform solution. The first step should always be to identify those flows and evaluate their possible impact on the whole business. From the engineering side, implementing human verification processes mitigates, yet does not

prevent, those types of attacks. Usage of rate limiting, email or phone identity verification, CAPTCHA, and blocklisting known VPNs, proxies, or TOR exit nodes will help. [34]

### 1.3.7 Server Side Request Forgery (API7:2023)

A section 1.2.10 in OWASP Web Top 10 was already dedicated to SSRF. The vulnerability itself is very common, even in API endpoints, as those commonly work with webhooks and SSO. Both the impact and the prevention are, at the core, the same. [35]

### 1.3.8 Security Misconfiguration (API8:2023)

Security misconfigurations were also discussed in OWASP Top 10. Nevertheless, even those are also related to the API endpoints. An API stack depends on many components requiring proper security hardening and staying up-to-date with the latest security patches.

Communications with the API endpoints, both external and internal, have to be ensured of being transmitted over an encrypted communication channel (TLS). API endpoints must be strictly configured to include sufficient security headers like Cross-Origin Resource Sharing (CORS) policy in communications and also correctly handle caching. Improper configuration of the HTTP response header might expose sensitive user data, as the absence of a `Cache-Control` header could lead to unintentionally locally storing sensitive information like customer details (if they were previously accessed by the victim during their latest session on the same device). [36]

### 1.3.9 Improper Inventory Management (API9:2023)

API endpoints grow over time[26]. Minor changes can be made at any time; on the other hand, significant changes or changes directly in the web application model structure require the endpoint to be reworked from scratch. As many dependent services might already work with the endpoint, the new version is often released separately, with the original only marked as deprecated, providing external parties enough time to update their dependence before completely shutting the old deprecated endpoint down and replacing it with the new one.

The issue arises when those old versions of endpoints are forgotten and lost in time. This creates a massive opportunity for attackers to try to exploit them, as they might use weaker security measures or even contain vulnerabilities in their implementation. Without proper inventory management, not only is it hard to keep track of all possible endpoints and their versions, but it is also difficult to patch possible vulnerabilities. Therefore, an inventory with a continuous plan of reviews and updated documentation is required to shorten the attack vector. API endpoints with unclear purpose and use cases should be revised. [37]

### 1.3.10 Unsafe Consumption of APIs (AP10:2023)

As with the user-provided data, the assumption that any third party's provided data is safe is, in fact, an insecure approach. API endpoints might rely on third-party APIs and, even worse, trust any data received from them. A quite amusing yet unpleasant example is an endpoint that allows the user's account to be connected to an external social network platform and fill in its information based on data received from an external provider, like the chosen username on the external platform. Attackers could create a fake account on a social networking platform with a username equal to an SQL injection payload and request that the vulnerable web application API connect to it. By mindlessly trusting the data received without further sanitization and validation, information disclosure or worse could occur. [38]

## 1.4 Threat Modeling

Even with a suitable testing methodology in mind, security does not exist independently. There will always be a context as to what is required for securing, how or where it is used, and so on. Without a proper plan, spending days on improving the already secure data in the database itself while the database uses default credentials such as `admin:admin` probably speaks for itself. This is where a threat modeling process comes in really handy. It maps components of the web application and potential threats to it. Those may include internal bad actors, like a fired colleague with access to the system, but also a corrupted database as an outcome of a natural disaster. The result of the threat modeling is a prioritized list of possible security improvements or, rather, required changes to be done in the web application. Threat modeling is often used as a continuous cycle, so any changes or updates from both components and possible threats are still up to date. [39]

## 1.5 Common Vulnerability Scoring System v3.x

Vulnerabilities may arise at any time and in any scale. Yet, not every vulnerability is the same. Security holes like improper user data sanitization resulting in SQL should be the highest priority. In contrast, inadequate authentication policies like checking against the top 10000 common passwords may wait a few weeks without any significant impact. As the number of vulnerabilities can also be overwhelming, a strategy has to be chosen appropriately in order to mitigate every single one of them optimally.

Every company differs in the services and products it provides. This also affects the business impact of possible vulnerabilities in their systems, therefore changing the whole strategy chosen for sufficient vulnerability mitigation, including its order of completion. However, without a broad context of the inner components from both the business and technical sides, a strategy may be impossible to define. This is where a common vulnerability scoring system (CVSS) comes in with its solution.

CVSS, created by the National Institute of Standard and Technology, is a method used to supply a measure of severity for given vulnerabilities by a standardized CVSS score, used widely by industries, organizations, and governments that need accurate and consistent vulnerability severity scores. CVSS comes with three metric groups: Base, Temporal, and Environmental. The final score ranges from 0 to 10 and is calculated by a given Base metric, which is then affected by both Temporal and Environmental metrics. Those metrics reflect the impact of a vulnerability on confidentiality, integrity, and availability while also reflecting the complexity of attack execution and its attack vector. The CVSS score can be represented by its numerical value, by a single word substitution based on table 1.1, or by a compressed textual representation, a vector string. [40]

■ **Table 1.1** CVSS v3.1 Ratings table conversion

Severity	Severity Score Range
None	0.0
Low	0.1-3.9
Medium	4.0-6.9
High	7.0-8.9
Critical	9.0-10.0



# Chapter 2

# GoOut

*The purpose of this chapter is to explain in detail how GoOut works and how it differs from other companies in the same industry. The chapter also covers the technical and business aspects of user roles and their related components integrated into their web application.*

## 2.1 Mission and Vision

The claim that there are countless cultural events around the world every year may not be too surprising. These can range from huge music festivals, international sports matches and gala evenings to small local art exhibitions or workshops. On reflection, it should be evident that the organizers' costs can vary significantly. After all, only some theatres can afford the same quality and robust software solution, such as its own system with sales, administration, and ticket management of its performances, compared to, for example, a multinational cinema chain.

GoOut addresses this problem. Their original intention was just to record all cultural events in the Czech Republic, thanks to which it can still be used as a search engine for cultural events without the need for registration in many countries, including the Czech Republic. Since 2014, which has been clarified in a conversation in 2023 with one of the original founders of GoOut, Vojtech Knyttl, they have even started selling tickets and providing a complete solution for organizers. The solution includes, among other things, event management, ticketing including many payment methods, customer support, remarketing and tracking codes, advanced financial statistics, social media promotion, widgets allowing the organizers to also integrate GoOut saleform with their website, web development, and many more. Some of those features are new, whereas others have been in place since 2014; in the end, the GoOut web application is still growing and may come with even more features in the future. Unlike other competitors in the ticketing industry, GoOut differentiates itself by not only selling tickets but also connecting people to the actual culture. It is not common for a seller to include events for which they do not sell tickets. [41]

It is possible to register to the GoOut platform either as an organizer whose capabilities have been mentioned in the previous paragraph or as a regular user. Functions for regular users include an easy discovery of new events based on their preferences or previously attended events or the use of a manual search. Searching can be done directly using event name, organizer, artist, or location, as well as by individual categories of music genres. Users have their own personal profiles, which they can hide from the public. Users can also gain cultural guide status if they actively attend and follow cultural events. It is also possible to discover new events just by following other users. The activity of the followed users is then displayed on the activity feed, so it is possible to find many exciting events just by scrolling without the need to search for them manually.



Regular users, as well as any visitors without registration, can purchase tickets for events contracted with GoOut directly using the web interface of the web application. What is worth mentioning, however, is that the overall payment integration is provided by an external company called PayU[42], which complies with the European Union regulations, so any sensitive data is never handled by the GoOut application. The existence of the registered user account allows one to view previously purchased tickets, including those that were bought before the actual account registration (if the provided email on the tickets corresponds with the registered one). The whole web application also comes with dedicated mobile applications both for Android and iOS. Registered users can view their tickets or discover new exciting events at any time, simply having a culture in the palm of their hand.

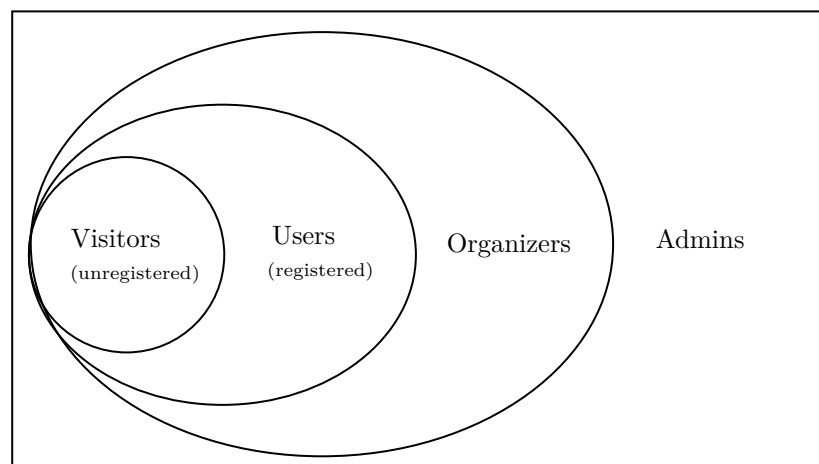
## 2.2 Components and Technologies

In order to perform the penetration test focused on OWASP's Top 10 methodologies, a brief overview of which components in conjunction create the web application and how they might be important from the business perspective is crucial. Therefore, I opened a discussion about these matters directly with GoOut, which is my primary source for the following findings.

Components within GoOut are not necessarily accessible to everyone as some privileges might be required. There are essentially four types of users:

1. unregistered visitors,
2. registered users,
3. organizers,
4. and admins.

