Vulnerability Discovery - Bug Bounty Programs

Adam Zvara
Faculty of Information Technology
Brno University of Technology
Brno, Czech Republic
xzvara01@vutbr.cz

***Abstract*—This paper aims to provide an outline of bug bounty programs. It offers an overview of some selected web vulnerabilities and explores various tools that can be employed for exploiting these vulnerabilities, showcasing some of their practical examples.**

***Index Terms*—bug bounty, web, vulnerability, access control, SQL injection, authentication, SSRF, path traversal, burp, ffuf, wfuzz, gobuster, sqlmap,**

I. Intruduction

This paper is split into several sections. Starting with an overview of bug bounty programs, continuing to the most common web exploits in section III, section IV goes over tools, which can be helpful in researching vulnerabilities, and finally section V describes my process of approaching bug bounty programs.

II. Hacker-powered security

In our digitally dominated age, ensuring cybersecurity has never been more crucial. As organizations deal with protecting their digital assets, a new approach has emerged termed ”hacker-powered security”. Hacker-powered security (or ethical hacking) refers to the practice of deliberately and systematically attempting to identify and exploit vulnerabilities in computer systems, networks, or applications [4].

Bug Bounty Program (BBP) is integral component of hacker-powered security (fig. 1), specifically within the domain of ethical hacking. BBPs are programs, whose initiatives involve crowd-sourced security testing, where ethical hackers who successfully identify and report vulnerabilities to organizations receive compensation [3].

Alternative approaches to vulnerability assessment exist in contrast to Bug Bounty Programs. One such method involves integrating vulnerability assessment into the development and deployment processes of applications within the organization. This includes practices like code reviews, security audits, and the use of automated testing and scanning tools. Another approach is to delegate certain aspects of vulnerability assessment to external companies specializing in penetration tests.

This paper specifically focuses on public bug bounty programs. Examples of organizations offering such programs, which were referenced in the composition of this paper, include HackerOne[1], Bugcrowd[2], and Hacktrophy[3]. An es-

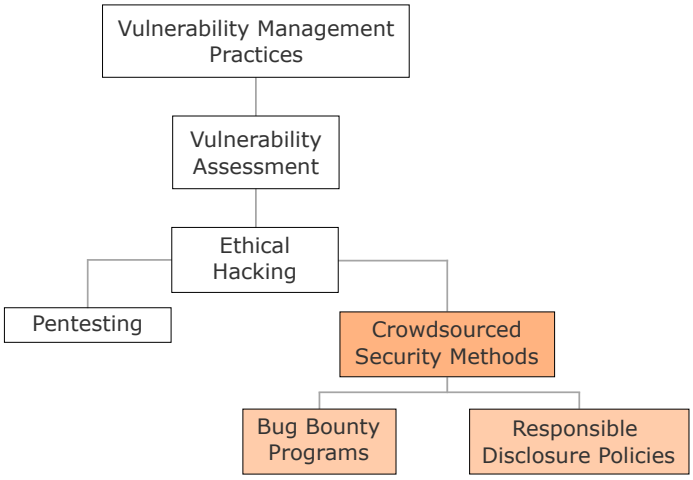

Fig. 1. BBP and Responsible Disclosure Policies as a part of Vulnerability Management Practices [3]

sential element within every BBP, contributing to the mutual protection of both the targeted company and ethical hackers against unlawful claims, is the taxonomy of the BBP. This taxonomy includes [5]:

Scope

- *in-scope* - specific systems, applications, and assets that are eligible for testing within the BBP
- *out-of-scope* - off-limit areas of the application or specific types of attacks that can not be used in BBP

Access

- shared credentials or specific accounts for testing

Rewards

- defining Vulnerability Rating Taxonomy (VRT)[4] - rating vulnerabilities within context and assigning rewards to them

Communication and guidelines

- *disclosure policies* - guidelines for responsible disclosure and reporting of vulnerabilities discovered by ethical hackers

III. Web exploting

Based on the selection of BBPs introduced in the previous section of this paper the focus will be on finding web vulner-

[1]HackerOne’s public website: https://www.hackerone.com/
[2]Bugcrowd’s public website: https://www.bugcrowd.com/
[3]Hacktrophy’s public website: https://hacktrophy.com/en/

[4]Bugcrowd example of VRT: https://bugcrowd.com/vulnerability-rating-taxonomy

abilities since web applications are among the most common targets in these programs. Before diving into the bug bounty adventure, it is necessary to know some of the most popular web vulnerabilities, and how to look for them and exploit them. This section concludes some of the most widespread vulnerabilities, which appeared on the OWASP Top Ten list[5] of web vulnerabilities in the year 2021 [2].

