Becoming a Bug Bounty Hunter

Aditya Kore

Master of Science

School of Informatics

University of Edinburgh

2024

Abstract

The process of identifying and reporting security vulnerabilities in a web application or software in return for monetary rewards is known as Bug Bounty Hunting. This paper presents an overview of the experiences of a bug bounty hunter (BBH), underlining the challenges faced, tools used, and methods employed to find the vulnerabilities in web applications. By showcasing the real-world experiences and the lessons learned, this study contributes by bringing practical insights for individuals interested in BBH as a career option and for people just getting started. This research will mainly focus on presenting the various types of vulnerabilities/bugs discovered in the process, ranging from Cross-Site Scripting (XSS) to HTTP request smuggling. Each discovery is explained in such a way that readers can understand and replicate the same using the tools and techniques employed to discover them. To simplify the process, this paper aims to develop a custom tool to streamline the enumeration part of bug hunting.

Furthermore, the paper discusses the strategies for choosing bug-bounty programs, the advantages of sticking to specific types of attack methods, and when to switch the attack method. By sharing valuable tips, techniques, and personal experiences, this paper aims to guide and encourage upcoming Bug Bounty Hunters, providing them with an approach to success in this dynamic yet fruitful realm.

Research Ethics Approval

This project was planned in accordance with the Informatics Research Ethics policy. It did not involve any aspects that required approval from the Informatics Research Ethics Committee.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Aditya Kore*)

Acknowledgements

I would like to express my deepest gratitude to my supervisor, Prof. David Aspinall, for his invaluable guidance, support, and encouragement throughout this journey. His mentorship helped me push the boundaries of my research. Through this project, I gained a great amount of knowledge in a surprisingly short period of time, which will undoubtedly benefit me in the future as well.

I am also profoundly thankful to my family and friends for their unwavering support, patience, and belief in me, which made this work possible.

Table of Contents

Chapter 1

Introduction

In today's world where technology is constantly evolving, cybersecurity has become a critical concern for companies all over the world. As the traditional old security measures struggle to keep pace with current new sophisticated cyber threats, bug bounty programs have evolved to be an innovative approach for identifying and mitigating vulnerabilities that go unnoticed by the developers. This dissertation explores the world of Bug Bounty Hunting (BBH) from our perspective as a beginner bug researcher. This dissertation aims to provide a unique perspective on bug bounty hunting from a beginner researcher's viewpoint, offering valuable insights for other beginner hunters entering this field. The development and analysis of a custom tool (DomainMiner) contribute to the growing toolkit available to security researchers. By documenting both successes and failures in the bug-hunting process, this study offers a realistic view of the challenges and opportunities hunters face in this field.

1.1 Motivation:

Bug bounty programs represent a model where organizations collaborate with ethical hackers & security researchers to find and report vulnerabilities/bugs present in their systems, websites, or applications. In return for finding these bugs, organizations reward these hunters with a monetary reward "bounty" or hall of fame inclusion. This creates a mutually beneficial ecosystem that enhances the security of the systems while offering rewards to the hunters for their efforts. They provide continuous testing, and diverse perspectives, and often uncover vulnerabilities that might have been missed by conventional methods. This research aims to explore the methodologies, tools, challenges, and impacts of bug bounty hunting, contributing to the ongoing conversation

on effective cybersecurity strategies in an increasingly complex digital landscape.

1.2 Research Objectives:

This study aims to provide an exploration of bug bounty hunting from the perspective of a novice researcher. By combining practical experience with analytical research, this dissertation aims to bridge the gap between the initial knowledge of aspiring bug bounty hunters and the complex, multifaceted nature of real-world vulnerability discovery. The research aims to fulfill the following primary objectives:

- To explore the process of bug bounty hunting from a beginner's perspective, documenting the learning curve, challenges faced, and bugs encountered.
- To evaluate the effectiveness of various tools and methodologies in identifying vulnerabilities within the context of bug bounty programs.
- To develop and assess a custom tool designed to enhance the efficiency of the bug-hunting process.
- To contribute practical insights and strategies that can benefit both aspiring bug hunters and organizations looking to implement or improve their bug bounty programs.

1.3 Structure of this Dissertation:

This dissertation is arranged correspondingly, chapter 2 provides a background on bug bounty programs, and common vulnerabilities, acting as a base for the findings in the following sections. Chapter 3 gives the methodology employed in this research, including tool selection, custom tool development, and the overall bug-hunting process which aims to answer the **"What"** and **"Why"** aspects of the research. Chapter 4 presents the experimentation phase, outlining the specific tests conducted, and how the tools were utilized to obtain the results, demonstrating the **"How"** part of the research. Chapter 5 discusses the results of the research, including successful bug discoveries and lessons learned from unsuccessful attempts. Chapter 6 presents our tool DomainMiner, and how successful it is by comparing the results obtained with other tools. Chapter 7 provides an evaluation of our research followed by chapter 8 to conclude the research.

Chapter 2

Background

This chapter provides a brief overview of the key concepts, vulnerabilities, and context necessary to understand before indulging in the field of bug bounty hunting.

2.1 Bug Bounty Programs

Bug bounty programs are a place where organizations invite ethical hackers to discover and report bugs on their websites in return for rewards. The bug bounty cycle includes various stakeholders including organizations running bug bounty programs, Bug bounty platforms like HackerOne [26], Bugcrowd [4], individual bug hunters, and security researchers. These programs have gained popularity as a cost-effective method to enhance cybersecurity, complementing traditional security measures. We found this paper [45] to provide a good idea for the bug submission quality for both hunters and companies, which we have applied in our project. This paper identifies best practices and areas for improvement, focusing on five key aspects: scoping, timing of crowd engagement, submission quality, firm-researcher communication, and managing hacker motivation which we feel is important in this domain. The paper [82] highlights the value of bug bounty programs as a complementary approach to traditional security practices and provides a basis for further studies into optimizing reward structures and understanding hacker motivations.

2.2 Vulnerabilities Encountered:

Before presenting our bug findings, we will first provide a brief overview of the vulnerabilities we discovered and discuss their impact to highlight the significance of

our results. We have used various HTTP status codes in our dissertation to provide credibility for our findings, the below figure 2.1 shows what each code is.

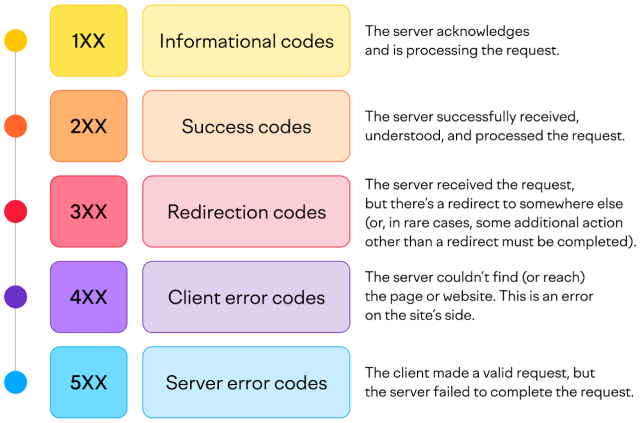

Figure 2.1: Different HTTP status codes [76]

Cross-Site Scripting (XSS) is a very common vulnerability allowing attackers to inject malicious scripts into websites viewed by other users. Leading to data stealing, session hijacking, and unauthorized actions on behalf of users. These are significant due to their potential to compromise user data and the integrity of web applications, making them critical in security assessments [24].

For **File Upload** bugs, we referred this study [63] which reveals that file upload vulnerabilities are a significant risk to web application security. These vulnerabilities arise from inadequate validation of user-uploaded files, potentially allowing attackers to upload and execute malicious code on the server. This can be fixed by including proper sanitizations and safety checks

Cross-Site Tracing (XST) bug exploits the HTTP TRACE method, where an HTTP request is reflected back to the user without sanitizations, which can be used to bypass security controls such as the same-origin policy. In brief, HTTP methods have different functionalities, GET retrieves data, POST sends data to create/update, and TRACE reflects the request for debugging. By leveraging XST [23], attackers can potentially access sensitive information, such as cookies and authentication tokens, leading to unauthorized access and data breaches [23].

WordPress is a content management system (CMS) [39] that powers more than 40% of all websites [9]. We discovered 2 types of bugs involving this CMS [39], one was Denial-of-Service (DOS) which impacts the availability of the resource, by exploiting `wp-cron.php` that is a WordPress feature that simulates a system cron job, allowing task scheduling & running at specific intervals, such as checking for updates. However,

it can be exploited by sending excessive requests to the `wp-cron.php` file, leading to Denial of Service (DoS) [8] attacks by overloading the server with unnecessary tasks, potentially causing performance degradation or downtime which causes loss of revenue, damage to reputation, and potential liabilities for failing to provide expected services [44].

Security Misconfigurations occur when security settings are incorrectly configured or left at default values, making systems vulnerable to attacks. This can have many implications, in our case we used the null byte to show improper sanitizations and bypassing input validation to cause server errors, implying buffer overflow errors which can mean the execution of arbitrary code, overwriting other storage space to view unauthorized data [10, 1].

HTTP Smuggling is a complex attack that exploits inconsistencies in the way front-end and back-end servers handle HTTP requests through various HTTP headers. HTTP headers are key-value pairs in a request/response that give metadata, such as content type, encoding, and authentication, influencing how data is processed and displayed. By manipulating HTTP headers, attackers can inject malicious requests that bypass security controls, leading to unauthorized data access or server compromise. The impact of HTTP Smuggling is severe, as it can lead to data breaches, session hijacking, and other critical security incidents [32]. There are various methods to smuggle requests, the main types include the **CL.TE (Content-Length Transfer-Encoding)** where, the front-end server interprets the HTTP request using the "Content-Length" header to determine the message body size, while the back-end server uses the "Transfer-Encoding" header. This discrepancy can be exploited to smuggle an additional request by manipulating the request body length [65]. **TE.CL (Transfer-Encoding Content-Length)**, the front-end server uses the "Transfer-Encoding" header to determine the message body, while the back-end server uses the "Content-Length" header [65]. **TE.TE (Transfer-Encoding Transfer-Encoding)** is exploited by both the front-end and back-end servers use the "Transfer-Encoding header", but one of the servers can be tricked into ignoring it through obfuscation. This can lead to a situation where the servers disagree on the boundary of HTTP requests, allowing for request smuggling [65]. Lastly, the **CL.0 smuggling (Content-Length only)** was used by us to find the bug, it involves sending a request with a "Content-Length: 0" header, leading the front-end server to think there's no body while the back-end misinterprets the following data as a new request. This desynchronization allows attackers to inject malicious requests, potentially bypassing security controls or accessing unauthorized resources.

2.3 Essential Tools for Bug Hunting:

Tools play a crucial role in the bug bounty hunting process, serving as the primary way by which hunters discover, analyze, and exploit vulnerabilities. The toolkit of a bug bounty hunter typically includes a diverse array of software designed for different aspects of security testing. Web application scanners like Burp Suite are essential for analyzing HTTP traffic and detecting common web vulnerabilities. Network mapping tools such as Nmap allow hunters to discover open ports and services on target systems. Additionally, specialized tools exist for specific types of vulnerabilities or technologies – for instance, SQLmap for SQL injection or WPScan for WordPress vulnerabilities. This paper [3] provides a good comparison of various vulnerability discovery techniques and the use of automated tools for discovery was advised by the authors, making it imperative to select good automated tools.

As the field evolves, so do the tools, with new ones constantly emerging to address novel attack vectors or improve existing techniques. Proficiency in selecting and utilizing the appropriate tools for each scenario is a key skill that distinguishes effective bug bounty hunters, allowing them to efficiently identify vulnerabilities that might be overlooked by automated scans or less experienced researchers which we have covered in our research.

Chapter 3

Methodology

This chapter details the methodology I employed in my bug bounty hunting research, emphasizing a beginner's learning journey. It covers tool selection, custom tool development, and the process of vulnerability hunting, laying the groundwork for the experimentation and results chapters. The progression in skill development is reflected in the results chapter, demonstrated by the increasing severity and complexity of the bugs found.

3.1 Tool Selection

Selecting the right tools was crucial for effective vulnerability assessment. As a beginner, we prioritized tools that balanced power with ease of use, enabling efficient learning and application of new techniques. Two books [40, 42] were particularly helpful in guiding us through the process. We evaluated tools based on beginner-friendliness, effectiveness in identifying vulnerabilities, community support, documentation, and cost. For each of these use cases, we used this tools, in the case for **Content Discovery** we search for hidden pages and directories on websites, for this, we used **FFUF** [20], a fast and efficient tool that helps us uncover potential entry points that aren't immediately visible. We typically run FFUF with custom wordlists against target URLs, analyzing the responses to identify valid endpoints. For **Network Scanning**, which involves mapping out the structure of a target's network, identifying active hosts, open ports, and running services, we relied on Nmap [54] for this task due to its versatility and powerful scanning capabilities. Our process includes running various Nmap scans [54], from quick ping sweeps to more comprehensive port and service scans. In **Web Application Testing** We analyze how web applications handle user inputs and manage data. Burp

Suite [43] is our go-to tool, allowing us to intercept, inspect, and modify web traffic to identify potential vulnerabilities. We use Burp's proxy [43] to capture traffic, then use its various modules to test for other vulnerabilities. Lastly, for **Technology Stack Identification** Wappalyzer [83] helps us quickly identify the tech stack before diving deep into the website, allowing us to tailor our testing approach accordingly. We simply browse the target website with Wappalyzer [83] enabled, which automatically detects and lists the technologies in use.