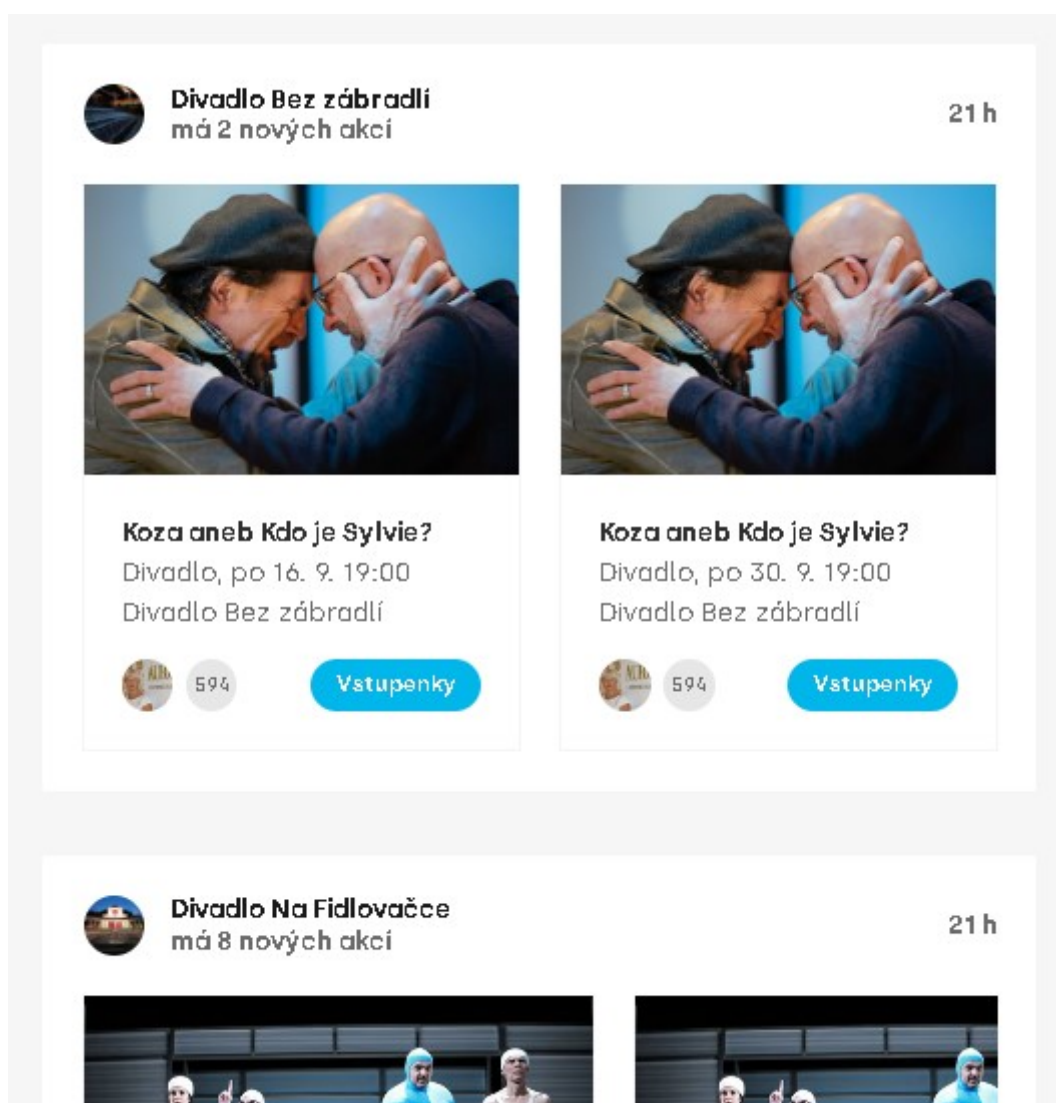
■ **Figure 2.1** Diagram showing the sets of rights for user type in GoOut



Each level unlocks additional components to work with. Admin accounts can interact with every single one, whereas organizers cannot access the admin-only settings like creating new events or sales, illustrated in the figure 2.1. Before diving further into each category, it is important to note that becoming an organizer in GoOut is not necessarily a simple process. Anyone representing an individual or a legal person can fill in a special organizer registration form to apply for cooperation. This form is then manually reviewed by someone from GoOut to decide whether the cooperation could be beneficial for both parties. If that is the case, the contact

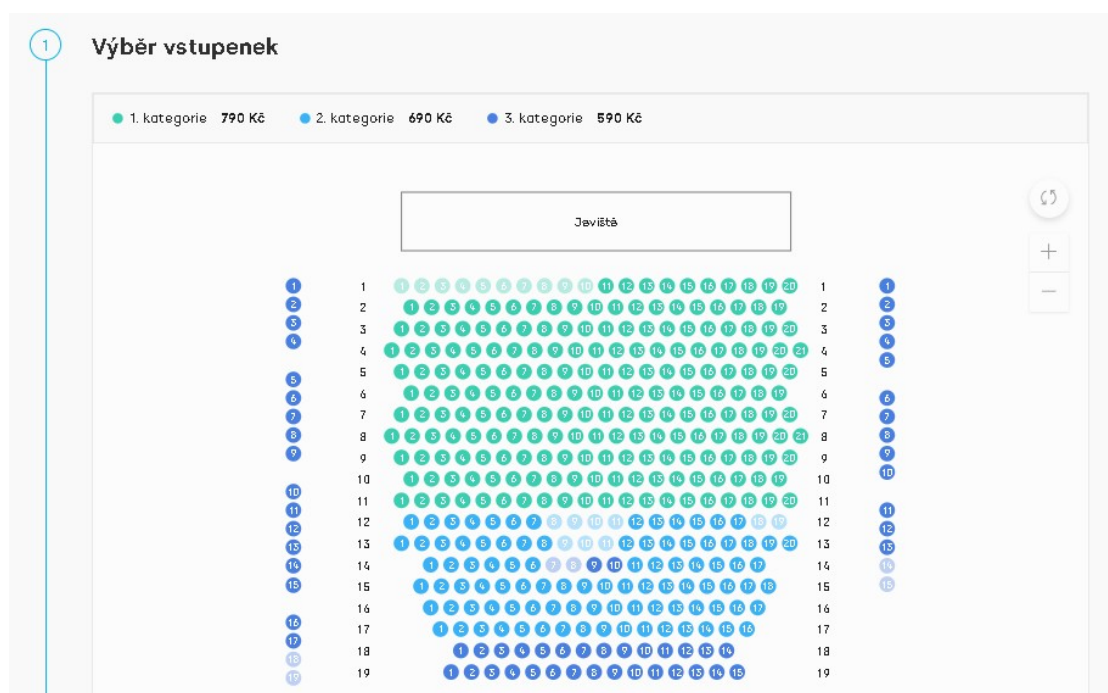
person is contacted to settle details about the cooperation, all before any access to the GoOut Admin is set up. The access is then based on roles given to each person from the organizing group if there are any other people needing the access as well. So simply said, getting access to the organizers features is definitely a complicated process for attackers to perform on their own, on the other hand it might be simple for someone who is working on behalf of an already-verified organizer group.

Now, when roles are clearly defined, let me start with the lowest privilege, unregistered visitor's view: Upon entering the homepage the activity feed section is loaded as shown in figure 2.2. The fetched feed in this section is based on many criteria, including interesting cultural events or recommended events by cultural guide users, whereas for registered users it is mainly based of their past attended events, liked events, followed performers, venues or other users. As a result, newly registered users might have their activity feed empty. However, for the sake of everyone else, the activity feed presents a particular load on the server. A very similar component is a search bar, allowing everyone to manually search for desired sales by their name or by listing specific artists, venues, or institutions.



■ Figure 2.2 Activity feed component of GoOut web application

Once a specific event's sale is selected, the sale-form component appears as shown in figure 2.3. This component plays a vital role from a business perspective and is also accessible to all user types. What is being presented as a simple HTML form, in reality, hides a complex system behind it. There are simple sale forms, like for a gallery exhibition, and then there are hall sale forms allowing users to select a specific seat from the plan of a theater. Seasoned sale forms combine multiple hall functionality like theatre subscriptions. Sale-form consisting of a several other sale-forms displayed as one, mainly for music festivals like RockForPeople, allowing users to not only select tickets for the event but also additional services like camping, parking, tents, and so on. Sale-form itself is the last step of conversion marketing, as the ticket purchase process can only be done through it.



■ **Figure 2.3** Hall sale-form component of GoOut web application

The registration process is a gate to account creation. The only requirement to advance in privilege is to provide a valid email address. Once registered, the user can access components related to account settings. Those settings include profile picture upload, bio status field, visibility of the account, connection to Google Calendar, and most importantly, the ability to view purchased tickets and their receipts. Users are also able to favorite any existing venue, artist, or event or follow other users on GoOut. Unlike other platforms, there are no comment or review sections available for users to create their own content unless they become organizers who are partially enabled to create certain content, like sale categories.

The last component is called GoOut Admin. This component stays hidden from most of the users as it is accessible only by selected organizers<sup>2</sup> or GoOut employees. I should note that this component itself is very complex and is created out of many other components, as everything needed for the administration is located there. Listings of existing transactions with many possibilities of their filtering, settings to set up a new sale or manage existing ones, or settings of sale categories, like changing their visibility, creating unique voucher codes, setting up a maximum number of tickets per the given category and so on, the same applies to events themselves and their connections to venues, performers, or institutions, it is also possible to

<sup>2</sup>Their roles might differ from one another.

manage organizers and their permissions by directly managing their roles. Even if the organizer is granted all possible access from the roles they can be given, they will still be restricted to access only their related sales, unlike admins who are not limited in their activities in GoOut Admin. For internal reasons, I am not able to disclose every single existing part of the GoOut Admin, but if the part is related to a possible vulnerability, it will be covered later to properly understand its scope and impact.