A. Broken Access Control (server-side)

Access control in the context of web security refers to the mechanisms and policies that regulate who can interact with and have access to resources within a web application or system [6]. It involves managing user permissions and privileges to ensure that only authorized individuals or entities can perform certain actions or access specific information. Common attacks on access control include:

Vertical privilege escalation [2] refers to the unauthorized elevation of an individual's privileges (from a basic user) to a higher level (to an administrator) within a hierarchical structure of permissions and access levels. In more complex cases, vertical privilege escalation might make a distinction between more (so-called fine-grained) user roles, granting access to specific functions allocated to that role.

Horizontal privilege escalation [2] occurs when an attacker with a certain level of access in a system attempts to gain access to the same level of privileges but in a different user account or system component (for example one user reading another user's emails).

B. Injections (server-side)

Most commonly SQL injections (SQLi) target the security of a database-driven application [11]. In an SQL injection attack, an attacker inserts malicious SQL code into the input fields or parameters of a web application, exploiting vulnerabilities in the application's handling of user input. Various types of SQL injections can be identified:

In-band SQLi - the simplest type of SQLi where the attacker uses the same communication channel to launch an attack and gather the results. Is often divided into more subcategories such as UNION-based injection (combining multiple sources of data, which are sent to the user) and error-based injection (relies on error messages thrown by the database server to obtain information about the database structure) [1].

Blind SQLi - in this type of attack, the attacker does not directly receive the results of the query. Instead, the attacker must infer the information indirectly based on the application behavior (for example different HTTP statuses, new pop-ups on the page, etc.) [1]. Is often divided into boolean-based SQLi (using boolean expressions in the query to introduce conditions, that the result is true or false) or time-based SQLi (attacker introduces delays into the query, which triggers based on condition).

Out-of-band SQLi - not very common, as it depends on some extra features being configured on the database server (DNS/HTTP protocols). This type of attack relies upon passing functions into queries, which send DNS or HTTP requests to deliver the data to an attacker [1].

C. Authentication Attacks (server-side)

User web authentication is the process of verifying the identity of a user attempting to access a web application or service [7]. It ensures that the user or entity requesting access is who they claim to be, providing a layer of security to prevent unauthorized access. There are three main types of authentication [7]:

1) something the user *knows*, such as password, keys, answers to security questions (also known as knowledge factors)
2) something the user *has*, such as mobile phone, security token (also known as possession factors)
3) something that *makes* the user or something he *does*, such as biometrics or behavioral patterns (also known as inherence factors)

Despite the apparent simplicity of authorization methods, creating secure authentication methods can be challenging. In real web applications, authentication can often be the weakest link, which enables the attacker to gain unauthorized access [2]. The most common attacks on authentication include:

Brute force attacks - when the attacker systematically tries all possible combinations of usernames and passwords until they find valid credentials. This attack relies on the assumption that weak or easily guessable passwords are in use. While brute force attacks can have generally high time complexity, there are some ways that the attacker can minimize the search space with techniques such as username enumeration. *Username enumeration* is a technique, in which the attacker can determine valid usernames by exploiting differences in server responses for valid and invalid usernames during the login process [7]. The differences can be spotted when reviewing:

- status codes - the valid responses usually contain different HTTP response codes
- error messages - the application displays distinct error messages based on whether only the password is invalid or if both the username and password are invalid
- response times - the application exhibits varying response times, with correct usernames resulting in different response times compared to incorrect usernames

Other types of authorization attacks might target multi-factor authentication (2FA simple bypass, flawed two-factor verification logic), vulnerable OAuth authentication (OAuth implicit flow, flawed CSRF protection, OAuth hijacking via redirect_uri), keeping users logged in and resetting/changing user passwords [7].