3.2 Creating a Custom Tool: DomainMiner

Recognizing the limitations of existing tools, for the crucial enumeration phase, we developed a custom tool called DomainMiner because existing subdomain enumeration tools provided varied outputs, and we wanted to simplify the merging process, gather more data, bypass usage limits, and use it as a learning exercise. DomainMiner is a shell script that runs multiple tools, merges their outputs, removes duplicates, get the IP addresses of the domain, checks HTTP status codes [51] for it, and filter out non-working subdomains, thereby streamlining the enumeration process. It is free, has no usage limits, and rivals paid tools in effectiveness. Users simply input a domain and receive a list of working subdomains with their corresponding IP addresses and HTTP codes [51].

3.3 Bug Bounty Hunting Process

This section outlines our structured approach to bug bounty hunting, detailing the methods and strategies we used to enhance my learning and effectiveness.

3.3.1 Program Selection Criteria

Selecting a bounty program was crucial, here are a few things we looked for, firstly, the **Program Scope**, we prioritized programs with broader scopes, as these offered a wider range of vulnerabilities to discover, providing more opportunities. We looked at **Statistical** data like last submission made, number of bugs reported, number of reports resolved and duplicates, most reported domain, and announcements to see if the program had enough opportunities to conduct bug hunting. We also looked at **Program Responsiveness** due to our limited research time, we chose programs for

their responsiveness to submitted reports, as quick feedback was vital for improving our understanding and refining our techniques.

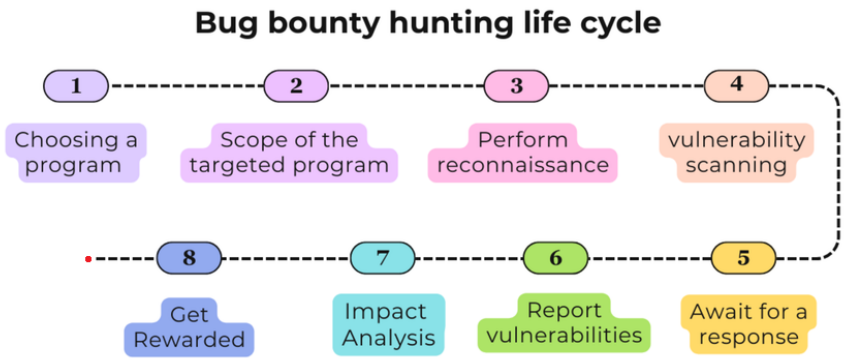

Figure 3.1: Bug Bounty Lifecycle [70]

3.3.2 Vulnerability Discovery Process

While reconnaissance, enumeration, and vulnerability scanning are the backbone, our bug-hunting process as beginners was more iterative and adaptive. This section outlines the realistic, often non-linear journey we experienced. **Target Assessment and Scope Definition:** Before analysis and vulnerability testing, we thoroughly assessed the target and understood boundaries. This involved studying the program's scope, clarifying ambiguous points with program managers to avoid out-of-scope testing and ethical violations, and expanding our target using techniques like Google Dorking and tools like Assetfinder [79], built around this paper [16]. Previously disclosed vulnerabilities helped gain insights into potential weaknesses. Next, we employed **Iterative Testing and Learning Loops** which featured continuous testing, learning, and refinement loops. We started with basic vulnerabilities like XSS [41], using simple tools and manual testing to gain confidence. As we grew more familiar with the target, we gradually incorporated advanced techniques and tools. Analyzing failed attempts led to valuable insights into the target's security. We frequently pivoted our attack vectors based on findings, such as shifting focus to API endpoints when the front end was well-secured. We also performed **Documentation and Reflection** where we maintained a detailed hunting journal to document findings and thought processes, including dead ends and false positives, which refined our intuition over time. After each session, we conducted retrospectives to analyze successes and failures, accelerating our learning curve. We also created a personal ledger to document techniques, tools, commands, and target-specific

notes, which became a valuable resource for future hunts.

3.4 Learnings, and Risks involved

Continuous learning and skill development were central to our approach. We regularly studied new and significant security vulnerabilities, such as HTTP smuggling, cache poisoning, and the OWASP Top Ten, which, when combined with analyzing public bug bounty reports, provided valuable insights and helped us discover new attack vectors.

A key aspect of our strategy was strict adherence to scope considerations. We made it a priority to follow each program's standard guidelines, ensuring we stayed within the defined scope, used non-intrusive methods, and provided detailed reports. Whenever we needed to test something out-of-scope, such as Denial of Service attacks, we always sought permission first to maintain ethical and legal integrity.

However, our journey was not without risks. Legal risks were a significant concern, as unauthorized testing or exceeding program boundaries could lead to serious consequences. We also faced technical risks, where our methods might unintentionally harm systems, potentially causing downtime or data loss. Managing these risks required careful consideration and strategic decision-making, adding a layer of complexity to our learning and bug-hunting activities.

Chapter 4

Experimentation

This chapter applies the methodology from the previous chapter, offering a step-by-step guide on our bug bounty hunting process. This chapter forms a bridge between our Methodology and Results chapter by answering **"How"** we performed bug bounty hunting. We'll cover tool configurations, effective usage, and practical implementation, providing use cases to illustrate how we found bugs. This section also aims to guide beginners through tool setup and usage, helping them replicate the results discussed in Chapter 5.

4.1 WPScan:

WPScan was used on websites using WordPress, on some instances where the Wappalyzer extension [83] couldn't guess if the website was using WordPress or not, WPScan [86] consistently gave us the correct information, along with the WordPress version number. We were particularly interested in the WordPress version number, as the vulnerabilities that even WPScan [86] might have missed could be discovered by doing a simple Google search. Mainly using the user enumeration function of this tool, as it can even identify users that aren't mentioned in the `\wp-json\v2\users` endpoint (for example: https://abc.com/wp-json/v2/users), we found an Information Disclosure bug using this successfully and listed users who weren't mentioned in this endpoint. The most used WPScan [86] command was this:
`wpscan --url http://website.com/ --enumerate u,p,t --random-user-agent`, depending on the use we changed the arguments to `--enumerate vp,vt`, this only enumerates vulnerable themes and plugins and requires "sudo" command to be enabled. This argument gives a clean output, making it easy to distinguish between unwanted plugins

and vulnerable ones when a website uses multiple plugins.

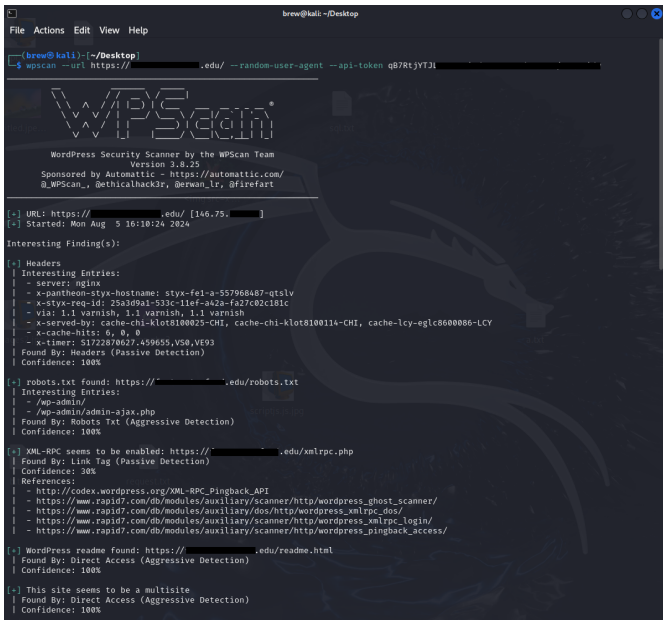

Figure 4.1: WPScan [86] detecting vulnerabilities on a website

Above figure 4.1 displays the WPScan interface and the vulnerabilities it discovered. WPScan was instrumental in identifying files like `xmlrpc.php` and `wp-cron.php`, which we used to submit a DoS (Denial of Service) bug. Additionally, as described in [74], we used it to brute-force WordPress admin credentials. For WordPress vulnerabilities, WPScan combined with Nikto/Nmap [54] is unmatched.

4.2 FFUF:

Every bug bounty hunter needs an automatic fuzzing tool in their kit. Tools like GoBuster [36], dirbuster [35], dirb [34], wfuzz [37], and FFUF [20] are tool directory fuzzing, filename, extensions, and sub-domains. Among these, we found wfuzz [37] and FFUF [20] to be the fastest, but we chose FFUF for its simple interface and easy-to-read output, despite wfuzz being slightly faster. To maximize FFUF's potential, we used SecLists, a comprehensive collection of data sets including usernames, passwords, URLs, filenames, sub-domains, and fuzzing payloads. SecLists proved crucial in identifying vulnerabilities, especially when compared to the smaller wordlist provided by "dirb" [34] in Kali OS, which missed several important endpoints detected by FFUF [20] and SecLists [47].

In the below figure 4.2, we see that FFUF tool combined with the SecLists wordlist,

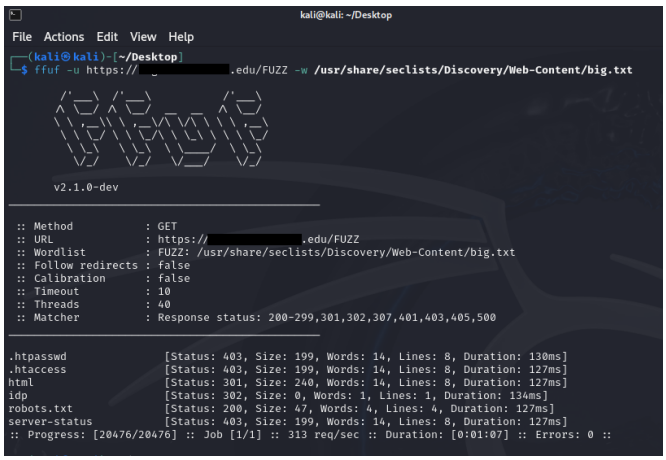

Figure 4.2: FFUF finding interesting files on a website

found some interesting endpoints such as the `.htaccess`, `.htpasswd`, `robots.txt` which we can try accessing by using other tools like 403 bypasser [88]. The most useful feature of this tool is that, in its output, for each file/endpoint it discovered, the tool gives us HTTP status, size of response, length of response, and response duration, just like the results section in BurpSuite Intruder [43] which is very helpful in the long run and saved a lot of time.

4.3 Nmap:

Network Mapper or Nmap [54] is a flexible tool for network discovery and port scanning, providing detailed insights into network structures, open ports, and running services, making it crucial in early vulnerability assessments. We relied on these books [58, 6] to learn and apply effective Nmap methodologies. We primarily used the following Nmap commands, especially for server information where Wappalyzer [83] fell short. Nmap's key uses include network sweeps to identify live hosts, comprehensive port scans, service and version detection, targeted script scanning, and OS profiling to streamline attacks [54].

Our first command `nmap -A -v example.com`, is our go-to for initial reconnaissance and ran on every website. We used the `-A` flag which enables aggressive scan options. This is similar to combining `-O (OS detection), -sV (version scanning), -sC (default script scanning)` flags. When we don't find the information we need, or if the scan fails, we try these below two commands,

`nmap -A -v example.com --script=vuln -p 80,443` was used to perform a vul-

nerability scan on specific ports we identified previously. These scripts are part of the Nmap Scripting Engine (NSE) [54] and are used to detect known vulnerabilities on the target. Sometimes the target responds with all the ports being open, this is done to obscure the port scanners. To combat this, we run the below:

The last command `sudo nmap -F example.com -v -sV -f -Pn --bad-sum` was used for a less aggressive, silent scan to identify open ports. The `-F` flag performs a fast scan of the 100 most common ports, while `-f` fragments packets to potentially bypass simple firewalls. The `-Pn` flag disables host discovery, useful when pings are blocked. `--bad-sum` sends packets with incorrect checksums to test how the target handles bad packets, aiding in evading detection or triggering specific responses. Below figure 4.3 shows Nmap being useful for finding bugs.