## 2.3 Defined Scope and Testing Notes

Upon thorough technical and business discussion with GoOut, the following scope was declared. My penetration test will mainly focus on the parts accessible from the unregistered visitor view, the registered user view, and the organizer view. Only a few admin-only components will be excluded from the testing for obvious internal reasons. Otherwise accessible components from those views are not further limited. The purchasing process handled by an external company, “PayU”, is out of the scope, as well as the internal databases or other services except for the web application located on `dev.goout.net` (a standalone copy of the web application’s production version).

As an approach, black box testing was selected, restricting me from both statically and dynamically analyzing source codes of the components and simply putting me into the position of an external attacker. Testing for common web application vulnerabilities to make the most effective use of the options in terms of time to attack versus potential success of the attack, thus sticking to the aforementioned top ten lists.

# The Penetration Test

*Building on the previous two chapters, this chapter demonstrates the practical application of the OWASP resources to a penetration test performed on the GoOut web application. The chapter also includes an evaluation of the vulnerabilities found during security testing and their CVSS score based on their potential impact on GoOut.*

## 3.1 Burp Suite

Building a house does not strictly require the usage of any tools. Nonetheless, they are still recommended and commonly used. The same applies to penetration testing; nevertheless the types of testing tools are still ubiquitous in their wide variety. Thanks to them, some parts of the process can be automated, saving time and resources for more sophisticated tasks.

One particular tool is Burp Suite[43], a web application security testing software widely adopted by professionals. Burp Suite allows to intercept and, where appropriate, to modify every request coming through the browser or easily manually create custom payload requests in a tool called repeater. It comes with many integrated automated tools for enumerating, brute forcing or decoding, and even a tool called a sequencer, allowing explicitly testing web application tokens, like cart or session tokens, for statistical tests of character-level and bit-level of their randomness. Those statistical tests are composed of a runs test, a spectral test, a correlation test, a compression test, and a few more. This is very crucial for black box testing approaches as source codes of the back-end part of the applications are not available for analysis; therefore, without tools like the sequencer, it is challenging to validate and evaluate the cryptography used properly. However, it is essential to note that those tests can reveal only that the randomness is weak; they cannot prove it is strong. [44]

To make the penetration testing even more structured and clear, Burp Suite allows to scope, filter, and search for past-made requests, crawl chosen domains, or even run automatized testing for known vulnerabilities. Burp Suite comes with even more integrated features. Hence, their software comes in three types: Community, Professional, and Enterprise. The community one is provided for free, but it has limited features, like missing vulnerability scan, advanced searching or filtering, saving project files, and having limits on a brute force tool called intruder.

I have used the Burp Suite Professional version for my penetration testing, as I have been given a one-month free license upon my direct request to PortSwigger. Unlike other tools, Burp Suite accompanied me during the whole testing until the very end, which is not surprising as based on a recent TechValidate[45] survey of Burp Suite Professional, 94 % of penetration testers said that Burp Suite is a “best in class” software.

■ **Table 3.1** Penetration test findings based on OWASP Top 10

Vulnerability name	CVSS v3.1 rating
Accessible DEV environment	High, 7.5
Authentication policies	High, 7.5
Sign-out does not sign-out	High, 7.5
Cross-site request forgery	Medium, 6.5
Leaking stack trace	Medium, 6.5
Password reset hash can be reused and never expires	Medium, 5.7
XSS in organizer view	Medium, 4.6
XSS in event's settings	Low, 3.8
Incorrect data validation of a phone number	Low, 3.6

## 3.2 Information Gathering

Before any active testing was done, I tried to gather as much information as possible about GoOut systems and functionalities from public resources, including open source intelligence (OSINT) and Wayback Machine. Documents found were either outdated or public anyway; however, from the source code of the homepage, a link leads to `/services/partners/swagger-ui/index.html`, which is a swagger UI API documentation version 1.0.6. which is affected by CVE-2022-24863[46]. Another interesting finding, thanks to the Nikto tool, an automatized web application scanner, is `/.well-known/apple-app-site-association` which can be found in the attached media. This is a special Apple's meta JSON file meant for secure association between domain and iOS application[47]. This file also allows to exclude possibly sensitive paths or endpoints. Thanks to that, I have gained further knowledge of the existence of otherwise hidden paths like `/ticketing/`, `/edit/` and `/legacy/`.

I have also searched the domain's DNS history to gain insight into possible servers' locations. However, the only notable information discovered from it was that servers were hidden behind the CloudFlare proxy. Thanks to this, techniques like fingerprinting the web server or scanning for opened ports will not work as intended because it is not possible to access the targeted machine directly without CloudFlare in the way.

## 3.3 OWASP Top 10 Testing

With a proper overview of the GoOut web application's components, roles, functionalities, and possible business impact, it is already possible to perform penetration testing based on the newest version of OWASP Top 10 released in 2021 and further researched in the chapter 1.2 with a declared scope defined in chapter 2.3. I have tried to test components from all possible user roles one by one with the vulnerabilities list in mind. Overall vulnerability findings evaluated by CVSS v3.1 are listed in table 3.1.

### 3.3.1 Broken Access Control

I was amazed that none of the tested components had been affected by broken access control vulnerability. As it is the number one on the Top 10 list, it is an excellent job from a security perspective. Yet some findings similar to broken access control were found to be related to API endpoints; information about those findings will be discussed in section 3.5.5.

### 3.3.2 Cryptographic Failures

As declared in the scope, how data is handled or stored in the database could not be tested. However, how they are being processed or stored locally is indeed within the scope. GoOut strictly enforces the usage of the HTTPS protocol by including the `Strict-Transport-Security` HTTP header with value `max-age=15724800; includeSubDomains`, while also redirecting any requests from HTTP to HTTPS. All locally stored cookies are restricted with the attribute `secure` set to `valid`. To mitigate possible attacks on confidentiality like XSS, sensitive cookies, namely `accessToken`, are also marked with the attribute `HttpOnly` set to true, allowing them to be only included in the HTTP request but not directly accessible by a locally running JavaScript.

I have made further inspections of the `accessToken` and `refreshToken` cookies. Based on their structure, a conclusion was made that those are most likely the JWT tokens, which were later indeed confirmed by successfully decoding their values. They were signed by HMAC using SHA512 and contained `iat`, which stands for Unix epoch time, and also user ID. It was impossible to spoof anyone's identity as the server declined any JWT tokens with no signature algorithm applied, and the server's private key used for signing was never found during the whole testing.

One particular component that came to mind is the sale-form. Each purchase made has to go through an external payment gate. However, GoOut somehow has to track the current state of the transaction so that it can correctly generate tickets, but only if the payment process is successful. During my investigations, I discovered that the `purchaseHash` is being sent along with the payment request to PayU. Thanks to this unique `purchaseHash`, it is possible to view purchased tickets and the receipt even without being logged in; as for the ticket purchase, no account is required in the first place. The idea is straightforward: if the generated hash was weak, anyone could access anyone's tickets. Luckily, `purchaseHash` has a length of 32 characters made out of characters ranging from the letter a to the letter z, resulting in  $\log_2(26^{32}) = 150.4$  bits of entropy, which is very strong. This implies that valid tickets are secured against unauthorized access.

### 3.3.3 Injection

GoOut lets the users provide data for every possible role. Even unregistered visitors, who are the most difficult ones to identify or block, are eligible to use the search bar by filling in any payload. Registered users can fill in details on their profile, which is what I focused on first. Every field related to this component is being correctly sanitized, except the phone number. It was not possible to inject an XSS payload, but it was possible to bypass client-sided phone number restrictions and change it to something invalid as shown in figure 3.1. To fully understand the potential impacts on the web application functionalities, I discussed it with GoOut developers[48]. The phone numbers provided by users may be used by the organizers to send informational SMS about events for which users purchased tickets. Based on the discussion, this SMS sending process, fortunately, creates a payload from all receiving numbers and sends it as one list to the external SMS provider, as shown in code listing 3.1. Then, it checks for a the number of successfully sent SMS by the external gate; therefore, an invalid number would not cause any harm to the sending process itself. On the other hand, injecting a premium SMS number could cause harm. Phone number length was limited only to a minimum of five digits, allowing for the insertion of custom premium SMS services and potentially costing financial resources. This, however, still depends on many variables as the external SMS gate might skip those numbers, the attacker would have to purchase tickets for incoming events, and the organizer of that event would need to send an SMS afterward containing a specific keyword, which on its own is very unlikely as organizers are charged upon sending them. Therefore, it is doubtful that it will occur in the first place for the attack to be successful.

Another component tested was sales and their settings in the GoOut Admin. Unlike other components, a few fields were found vulnerable to the XSS injection attack. Starting with the

■ **Listing 3.1** Function responsible for sending group SMS to an external gateway

```
fun sendSms(numbers: List<String>, text: String) =
    "https://[REDACTED]/"
        .httpPost(
            mutableListOf<Pair<String, Any?>>(
                "to" to numbers.joinToString(","),
                "message" to text,
                "format" to "json",
                "test" to if (production) 0 else 1,
            ),
        )
        .authentication().bearer(smsApiToken)
        .fetch<SmsApiResponse>()
```

`reason` field, which is required when making any changes to the specific sale category as a reason for the change. This field then displays in the overall history of the sale. Nothing prevents the organizer from injecting malicious payload to attack directly GoOut Admin accounts viewing the tampered sale. Luckily, as discussed in section 3.3.2, sensitive cookies, namely `accessToken` and `refreshToken`, are correctly restricted from direct JavaScript access, thus mitigating the risk of malicious payloads targeting them. On the other hand, one specific attack scenario affecting the company's reputation that comes to my mind, is when a departing colleague from the organizer's company seek to resell as many tickets as possible for personal profit, thereby also damaging the company they were about to leave. Thanks to the malicious payload, they could change the price of the ticket category to zero. They could also order a particular amount of tickets on unregistered email accounts, revert the change, and use the malicious XSS payload to either completely hide the changes in the locally displayed sale changes history or alter it in some other way to avoid suspicion.

