
Bad Service: Towards detection of malicious web servers using a client-side honeypot.



Master Thesis

María Huerga Espino
Ana Isabel Asencio Martín

Aalborg University CPH
Electronics and IT



Electronics and IT
Aalborg University CPH
<http://www.aau.dk>

AALBORG UNIVERSITY

STUDENT REPORT

Title:

Bad Service: Towards detection of malicious web servers using a client-side honeypot.

Theme:

MSc. Cyber Security - Master Thesis

Project Period:

Spring Semester 2024

Participant:

María Huerga Espino
Ana Isabel Asencio Martín

Supervisor(s):

Shreyas Srinivasa shsr@es.aau.dk

Copies: 1

Page Numbers: 99

Date of Completion:

May 2024

Abstract:

The rise of malicious web servers poses significant cyber security threats. This thesis presents *MaiBee*, a client-side honeypot in the form of a browser extension designed to detect and analyse suspicious advertisements. The primary objective is to enhance the detection of malicious activities through filtering techniques and suspicion scoring based on multi-criteria.

MaiBee employs various functionalities, including ad detection, filtering, DNS lookup and reverse lookup, analysis of IP characteristics and reports of external sources such as AbuseIPDB and VirusTotal, to scrutinise potentially harmful web elements. By conducting an experiment with the honeypot on 10,000 sites, this thesis demonstrates *MaiBee's* effectiveness in identifying suspicious ads and its detection capabilities. The findings offer valuable insights into the nature of malicious advertisements and contribute to a deeper understanding of current cyber threats.

Contents

List of Figures	v
List of Tables	vii
1 Introduction	1
1.1 Problem Statement	2
1.2 State of The Art	3
1.3 Knowledge Gap	5
1.4 Contributions	6
2 Background	9
2.1 Client Honeypots	10
2.2 Online Advertisement	11
2.2.1 Ad Blockers	12
2.2.2 EasyList	13
2.2.3 Anti-Ad Blockers	14
2.3 Web Browsers	15
2.3.1 Reference Browser Architecture	16
2.3.2 Security	18
2.3.3 Browser Extensions	19
2.4 Web-Based Attacks	20
2.4.1 Browser Exploitation Framework	20
2.4.2 Maliciousness in ads	22
3 Methodology	25
3.1 Architecture of <i>MaiBee</i>	25
3.1.1 User Interface	26
3.1.2 Back-End	26
3.2 Design choices	27
3.2.1 Design of a Browser Extension	27
3.2.2 Deception Approach	30
3.3 Extension’s functionalities	31

3.3.1	Ad Detection	31
3.3.2	Filtering Suspicious Ads	32
3.3.3	DNS Lookup and Reverse Lookup	32
3.3.4	Content Security Policy	34
3.3.5	VirusTotal Analysis	35
3.4	Workflow in MaiBee	36
3.5	MaiBee Use Case	38
3.6	Weighted Scoring Model	41
3.7	Comparison with Thug	43
4	Results	45
4.1	Data Collection	45
4.1.1	Web Navigation	46
4.1.2	Preliminary Tests	48
4.2	Data Analysis	50
4.2.1	Characterisation of The Results	51
4.2.2	Scoring of the Data	54
4.2.3	Scoring Evaluation Results	57
5	Discussion	59
5.1	Overview of Final Results	59
5.1.1	General Findings	60
5.1.2	Top 3 Most Suspicious Sites	60
5.2	Limitations	65
5.2.1	Browser Extension	65
5.2.2	Selenium	66
5.2.3	Usage of APIs	67
5.3	Malicious JavaScript: Proof of Concept	68
5.3.1	JStap	68
5.3.2	Thug Discussion	69
6	Conclusion	73
6.1	Future Work	74
A	Suspicious Site A	77
B	Suspicious Site B	83
C	Suspicious Site C	87
	Bibliography	93