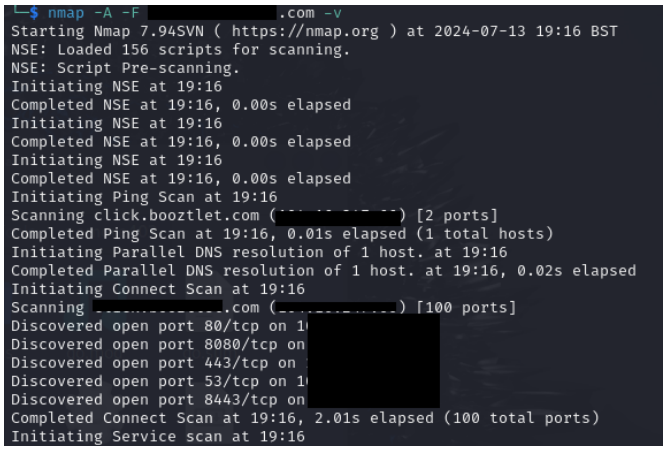

Figure 4.3: Nmap tool output finding open ports on a website

4.4 BurpSuite:

BurpSuite was used on almost all websites, a toolset for web-application security testing [43]. While researching the best settings and tools to use inside BurpSuite, we found these two books [84, 68] to offer a comprehensive guide on how to set up and use BurpSuite, building on these two books, through testing and research, we found the below approaches to its tools, fit the best for our project:

4.4.1 Proxy:

Our browser's proxy was connected to BurpSuite at all times as BurpSuite's inbuilt vulnerability scanner continuously scans visited websites in the background, providing

valuable insights. We frequently used the proxy to intercept requests, particularly during user authentication, file uploads, and API calls. Additionally, BurpSuite's [43] feature to intercept server responses (`right click on Intercept > Do Intercept > Response to this request`) was invaluable for examining server response formats and HTTP status codes. In some cases, we bypassed restricted or redirected pages (e.g., 403, 301/302) by modifying the status code to 200 OK in the intercepted response.

4.4.2 Repeater:

Repeater tool in BurpSuite [43] allows manual manipulation and sending of individual requests, particularly when interception isn't sufficient. Repeater was our most used tool due to its versatility to modify requests and analyze server responses, especially for login pages using HTTP POST with JSON bodies, as detailed in [46]. By injecting SQL commands into headers via Repeater, we could determine if further vulnerability exploration was needed. Articles [30, 62] were helpful for manipulating password reset requests, allowing us to send OTP codes to multiple accounts, repeater also aided in detecting HTTP smuggling vulnerabilities by grouping requests and using "send group in sequence" to observe desync attacks. We recommend trying methods like sending POST request JSON bodies in URLs or different formats as shown in fig 4.4.

Figure 4.4: Comparison of different body JSON request methods

4.4.3 Intruder:

Intruder's functionality to send multiple requests in a short span of time was very useful to brute-force admin panel usernames and passwords, we also used a directory wordlist to brute-force different endpoints and files in a website. Intruder's Sniper function was useful for sending multiple injection queries, which we could then observe in the

results tab. This article [72] summarizes all the different functions of the BurpSuite intruder. For example, in the below fig 4.5, we can see the results page of the BurpSuite intruder, where all the different request payloads are listed with columns response, response length, response received time, making it easier to find the outlier when brute-forcing data. Brute-forcing is sometimes mentioned in the Out-of-scope section in some of the bounty programs, so it's advisable to check the program description before proceeding with any attacks, as sending these many requests in a short time, may be harmful, as the server might block our requests, specifically throttle our requests.

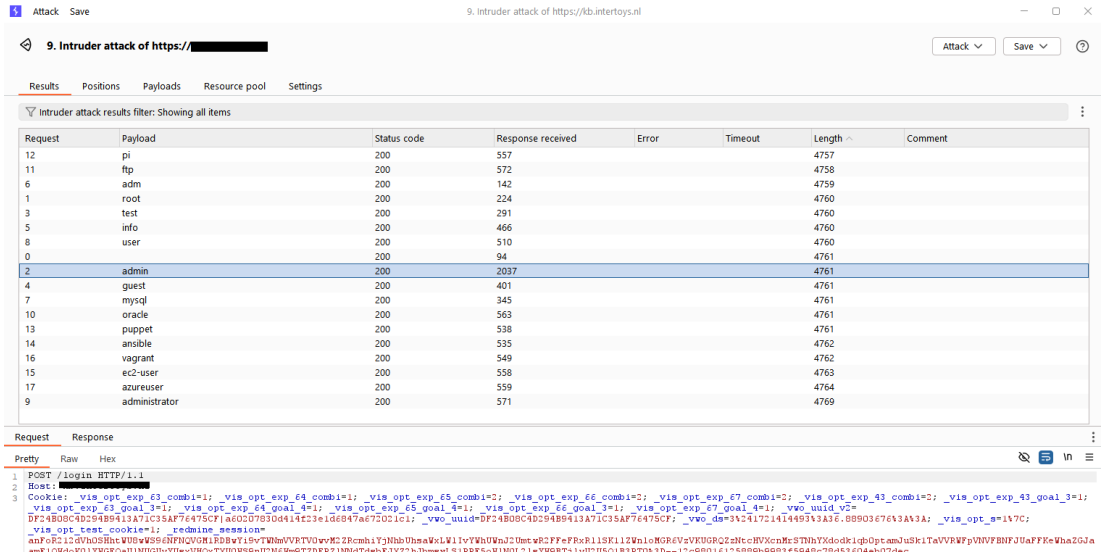

Figure 4.5: BurpSuite Intruder results tab.

The above figure 4.5 shows my attempt to brute-force a login page of a website using a username wordlist, in the figure you observe that the highlighted request with **"admin"** is different from other requests, while the status code, response length being almost similar, but the response received is significantly different, this implies that the database is running additional checks if the username is found, the database looks if the password is also similar, this causes delay to receive the response, we advise to look for such inconsistencies in your findings to find new vulnerabilities or hints on how to approach a particular bug.

4.4.4 Decoder:

Another tool that we used apart from the above was the decoder tool. BurpSuite's decoder allows us to decode or encode any data into Plain Text, URL, HTML, Base64, ASCII Hex, Hex, Octal, Binary, and Gzip [43]. Decoder is useful in scenarios where we want to URL encode certain characters or HTML encodes them to bypass certain

firewalls or sanitization. XSS bugs [41] found by us were possible only because we could URL encode the payloads in the intruder, this also is possible in the intercept request part of BurpSuite, where if we select the text we want to encode/decode and `right click text > Convert select> URL/HTML/Base64 > decode/encode` this simplicity helps us to URL encode characters on the go in the URL, thereby simplifying our hunting. We can see in the below figure 4.6, we URL encoded our XSS payload.

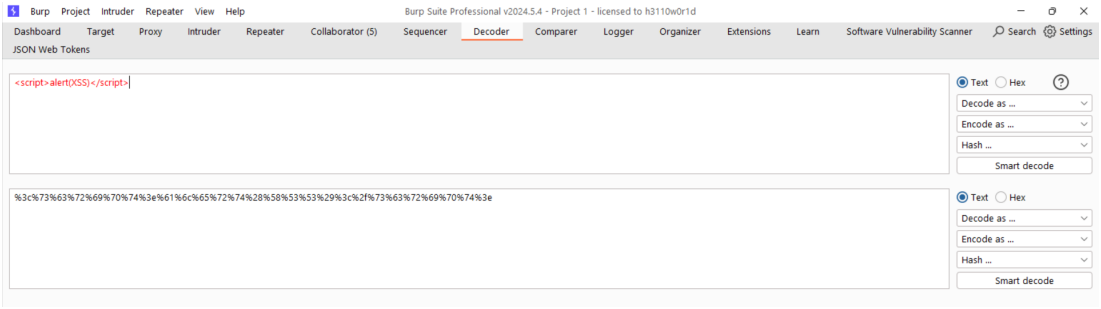

Figure 4.6: BurpSuite Decoder

Decoder tool can be used to double encode characters to perform path traversal exploits [2]. Path traversal attacks occur when we traverse out of the current directory of the web server, for example, there is a server that has the **"index.php"** in the base directory using a Linux file system. To find Javascript files or any other pages, the document makes a call to the server from URL, like `https://example.com/pages?url=/images/apple.png`, so now to perform a directory traversal, we just edit this URL as `https://example.com/pages?url=/../../etc/passwd`, giving us `passwd` file on the server, which is critical. But, nowadays servers are up-to-date & have URL sanitization [19] in place to remove **"../"** characters from URL, so we use the Decoder tool here to modify the payload and only encode part of the payload such as - `%2e%2e/%2e%2e/etc/passwd`, `..%2f..%2fetc/passwd` or `%2e%2e%2f../%2e%2e%2f../etc/passwd`. We encourage hunters to try different payloads for directory traversal, here is an example of an attack vector:

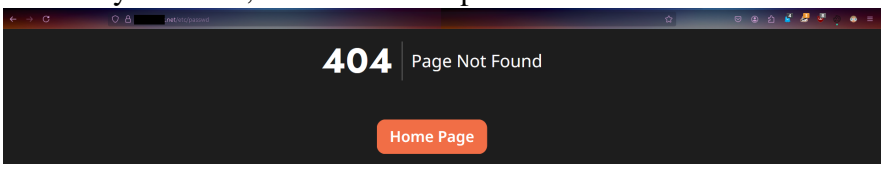

Figure 4.7: Normal Directory Traversal

Figure 4.8: Directory Traversal through nextjs [80] server

Here in the above two figures 4.7 & 4.8 we observe that performing directory traversal directly on the base URL will give us a **404 not found error**, as the server firewall sanitized URL and removed the `../../` from the URL as it occurs after the domain. However, if we try doing the Directory Traversal through the nextjs [80] `image?url=` functionality, we can see that the server doesn't remove `../../` in this case and we can traverse out of the base directory, this is supported by the fact that we receive **403 Forbidden error** and get a text that the firewall blocking us is the "Microsoft Azure Application Gateway" Web Application Firewall (WAF) [18]. This finding tells us that we know there is a folder **etc** on the server, but we can't access it due to the WAF [18], now to bypass the WAF [18] and the 403 error, we can use different encoded payloads in the URL, such as encoding the `../../` through BurpSuite Decoder, or use a specific tool for this, which we will see in further sections.

4.4.5 Burp Extensions:

BurpSuite offers various extensions in its **BApp Store** [43], enhancing its functionality. We primarily used the **"HTTP Smuggler"** [64] and **"Software Vulnerability Scanner"** extensions. These can be installed directly from the BApp Store or via external links [64]. Author Peter De Witte of this article [85] provides guidance on using the HTTP Request Smuggler extension for automating HTTP smuggling exploits. While we explored several methods of HTTP smuggling through this extension and found some interesting results, we still preferred the manual approach of editing the Content-Length and Transfer-Encoding headers. Authors huang et al. of this paper [29] offers insights into different types of HTTP smuggling (CL.TE, TE.CL, CL.CL, and TE.TE) and explains how servers interpret smuggled requests. The **Vulnerability Scanner** extension is another valuable tool, especially for beginners. It automatically analyzes responses within the target scope, identifies specific software versions, and cross-references them with the Vulners database to detect associated vulnerabilities, including CVEs and relevant advisories. This extension can serve as a secondary passive vulnerability scanner, saving time and enhancing the effectiveness of vulnerability detection. Users can download this extension through the BApp Store or from this Github repository [66].

4.5 403 Bypasser:

During our experimentation, we regularly encountered the HTTP 403 Forbidden error, which means that the Web-Application-Firewall (WAF) was blocking our access to a particular resource. To circumvent this, we used various tools to bypass 403 errors. This particular method is described by authors Sharma et al. of this paper [75], authors of this paper provide us with several 403 bypass methods including HTTP Method Manipulation, Request Header injection/manipulation, Path injection, and brute-force directories. These methods are implemented in tools such as the one we used [88]. This tool was used by us on pages where we encountered the 403 status error. Readers can find various tools with similar names on GitHub, but most of the tools perform just the same. This tool sends multiple requests with different payloads such as using different HTTP methods, adding special characters in the URL, and using different Host headers like `X-Forwared-Host`, `X-Host`, `Client-IP`, etc.

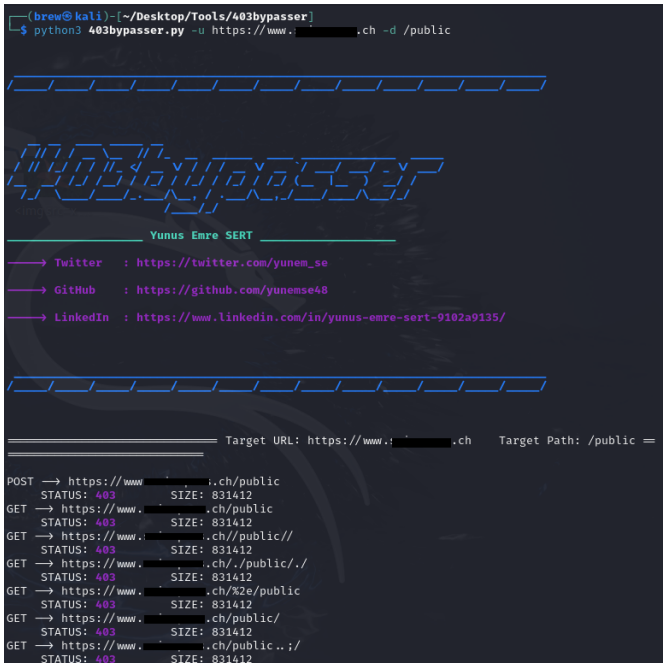

Figure 4.9: 403bypasser tool output.

We suggest users use this tool, and other techniques mentioned in the paper [75] in their bug bounty hunting if they encounter any 403 errors. Getting 403 errors does not mean the path is closed, 403 errors give us a lot of information, first, it's a clear indication that the file is present on the server and we can connect to it. Second, some WAF [18] also give us the server version on the 403 page which can be searched to find different access techniques.