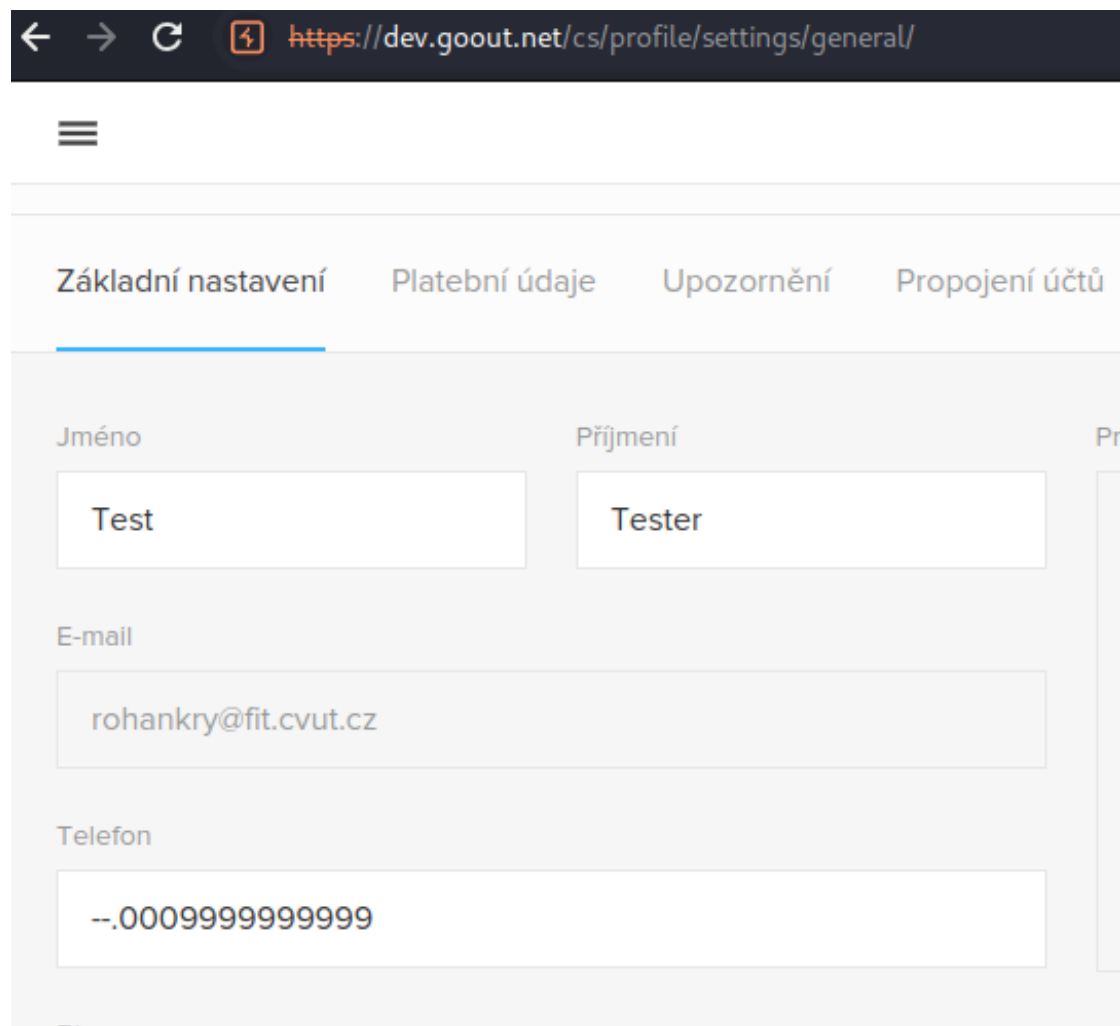
One more particular field vulnerable to the XSS injection was a part of the event settings. This is, unlike the sale settings, accessible only to GoOut admins. Therefore, the possible threat is much smaller than in the previously discussed field. This field, named `adminInfo`, serves as an informational field for other admins to know about policies related to the event or who to contact about the event or related sales if needed. This vulnerability itself may present a low impact, yet it is way more interesting than it seems; details are discussed in section 3.5.5.

Otherwise, input fields and parameters were correctly sanitized or escaped, preventing possible SQL injection.

### 3.3.4 Insecure Design

From the role model applied in the organizer's permissions, which can be seen in figure 3.2, there would not be much to point out; meanwhile, GoOut admins' permissions are something to focus on. Becoming an official GoOut admin does not immediately allow viewing unpublished events, sale categories, and transactions, as a role model is applied even there, however, those permissions granted via the applied role are not fully documented anywhere and, therefore, they might be wrongly set up in the first place, allowing, for example, access to internal business numbers someone from the customer support team. This is not necessarily a vulnerability, but as much as I would like to believe that no one with malicious intentions would ever come in contact with it, it is nothing that can be relied on and should be properly revised, documented, and set up to prevent any potential security incident.





The screenshot shows a web browser window with the address bar containing the URL `https://dev.goout.net/cs/profile/settings/general/`. Below the browser, a navigation menu is visible with the following items: `Základní nastavení`, `Platební údaje`, `Upozornění`, and `Propojení účtů`. The `Základní nastavení` section is active and contains several form fields:

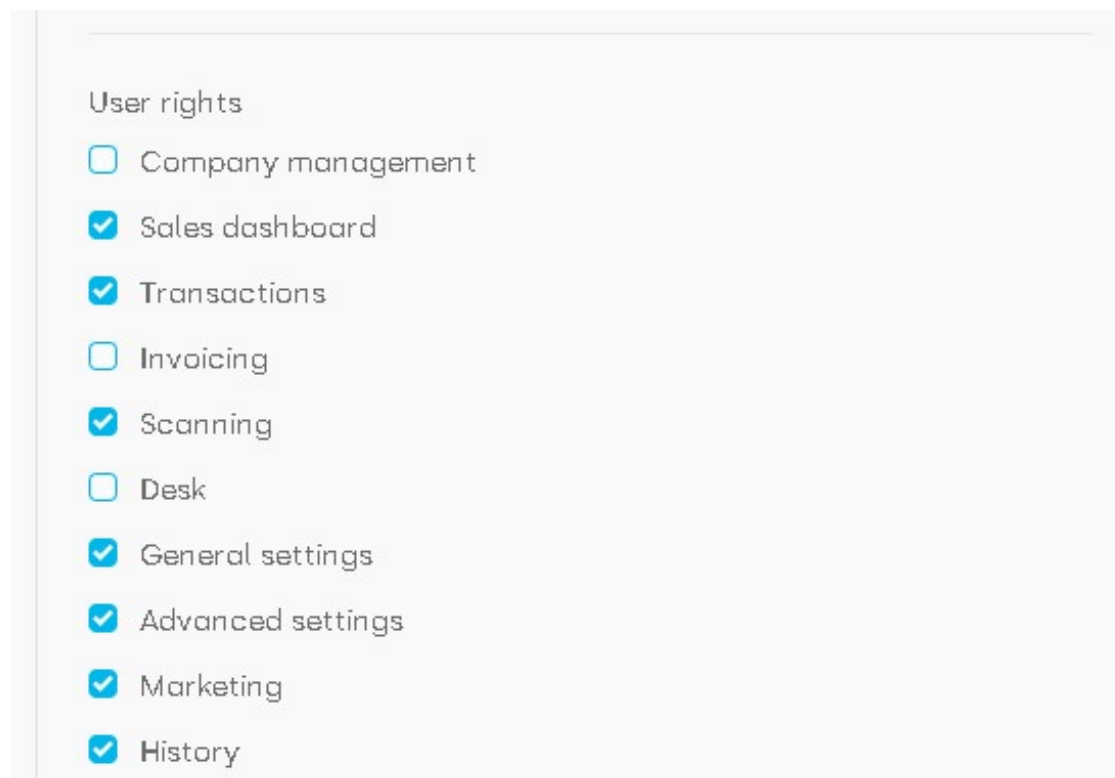
- `Jméno`: `Test`
- `Příjmení`: `Tester`
- `E-mail`: `rohankry@fit.cvut.cz`
- `Telefon`: `--.000999999999999`

■ **Figure 3.1** Proof of concept of the invalid phone being saved in the database

### 3.3.5 Security Misconfiguration

During my testing, I have managed to cause several HTTP error responses, including HTTP 500, and other internal errors. At a first glance, it seemed that there was a potential stack trace leak occurring when an internal error was thrown. This was immediately refuted as I realized that those errors were left unhandled by the developers intentionally. My testing was scoped only on the development environment, where is a different server configuration to provide a detailed error log of exceptions, including a stack trace, for the purpose of easier debugging.

Nonetheless, I discovered a special occurrence of an error response, an HTTP 400 bad request, which ultimately led to leaking stack trace even on the production environment. By purchasing tickets for any event, a unique purchase hash is generated. This hash also acts as a secret, as tickets can be purchased even by unregistered visitors. GoOut allows the generation of tax receipts using some particular component, which has not been discovered until now. The communication with this component relies on a parameter called `kind`. This parameter can be set for example to value `document_full` which specifies the requested document type of the generated tax receipts. Upon requesting an empty value, the component correctly handles the request with some fallback default selected document type. However, upon entering an invalid



■ **Figure 3.2** Individual organizer account permissions settings in GoOut Admin

HTTP character, reverse slash \, an unhandled HTTP error 400 bad request was thrown, ultimately leaking the stack trace and HTTP server type with the currently running version. This vulnerability was given a CVSS v3.1 rating of 6.5, based on information disclosure, expanding possible attack vectors, and being accessible by anyone on the internet.