List of Figures

2.1	“Honeypot system placement in an organizational network” [71]. . .	10
2.2	Basic client-side honeypot. The honeypot initiates a communication with a potentially malicious server and waits for the server’s response to analyse it.	11
2.3	Message from the anti-ad blocker of Forbes.	15
2.4	Message from the anti-ad blocker of El País.	15
2.5	Web browser architecture.	16
2.6	Browser Exploitation Framework architecture diagram [10]	21
3.1	Architecture overview of <i>MaiBee</i>	27
3.2	Overall anatomy of a web browser extension [58]	28
3.3	Overall functionality of the ad detection phase.	31
3.4	Overview of the workflow of <i>MaiBee</i>	37
3.5	Extension already installed temporarily on Firefox.	38
3.6	How <i>MaiBee</i> looks when running the ad detector (“BEES”)	39
3.7	Options page of <i>MaiBee</i>	40
3.8	Structure of the files provided by Thug.	44
4.1	Overview of navigating multiple sites with an implementation inside <i>MaiBee</i>	47
4.2	Overview of navigating multiple sites with Selenium.	49
4.3	Proportion of suspicious URLs following the comparison with our Whitelist.	52
4.4	Overview of the tags discovered in the analysis.	53
4.5	Overview of the countries discovered in the analysis.	54
5.1	Malwarebytes Ad blocker showing adware detection on Site C.	65
5.2	Terminal view of one of the tests conducted by Thug (part 1).	70
5.3	Terminal view of one of the tests conducted by Thug (part 2).	71

List of Tables

1.1	Summary table of the state-of-the-art client honeypots.	5
2.1	Security properties involved in the definition of malicious ads. . . .	22
3.1	Key manifest elements used in <i>MaiBee</i>	29
3.2	Explained fields provided in the JSON logging mode by Thug. . . .	44
4.1	Overview of malicious JavaScript files.	46
4.2	Relevant results after testing Selenium and internal browsing functionality against one hundred sites	50
4.3	Relevant results from the analysis on the Top 10k Majestic.	51
4.4	Weights for our given criteria.	55
4.5	Scoring rubric applied to our given criteria.	56
4.6	Percentage of sites that were marked as <i>Shady</i> by the initial filter that received scores from each criterion	57
4.7	Highest, lowest, and average scores for each criterion.	58
5.1	Ranking position on the Majestic dataset of affected domains by Site A.	60
5.2	Final scores for Site A.	61
5.3	Final scores for Site B.	62
5.4	Ranking position on the Majestic dataset of affected domains by Site C.	64
5.5	Final scores for Site C.	64
5.6	Confusion matrix of JStap's experiments.	69
5.7	Datasets size used for JStap experiments.	69

Chapter 1

Introduction

With the growing interest in cyber security, the spectrum of related solutions for both offensive and defensive applications has expanded. Offensive security involves proactive measures taken to simulate and understand potential attack vectors, often done through penetration testing. These techniques help organisations identify vulnerabilities before malicious actors exploit them.

Contrariwise, defensive security focuses on protecting systems and networks. Various strategies can be included as defensive security, like firewalls, intrusion detection systems, and intrusion prevention systems. Among the many defensive strategies, the application of honeypots emerges as a significant tool for analysing attacker behaviour and trends. Honeypots are described as "a resource whose value lies in being probed, attacked, or compromised" [75, 72]. To align with this definition, honeypots must simulate technological or human vulnerabilities that can deceive cyber criminals, facilitating the study of their methodologies.

Honeypots can be deployed on both the server side and the client side to address a wide range of potential attack vectors. On the server side, they serve as bait, awaiting unauthorised access attempts. This allows them to gather data on the tactics used by malicious actors, which can then be analysed for further insights. On the client side, honeypots simulate user behaviour and interact with servers to identify those that exploit client-side vulnerabilities [80]. Although client-side honeypots are typically used to interact with web servers, their functionality extends beyond this scope. They can also be effectively used to gather data from a variety of other sources, including emails, FTP servers or SSH connections.

In this thesis, the focus will be on developing and utilising client-side honeypots to study potentially malicious advertisements. Advertisements are present all across the Internet and are often overlooked. By targeting advertisements, honey clients can provide valuable insights into their current usage to deliver malicious content.

The following sections will further explore this idea, including its relevance in

the current context, this thesis's specific contributions, the state of the art in honeypots, the knowledge gap that our solution aims to address, and the contributions.

1.1 Problem Statement

In our contemporary era, online content consumption has become an integral part of daily life, increasing steadily. Capitalising on this high level of engagement, companies have discovered online advertising as a lucrative opportunity, attracting not only legitimate businesses but also cyber attackers. These attackers see users as the weakest link in the security chain, making them the primary attack targets.