D. Server-Side Request Forgery (server-side)

Server-side request forgery (SSRF) is a type of vulnerability, that allows the attacker to cause server-side application to make requests to an unintended location [10]. In its most simplistic form, SSRF can make the server post HTTP requests

[5]OWASP Top Ten list: https://owasp.org/www-project-top-ten/

against itself, via the loopback network interface. This becomes useful if the location of the attack requires higher access privileges than the attack has, but normal access controls are bypassed when the request comes from a local machine.

Another form of SSRF could be targeted to back-end systems, that are not directly reachable by users.

E. Path Traversal (server-side)

Path traversal, also known as directory traversal, is a vulnerability, that enables the attacker to read arbitrary files on the server, which could include application code and data, credentials to other back-end systems, and sensitive operating system files [9].

IV. Tools and Systematic Approach

Looking for bugs in bug bounty programs is in some aspect similar to developing and application. You need to have a standardized plan, otherwise, one can get lost right away trying out SQL injections on every input field in every site. The very first step is to practice finding and exploiting vulnerabilities on purposely weakly secured websites, so you can get first-hand experience and try out different tools.

A good place to start learning about different web vulnerabilities is the PortSwigger Academy[6], which has numerous types of web-based content and "labs", where you can practice finding and exploiting vulnerabilities on real (purposefully) vulnerable web applications. The core parts of PortSwigger Academy are based on the book *The Web Application Hacker's Handbook* [12], which offers a more comprehensive and in-depth guide to various web vulnerabilities.

A systematic approach to testing web applications for vulnerabilities is denoted in the *Web Security Testing Guide*[7] (WSTG) developed by OWASP organization. The WSTG provides a comprehensive resource for ethical hackers. WSTG is designed to be a framework-agnostic guide that covers a wide range of topics related to web application security testing. It includes detailed testing checklists, methodologies, and best practices for various stages of the software development lifecycle. The following subsections are devoted to some parts of the WSTG, which was applied in the process of participation in bug bounty programs in this paper:

A. Information Gathering

Information gathering is the first phase of researching vulnerabilities in web applications. In this phase, publicly available information or internal information about the target is collected. This is accompanied by active reconnaissance (engaging with the target to get some information - can be detected by the target) as well as passive reconnaissance (collecting publicly available information about the target without engaging with the target). Methods for gathering information include [14]:

Search engine discovery for information leakage:

- robots.txt file - files not to be indexed by google crawlers might contain sensitive information
- search operators in different search engines (`site:`, `inurl:`, `filetype:`)
- Google hacking/Dorking - chaining multiple search operators to find sensitive files - Google Hacking Database[8]

Fingerprinting web server to identify its type and version:

- banner grabbing - examining responses from web server
- sending malformed packets and examining responses

Identify application entry points

- understanding the application flow
- detecting entry points, used methods and parameters

Fingerprinting web application

- identify used technologies and frameworks (CMS[9], security frameworks, databases), check for known vulnerabilities

Many parts of this process can be simplified using automated tools, such as:

Wappalyzer[10] - browser extension and online service that helps users identify the technologies used on websites. Wappalyzer analyzes visited websites and provides information about the software, frameworks, content management systems (CMS), web servers, database servers, and other technologies being used.