What was, in fact, far worse than just a leaking stack trace is that even though the development version of the application was hidden, it was still accessible from the internet without any authentication required. The development environment is an altered copy of the production version, with no real connection to it and containing only limited data. Yet when compared to the production environment, some of the debugging settings were enabled there. While not affecting integrity anyhow, confidentiality was definitely affected. Causing a stack trace or other type of information disclosure was far easier there, therefore elevating the potential impact and creating a larger problem.

### 3.3.6 Vulnerable and Outdated Components

One particular component, which has already been discussed in section 3.2, is Swagger. The version used is not only outdated but also vulnerable to denial of service attacks[46] by crafting POST requests to the endpoint that expects only GET ones. Fortunately, this has been either manually patched or secured by the usage of Cloudflare's web application firewall (WAF).

Thanks to the misconfiguration issues further discussed in section 3.5.8, which ultimately led to informational disclosure, I was able to identify the technologies internally used. One of them was an outdated version of Java 17.0.7, which was directly by Oracle[49] recommended to stop using after a critical patch release scheduled on 18 July 2023. I was personally unable to exploit the version.

### 3.3.7 Identification and Authentication Failures

Several problems were found in terms of authentication failures. Some of them present a higher security risk, while others do not.

Let's start with the one with the lowest risk, enforced password policies. GoOut allowed me to create an account or change the password to at least seven characters long secret. This secret could have been written in a lowercase format; it could have also been some very well-known key-phrase or commonly used password like `password`. GoOut is used mainly in the Czech Republic. Therefore, one of the most common passwords is `Hes1o123`, which was also allowed to be used even with the first letter being lowercase. Admin accounts are disallowed from using passwords to authenticate, so those are not affected, yet organizer accounts were. The reason why I consider this as the one with the lowest risk is that in order to successfully execute the attack, the victim would have to consciously decide on a weak, common, or leaked password, like using the very same one for every web application registration.

Unfortunately even if they decided to use a strong password it may not be enough against a sophisticated targeted attack. With knowledge of the OSINT of the victim and some of the passwords they used that have gotten leaked in past database breaches, it may be possible to guess their currently used strong password. Multi-factor authentication, which could have helped, was lacking in the GoOut application.

Besides the policies, something worse related to the sign-in component was discovered. GoOut used no rate-limiting or prevention against attackers from launching targeted brute-force attacks or any more beneficial attacks in terms of speed-result ratio, for example credential stuffing.

Following up with the session handling, the `AccessToken` cookie provides, seemingly a secure way to handle user sessions. Unfortunately, this was not entirely true, as even with the token's integrity being strictly validated, there was a flaw in the invalidation during sign-out. The token was valid for a few weeks; if the sign-out process was called at any point in time during the token's existence, a cookie was locally destroyed and the user was signed out. On the other hand, this invalidation was done only on the client's side. The server did not receive any information about the `accessToken` invalidation upon request, as the only request it processed was to either validate the provided cookie, its signature and time, or craft and sign a new one upon a successful sign-in. Even with secure authentication policies in place, session hijacking may be possible. Once the attacker has gained control of the session, without the possibility of invalidating it on the server's side there is nothing left to do but directly ask for help from GoOut developers.

This is not the only flaw related to the `accessToken` cookie affecting the overall security. The cookie was not set with the attribute `SameSite`, resulting in it being vulnerable to cross-site request forgery (CSRF) attacks. As a proof of concept, I created a friendly-looking website with a simple button saying "Click to get your free gift!"; a key part of the HTML content can be seen in listing 3.2. If the victim clicks on the button, their browser will create a POST request to the official GoOut endpoint `/services/social/follow/v2/follow` with the payload set to the values visible in the code snippet. That signed up the user to follow my testing account. This might seem like a harmless play, but the opposite is true. Nothing stops attackers from crafting requests to, for example, a payment or a sale settings endpoint, while also executing the request right upon the page loading, rather than requiring the user interaction by clicking on a button. Unlike the follow endpoint I used for the proof of concept, a password change would not be possible to exploit as a unique hash is required for it. Overall, this vulnerability affects integrity; on the other hand, it requires attackers to trick the victim into visiting their website, unless they combine the usage of other vulnerabilities, such as XSS or a man-in-the-middle attack scenario.

The last problem from this section was found in the password reset component. In order to change the currently used password, a password reset process has to be authenticated through email. The email received contains a unique hash allowing one to change the password. The issue with it was that not only could this hash be used more than once to change the password, but it never really expired. If a bad actor gained access to the victim's email account with an

■ **Listing 3.2** Malicious HTML form exploiting CSRF vulnerability to follow the attacker's account on GoOut

```
<form action="https://dev.goout.net/services/social/follow/v2/follow"
↳ method="POST">
  <input name="type" id="type" value="user" type="hidden"/>
  <input name="ids" id="ids" value="2951547" type="hidden"/>
  <input name="action" id="action" value="LIKE" type="hidden"/>
  <div>
    <button>Click to get your free gift!</button>
  </div>
</form>
```

■ **Listing 3.3** Commented out code checking for invalid password reset hashes

```
//      Commented out because of issues with finishing registrations. Should be
↳ refactored when reimplementing for Next frontend.
//      if (userRequest.isClosed) {
//          log.error("Request has already been closed: $userRequest")
//          throw BadRequestException()
//      }
```

old password reset request, an account takeover would still be possible. How likely would it be to change the password of a random user who has requested the password change process at any point in time until now? The reset hash consists of 37 characters with 26 possible values ranging from letter a to z. Let's say that 500 password reset requests are done daily. After seven years of running, we would end up with  $500 \cdot 365 \cdot 7 = 1277500$  hashes, resulting in a  $1$  in  $\frac{37^{26}}{1277500} \cong 4.64 \cdot 10^{34}$  chance of successfully brute forcing hash on the first try, which is negligible.

Upon discussion with GoOut developers[48], it was found that the issue with the password reset hash expiration was not there from the very beginning. One day, during further development of the platform, someone commented out a part of the script responsible for the password change request, which can be seen in code listing 3.3.

### 3.3.8 Software and Data Integrity Failures

I could not find any software or data integrity failures during my security testing or any vulnerable components that would be exploitable from this perspective.

### 3.3.9 Security Logging and Monitoring Failures

Due to the sequential security testing of every component, I have tried illegal actions from the application design perspective like injecting malicious payloads numerous times. What surprised me is that there was no active monitoring for potential security threats except error and warning logging for the development and debugging purposes. All those and many others are being actively logged in Google Console. Some specific thrown exceptions on the back-end are directly integrated to be sent over the Slack chatting platform, as can be seen in figure 3.3. While these logs usually contained data about related users and also were properly sanitized to prevent stored attacks, both the channel on Slack which was used for this integration and the Google

Console itself were being constantly spammed with other logs or exceptions, often becoming disorganized. Whereas thrown errors like internal server errors were correctly logged, brute-force password attack attempts were never registered in the first place.

In addition to the internal error logging, additional logging is done directly in the web application. This applies to event or sale changes in order to keep a history of past changes.

```
net.goout.messaging.PubSubService$consumeWithKClass$1.invoke(PubSubService
[app.jar:?])

webapp [REDACTED] APP 16:15
2023-12-07 16:15:15.520 FATAL [.] ExceptionHandlingController:176 -
java.lang.IllegalStateException: Field passwordFirs is missing in
RegistrationForm, available fields were:

webapp [REDACTED] APP 16:20
2023-12-07 16:20:36.968 FATAL [.] RegistrationForm:103 - Request not valid
vivost@email.cz <
UserRequest[3134903,RECOVER_PASSWORD,rohankry@fit.cvut.cz,pmoftorsslz1frg
uhvkiqifhrafgc]
```

■ **Figure 3.3** Error and warning messages logged in chat by Slack integration

### 3.3.10 Server-Side Request Forgery

SSRF was nowhere to be found, as the web application did not seem to fetch remote resources based on the user-requested data.

## 3.4 OWASP Top 10 Testing Evaluation

Security testing focused on the OWASP Top 10 list yielded several interesting findings. Those vary both in severity of the impact and the complexity of execution. Overall, the web application of GoOut was well designed, with just a few flaws that commonly appear in any web application. There were several findings related to authentication failures, like weak policies, lack of rate limiting, improper session invalidation, and password reset hash re-usability.

## 3.5 OWASP API Security Top 10 Testing

This section is dedicated to the examination of security vulnerabilities in API endpoints, which have been discovered during the process of information gathering or any previous security testing. With the knowledge of the newest OWASP API Security Top 10 list from 2023, further discussed in section 1.3, found vulnerabilities are listed in table 3.2.

### 3.5.1 Broken Object Level Authorization

One particular feature plays a key role in event administration. The endpoint `/services/contentadmin/schedule/v1/setPublishOn` allows organizers or GoOut admins to

■ **Table 3.2** Penetration test findings based on OWASP API Security Top 10

Vulnerability name	CVSS v3.1 rating
“Locking out” the entire sale without any purchase	Critical, 9.3
API event settings BFLA	Medium, 5.9
API schedules settings BOLA	Medium, 5.5
API user enumeration	Medium, 5.3
API leaking an internal exception	Medium, 5.3

set a specific event to be scheduled on a given date, which can be used to unpublish the event by setting any future date or publish the event right away by setting a current date. The endpoint accepts POST requests with two parameters. One of them is `id`, which specifies the desired event, and `publishOn`, which defines the date (past, present, or future). The endpoint correctly requires authentication as organizer or admin in order to process the request, but it does not check for object level authorization. Due to this, it is possible for any organizer, even the smallest one who organized only a single event with a few attendees, to unpublish or publish any existing event in the application. What is important to note is that the organizer cannot access the targeted event settings or alter anything else. To execute this attack, the attacker would have to guess the event ID or retrieve it using a different approach. I have found out that the front-end part of the web application, upon viewing specific venues, calls the `/services/entities/v1/schedules` endpoint to retrieve incoming or past events. Those are referred to by their ID, which is exactly what the attacker needs to successfully unpublish any published event. This affects both the availability and confidentiality of the component. Availability is impacted when a published event becomes suddenly unpublished, preventing customers from ordering tickets, and confidentiality is affected if an attacker manages to publish any unpublished event. However, even this may be more challenging than it seems as the event object has to be both approved and published (in the past or the present) in order to be visible to the general public, and this endpoint allows modification of the `publishedOn` attribute only.