4.6 Browser and Extensions:

Selecting a browser is not a significant task as most modern browsers offer the basic functionalities required for bug bounty hunting. We chose the Firefox Developer browser for our hunting because it performed better than Google Chrome [56], used fewer system resources, and offered useful features like Firefox containers. Containers act like separate browser instances within the same window, with no data shared between them, similar to sandboxing [7]. We configured the Burp proxy(target domain) in one container while using normal tabs for other purposes, which helped us keep the Burp scope focused and avoid unnecessary data collection.

We tested various extensions to aid our process, allowing us to run enumeration tools in the background while conducting manual website inspections. Extensions provided instant information without the need to open command prompts or wait for tool outputs. We extensively used three extensions: Wappalyzer [83], Cookie-Editor [50], and ModHeader [49]. Some bug bounty programs require hunters to include specific headers in their HTTP requests, such as `X-Intigriti-username`, `X-Bug-Bounty`, to distinguish bounty hunters from actual attackers. We used the ModHeader extension [49] to modify all requests made from the browser, which also applied to BurpSuite Proxy. Although BurpSuite can handle custom headers, editing them repeatedly was time-consuming, and the extension significantly improved our efficiency.

The Cookie-Editor extension [50] was used to view and edit cookies, making it easier to inspect data transferred through cookies and perform attacks like cookie hijacking, poisoning, and injection which are mentioned by authors Zheng et al. in the paper [89].

The Wappalyzer extension [83] was indispensable in our research. It allowed us to quickly identify the technology stack used on any website, including database types, content management systems, web servers, frameworks, analytics tools, authentication methods, and firewalls. This insight was crucial in bug bounty hunting, as understanding the underlying technologies helped us identify potential vulnerabilities and attack vectors. By integrating Wappalyzer [83] into our workflow, we could focus on specific technologies with known security issues, improving the efficiency and effectiveness of our assessments and increasing our chances of finding exploitable vulnerabilities.

Chapter 5

Results

In this section, we will look at the results obtained after following the methodology and performing experimentation in sections 3 and 4. Below are the vulnerabilities we encountered during our bug-hunting process, after implementing the techniques discussed. The vulnerability types found were XSS [41], File upload, WordPress, and HTTP Smuggling, we will also show the results obtained by our tool DomainMiner.

5.1 Cross-Site Scripting (XSS)

Description:

The first type of bug we found during our testing was a Cross-Site Scripting vulnerability, which was found on two websites, one on an image hosting website (bug-1) and another on an Indian Government website (bug-2). These bugs are a type of XSS called a Reflected XSS attack [41] where an attacker crafts a malicious URL of a website, where if a user clicks on the link the malicious script gets executed in the user's browser. These types of attacks are the most common attacks [60] performed by bounty hunters.

Severity:

Our XSS bug on an image hosting website and a government website mirrors CVE-2023-39575 [15]. This finding highlights how XSS vulnerabilities persist across various web applications, from specialized software to public-facing government sites, emphasizing the ongoing challenge of input validation and the importance of robust security measures in diverse web environments.XSS bugs like these have a severity rating of around 6.0 to 7.0 in the CVSS 3.x metric system [52] which has the severity of medium. Reflected

XSS is considered a serious vulnerability in websites where user logins and session cookies are implemented, as an attacker can craft a malicious script that sends the victim's credentials to the attacker's server or hijack sessions as mentioned in this paper [71] without the knowledge of the victim. For other websites, phishing attacks can be caused using this attack vector.

Steps to Reproduce:

For bug-1, on the search part of the website, we observed that the words entered in the search bar were getting displayed on the webpage inside a `<strong>` HTML tag, and we could escape sanitizations through this particular HTML tag `<a href>`, we tried a wordlist with all the HTML tags in BurpSuite Intruder [43], and checked the response length for all request to see which tag gave us an alert box in the HTML page. Now, to trigger XSS from the `<a>` tag, we used the `onmouseover` function to trigger an alert box, based on this paper [33]. HTML event attributes [81] are used to record events and execute javascript code in webpages, taking advantage of this, we crafted this payload:

```
</strong><a href=blahblah onmouseover=alert("XSS")>XSS Here</a>
```

that closed the strong tag and inserted the **a href** tag with the `onmouseover` attribute to trigger the XSS when a user hovers over the text. For bug-2, we checked the HTML code to find out that the search terms were reflected inside the search bar on the HTML page without any HTML sanitization [19], now to execute a script here, we used this code: `"><script>alert('XSS')</script>"`

This closed the HTML event attribute `value=""` so we can escape it. We used different quotation marks, so that our script doesn't break the HTML code.

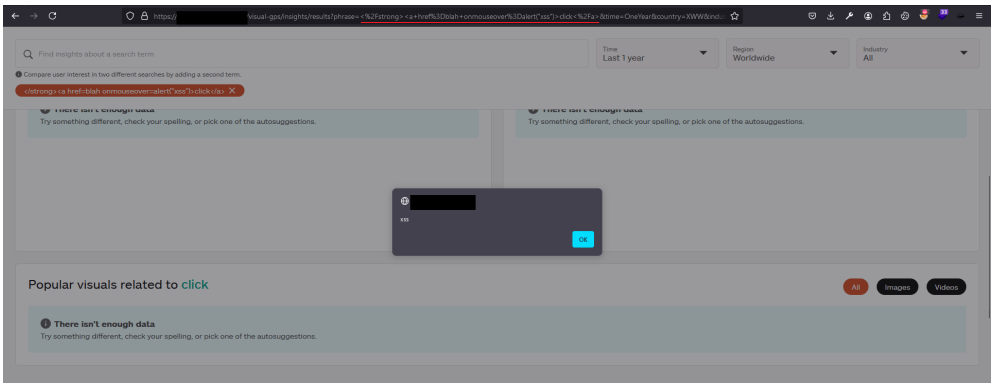

Figure 5.1: Alert Box triggered by XSS on an image hosting website

Above figure 5.1 shows the alert popup box triggered due to our malicious script.

We can also see the script reflected on some parts of the page as the sanitization [19] is being performed in some areas of the website but not all.

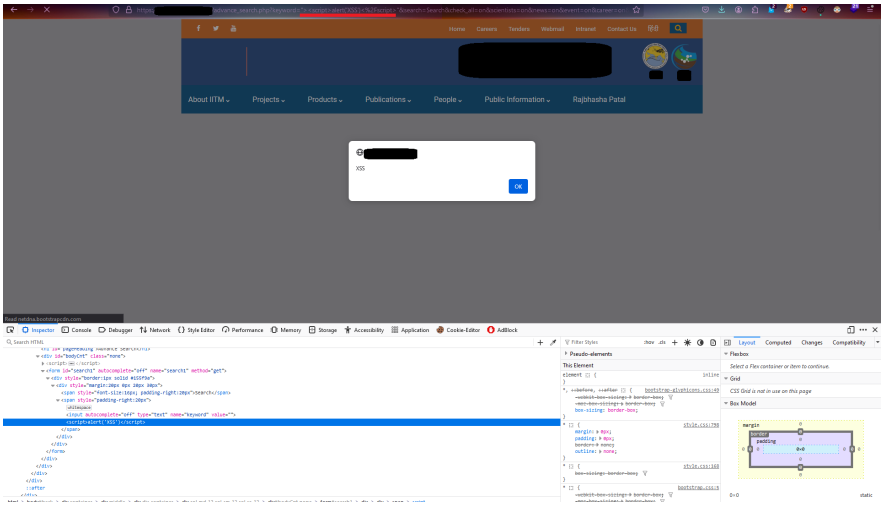

Figure 5.2: Alert Box triggered by XSS on a government website

Outcome:

We reported this bug to the openbugbounty [57] website, and got a reply as "Not Applicable" as the bug couldn't be reproduced by the website owners, we checked on Google for any other solutions to this and found out that many BBH had the same issue with the website where their bug was being disclosed as Not Applicable or Informative because the website owners didn't want to disclose that their website has any vulnerabilities.

5.2 File upload to XSS

Description:

This bug was found on a university website, where an attacker can change the filename of any PDF, JPG, or PNG files to include the JavaScript code in the filename which gets executed when the file is uploaded. A type of XSS called a Self-XSS attack [41] where an attacker makes the victim execute commands or perform actions on behalf of the attacker through social engineering.

Severity:

Our bug of XSS through file upload is very similar to CVE-2023-28158 [14]. We found that changing filenames to include JavaScript code could lead to XSS execution upon upload, similar to the privilege escalation vulnerability described in the CVE, demonstrating how simple features like file uploads can become attack vectors when input validation is inadequate, highlighting the importance of thorough security checks at all potential entry points, even in educational institution environments. XSS bugs like these have a severity rating of around 4.0 to 6.0 in the CVSS 3.x metric system [52] which has a severity of low or even medium in some cases. These bugs can pose harm to the website as attackers can execute SQL commands in the backend servers, due to no sanitization [19]. These SQL commands provided by user input are one of the common ways attackers are using to compromise the systems [28].

Steps to Reproduce:

This bug was found on the Contact-Us page of this website where people send university admission queries, this form had many text fields that were sanitized except the filename of the uploaded file, that was getting reflected on the website, so on Kali OS, we changed the filename of a pdf file to `<script>alert(document.cookie)</script>.pdf` to display the session cookie in the alert box. We used Linux OS for this as Windows doesn't support special characters in the filename of any files. To exploit further, we are trying to execute SQL commands through this way.

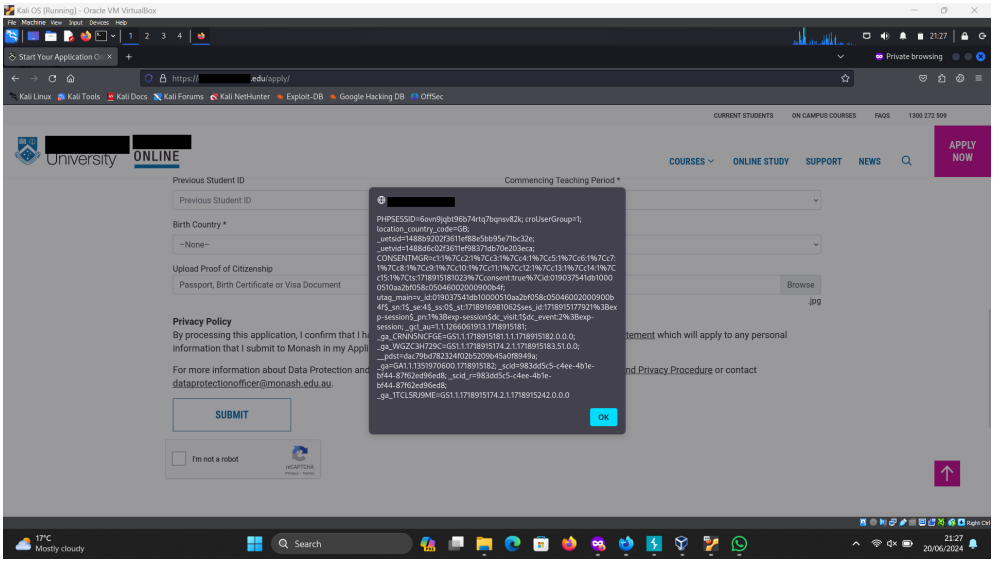

Figure 5.3: Alert Box triggered by XSS [41]

Above figure 5.3 shows the session cookies of the browser displayed in an alert box.

Outcome:

We reported this vulnerability to Bugcrowd [4] and got a reply as "Duplicate" as another researcher submitted the same vulnerability, and our submission was closed. If this happens, bounty hunters don't receive any rewards for it, we get points for the submission if the original reports progresses. Points determine the ranking of a hunter on the platform which may or may not be useful.

5.3 Cross-Site Tracing (XST)

Description:

This bug was discovered on a top university's website, where the developers had left the TRACE method enabled which led to Cross-Site Tracing(XST) [48]. This method can execute large lines of javascript code on the victim's browser. This attack was prevalent in the 2000s but is not used nowadays as most browsers block the use of the TRACE method [23]. We can increase severity by finding another XSS bug in the website and inject our script into it to send the data of the TRACE method over to our computers to hijack the session. There are methods to take advantage of this vulnerability such as using xhttp [87] request or using flash plugins [22], but most programs don't accept them.

Severity:

Our discovery of the Cross-Site Tracing (XST) vulnerability here mirrors CVE-2018-11039 [11], where the TRACE method was enabled, potentially allowing attackers to escalate existing XSS vulnerabilities to more severe XST attacks. XST [48] bugs like these have a severity rating of around 0.1 to 4.0 in the CVSS 3.x [52] metric system depending on the different attacks being chained with it. It has a severity of low. While looking for another attack to chain with the TRACE method to exploit the XST bug for higher severity, we stumbled upon another method named HTTP Smuggling which was used in conjunction with the TRACE method for performing Client-Desync attack as mentioned in this report [69], which we will talk in further sections.