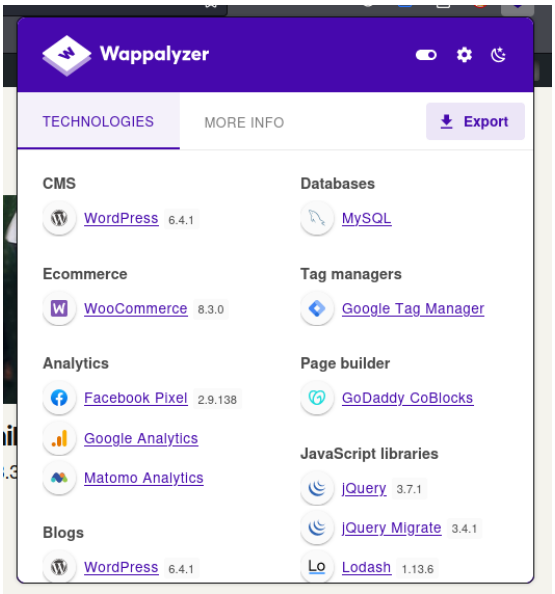

Fig. 2. Wappalyzer extension used on https://juraj.bednar.io/

Sublist3r[11] - automated python tool for subdomain enumeration using OSINT (gathers data from multiple search engines: Google, Yahoo, Bing). Example output of running Sublist3r on https://www.hackthissite.org/ shown in figure 3.

[6]PortSwigger Academy website: https://portswigger.net/web-security

[7]OWASP WSTG v4.2 website: https://owasp.org/www-project-web-security-testing-guide/v42/

[8]https://www.exploit-db.com/google-hacking-database

[9]https://en.wikipedia.org/wiki/Content_management_system

[10]https://www.wappalyzer.com/

[11]https://github.com/aboul3la/Sublist3r

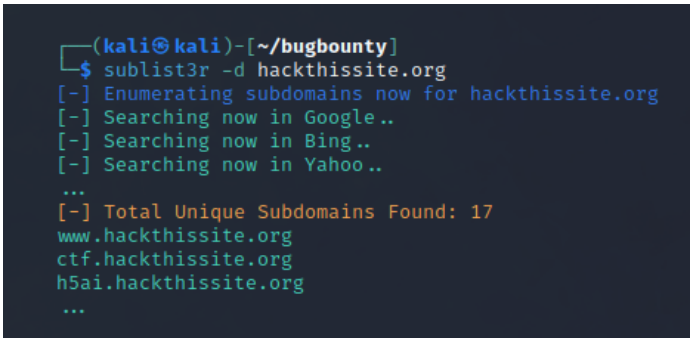

Fig. 3. Domains obtained by Sublist3r on hackthissite.org

ffuf[12]**/wfuzz**[13] - detecting subdomains using fuzzing, which is a testing technique where automated tools input various combinations of data into a system to discover vulnerabilities, error or generally discovering content [13]. Users can define custom wordlists or patterns to help the fuzzing process, which allows for more targeted and customizable testing. An example of subdomain detection using ffuf and wordlist is shown in figure 4.

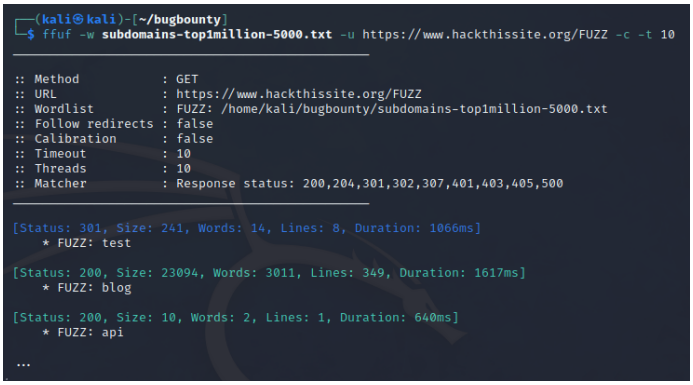

Fig. 4. ffuf obtained subdomains - hackthissite.org

B. Configuration and deployment testing

This section covers testing aspects related to server configuration and deployment. In addition to ensuring the proper configuration necessary for maintaining the application security (the application does not leak source files, configuration files, and so on), attention is directed towards different parts of used frameworks. Tests within this part might include [14]:

- validating that used frameworks are secure and not susceptible to known vulnerabilities
- testing file extensions handling for sensitive information (suspicious file extensions that might leak information include: .asa, .inc, .config, .phps, .java, .txt, .zip, .tar ...)
- discovering supported methods for different user actions
- testing for access control bypass via redirection (finding a page accessible after logging in and trying to access it with different methods)

[12]ffuf testing tool: https://github.com/ffuf/ffuf
[13]wfuzz testing tool: https://github.com/xmendez/wfuzz

List of useful tools during this phase:
Burp suite[14] **(Proxy and Repeater)** - intercepting proxy allowing users to examine and modify requests between web browser and target application. Burp repeater is a tool designed for easier manual testing of HTTP requests providing the ability to modify/repeat requests and show their response.
gobuster[15] - brute forcing URIs (directories and files - with extensions), DNS subdomains, virtual hostnames on target web servers.