In this digital landscape, malicious advertisements blend in with legitimate content, poised to deceive users and compromise their security. This is a significant problem because these deceptive ads can have harmful consequences. When users click on them, they may not realise they are malicious, leading to the exploitation of system vulnerabilities, which can result in data breaches and unauthorised access. However, sometimes it is not necessary for the user to interact with the advertisement, as by just visiting the site displaying it, attackers can make unauthorised use of their computer's resources, for instance, through crypto mining.

As previously introduced, honeypots are an important security tool designed to attract attackers by mimicking real systems and capturing their behaviour for analysis. Their applicability extends to both server-side and client-side attacks. Client honeypots have gained importance within client-side exploitation methods because they focus on interacting and identifying suspicious content in the user's browser and system. This type of honeypot exhibits considerable potential, as it can be adapted to study the different trends in cyber security. One such adaptation could focus on malicious advertisements, thereby raising awareness about this issue and helping users recognise patterns.

To achieve this, research into the behaviour and characteristics of malicious advertisements is essential, focusing on how they can be detected to protect users. This thesis aims to develop a client honeypot system, implemented as a browser extension, to detect suspicious advertisements and explore the potential for classifying them based on observed maliciousness.

In this thesis, we study the possible strategies employed by server-originating attacks, specifically in the form of malicious ads, that aim at exploiting clients. To break down this problem statement, we form the following research questions for our thesis:

- **RQ1:** What strategies are used by malicious servers in online advertising today?
- **RQ2:** What are the attack vectors originating from the above strategies, and what is the impact on clients?

1.2. State of The Art

- **RQ3:** How could we capture such malicious attempts on the client side for comprehensive analysis?

RQ1 aims to uncover the techniques employed by malicious servers to deliver deceptive advertisements to users. The focus is on understanding various methods used by attackers to integrate malicious content into advertising networks. Once the strategies of malicious servers are identified, the next step is to define the attack vectors used. Thus, the objective in **RQ2** is to understand them to help in evaluating the potential impact on clients. The last research question **RQ3** focuses on the practical implementation of capturing and analysing malicious attempts targeting users. It involves the design and development of a client honeypot capable of detecting and analysing advertisements to discover suspicious ones.

1.2 State of The Art

This section will provide an overview of the state-of-the-art client honeypots. These honeypots are classified into two interaction levels: high or low. As the name suggests, the difference between these levels is based on the interaction abilities of the honeypot towards the attacker. In essence, high-interaction honeypots offer full interaction capabilities to the attacker as they are real systems and not a simulation, for example, by interacting with an actual application. In contrast, low-interaction honeypots offer a minimal simulation of the system and, hence, minimum interaction capabilities to the attacker, for example, by interacting with a simulated web browser. An in-depth definition of how this applies to honeyclients is discussed in 2.1.

In HoneyC [80], Seifert et al. introduce a “low-interaction client honeypot,” which diverges from the predominant high-interaction classification of existing honeypots. The authors delve into the landscape of client honeypots, emphasising a common characteristic: they are uniformly classified as high-interaction. HoneyC offers a novel approach, focusing primarily on the HTTP1.1 protocol in its initial version. The honeypot, consisting of three loosely coupled components (the queuer, the visitor and the analysis engine), enables seamless communication between them by processing standard input and generating standard output.

The paper indicates that the analysis engine uses Snort signatures, which can lead to false positives. However, the authors are optimistic that this innovative approach’s performance will compensate for such occurrences.

In *PhoneyC: A Virtual Client Honeypot* [59], the author proposes a new category for client honeypots that goes beyond the traditional consideration of interaction levels. This new category differentiates between whether the vulnerable application is actively used (real context) or emulated (virtual context). According to the article, the advantage of the virtual approach is its ability to present a wider range of potential victims that may attract attackers.

This low-interaction virtual honeyclient consists of a web crawler and an analysis engine. It features dynamic content analysis, such as JavaScript, and can deobfuscate it as needed, avoiding reliance on "external patterns or antivirus" alone. PhoneyC's application is specifically designed for limited environments instead of the broader World Wide Web.