Steps to Reproduce:

We ran Nmap [54] and Nikto [78] tools on this website and we found that some endpoints had TRACE method enabled. We tried sent a TRACE request and found that our entire request was being reflected by the server including internal IPs and other server information which are useful info. We sent a TRACE request via both the BurpSuite Repeater and curl command on Windows with the XSS script [41] inserted in an HTTP header, in both methods we can confirm our script being reflected back.

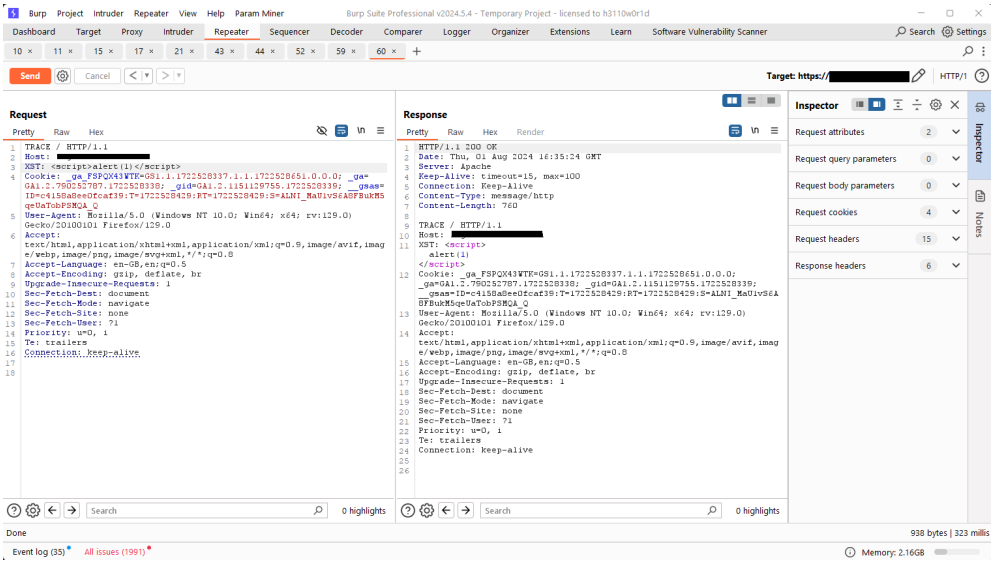

Figure 5.4: XSS script being reflected via TRACE method

Below figure 5.4 shows how the servers reflect back our XSS script as it is along with some server information.

Outcome:

We reported this vulnerability to HackerOne [26] and got a reply as “Informative” from the HackerOne team, as they wanted a working Proof of Concept that involved no victim interaction, this can be done using HTTP Smuggling [64], but we found that the website was not susceptible to HTTP Smuggling. We found some interesting information like internal IP address and server configuration information.

5.4 WordPress

We found three WordPress vulnerabilities on websites using this, one of them was a Denial of Service (DoS) [8] attack on a European website for Human resources, and another was an information disclosure vulnerability on a university website.

5.4.1 DoS Attack

The vulnerability stemmed from the misuse or misconfiguration of wp-cron.php, which could be triggered externally by unauthorized users due to its accessibility via a simple URL request. This exposure allows attackers to manipulate the scheduler to execute tasks at unscheduled times, potentially causing denial of service (DoS) [8] attacks by overloading the system with frequent, unnecessary jobs. Additionally, if combined with other vulnerabilities, this could lead to more severe exploits such as unauthorized data access or code execution. This discovery emphasizes the importance of securing scheduled task configurations and restricting access to critical system files to prevent potential abuse.

Description:

We found a DoS [8] vulnerability on a German software company website where any attacker can execute this DoS attack by sending multiple requests to this `wp-cron.php` WordPress file which saturates the server, ultimately slowing it down or completely shutting it down. Wp-cron is used by WordPress to handle and schedule tasks such as core updates, running backups, and other scheduled tasks [9, 77]. This is all that its supposed to do, the problem comes when a visitor requests this file, WordPress will create an extra request from its server to the `wp-cron.php` file which is fine for a small server with few visitors, for larger servers or with multiple visits, the server gets overwhelmed with scheduling each request which leads to DoS [8]. This particular feature of WordPress is what we tried to exploit.

Severity:

Our discovery of the wp-cron.php vulnerability aligns with CVE-2023-22622 [13], demonstrating a critical WordPress security issue. We found that default configuration could allow attackers to overwhelm servers and delay crucial updates, especially on less-visited sites. however, most of the bug bounty programs put the DoS vulnerability

in “Out of Scope”, the reasoning behind this is that some researchers while trying to prove a DoS vulnerability may cause a disruption in the live server [73]. But in reports where it is considered, its rating can range from 6.0 to 7.0 with medium severity or in some cases even have a rating of 9.0 to 10 with critical severity such as in this report [25].

Steps to Reproduce:

We used the Wappalyzer [83] extension to find out that the target website was using WordPress, so by scanning it with WPScan [86] we found some interesting endpoints and files such as `xmlrpc.php` which is also another attack vector, but we couldn’t access the file so we started looking for something else, we didn’t find the DoS causing file `wp-cron.php` appear on our WPScan tool, so we decided to use a wordlist [47] which had all the WordPress file endpoints which where critical, and we loaded it into our FFuf fuzzing tool and we found out that the `wp-cron.php` was enabled and we also confirmed that we could send both GET and POST requests from BurpSuite Repeater. Now the bug was found, but we wanted to verify if it is possible to perform a DoS [8] attack by running a script, this is not advisable as this is a legitimate attempt to disrupt services, so instead we did multiple page reloads accessing the file, and to our surprise, we found the request getting delayed and ultimately we received a 502 server error, and further requests weren’t getting through, confirming the bug.
If a hunter wants to confirm this bug, they can download this tool [67] from GitHub and run this command

```
./doser -t 999 -g 'https://target-website.com/wp-cron.php'
```

to send multiple requests to the server in a short time, after they will find a 502 server error on the website confirming the bug, this is extremely severe and shouldn’t be tested in a live environment.

Above figure 5.5 shows how the server has enabled `wp-cron.php` file and can be accessed by the POST request method.

Outcome:

We reported the first vulnerability to Intigriti [31] and got a reply as “Out-of-Scope” from the Intigriti team, and another to HackerOne [26] and got a reply as “Not-Applicable”, even if it was a valid vulnerability if the team doesn’t acknowledge it, we won’t get a reward for it. Thus if it is mentioned in the scope that a DoS bug won’t be considered,

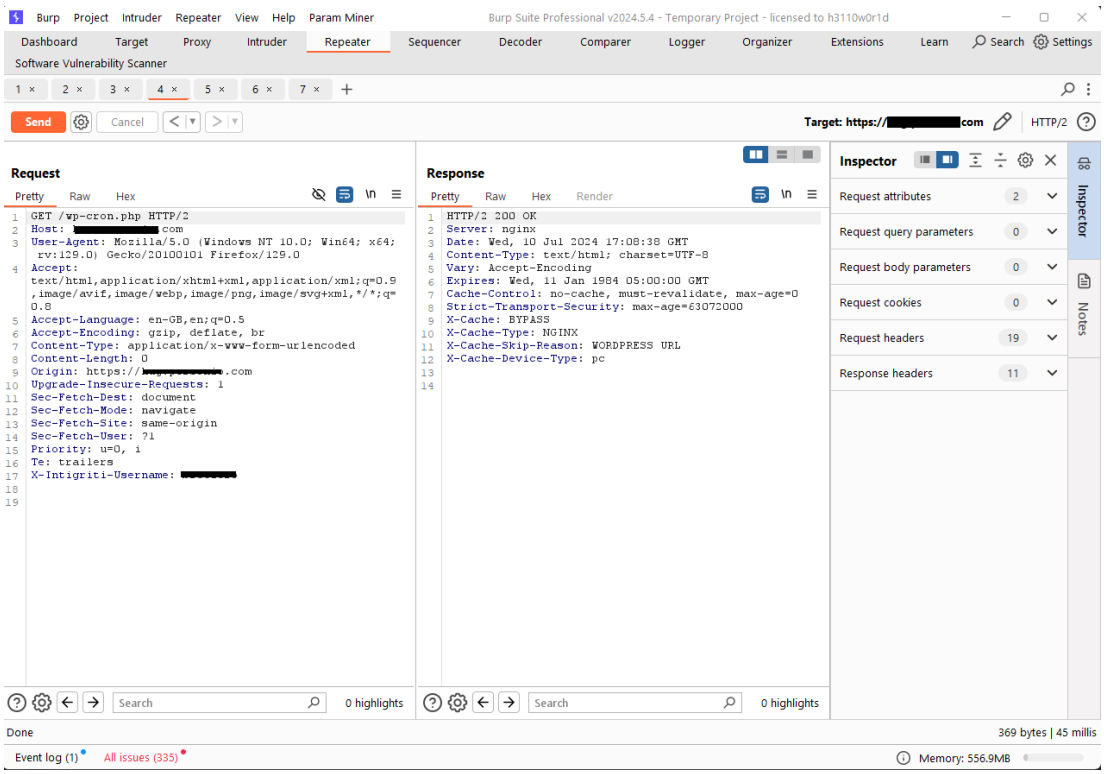

Figure 5.5: wp-cron.php file being enabled

then it will be wise to stay clear of it, some programs may recognize it and reward it, but it's better to look for something else.

5.4.2 Information Disclosure

Description:

We found two WordPress-enabled domains of a university, where a user can visit the path: `"/wp-json/wp/v2/users"` which is a JSON file containing information on all the users and authors of this website. Information Disclosure can range from personal data to private, financial, or intellectual property, in our case we were able to view the user's email ID, username, and profile picture.

Severity:

Our discovery of information disclosure in WordPress is similar to CVE-2020-26876 [53], where, the wp-courses plugin unintentionally exposed course content via the REST API, similar to how user data was exposed through the /wp-json/wp/v2/users endpoint in our case. Our finding demonstrates how default WordPress configurations

can inadvertently reveal sensitive information. The severity of Information Disclosure depends on the type of data being disclosed and can range from 4.0-6.0 low severity to 6.0-7.0 with medium severity. It can even be critical if information like `.htaccess` or `.htpasswd` files are being exposed which contain server configuration and username & passwords respectively.

Steps to Reproduce:

To be able to look for bugs similar to this, we need to know that the `/wp-json` endpoint is used by WordPress REST API to allow developers access to WordPress functionality like editing posts, comments, etc. We could see the REST API when we visit the endpoint `https://yourwebsite.com/wp-json` on this website, and the user's endpoint too. Additionally, we used this WPScan [86] command: `wpscan --url http://website.com/ --enumerate u,p,t --random-user-agent --api-token yourtoken` which gave us an in-depth enumeration of users, plugins, and themes, and also gave us some author names and email-ids which were not visible in the `"/wp-json/wp/v2/users"` endpoint. We used the random-user-agent argument to make the tool use random user agents for each request to bypass some firewalls, and we used the API-token which is needed to scan for vulnerabilities.

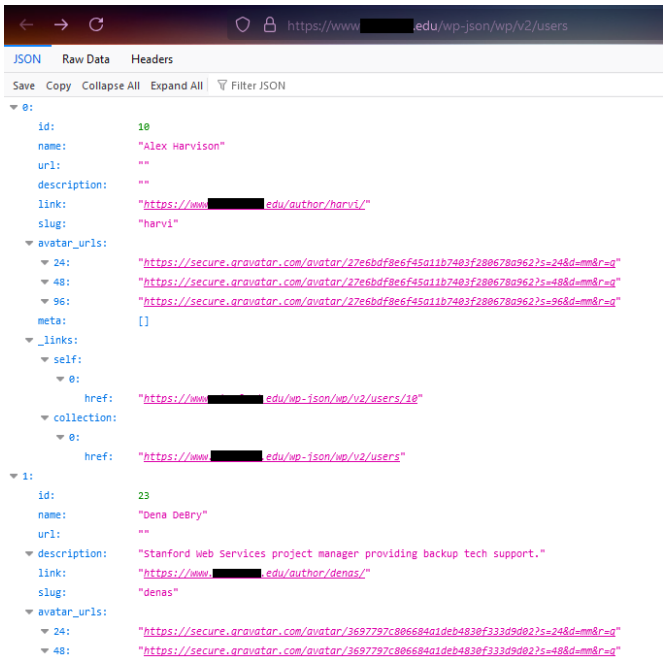

Figure 5.6: usernames visible in wp-json endpoint

Above figure 5.6 shows how the usernames and profile pictures can be viewed

through the `/wp-json` endpoint.

Outcome:

We reported both of these bugs to HackerOne [26] and got a reply as "Informative" from the team, as they feel this was not sensitive information. This depends a lot on the program too, some programs reward generously for this type of vulnerability, as the usernames can be used to bruteforce passwords for account takeovers, and there is a lot of trial and error involved to get the reward from the program.

5.5 Security Misconfiguration

Description:

We found a Security Misconfiguration [10] error on a shareholding and investing website. This bug was found in the password reset section of this website. The email id that we entered in the password reset dialog box was being sent to the server as a JSON body request, this meant that the website was making an API call to the backend to fetch the account from the email ID. We tried SQL injection and other attacks that could be performed, but we couldn't get anything due to the firewall and sanitization. So we referred to this report [10] and found this site to have the same behavior.