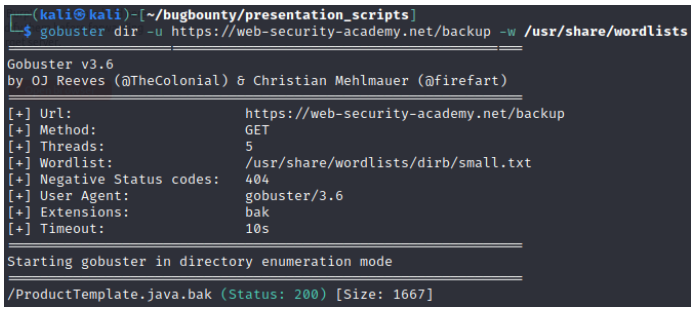

Fig. 5. gobuster enumeration of files on PortSwigger Academy lab

C. Identity management testing

Involves testing role definitions - what types of user roles exist within the system, changing between roles, and reviewing role permissions. Other activities include testing:

- user registration process
- account enumeration and guessable usernames
- weak or unenforced username policy

Tools:
Burp suite (Intruder) - designed for automating customized attack patterns, parameter manipulation, and payloads with different attack modes:

1) *sniper* - targets single position with various payloads
2) *battering ram* - apply single payload to all positions
3) *pitchfork* - apply multiple payloads to positions with a given order
4) *cluster bomb* - apply multiple payloads to multiple positions in every order (permutations of payloads)

D. Authentication testing

Evaluation of mechanisms implemented to verify the identity of users. Key aspects of this stage involve [14]:

- testing credential strength - assessing strength and complexity of user credentials, password policies, default credentials
- evaluate the correctness of multi-factor authorization if available
- bypassing authentication mechanisms via direct page requests, parameter modification, session ID prediction, SQL injection

[14]https://portswigger.net/burp
[15]https://github.com/OJ/gobuster

- testing out lockout mechanisms, ”remember password” functionality, cache weaknesses, password change or password resetting

E. Input validation testing

The most extensive part of the WSTG goes into detail about testing whether a web application handles and validates user inputs. Proper input validation is a crucial aspect of web application security, as it helps prevent various types of attacks, including injection attacks like SQL injection and cross-site scripting (XSS) [14]. Areas mentioned in this part include:

- testing for different types of XSS - reflected, stored, bypassing filters
- HTTP verb tampering, parameter pollution, smuggling request
- different types of SQL injections on various database management systems
- other types of injections: LDAP, XML, SSI, XPath ...
- code injection, command injection, SSRF testing

Tools:

Burp suite (Collaborator) - a network service that enables detecting out-of-band vulnerabilities by using its own server to catch requests from exploited applications [8]. Burp sends Collaborator payloads to the target application, which contain subdomains of the Collaborator server's domain. The targeted application uses the injected payload to interact with this server, which is polled by Burp to detect interactions.

Sqlmap[16] is an open-source penetration testing tool for detecting and exploiting SQL injections. It supports many database management systems and performs automatic testing of the most common SQL injections. Sqlmap also supports directly connecting to the database, enumerating users, passwords, hashes, automatic recognition of password hashed, dumping database schema, and many more functionalities. The following is an overview of the basic functionality of sqlmap tool from the PortSwiggers Academy exercise in Lab: *SQL injection attack, listing the database contents on non-Oracle databases* injection[17]:

- Figure 6 demonstrates a simple detection of stacked query (comment) injection. It also shows that sqlmap is able to infer the database type of the web application.
- The goal of this lab is to obtain the administrator's username and password, which is stored in an unknown table. As shown in figure 7, sqlmap can detect table names and also retrieve them using UNION-based request (while detecting the number of columns required for UNION injection to work).
- The Final step is to dump the database entries in the table found in the previous step, which can be seen in figure 8.