### 3.5.2 Broken Authentication

The Authentication process itself is properly handled, using the JWT tokens, which were further discussed in the cryptographic failures section 3.3.2, yet there was a major difference between the web application and the API authentication processes. That was an inconsistent approach. The web application focuses on security policies, avoiding leaking any information about user account existence in the system: for example during an incorrect sign-in attempt, the generic message `Invalid credentials` is displayed rather than `Provided username does not exist in the system` and `Password for a given account is incorrect`. In contrast, the API design focuses more on the user experience (UX) by providing feedback for customer comfort. This inconsistency, however, causes both wrong security policies and insufficient UX simultaneously. API endpoint `/services/user/v1/accountExists` which takes one parameter `email` in a GET HTTP request and responds with either HTTP 404 when the account does not exist in the system or with HTTP 200 which includes real user ID as can be seen in the example response in listing 3.4.

### 3.5.3 Broken Object Property Level Authorization

None of my created requests resulted in a broken object property level authorization flaw, from both information disclosure and tampering point of view.

■ **Listing 3.4** Payload in response from `/services/user/v1/accountExists` endpoint

```
{
  "userId": 1214821,
  "status": 200
}
```

### 3.5.4 Unrestricted Resource Consumption

Analytics of sale transactions are a necessary part of the web application from the organizer's business perspective. GoOut implemented various query options to fetch any desired data organizers would ever think of. This component is, however, very complex and so brought attention to me. All queries are being handled by the `/services/reporting/v0/purchase-stats` API endpoint which processes HTTP POST requests with JSON payload data. It is possible to filter specific time ranges, transaction states like only those successfully paid, payment methods, sales related to only specific sales or keywords, and so on. One key parameter, `saleIds`, was either incorrectly designed or contained a security flaw in it. If the parameter was included in the request query payload but left with an empty value, an unrestricted resource consumption would occur.

Based on communication with GoOut[48], what did actually happen on the back-end is that the component would iterate over all sales as the `saleIds` was empty while also evading the authorization checks and ultimately causing the massive overload on the server to calculate correct values over tons of existing sales. Luckily, the returned data of the JSON type contained only numbers, and no further sensitive information about sales of other organizers or customer related data was disclosed.

A denial of service could very quickly result from multiple calls with an empty `saleIds` value to this endpoint. This affects primarily availability, yet also confidentiality from the internally leaked numbers of sales; however, as it was already stated, attackers could have no idea where those numbers came from or what they are even related to, and the overall confidentiality impact is, therefore, smaller.

### 3.5.5 Broken Function Level Authorization

A case of BFLA I had stumbled upon was in `/legacy/forms/event/submit` endpoint, which is a part of the GoOut Admin available only to the highest privileged users. Without being admin, no requests are sent to this endpoint. On the other hand, a very similar endpoint `/legacy/forms/sale/submit` is available for organizers, and relying on security through obscurity that no one would ever guess or find this endpoint would not be a good idea. Upon further testing, if the attacker knew the payload's structure sent to this endpoint, nothing would have prevented them from crafting their own malicious one to alter existing events. Luckily, as seen in the listing 3.5, this minimal structure of a payload is nowhere near to being just randomly guessed by the attacker, and also, the `revisionId` did not seem to be visible anywhere, therefore the severity of this vulnerability is lowered by the attack complexity even though the attacker would be able to alter the whole event.

### 3.5.6 Unrestricted Access to Sensitive Business Flows

A crucial role in the conversion marketing on the GoOut web application is definitely the sale-form, which stands as a sensitive business flow, allowing even unregistered users to purchase tickets. Once the ticket is selected and placed into an imaginary cart, for a small period of time

**■ Listing 3.5** Required JSON payload structure to exploit the BFLA

```
{
  "strings": {
    "revisionId": "1651659",
    "state": "APPROVED",
    "adminInfo": "[payload]"
  }
}
```

a reservation is made to prevent anyone else from buying out in the ticket in meanwhile. This time period differs in particular scenarios, for example it is extended upon the start of the payment process so customers can still successfully buy their selected tickets in case of insufficient funds or wrong payment details provided, requiring more time to complete the order. The main issue of this otherwise well-intended functionality is that it can also be used for malicious purposes.

During testing, I discovered that the `/services/saleform/selection/v1/ticketByCount` endpoint is being called upon selecting or removing tickets. HTTP POST request consists of `saleId`, `dealId` (sale category), `totalPriceBefore` which contains the total price of the cart before the asked change (I was unable to exploit this parameter) then there are also a few more uninteresting attributes based on my findings and the very last one is `numTickets`. The front-end loads and sets proper ticket limit per sale category in the sale-form based on `/services/feeder/v2/sales` data. Clearly, a limitation value per ticket category exists in the database. Yet, it was not correctly checked for and upon altering request to the `/services/saleform/selection/v1/ticketByCount` with `numTickets` exceeding local sale category limit, it was possible to order all of the requested tickets as long as their count did not exceed the total amount of tickets per targeted sale category. This also implies that it was possible to block out a whole sale using a single POST request to the endpoint even without creating an account, ultimately affecting the tickets availability.

### 3.5.7 Server Side Request Forgery

SSRF was not found anywhere, even in the API endpoints; no endpoint seemed to be accepting URLs to make requests for or to be crafting URLs based on the string input provided, which would ultimately lead to the vulnerability.

### 3.5.8 Security Misconfiguration

A sale-form component allows several payment methods for customer needs. Those methods are selectable in the form itself and, in case of a payment failure, can even be changed afterward. Once sale categories, ticket count, contact details, and a payment method were selected, the `/services/saleform/purchase/v1/pay` endpoint was called via HTTP POST request with an example payload visible in the listing 3.6.

Upon changing the type parameter to something illegal, like a single dot, the exception was thrown and the endpoint replied with an HTTP 400 Bad Request. However, this response also contained a JSON reply with the thrown exception as seen in listing 3.7.

Another occurrence of API security misconfiguration was located in event administration, which was accessible by both GoOut admins and organizers. It is the very same component discussed upon in the injection vulnerability findings in section 3.3.3, used for sale category configuration. There was no limit to how many tickets could be set up as the maximum capacity per the specified ticket category. When setting 999999 as the count parameter in the request,



**Listing 3.6** Payload send via HTTP POST to /services/saleform/purchase/v1/pay

```
{
  "purchaseHash":"wlwnhnhnqbmxfegwivofursfyrqeocsiy",
  "firstName":"Krystof",
  "lastName":"Tester",
  "email":"rohankry@fit.cvut.cz",
  "phone":"",
  "ticketCount":2,
  "ticketPriceCents":100500,
  "saleDiscount":null,
  "type":"PAYU",
  "mailingAgreed":true,
  "language":"cs",
  "isDesk":false
}
```

**Listing 3.7** Thrown error contained in JSON response

```
{
  "message":"JSON parse error: Cannot deserialize value of type from String
  ↵ "": not one of the values accepted for Enum class: [REDACTED]",
  "status": 400
}
```



## 3.6 OWASP API Security Top 10 Testing Evaluation

Compared to the previous security testing based on OWASP Web Top 10, more severe vulnerabilities were discovered. Namely, unrestricted access to sensitive business flow sale-form tickets, which could have been exploited to prevent any other visitor from buying tickets using just one simple POST request exceeds the magnitude of the discovered vulnerability severities due to its direct impact on the key business flow. Some potential problems regarding the code sustainability were also discovered, such as improper inventory management of the existing API endpoints.

## 3.7 Recommendations

In the discussion of my vulnerability findings and their possible impacts in the previous sections, I might have also hinted at what could be the possible mitigation or patches for them. However, this was just a by-product of my explanation, and the overall recommendations, considering the gathered information about both the technical and business sides, are listed in this section.

### 3.7.1 “Locking out” the entire sale without any purchase

A very straightforward mitigation of the lockout is to enforce checks on the back-end for maximum selected ticket number. On the other hand, the core problem with potential attackers using many VPNs or proxies to bypass the selected ticket amount will still persist. I would personally recommend the following steps:

1. Validate maximum ticket selection on the back-end and log any illegal requests.
2. Blocklist well known VPN, proxies, or TOR exit nodes.
3. Implement CAPTCHA checks as an optional setting per sale.
4. Monitor incoming traffic or suspicious traffic spikes.

Those will never completely prevent attackers from doing so, but it mitigates the issue and lowers the possible impact.

### 3.7.2 Accessible DEV Environment

The development environment still requires users to sign in as they would also have to in the production environment. With DEV being accessible from the wide internet by anyone, this is the core issue that has to be resolved. Possibly, the fastest approach is to apply a allowlist on the firewall to completely restrict access from all but explicitly allowed IP addresses. Even though this would do the job, it still has many flaws. First of all, IP addresses may rotate and change over time, so some developers could lose their access while a random person under the same internet service provider would be suddenly granted it. Sometimes, the IP address is used on a wider range of devices thanks to the network address translation, granting the access not only to the desired developer but also to many other people living or being in the same area as them.

However, a better approach is to apply rules to a firewall or a router in order to block the incoming traffic to the server completely. Allowing only devices connected to the local network access to the DEV environment. Even with this more secure approach, one problematic area still exists: developers working remotely. In order to allow the remote access only to authenticated developers and nobody else, a VPN or a SSH connection should be in place. The last thing I would personally recommend to improve overall security is to sufficiently log accesses from the VPN or SSH and monitor any suspicious sign-in attempts or actions.