Severity:

Severity is subjective here, in our case the website was treating this overflow as a status 500 internal server error, this can mean a lot of things running from buffer overflow to improper sanitization [19]. We reported this bug with a 0.1 to 4.0 rating and low severity, if we had more proof that this bug was interfering with other resources, our severity would also be higher, for example, this bounty report [10] earned $40,000 for the same vulnerability. Improper null-byte sanitization causes this vulnerability.

Steps to Reproduce:

While using BurpSuite to intercept the password reset request, we saw that when the email ID passed through the JSON body is in the format `abc%00@gmail.com` or `abc@%00%00%00@gmail.com` the email id appearing in the response is `abc@gmail.com`.

This behavior can be explained with an example, let's assume `abc%00` is passed, the front end treats `%00` as a null byte and assigns a storage byte for it, 4 bytes in this case, the backend, however, removes the null byte `%00` due to sanitization being implemented, the storage is still 4bytes, but the data being filled i.e abc is 3 bytes, the extra 1-byte storage space is empty when sending multiple null-bytes in a series of request, this creates large undefined storage spaces in the server which causes a buffer overflow in some cases. In our, when sending a few null byte characters we found the endpoint was still working, but when we sent 100s of null bytes, the server gave us a 500 error, this was not the same case when we sent a similar amount of alphabets, this proves that the server is behaving differently towards null byte. We have enough proof to say buffer overflow has happened in the server, but as the response being sent is sent as a JSON body inside a pre-defined message, we can't confirm the vulnerability as the server treats this as a 500 status error, which should not occur in the first place.

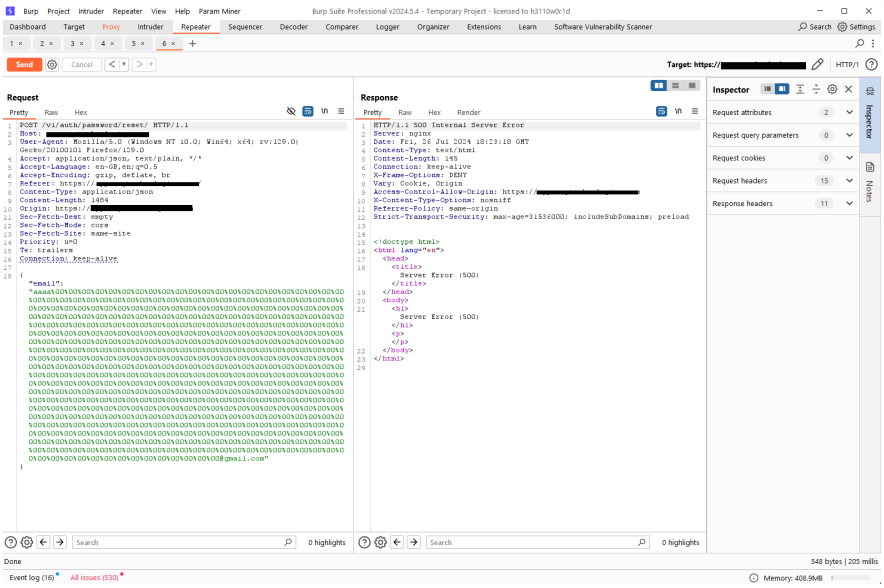

Figure 5.7: server error due to excessive null bytes

Above figure 5.7 shows how the server throws 500 internal server errors when an excessive amount of null bytes are sent.

Outcome:

We reported this vulnerability to Intigriti [31] and got a reply as "Not-Applicable" from the Intigriti team as they needed more concrete evidence that a system resource was being modified or any sensitivity was gained. If we could have shown that buffer overflow was visible to use in the response, then they would have accepted the bug.

5.6 HTTP Smuggling

Description:

We found an HTTP Request Smuggling vulnerability, specifically CL.0 (Content-Length: 0), where we trick the server into misinterpreting our single request as 2 different requests. This is slightly different from HTTP pipelining [21] where by using HTTP/1.1 protocol we send multiple HTTP requests through the same connection, and the server responds to these requests through the same connection. This is the functionality of HTTP/1.1 protocol which was designed to work in this manner and has no inherent vulnerabilities, we tried different attack strategies like CL.TE, TE.TE, TE.CL, and found that the server was vulnerable to CL.0 attack. We referred to this publication [38] by a James Kettle, which helped us come up with this approach for finding HTTP smuggling vulnerabilities.

Severity:

Our bug specifically the CL.0 variant, is similar to the CVE-2022-26377 [12]. We found that we could trick the server into misinterpreting a single request as two separate ones, similar to the vulnerability in Apache HTTP Server's mod_proxy_ajp [12]. Its rating can range from 7.0 to 9.0 with High severity and even critical in some cases. This bug has severe impacts, potentially letting attackers to circumvent security controls, gain unauthorized access to sensitive data, compromise the integrity of the web application, poison other users' requests, and even credential stealing and session hijacking.

Steps to Reproduce:

There are some conditions we researched that must be met for this bug, firstly, "find an endpoint that accepts POST requests", secondly, "the endpoint must be able to accept HTTP/1.1 request", and lastly, "the server ignores the content-length(CL) header (For CL.0 attack)" or "the server accepts both CL and Transfer-Encoding header (For CL.TE and TE.CL attack)"

Next, when we have the conditions met, we send a POST request with another HTTP request as our POST body, this should normally not work on HTTP/2.0 requests, so we perform a downgrade attack by editing the protocol manually from 2.0 to 1.1, this gives a 400 error if a server is properly configured, for us this worked so we moved

on to the next step. In BurpSuite Repeater we sent two requests as a group, one the poisoned request(main request + payload request as body) and another a normal request to demonstrate smuggling. In the poisoned request, we kept the Content-Length of the first request to end before the payload request in the body, as shown in the figures 5.8 & 5.9, then when the request is sent as a group, we could see that we can poison our other normal request on the same connection, this confirms our Client-Desync attack. Now, to poison other users, for confirming request smuggling, we had to somehow keep the response of the payload request alive, and let the server think of it as a second request with no body, this will make the server wait for another request(victim's request) and attach that request as a body for our poisoned request. To do this, we made the second request be POST request and made the Content-Length header for the poisoned request have a value of around 200-500 (this will vary on the request length), so that we can accommodate the entire victim's request along with their session credentials. We tested this on our browsers and we could see that our other request received the response for our poisoned request.

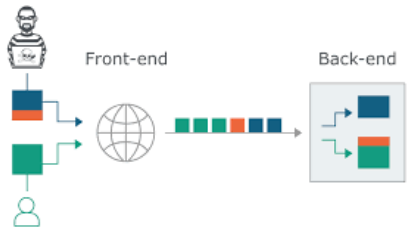

Figure 5.8: Typical CL.0 smuggling attack

We can see in the above figure 5.9 the CL.0 Client Desync attack.

Outcome:

We reported this vulnerability to Intigriti [31] and got a reply as "Duplicate" from the Intigriti team, as another researcher had already submitted this vulnerability, if we had submitted this earlier, maybe we could have gotten a monetary reward. These vulnerabilities normally pay quite a lot as it is on the easier side to execute and can cause severe damage to the company. We also used this automated tool [17] to search for these vulnerabilities on websites, although we recommend a manual way to check for them and also to cross-verify any false positives.

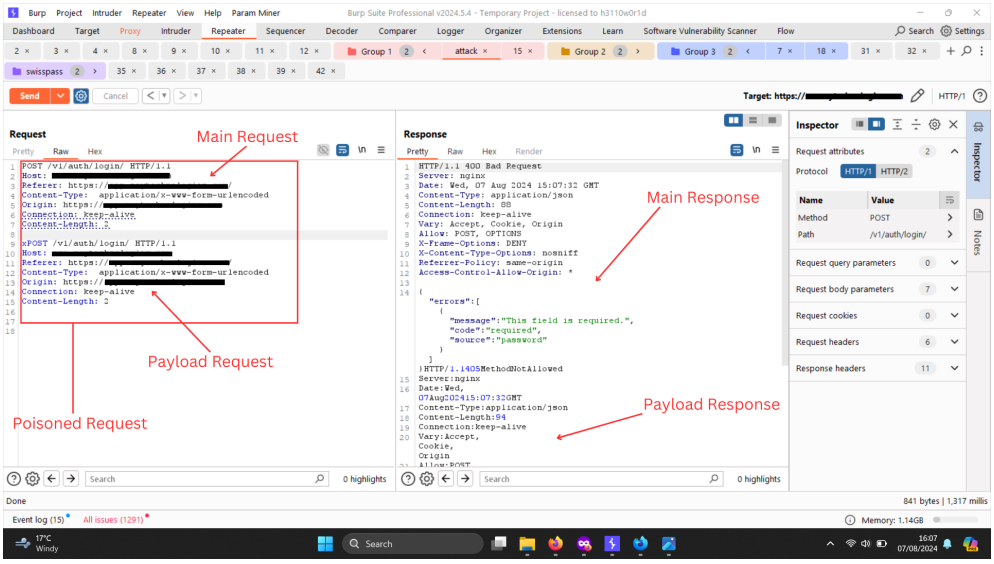

Figure 5.9: CL.0 attack

Summary:

Sr	Vulnerability	CVSS	Severity	Platform	Outcome
1	Cross-Site Scripting	6.0 to 7.0	medium	openbugbounty	Not Applicable
2	Cross-Site Scripting	6.0 to 7.0	medium	cyber helpline	Informative
3	File Upload	4.0 to 6.0	low	BugCrowd	Duplicate
4	Cross-Site Tracing	0.1 to 4.0	low	HackerOne	Informative
5	Denial of Service	6.0 to 7.0	medium	HackerOne	Not Applicable
6	Denial of Service	9.0 to 10.0	critical	Intigriti	Out-of-Scope
7	Information Disclosure	4.0 to 6.0	medium	HackerOne	Informative
8	Information Disclosure	4.0 to 6.0	medium	HackerOne	Informative
9	Security Misconfig.	0.1 to 4.0	low	Intigriti	Not Applicable
10	HTTP Smuggling	7.0 to 9.0	high	Intigriti	Duplicate

Table 5.1: Vulnerability Report Summary

Above table 5.1 summarizes all the vulnerabilities we encountered during our testing.

Chapter 6

DomainMiner

DomainMiner, a bash script tool we developed, which combines the output of 10 other tools to streamline the subdomain enumeration phase of bug bounty hunting, making the recon process of BBH more efficient and time-saving. It simplifies finding all subdomains of a target website, combining results with IP addresses and HTTP status codes, and filtering out non-functioning domains. This tool significantly reduces the workload for bug bounty hunters by automating much of the initial discovery process, providing outputs in text or JSON formats with just a domain name input.

Figure 6.1: DomainMiner Tool

This tool consists of 3 bash script files: **install.sh** which handles the installation of all necessary tools and their dependencies required for DomainMiner to function. It installs an additional 10 tools for subdomain enumeration. The main script **DomainMiner.sh** runs all the tools and combines their output to a text file `all_tools_output.txt`. **resolver.sh** is a helper script that handles the cleaning of the tool's output, removing duplicates, resolving domain names to IP addresses & its HTTP status codes. To use DomainMiner, users need to make these scripts executable using the command `chmod +X scriptname.sh` for all of the scripts, and then run the `./DomainMiner.sh` command

to execute the tool. The output is saved as `resolved_domains.txt`, users can use `-j` or `--json` for JSON output which may be useful as an input for other tools.

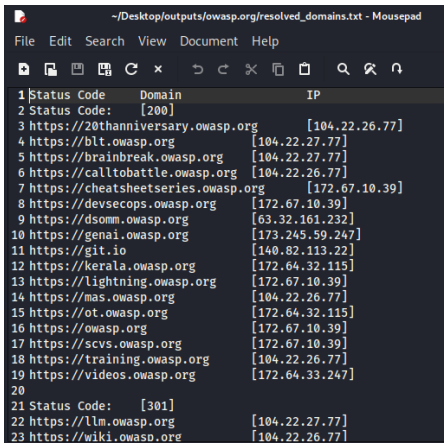

Figure 6.2: Tool output in text format.

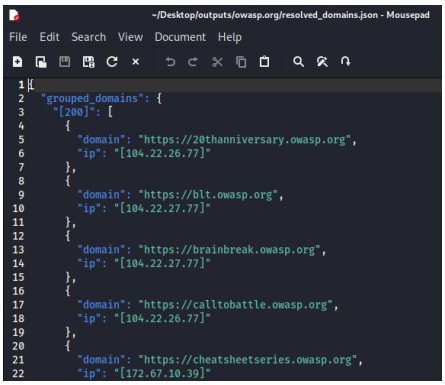

Figure 6.3: Tool output in JSON format

Advantages over other tools; It's **cost-effective**, being completely free to use without any restrictions or daily limits. The tool provides **detailed output**, including HTTP status codes and IP addresses, with customizable options for more aggressive or passive results. **Multiple tools** are combined within DomainMiner, effectively running and merging outputs from 10 different subdomain enumeration tools. Additionally, it features **bad domain filtration**, automatically filtering out non-functioning domains, saving users time on manual verification.