[16] https://sqlmap.org/

[17] portswigger.net/web-security/sql-injection/examining-the-database/lab-listing-database-contents-non-oracle

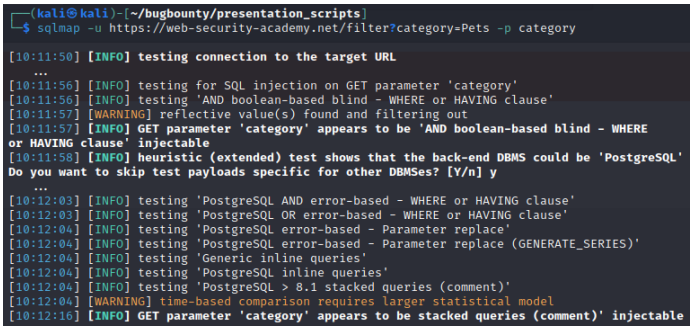

Fig. 6. sqlmap comment injection detection

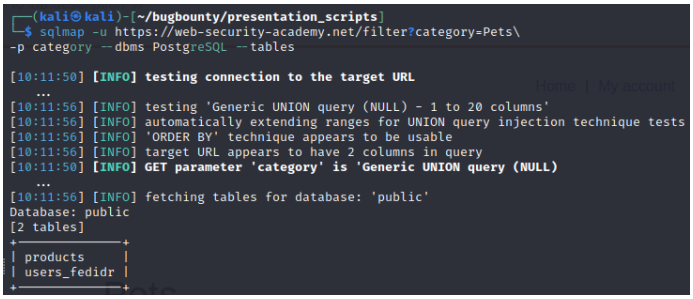

Fig. 7. sqlmap detecting table names

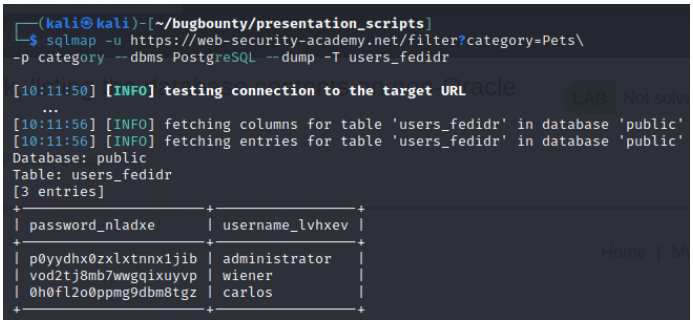

Fig. 8. sqlmap dumping table entries

V. Practicing Bug Hunting

The final section of this paper is dedicated to practicing discussed testing methodologies in previous sections. The target website is https://juraj.bednar.io, which is a part of public BBP on Hacktrophy.

A. Reconnaissance phase

First step is visual examination the website to discover its page structure and application flow:

- `/blog` - links to blog posts posted by the author
- `/shop`, `/product-category` - displaying books and courses, which the (unlogged) user can purchase
  - `/digitalny-obsah/kurzy`
  - `/knihy`
  - shopping workflow: `/cart` → `/checkout`

- `/my-account` - logging in and signing up
- other pages with names in English or Slovak language: `/kryptomeny-info`, `/reci-o-zivote`, `/o-mne` ...

Trying to ping the server address (91.210.181.167) given in the scope of the BPP returns no address. Looking up domain `juraj.bednar.io` on whois[18] only returns DNS records of servers outside the scope. Using the Wappalyzer extension gives the following information:

- CMS: WordPress v6.4.1 with interesting plugins:
  - Jetpack - security
  - LearnDash - creating and selling online courses
- Databases: MySQL
- Web servers: Nginx
- Programming languages: PHP

Other interesting subdomains detected with ffuf and SecLists[19] - the requests need to be throttled to 3 requests per second:

- `/admin`, `/dashboard` - WordPress interface (requires authorization)
- `/members` - all registered user profiles
- `/assets` - returns 403 Forbidden return code
- `/meeting` - sends request to nextcloud.bednar.io
- `/activate` - activating your account (probably after signing up)

Google Hacking results:

- using `filetype:pdf` returned some presentations from the `/assets` page
- using `filetype:txt` returned keybase.txt file

B. Attempted attacks

Since registered users can be directly observed at `/members`, one possibility of attack is password brute-forcing. The first step is to identify users, who might have weak passwords. This was done by examining usernames and picking accounts, which had short or random usernames (the thought process behind this decision was that the users were just testing the application, so they did not pick strong passwords). After picking candidate usernames the brute-force attack was realized with burp intruder and wordlist from the SecLists[20] collection. Incorrectly guessed passwords could be filtered out by regular expression matching, but the attack was soon blocked by WAF, which is present in the system (Wordfence[21]).