Released in The HoneyNet Project Workshop in 2012 by Angelo Dell'Aera, Thug is an open-source, Python low-interaction client honeypot designed to mimic the behaviour of a web browser to detect and emulate malicious content [24]. Unlike earlier tools, Thug addresses limitations observed in other client honeypots like PhoneyC by implementing a complete Document Object Model (DOM) and supporting a more comprehensive array of browser versions, ActiveX controls, and plugin modules. This honeypot follows a hybrid static and dynamic analysis approach, allowing for a complete study of the malicious scripts and web pages encountered. It uses the Google V8 JavaScript engine to analyse malicious JavaScript code and the Libemu library to detect and emulate shellcodes [24].

Furthermore, Thug emulates 46 different "personalities", including mobile devices, tablets, and computers. The emulated browser can also be configured to use a specific version of plugins. Combined with other functionalities, this tool is powerful enough to trick malicious pages into believing they interact with real users, allowing Thug to analyse possible client-side attacks realistically.

A recurring theme in the discussions of several papers on future work involves the potential integration between low-interaction and high-interaction honeyclients. This combination is actively explored in [7]. This doctoral study seeks to investigate web-based attacks to predict potentially malicious behaviour. A foundational element in this research involves the use of client honeypots. The proposal suggests incorporating a fusion of Capture-HPC, characterised as a high interaction honeyclient, and a state machine model, coupled with Honeyware, identified as a low interaction one. A comparison of the state-of-the-art client honeypots discussed is presented in Table 1.1.

Limitations of Current Client Honeypots

Despite their multiple benefits, current honeypots like HoneyC, PhoneyC, and Thug have limitations that impact their effectiveness. HoneyC, while providing a novel low-interaction approach, relies heavily on Snort signatures for detection, which can result in a high rate of false positives. This reliance on signature-based detection means that HoneyC may not detect new or unknown threats that do not match existing signatures.

PhoneyC offers a virtual environment that can dynamically analyse content such as JavaScript, but its design limits it to specific environments. This restriction means it may not effectively analyse threats across the broader web.

1.3. Knowledge Gap

Honeypot	Interaction Level	Key Features	Deception Techniques	Limitations
HoneyC [HoneyC]	Low-Interaction	<ul style="list-style-type: none"> - Signature-based - Crawling capabilities - Fast identification of malicious servers 	<ul style="list-style-type: none"> -No specification -Relies on being the first low-interaction honey-client 	<ul style="list-style-type: none"> -Dependence on complimentary systems - Performance
PhoneyC [59]	Low-Interaction	<ul style="list-style-type: none"> - Virtual honeyclient - Dynamic content analysis (e.g., JavaScript) - Modular 	<ul style="list-style-type: none"> - User-agent headers and JavaScript navigator to create browser “personalities” - Mimics normal behaviour by including referring URLs in the HTTP client headers 	<ul style="list-style-type: none"> - Limited scalability - Might require integration with shell-code analysis engines for understanding next stages of attacks
Thug [24]	Low-Interaction	<ul style="list-style-type: none"> - Complete DOM implementation - Supports various browser versions - Hybrid static/dynamic analysis - Uses Google V8 JavaScript engine and Libemu library 	<ul style="list-style-type: none"> - Emulates 46 different entities 	<ul style="list-style-type: none"> - Challenges in scalability and performance - High resource consumption
Capture-HPC & Honeyware (Integrated Approach) [7]	Integration of High-Interaction & Low-Interaction	<ul style="list-style-type: none"> - Fusion of Capture-HPC (high-interaction) and state machine model with Honeyware (low-interaction) - Combines strengths of high and low-interaction honeypots - Comprehensive analysis of web-based attacks 	<ul style="list-style-type: none"> - Honeyware uses clients to be able to send various requests to the servers while maintaining IP consistency. - Honeyware can mimic Internet Explorer, Firefox, Opera, Chrome, Safari and Konqueror 	<ul style="list-style-type: none"> - Complexity in implementation - Integration challenges

Table 1.1: Summary table of the state-of-the-art client honeypots.

Thug addresses some limitations of previous honeyclients like PhoneyC, supports different “personalities” and follows a hybrid approach. However, it still faces challenges in scalability and performance. The nature of its analysis can lead to significant resource consumption, making it less practical for large projects.

1.3 Knowledge Gap

The study of client-side honeypot applications shows their varied purposes and reveals a research gap in the detection of malicious web servers hidden in online advertisements. Current approaches, as observed in Table 1.1, often fall short in terms of scalability, accuracy, and comprehensiveness, especially when identifying threats embedded within dynamic and user-interactive web content.