Analysis of Results; Based on the results presented in Table A.1, where we compare DomainMiner with other online tools like Hackertarget [27], Subdomain Finder [5], nmmapper [55], pentest-tools [61], and osint.sh [59], we can observe several interesting patterns in the performance of different subdomain enumeration tools across various domains. DomainMiner, consistently outperforms other tools in terms of the number of subdomains found for most domains. This is particularly evident for domains like "quora.com", where DomainMiner discovered 10,921 subdomains, significantly more than any other tool. For domains like "medium.com" and "github.com", DomainMiner also found the highest number of subdomains (944 and 830 respectively). However, it's worth noting that for some domains like "duolingo.com", other tools like "osint.sh" performed better, finding 421 subdomains compared to DomainMiner's 430. The table also shows that the effectiveness of these tools can vary depending on the target domain, suggesting that using a combination of tools, as DomainMiner does, can lead to more comprehensive results in subdomain enumeration efforts.

Chapter 7

Evaluation & Discussion

7.1 Evaluations in the Project

Throughout this dissertation, we evaluated our bug bounty hunting approach through various aspects, the **methodologies applied** includes our approach of using a combination of manual and automated testing proved to be beneficial in discovering various bugs as we encountered bugs by using both methods. By following our approach which combined the tools, structured hunting process, and our custom tool, we found that our efficiency increased as we refined our approach and learned more.

For **tool selection**, tools selected such as BurpSuite [43], FFUF [20] were instrumental in our hunting, each tool has served its purpose and complemented our manual testing. For **vulnerability discovery**, We successfully identified a range of vulnerabilities, ranging from less complex Cross-Site Scripting (XSS) [41], WordPress issues to more complex issues like Security Misconfigurations [10], and HTTP Smuggling [64]. This diverse set of findings demonstrates the breadth of our approach. As for **bug reporting**, our experience with reporting bugs to various platforms like Bugcrowd [4], Hackerone [26], Intigriti [31], and OpenBugBounty [57] provided insights into how different organizations handle vulnerability reports. While some reports were marked as duplicates or out of scope, this feedback helped us refine our bug hunting and reporting strategies. Lastly, **DomainMiner** was found reasonably good against the current offerings, performing very well. There are a few things we hope to address in future, which include its Linux-only nature due to being developed in a bash script, time-consuming due to running multiple tools, and having a noisy operation due to high network traffic which may trigger firewalls. Despite these, its capabilities and functionalities make it a successful project.

7.2 Evaluation of the overall Project

The overall project is evaluated on these key outcomes, such as the **learning curve** proved to be steep, in understanding various web vulnerabilities, exploit techniques, and the workings of bug bounty programs. We learned quite a lot during a short span of time, this knowledge acquisition is valuable for anyone pursuing a career in cybersecurity. We gained **hands-on experience** by actively participating in bug bounty programs, and testing websites for vulnerabilities. This practical exposure is invaluable and difficult to replicate in controlled environments. The creation of DomainMiner as a **tool development**, demonstrated our ability to identify gaps in existing tools and develop custom solutions to address specific needs in the bug-hunting process. For our **contribution to security**, although not all reported vulnerabilities were accepted or rewarded, our efforts contributed to improving the security posture of various websites by bringing potential issues to their attention and also provided the readers with a realistic view of the challenges we faced as a bounty hunter.

Based on these factors, we would say that our project was successful, this experience was very fruitful and made us refine our approach. In retrospect, if we started from scratch there would be a few things that we would have done differently, like adding more tools to our toolset, increasing our knowledge base on vulnerability types, and many more, which we would hope the readers research on.

Chapter 8

Conclusions

This project has provided valuable insights into the world of bug bounty hunting and web application security. Through our approach and hands-on experience, we've gained a deeper understanding of various vulnerability types, their potential impact, and the processes involved in identifying and reporting them.
Our attempt to bug bounty hunting, combining manual testing with automated tools, proved successful in uncovering a diverse range of vulnerabilities, underscoring the critical value of a good approach. Mastery of essential tools was crucial, and the ability to develop custom tools like DomainMiner significantly enhanced our productivity. Given the constantly evolving landscape, continuous learning is essential to stay updated with the latest vulnerability types, attack vectors, and defensive measures. Persistence and patience were also vital, as not all discovered vulnerabilities were accepted or rewarded; refining techniques and coping with duplicate or out-of-scope submissions required perseverance. Despite these challenges, our efforts made a tangible impact on the security of various web applications, highlighting the real-world benefits of bug bounty programs in enhancing overall internet security.

8.1 Future Directions:

Future work could focus on specializing in specific types of vulnerabilities or technologies to develop deeper expertise. We are also keen on exploring more complex vulnerability chains and their potential impacts that could lead to higher-value discoveries. Automated testing is what we to explore further, in combination with this, the development of custom tools for automated vulnerability analysis could enhance the efficiency of the bug-hunting process.

Bibliography

[1] 0xold. Null byte on steroids. https://medium.com/@0xold/null-byte-on-steroids-23f8104a25ec, 2024. Accessed: 2024-08-20.

[2] Akpar Asadov. Directory traversal attack. In *1st INTERNATIONAL CONFERENCE ON THE 4th INDUSTRIAL REVOLUTION AND INFORMATION TECHNOLOGY*, volume 1, pages 174–177. Azrbaycan Dövlt Neft v Snaye Universiteti, 2023.

[3] Andrew Austin and Laurie Williams. One technique is not enough: A comparison of vulnerability discovery techniques. In *2011 International Symposium on Empirical Software Engineering and Measurement*, pages 97–106. IEEE, 2011.

[4] Bugcrowd. Crowdsourced cybersecurity platform. https://www.bugcrowd.com, 2024.

[5] C99. Subdomain finder. Available at: https://subdomainfinder.c99.nl, 2024. Accessed: 2024-08-24.

[6] Paulino Calderon. *Nmap Network Exploration and Security Auditing Cookbook: Network discovery and security scanning at your fingertips*. Packt Publishing Ltd, 2021.

[7] Check Point Software Technologies. What is sandboxing? Available at: https://www.checkpoint.com/cyber-hub/threat-prevention/what-is-sandboxing/, 2024. Accessed: 2024-08-24.

[8] Cloudflare. What is a denial-of-service (dos) attack?, 2024. Accessed 22 August 2024.

[9] WordPress Contributors. Cron plugin handbook. https://developer.wordpress.org/plugins/cron/, 2024.

[10] Sam Curry. Filling in the blanks: Exploiting null byte buffer overflow for a $40,000 bounty. https://samcurry.net/filling-in-the-blanks-exploiting-null-byte-buffer-overflow-for-a-40000-bounty. Accessed: 2024-08-15.

[11] CVE-2018-11039. https://nvd.nist.gov/vuln/detail/CVE-2018-11039.

[12] CVE-2022-26377. https://nvd.nist.gov/vuln/detail/CVE-2022-26377.

[13] CVE-2023-22622. https://nvd.nist.gov/vuln/detail/CVE-2023-22622.

[14] CVE-2023-28158. https://nvd.nist.gov/vuln/detail/CVE-2023-28158.

[15] CVE-2023-39575. https://nvd.nist.gov/vuln/detail/CVE-2023-39575.

[16] Pedro Daniel Carvalho de Sousa Rodrigues. An osint approach to automated asset discovery and monitoring. 2019.

[17] defparam. smuggler. https://github.com/defparam/smuggler, n.d. Automated Tool for checking HTTP request smuggling.

[18] F5 Networks. What is a web application firewall (waf)? Available at: https://www.f5.com/glossary/web-application-firewall-waf, 2024. Accessed: 2024-08-24.

[19] Faun. Url sanitization: The why and how. Available at: https://faun.pub/url-sanitization-the-why-and-how-9f14e1547151, 2024. Accessed: 2024-08-24.

[20] ffuf. ffuf - fuzz faster u fool. https://github.com/ffuf/ffuf, 2024. Accessed 22 Aug 2024.

[21] Roy Fielding, Jim Gettys, Jeffrey Mogul, Henrik Frystyk, Larry Masinter, Paul Leach, and Tim Berners-Lee. Hypertext transfer protocol–http/1.1. Technical report, 1999.

[22] GNOME Project. Install adobe flash. Available at: https://help.gnome.org/users/gnome-help/stable/net-install-flash, 2024. Accessed: 2024-08-24.

[23] Jeremiah Grossman. Cross site tracing (xst). *WhiteHat Security White Paper*, 2003.

[24] Shashank Gupta and Brij Bhooshan Gupta. Cross-site scripting (xss) attacks and defense mechanisms: classification and state-of-the-art. *International Journal of System Assurance Engineering and Management*, 8:512–530, 2017.

[25] HackerOne. Hackerone report #1888723. https://hackerone.com/reports/1888723, 2024. Wordpress DoS bug reported.

[26] HackerOne. No 1 trusted security platform and hacker program. https://hackerone.com, 2024.

[27] HackerTarget. Find dns host records. Available at: https://hackertarget.com/find-dns-host-records/, 2024. Accessed: 2024-08-24.

[28] William GJ Halfond, Jeremy Viegas, Alessandro Orso, et al. A classification of sql injection attacks and countermeasures. In *ISSSE*, 2006.

[29] Qi-Xian Huang, Min-Yi Chiu, Ying-Feng Chen, and Hung-Min Sun. Attacking websites: Detecting and preventing http request smuggling attacks. *Security and Communication Networks*, 2022(1):3121177, 2022.

[30] Tommaso Innocenti, Seyed Ali Mirheidari, Amin Kharraz, Bruno Crispo, and Engin Kirda. You've got (a reset) mail: A security analysis of email-based password reset procedures. In *Detection of Intrusions and Malware, and Vulnerability Assessment: 18th International Conference, DIMVA 2021, Virtual Event, July 14–16, 2021, Proceedings 18*, pages 1–20. Springer, 2021.

[31] Intigriti. Bug bounty agile pentesting platform. https://www.intigriti.com, 2024.

[32] Bahruz Jabiyev, Steven Sprecher, Kaan Onarlioglu, and Engin Kirda. T-reqs: Http request smuggling with differential fuzzing. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 1805–1820, 2021.

[33] Ashar Javed and Jörg Schwenk. Systematically breaking online wysiwyg editors. In *Information Security Applications: 15th International Workshop, WISA 2014, Jeju Island, Korea, August 25-27, 2014. Revised Selected Papers 15*, pages 122–133. Springer, 2015.

[34] Kali Linux. Dirb package description. Available at: https://www.kali.org/tools/dirb/, 2024. Accessed: 2024-08-24.

[35] Kali Linux. Dirbuster package description. Available at: https://www.kali.org/tools/dirbuster/, 2024. Accessed: 2024-08-24.

[36] Kali Linux. Gobuster package description. Available at: https://www.kali.org/tools/gobuster/, 2024. Accessed: 2024-08-24.

[37] Kali Linux. Wfuzz package description. Available at: https://www.kali.org/tools/wfuzz/, 2024. Accessed: 2024-08-24.

[38] James Kettle. Browser-powered desync attacks: A new frontier in http request smuggling. https://portswigger.net/research/browser-powered-desync-attacks, 2022.

[39] Kinsta. What is a content management system (cms)?, 2024. Accessed: 2024-08-23.

[40] Vickie Li. *Bug Bounty Bootcamp: The Guide to Finding and Reporting Web Vulnerabilities*. No Starch Press, 2021.

[41] Miao Liu, Boyu Zhang, Wenbin Chen, and Xunlai Zhang. A survey of exploitation and detection methods of xss vulnerabilities. *IEEE Access*, 7:182004–182016, 2019.

[42] Carlos A Lozano and Shahmeer Amir. *Bug Bounty Hunting Essentials: Quick-paced guide to help white-hat hackers get through bug bounty programs*. Packt Publishing Ltd, 2018.

[43] PortSwigger Ltd. Burp suite. https://portswigger.net/burp, 2024. Accessed: 2024-08-17.

[44] Sefat Mahjabin. Implementation of dos and ddos attacks on cloud servers. *Periodicals of Engineering and Natural Sciences*, 6(2):148–158, 2018.

[45] Suresh S. Malladi and Hemang C. Subramanian. Bug bounty programs for cybersecurity: Practices, issues, and recommendations. *IEEE Software*, 37(1):31–39, 2020.

[46] Tom Marrs. *JSON at work: practical data integration for the web*. " O'Reilly Media, Inc.", 2017.

[47] Daniel Miessler. Seclists. Available at: https://github.com/danielmiessler/SecLists, 2024. GitHub repository.

[48] MITRE Corporation. Cross site tracing (xst). https://capec.mitre.org/data/definitions/107.html, 2024.

[49] ModHeader. Modheader: Modify http headers. https://modheader.com, 2024. Accessed: 2024-08-18.

[50] Moustachauve. Cookie-editor. https://cookie-editor.com, 2024. Accessed: 2024-08-18.

[51] Mozilla Developer Network. Http response status codes, 2024. Accessed: 2024-08-23.

[52] National Institute of Standards and Technology (NIST). Vulnerability metrics. https://nvd.nist.gov/vuln-metrics/cvss, 2024.

[53] National Vulnerability Database. Cve-2020-26876, 2024.

[54] Nmap Project. Nmap: Discover your network. https://nmap.org/, 2024. Accessed 22 August 2024.