After creating a new profile, the user can update his profile picture. This type of interaction can be attacked by modifying the file type before sending the request. However, this attack is also protected on the server side.

Trying to access `/settings` page of another user as a form of bypassing authentication schema is also blocked by the server.

Using sqlmap to test SQL injections into multiple parts of the website (search bar, coupon) returned no results and was also blocked by the WAF, which could be partially solved by using sqlmap tamper option.

[18] https://who.is/

[19] Discovery/DNS/subdomains-top1million-5000.txt

[20] SecLists: https://github.com/danielmiessler/SecLists

[21] https://www.wordfence.com/

References

[1] ACUNETIX. *Types of SQL Injection (SQLi)* [online]. (accessed 17.11.2023). Available at: https://www.acunetix.com/websitesecurity/sql-injection2/.

[2] D. STUTTARD, M. P. *The Web Application Hacker's Handbook*. Wiley Publising, Inc., 2007. ISBN 978-1-118-02647-2.

[3] DING, A. Y., DE JESUS, G. L. and JANSSEN, M. Ethical Hacking for Boosting IoT Vulnerability Management: A First Look into Bug Bounty Programs and Responsible Disclosure. In: *Proceedings of the Eighth International Conference on Telecommunications and Remote Sensing*. New York, NY, USA: Association for Computing Machinery, 2019, p. 49–55. ICTRS '19. DOI: 10.1145/3357767.3357774. ISBN 9781450376693. Available at: https://doi.org/10.1145/3357767.3357774.

[4] HACKERONE. *The Hacker-Powered Security Report - Who are Hackers and Why Do They Hack p. 23* [online]. 2017. (accessed 16.11.2023). Available at: https://ma.hacker.one/rs/168-NAU-732/images/hacker-powered-security-report-2017.pdf.

[5] KONSTATINI, E. *Getting Started with Bug Bounty* [online]. (accessed 16.11.2023). Available at: https://owasp.org/www-pdf-archive/Getting_Started_with_Bug_Bounty..pdf.

[6] MICROSOFT. *What is access control?* [online]. (accessed 16.11.2023). Available at: https://www.microsoft.com/en-us/security/business/security-101/what-is-access-control.

[7] PORTSWIGGER. *Authentication vulnerabilities* [online]. (accessed 17.11.2023). Available at: https://portswigger.net/web-security/authentication.

[8] PORTSWIGGER. *Burp Collaborator* [online]. (accessed 19.11.2023). Available at: https://portswigger.net/burp/documentation/collaborator.

[9] PORTSWIGGER. *Path Traversal* [online]. (accessed 17.11.2023). Available at: https://portswigger.net/web-security/file-path-traversal.

[10] PORTSWIGGER. *Server-side request forgery (SSRF)* [online]. (accessed 17.11.2023). Available at: https://portswigger.net/web-security/ssrf.

[11] PORTSWIGGER. *SQL injection* [online]. (accessed 16.11.2023). Available at: https://portswigger.net/web-security/sql-injection.

[12] PORTSWIGGER. *The Web Application Hacker's Handbook* [online]. (accessed 17.11.2023). Available at: https://portswigger.net/web-security/web-application-hackers-handbook.

[13] THE OWASP FUNDATION. *Fuzzing* [online]. OWASP. (accessed 19.11.2023). Available at: https://owasp.org/www-community/Fuzzing.

[14] THE OWASP FUNDATION. *Owasp Web Security Testing Guide* [online]. OWASP, 2020. (accessed 19.11.2023). Available at: https://owasp.org/www-project-web-security-testing-guide/.