■ **Listing 3.8** Example of a counting expression as the rate limiting

```
http.request.uri.path eq "/services/user/auth/v2/login" and http.request.method  
↪ eq "POST" and http.response.code eq 401
```

### 3.7.3 Identification and Authentication Failures

Several vulnerabilities were found in the authentication process, but fortunately, none of them were severe enough to allow for authentication bypass or identity spoofing. I would suggest the following mitigation to reinforce the current security:

1. Minimum password length should be set to eight characters based on the NIST guidelines[50].
2. The password hash should be checked against commonly used passwords, rockyou<sup>3</sup> could be used; however, a Czech based list should also be used.
3. reCAPTCHA or any other form of robot detection should be in place in sign-in form.
4. Proper rate limiting should be applied to all authentication related endpoints and flows; an example of a possible rule applied directly through CloudFlare WAF can be seen in code listing 3.8, this expression counts the number of failed logging attempts by counting HTTP 401 replies to requests for login endpoint `/services/user/auth/v2/login`. If the number of attempts exceeds a certain threshold, human verification will be required.
5. Suspicious login attempts, especially to the organizer or admin accounts, should be properly logged and actively monitored.
6. 2FA should be required for organizer and admin accounts while also being optional for normal users.
7. Session cookies like `accessToken` and other sensitive ones should properly set the attribute `SameSite` to `Strict` to mitigate mentioned CSRF attacks.
8. Session cookies need to be also revoked on the server by implementing a blocklist for the JWT and shortening its expiry time to rotate them more often.
9. Password reset hash reuse prevention by uncommenting the existing code in the listing 3.3 while also implementing its expiration time to one hour to be checked before processing.

### 3.7.4 XSS

Cross-site scripting can be prevented by properly escaping or sanitizing user-provided data. Upon discussing with GoOut developers[48], this was a quick inline fix, as can be seen in the difference between the original code snippet without escaping 3.9 and the patched one in the code snippet 3.10. Unfortunately, this is the only part of the mitigation; to entirely prevent XSS, user-provided data should only go through a dedicated flow that is correctly sanitized in order to avoid any other possible components from being or becoming vulnerable to injection. I may not have found all instances of this vulnerability, and without proper mitigation, it could still be exploited at some point.

---

<sup>3</sup>Huge list of leaked passwords commonly used for password cracking.

**■ Listing 3.9** Vulnerable code to XSS

```
<p>${diff.key}: ${oldValue} <span class="iconfont">&#xF131;</span>  
↪  ${newValue}</p>
```

**■ Listing 3.10** Properly escaped code preventing XSS

```
<p>${diff.key}: ${oldValue?html} <span class="iconfont">&#xF131;</span>  
↪  ${newValue?html}</p>
```



## Chapter 4

# Conclusion

The goal of this thesis was to strengthen the security of the GoOut web application by finding possible vulnerabilities by using penetration testing.

An analysis of the importance of security testing in today's world, what key concepts need to be understood in order to perform it properly, pointed to existing resources published by OWASP. Another analysis was done on what GoOut is. This analysis answers what differentiates GoOut from other companies in the field, what components are essential for the application from both a technical and business perspective, and what types of user roles can be found there. Based on this analysis, it is possible to build on the knowledge gained and focus entirely on penetration testing the GoOut web application. Identified vulnerabilities are evaluated and discussed in terms of potential business impact. For the most serious ones, a dedicated remediation section aims to provide a possible and appropriate mitigation.

Although many vulnerabilities were found, many other attacks were unsuccessful. GoOut's developers continuously consulted with me about possible solutions to the vulnerabilities found, as well as the technical cause behind them. Fortunately, at the time of writing, most of them have already been mitigated, further improving the overall security. After all, there are many other products or services that were left out of scope and therefore untested. This leaves room for potential additional security testing in the future. I hope to be able to contribute to additional security testing in the future.

# Appendix A

## Evaluated vulnerability findings

■ **Table A.1** Full list of found vulnerabilities with their CVSS vector string

Vulnerability name	CVSS v3.1 rating
“Locking out” the entire sale without any purchase	AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:H/CR:X/IR:X/AR:H
Accessible DEV environment	AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N
Authentication policies	AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:H/A:N
Sign-out does not sign-out	AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N
Cross-site request forgery	AV:N/AC:L/PR:N/UI:R/S:U/C:N/I:H/A:N
Leaking stack trace	AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:N/A:N
Password reset hash can be reused and never expires	AV:N/AC:L/PR:L/UI:R/S:U/C:H/I:N/A:N
API event settings BFLA	AV:N/AC:H/PR:N/UI:N/S:U/C:N/I:H/A:N
API schedules settings BOLA	AV:N/AC:L/PR:H/UI:N/S:U/C:L/I:N/A:H
API user enumeration	AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N
API leaking internal exception	AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N
XSS in organizer view	AV:N/AC:L/PR:H/UI:N/S:C/C:H/I:N/A:N/CR:L/IR:X/AR:X
XSS in event’s settings	AV:N/AC:H/PR:N/UI:R/S:C/C:H/I:N/A:N/CR:L/IR:X/AR:X
Incorrect data validation of phone number	AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:L/A:N/CR:X/IR:L/AR:X

# Bibliography

1. SITELOCK. *2022 SITELOCK ANNUAL WEBSITE SECURITY REPORT* [online]. 2022. Available also from: <https://s3.us-east-1.amazonaws.com/sectigo-sites-web/global/uploads/2022-SiteLock-Website-Security-Report-FINAL.pdf>. [Accessed 20-02-2024].
2. SOUPPAYA, Murugiah; SCARFONE, Karen. *Technical Guide to Information Security Testing and Assessment*. Special Publication (NIST SP), National Institute of Standards and Technology, Gaithersburg, MD, 2008. Available also from: [https://tsapps.nist.gov/publication/get\\_pdf.cfm?pub\\_id=152164](https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=152164).
3. OWASP FOUNDATION. *OWASP Web Security Testing Guide* [online]. [N.d.]. Available also from: <https://owasp.org/www-project-web-security-testing-guide/>. [Accessed 20-02-2024].
4. OWASP FOUNDATION. *OWASP Top Ten* [online]. [N.d.]. Available also from: <https://owasp.org/www-project-top-ten/>. [Accessed 20-02-2024].
5. OWASP FOUNDATION. *A01 Broken Access Control — OWASP Top 10:2021* [online]. 2021. Available also from: [https://owasp.org/Top10/A01\\_2021-Broken\\_Access\\_Control/](https://owasp.org/Top10/A01_2021-Broken_Access_Control/). [Accessed 23-03-2024].
6. GARY L. BRASE Eugene Y. Vasserman, William Hsu. Do Different Mental Models Influence Cybersecurity Behavior? Evaluations via Statistical Reasoning Performance. *Information*. 2017. Available also from: <https://ncbi.nlm.nih.gov/pmc/articles/PMC5673648/>.
7. STEVENS, Marc; SOTIROV, Alexander; APPELBAUM, Jacob; LENSTRA, Arjen; MOLNAR, David; OSVIK, Dag Arne; WEGER, Benne de. Short Chosen-Prefix Collisions for MD5 and the Creation of a Rogue CA Certificate. In: HALEVI, Shai (ed.). *Advances in Cryptology - CRYPTO 2009*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009. ISBN 978-3-642-03356-8.
8. OWASP FOUNDATION. *A02 Cryptographic Failures — OWASP Top 10:2021* [online]. 2021. Available also from: [https://owasp.org/Top10/A02\\_2021-Cryptographic\\_Failures/](https://owasp.org/Top10/A02_2021-Cryptographic_Failures/). [Accessed 23-03-2024].
9. GOLDBERG, Lewis R.; ROSOLACK, Tina K. The Big-Five factor structure. *Journal of Personality and Social Psychology*. 1990, vol. 59, no. 6, pp. 1216–29.
10. OWASP FOUNDATION. *A03 Injection — OWASP Top 10:2021* [online]. 2021. Available also from: [https://owasp.org/Top10/A03\\_2021-Injection/](https://owasp.org/Top10/A03_2021-Injection/). [Accessed 23-03-2024].
11. KALUŻNY, Jakub. *Threat Modeling — how to start doing it?* [Online]. 2021. Available also from: <https://www.securing.pl/en/threat-modeling-how-to-start-doing-it/>. [Accessed 25-02-2024].