[55] NMMapper. Subdomain finder. Available at: https://www.nmmapper.com/sys/tools/subdomainfinder/, 2024. Accessed: 2024-08-24.

[56] NordVPN. Firefox vs. chrome: Which browser is better? Available at: https://nordvpn.com/blog/firefox-vs-chrome/, 2024. Accessed: 2024-08-24.

[57] Open Bug Bounty. Free bug bounty program and coordinated vulnerability disclosure. https://www.openbugbounty.org, 2024.

[58] Angela Orebaugh and Becky Pinkard. *Nmap in the enterprise: your guide to network scanning*. Elsevier, 2011.

[59] OSINT.SH. Subdomain finder. Available at: https://osint.sh/subdomain/, 2024. Accessed: 2024-08-24.

[60] OWASP Foundation. About owasp. https://www.owasp.org, 2013.

[61] Pentest-Tools.com. Find subdomains of domain. Available at: https://pentest-tools.com/information-gathering/find-subdomains-of-domain, 2024. Accessed: 2024-08-24.

[62] Carlos Polop. Reset/forgotten password bypass. https://book.hacktricks.xyz/pentesting-web/reset-password, 2024. Accessed: 2024-08-17.

[63] Karishma Pooj and Sonali Patil. Understanding file upload security for web applications. *International Journal of Engineering Trends and Technology*, 42(7):342–347, 2016.

[64] PortSwigger. Http request smuggler. https://github.com/PortSwigger/http-request-smuggler, 2024. Accessed: 2024-08-18.

[65] PortSwigger. Http request smuggling. https://portswigger.net/web-security/request-smuggling, 2024. Accessed: 2024-08-20.

[66] PortSwigger. Software vulnerability scanner. https://github.com/portswigger/software-vulnerability-scanner, 2024. Accessed: 2024-08-18.

[67] Quitten. doser.go: Dos tool for http requests. https://github.com/Quitten/doser.go, 2024.

[68] Sagar Rahalkar. *A Complete Guide to Burp Suite: Learn to Detect Application Vulnerabilities*. Apress, Berkeley, CA, 1 edition, November 2020. Copyright Sagar Rahalkar 2021.

[69] PortSwigger Research. Making desync attacks easy with trace. https://portswigger.net/research/trace-desync-attack, March 2024.

[70] ResearchGate. An overview of bug bounty hunting life cycle. https://www.researchgate.net/figure/An-overview-of-bug-bounty-hunting-life-cycle$_{f}ig2_{3}82126794, 2024. Accessed 22 Aug 2024$.

[71] Germán E. Rodríguez, Jenny G. Torres, Pamela Flores, and Diego E. Benavides. Cross-site scripting (xss) attacks and mitigation: A survey. *Computer Networks*, 166:106960, 2020.

[72] ShadowGirlInCyberLand. Tryhackme — burp suite: Intruder. https://medium.com/@shadowgirlincyberland/tryhackme-burp-suite-intruder-e73f000eb2d8, 2024. Accessed: 2024-08-18.

[73] Saman Shafigh, Boualem Benatallah, Carlos Rodríguez, and Mortada Al-Banna. Why some bug-bounty vulnerability reports are invalid? study of bug-bounty reports and developing an out-of-scope taxonomy model. In *Proceedings of the 15th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, ESEM '21, New York, NY, USA, 2021. Association for Computing Machinery.

[74] Pritam Gajkumar Shah and John Ayoade. An empricial study of brute force attack on wordpress website. In *2023 5th International Conference on Smart Systems and Inventive Technology (ICSSIT)*, pages 659–662. IEEE, 2023.

[75] Ashish Sharma, Aniket Tyagi, Prakrati Khatri, and Rinku Garg. Enhanced 403 bypass mechanism for web security. In *2024 4th International Conference on Advance Computing and Innovative Technologies in Engineering (ICACITE)*, pages 1858–1862, 2024.

[76] Carlos Silva. Http status codes: What they are and how to fix errors. https://www.semrush.com/blog/http-status-codes/, 2024. Accessed: 2024-08-20.

[77] Georgi Stoyanov, Adelina Aleksieva-Petrova, and Milen Petrov. Analysis of modern security plugins for wordpress. In *AIP Conference Proceedings*, volume 3084. AIP Publishing, 2024.

[78] Chris Sullo. Nikto. Available at: https://github.com/sullo/nikto, 2024. GitHub repository.

[79] TomNomNom. assetfinder, 2024. Accessed: 2024-08-24.

[80] Vercel. Next.js by vercel - the react framework. Available at: https://nextjs.org/, 2024. Accessed: 2024-08-24.

[81] W3Schools. Html event attributes. https://www.w3schools.com/tags/ref_eventattributes.asp, 2024.

[82] Thomas Walshe and Andrew Simpson. An empirical study of bug bounty programs. In *2020 IEEE 2nd international workshop on intelligent bug fixing (IBF)*, pages 35–44. IEEE, 2020.

[83] Wappalyzer. Wappalyzer: Identify technologies on websites. https://www.wappalyzer.com, 2024. Accessed: 2024-08-18.

[84] Sunny Wear. *Burp Suite Cookbook: Practical recipes to help you master web penetration testing with Burp Suite*. Packt Publishing Ltd, 2018.

[85] Peter De Witte. How to use the http request smuggler extension to perform an attack. https://peter-de-witte.medium.com/how-to-use-the-http-request-smuggler-extension-to-perform-an-attack-8a09c1a6801b, 2024. Accessed: 2024-08-18.

[86] WPScan. Wpscan: Wordpress security scanner. https://wpscan.com, 2024. Accessed: 2024-08-18.

[87] xhttp.org. xhttp - http/2 downgrade attack tool. Available at: https://xhttp.org/, 2024. Accessed: 2024-08-24.

[88] yunemse48. 403bypasser. https://github.com/yunemse48/403bypasser, 2024. Accessed: 2024-08-18.

[89] Xiaofeng Zheng, Jian Jiang, Jinjin Liang, Haixin Duan, Shuo Chen, Tao Wan, and Nicholas Weaver. Cookies lack integrity:{Real-World} implications. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 707–721, 2015.

Appendix A

DomainMiner Tool Results

A.1 Results Table

Here are the different website URLs for each tool:

1. https://hackertarget.com/find-dns-host-records/
2. https://subdomainfinder.c99.nl
3. https://www.nmmapper.com/sys/tools/subdomainfinder/
4. https://pentest-tools.com/information-gathering/find-subdomains-of-domain
5. https://osint.sh/subdomain/

Domain	Tool Name	Subdomains Found
duolingo.com	hackertarget.com	221
	Subdomain Finder	50
	nmmapper.com	15
	pentest-tools	152
	osint.sh	421
	DomainMiner	**430**
github.com	hackertarget.com	189
	Subdomain Finder	86
	nmmapper.com	15
	pentest-tools	288
	osint.sh	7
	DomainMiner	**830**
medium.com	hackertarget.com	500
	Subdomain Finder	13
	nmmapper.com	15
	pentest-tools	23
	osint.sh	529
	DomainMiner	**944**
mirror.co.uk	hackertarget.com	50
	Subdomain Finder	52
	nmmapper.com	15
	pentest-tools	89
	osint.sh	18
	DomainMiner	**365**
owasp.org	hackertarget.com	22
	Subdomain Finder	30
	nmmapper.com	15
	pentest-tools	27
	osint.sh	24
	DomainMiner	**38**
quora.com	hackertarget.com	67
	Subdomain Finder	1875
	nmmapper.com	15
	pentest-tools	5
	osint.sh	339
	DomainMiner	**10921**

Table A.1: Subdomains found by different tools for each domain

Appendix B

Tools and Summary

B.1 Tool Info

Below figure B.1 shows us the tool layout and how the data is presented, we can also identify the version number of some of the technologies used.

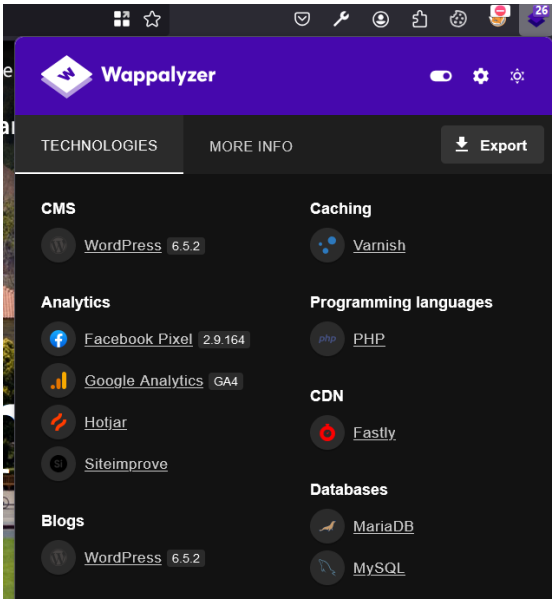

Figure B.1: wappalyzer tool output.

Below figure B.2 shows us the tool layout and how the data is presented, we can also edit some of the cookies.

In the below figure B.3 shown are the different 403 and 404 error pages we encountered in our tests which disclose the server type and version.

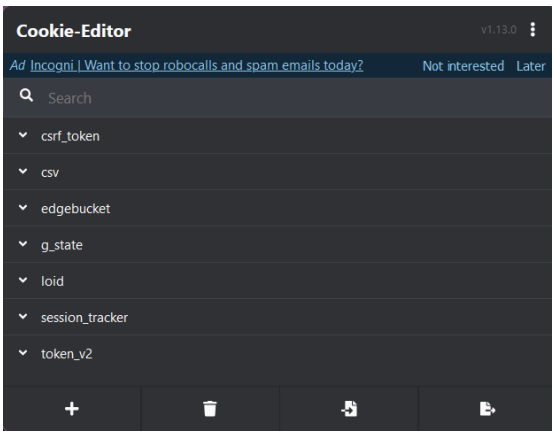

Figure B.2: cookie editor tool.

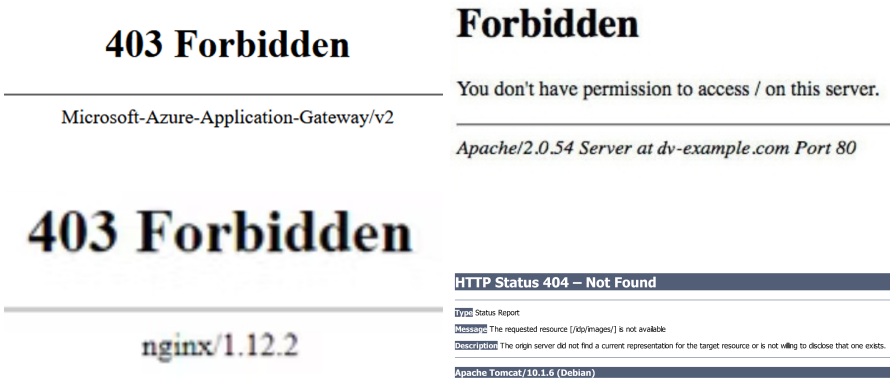

Figure B.3: 403 & 404 error pages

B.2 Summary of Bug Reports

Below figure B.4 shows the different outcomes of all the bug reports submitted over the period, and the figure B.5 shows the breakdown of different types of vulnerabilities found over our bug bounty hunting period.

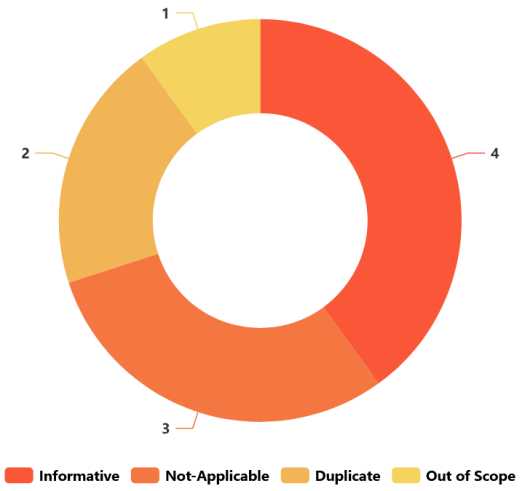

Figure B.4: Different outcomes of bug reports.

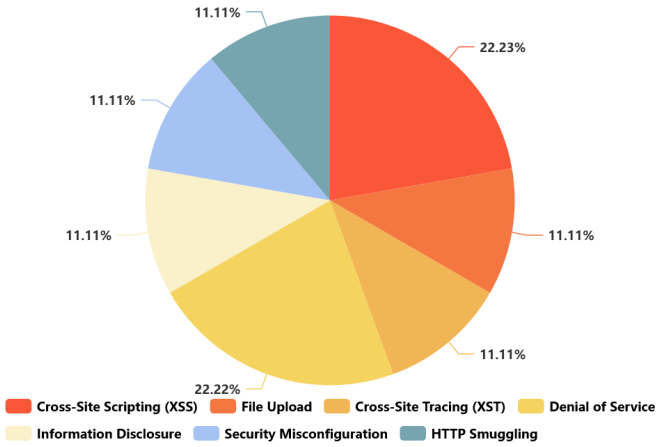

Figure B.5: Vulnerability types found.