12. OWASP FOUNDATION. *A04 Insecure Design — OWASP Top 10:2021* [online]. 2021. Available also from: [https://owasp.org/Top10/A04\\_2021-Insecure\\_Design/](https://owasp.org/Top10/A04_2021-Insecure_Design/). [Accessed 23-03-2024].
13. ANAND, Abhineet; CHAUDHARY, Amit; ARVINDHAN, M. The Need for Virtualization: When and Why Virtualization Took Over Physical Servers. In: HURA, Gurdeep Singh; SINGH, Ashutosh Kumar; SIONG HOE, Lau (eds.). *Advances in Communication and Computational Technology*. Singapore: Springer Nature Singapore, 2021. ISBN 978-981-15-5341-7.
14. ELMERS, Robbie. *LogicMonitor Provided Default And Weak Passwords To Its Customers* [online]. 2023. Available also from: <https://technewsspace.com/logicmonitor-provided-default-and-weak-passwords-to-its-customers/>. [Accessed 25-02-2024].
15. OWASP FOUNDATION. *A05 Security Misconfiguration — OWASP Top 10:2021* [online]. 2021. [https://owasp.org/Top10/A05\\_2021-Security\\_Misconfiguration/](https://owasp.org/Top10/A05_2021-Security_Misconfiguration/) [Accessed 23-03-2024].
16. Q-SUCCESS. *Usage Statistics and Market Share of WordPress, March 2024* [online]. [N.d.]. Available also from: <https://w3techs.com/technologies/details/cm-wordpress>. [Accessed 04-03-2024].
17. Q-SUCCESS. *Usage Statistics and Market Share of PHP for Websites, March 2024* [online]. [N.d.]. Available also from: <https://w3techs.com/technologies/details/pl-php>. [Accessed 25-02-2024].
18. SONATYPE. *The Pervasive Influence of Open Source: Trends, Adoption, and Security Concerns* [online]. 2022. Available also from: <https://www.sonatype.com/state-of-the-software-supply-chain/open-source-supply-and-demand>. [Accessed 25-02-2024].
19. OWASP FOUNDATION. *A06 Vulnerable and Outdated Components — OWASP Top 10:2021* [online]. 2021. Available also from: [https://owasp.org/Top10/A06\\_2021-Vulnerable\\_and\\_Outdated\\_Components/](https://owasp.org/Top10/A06_2021-Vulnerable_and_Outdated_Components/). [Accessed 23-03-2024].
20. OWASP FOUNDATION. *A07 Identification and Authentication Failures — OWASP Top 10:2021* [online]. 2021. Available also from: [https://owasp.org/Top10/A07\\_2021-Identification\\_and\\_Authentication\\_Failures/](https://owasp.org/Top10/A07_2021-Identification_and_Authentication_Failures/). [Accessed 23-03-2024].
21. UNIT42. *SolarStorm Supply Chain Attack Timeline* [online]. 2020. <https://unit42.paloaltonetworks.com/solarstorm-supply-chain-attack-timeline/> [Accessed 25-02-2024].
22. OWASP FOUNDATION. *A08 Software and Data Integrity Failures — OWASP Top 10:2021* [online]. 2021. Available also from: [https://owasp.org/Top10/A08\\_2021-Software\\_and\\_Data\\_Integrity\\_Failures/](https://owasp.org/Top10/A08_2021-Software_and_Data_Integrity_Failures/). [Accessed 23-03-2024].
23. THEHACKERNEWS. *Ex-Google Engineer Arrested for Stealing AI Technology Secrets for China* [online]. 2024. Available also from: <https://thehackernews.com/2024/03/ex-google-engineer-arrested-for.html>. [Accessed 09-03-2024].
24. OWASP FOUNDATION. *A09 Security Logging and Monitoring Failures — OWASP Top 10:2021* [online]. 2021. Available also from: [https://owasp.org/Top10/A09\\_2021-Security\\_Logging\\_and\\_Monitoring\\_Failures/](https://owasp.org/Top10/A09_2021-Security_Logging_and_Monitoring_Failures/). [Accessed 23-03-2024].
25. OWASP FOUNDATION. *A10 Server Side Request Forgery (SSRF) — OWASP Top 10:2021* [online]. 2021. Available also from: [https://owasp.org/Top10/A10\\_2021-Server-Side\\_Request\\_Forgery\\_%5C%28SSRF%5C%29/](https://owasp.org/Top10/A10_2021-Server-Side_Request_Forgery_%5C%28SSRF%5C%29/). [Accessed 23-03-2024].
26. DI LAURO, Fabio; SERBOUT, Souhaila; PAUTASSO, Cesare. A Large-Scale Empirical Assessment of Web API Size Evolution. *Journal of Web Engineering*. 2022, vol. 21, no. 6, pp. 1937–1979. Available from DOI: 10.13052/jwe1540-9589.2167.

27. OWASP FOUNDATION. *OWASP API Security Top 10 Introduction* [online]. 2023. <https://owasp.org/API-Security/editions/2023/en/0x03-introduction/> [Accessed 02-03-2024].
28. OWASP FOUNDATION. *API1:2023 Broken Object Level Authorization — OWASP API Security Top 10* [online]. 2023. Available also from: <https://owasp.org/API-Security/editions/2023/en/0xa1-broken-object-level-authorization/>. [Accessed 23-03-2024].
29. OWASP FOUNDATION. *API2:2023 Broken Authentication — OWASP API Security Top 10* [online]. 2023. Available also from: <https://owasp.org/API-Security/editions/2023/en/0xa2-broken-authentication/>. [Accessed 23-03-2024].
30. OWASP FOUNDATION. *API3:2023 Broken Object Property Level Authorization — OWASP API Security Top 10* [online]. 2023. <https://owasp.org/API-Security/editions/2023/en/0xa3-broken-object-property-level-authorization/> [Accessed 23-03-2024].
31. OWASP FOUNDATION. *API4:2023 Unrestricted Resource Consumption — OWASP API Security Top 10* [online]. 2023. Available also from: <https://owasp.org/API-Security/editions/2023/en/0xa4-unrestricted-resource-consumption/>. [Accessed 23-03-2024].
32. WORDPRESS. *Roles and Capabilities* [online]. [N.d.]. Available also from: <https://wordpress.org/documentation/article/roles-and-capabilities/>. [Accessed 02-03-2024].
33. OWASP FOUNDATION. *API5:2023 Broken Function Level Authorization — OWASP API Security Top 10* [online]. 2023. Available also from: <https://owasp.org/API-Security/editions/2023/en/0xa5-broken-function-level-authorization/>. [Accessed 23-03-2024].
34. OWASP FOUNDATION. *API6:2023 Unrestricted Access to Sensitive Business Flows — OWASP API Security Top 10* [online]. 2023. Available also from: <https://owasp.org/API-Security/editions/2023/en/0xa6-unrestricted-access-to-sensitive-business-flows/>. [Accessed 23-03-2024].
35. OWASP FOUNDATION. *API7:2023 Server Side Request Forgery — OWASP API Security Top 10* [online]. 2023. Available also from: <https://owasp.org/API-Security/editions/2023/en/0xa7-server-side-request-forgery/>. [Accessed 23-03-2024].
36. OWASP FOUNDATION. *API8:2023 Security Misconfiguration — OWASP API Security Top 10* [online]. 2023. Available also from: <https://owasp.org/API-Security/editions/2023/en/0xa8-security-misconfiguration/>. [Accessed 23-03-2024].
37. OWASP FOUNDATION. *API9:2023 Improper Inventory Management — OWASP API Security Top 10* [online]. 2023. Available also from: <https://owasp.org/API-Security/editions/2023/en/0xa9-improper-inventory-management/>. [Accessed 23-03-2024].
38. OWASP FOUNDATION. *API10:2023 Unsafe Consumption of APIs — OWASP API Security Top 10* [online]. 2023. Available also from: <https://owasp.org/API-Security/editions/2023/en/0xaa-unsafe-consumption-of-apis/>. [Accessed 23-03-2024].
39. SHOSTACK, Adam. *Threat Modeling: Designing for Security*. John Wiley & Sons, Incorporated, 2014. ISBN 9781118822692.
40. NIST. *Vulnerability Metrics* [online]. 2023. Available also from: <https://nvd.nist.gov/vuln-metrics/cvss>. [Accessed 03-03-2024].
41. GOOUT. *GoOut* [online]. [N.d.]. Available also from: <https://goout.net/cs/pro-poradatele/funkce/>. [Accessed 02-03-2024].

42. PAYU. *Zabezpečení plateb* [online]. 2024. Available also from: <https://czech.payu.com/zabezpeceni-plateb/>. [Accessed 20-03-2024].
43. PORTSWIGGER. *Burp Sequencer randomness tests* [online]. [N.d.]. Available also from: <https://portswigger.net/burp/documentation/desktop/tools/sequencer/results/tests>. [Accessed 04-03-2024].
44. WEAR, Natalie Sunny. *Burp Suite Cookbook: Practical recipes to help you master web penetration testing with Burp Suite*. *Information*. 2018. Available also from: [https://www.amazon.com/Burp-Suite-Cookbook-Practical-penetration-ebook/dp/B07HRHPK6L/ref=sr\\_1\\_1](https://www.amazon.com/Burp-Suite-Cookbook-Practical-penetration-ebook/dp/B07HRHPK6L/ref=sr_1_1).
45. PORTSWIGGER. *Case Studies* [<https://portswigger.net/customers>]. 2024. [Accessed 23-03-2024].
46. NIST. *CVE-2022-24863* [online]. [N.d.]. Available also from: <https://nvd.nist.gov/vuln/detail/CVE-2022-24863>. [Accessed 19-03-2024].
47. APPLE. *Supporting associated domains — Apple Developer Documentation* [online]. [N.d.]. Available also from: <https://developer.apple.com/documentation/xcode/supporting-associated-domains>. [Accessed 10-03-2024].
48. *Discussions with GoOut developers in person*. Prague, 2023-2024.
49. ORACLE. *JDK 17.0.7 Release Notes* [online]. [N.d.]. Available also from: <https://www.oracle.com/java/technologies/javase/17-0-7-relnotes.html>. [Accessed 16-03-2024].
50. NIST. *NIST's New Password Rule Book: Updated Guidelines Offer Benefits and Risk* [online]. [N.d.]. Available also from: <https://www.isaca.org/resources/isaca-journal/issues/2019/volume-1/nists-new-password-rule-book-updated-guidelines-offer-benefits-and-risk>. [Accessed 23-03-2024].

# Contents of the attachment

readme.txt.....	a brief description of the content of the medium
src	
├ thesis.zip.....	source code of the thesis in format $\text{\LaTeX}$
├ gobuster.txt.....	output of Gobuster tool used in information gathering
├ nikto.txt.....	output of Nikto tool used in information gathering
├ nmap.txt.....	output of Nmap tool used in information gathering
├ whatweb.txt.....	output of Whatweb tool used in information gathering
├ apple-app-site-association....	Apple's meta file found during information gathering
text.....	contents of the thesis
└ thesis.pdf.....	text of the thesis in PDF format