This gap indicates the potential need for an approach integrating web extension technologies with client-side honeypot mechanisms. By focusing on this aspect,

the present thesis aspires to take the lead in identifying and countering the threats associated with malicious online advertisements. The integration of a web browser extension designed to detect suspicious advertisements and the analysis following advanced client honeypot techniques could provide a more effective and comprehensive solution.

Choosing to develop a browser extension is strategic for several reasons. Firstly, it allows the system to be “closer” to the browser, providing more direct and immediate access to the content and interactions within the browser environment. Secondly, developing a client honeypot as a browser extension offers a unique opportunity to employ a different deception technique. By embedding the honeypot directly into the browser, it can interact with web content and capture more sophisticated attack vectors that traditional methods might miss. Additionally, inspiration was drawn from ad blocker extensions, as they have been proven effective in identifying and filtering out unwanted ads, and by building on their methodologies, we aim to enhance our honeypot’s capability to detect and analyse suspicious ads.

1.4 Contributions

In addressing the identified knowledge gap, this work makes the subsequent contributions:

- We introduce a novel approach to implementing honeyclients through browser extensions.
- We develop and release *MaiBee*, an open-source client honeypot in the form of a Firefox add-on, capable of detecting advertisements and analysing their characteristics to provide insights into their security [38].
- We investigate how malicious actors exploit advertisements as an attack vector.
- We propose a method for using collected data to assess the suspiciousness of advertisements.
- We conduct a security-oriented study of the landscape of online advertisement by exploring the Internet with our add-on.

The rest of the report is structured as follows. Chapter 2 provides the foundational knowledge necessary to understand the thesis. This covers client honeypots, the landscape of online advertisements, ad blockers, and the role of web browsers and web-based attacks. Chapter 3 outlines the design and implementation of *MaiBee*. It details the architecture, functionalities, and a use case of *MaiBee*

1.4. Contributions

and includes a comparison with a different honeypot. Chapter 4 presents the data collected using *MaiBee* and characterises the results. Chapter 5 discusses the research findings and limitations. It also presents a proof of concept for classifying malicious JavaScript. Finally, Chapter 6 concludes the thesis, summarising the findings and suggesting areas for future work.

Chapter 2

Background

This chapter delves into the essential areas of knowledge required to study a client-side honeypot, particularly a browser extension that addresses suspicious advertisements. This exploration includes defining client honeypots, understanding online advertisements, examining the mechanics of ad blockers, studying web browser functionality, and analysing web-based attacks executed through malicious ads. First, it is crucial to define and distinguish client honeypots from server-side honeypots.

Next, understanding online advertisements is key. This section covers the various types of online ads, their characteristics, and the entities involved in the ad ecosystem. By examining these, we can better understand how malicious ads can appear.

Given the similarities between the proposed client honeypot and ad blockers, we will study the mechanics of ad blockers. This involves exploring how ad blockers identify and block ads, focusing on detection mechanisms and filtering rules. This knowledge will guide the development of our honeypot's detection mechanisms for identifying advertisements.

Understanding web browsers is also relevant, as the development of our client honeypot involves creating a browser extension. Honeypots often need to mimic browsers to deceive attackers. Although this proposed honeypot does not need to mimic a browser, it is developed as a browser extension, so understanding what this entails is important. Additionally, since malicious ads are viewed within the browser context, the browser plays a key role in providing security layers to the user.

Finally, we will examine various web-based attacks and how these can be executed through malicious advertisements. Each section builds upon the previous, providing the necessary background to study and develop a client-side honeypot for detecting and mitigating the threat of malicious advertisements.

2.1 Client Honeybots

As previously established, a honeypot is a deception technology used to analyse the behaviour of attackers in different contexts. This analysis can have different goals, such as characterising malicious actors or learning how to detect them in real systems [39]. The most common implementation of honeypots is by simulating a server with which the malicious actor interacts. The placement of this category of honeypots across the network may vary depending on their purpose. This is exemplified in Figure 2.1, which shows three different possibilities of honeypot arrangements. From left to right, the first honeypot is positioned outside a firewall, which could be, for instance, to attract large amounts of traffic; the second honeypot is lined after a router immediately after an external firewall; this could lead to the study of more sophisticated attacks; the third honeypot is sitting after a switch, in the same level as other relevant systems, such as the internal firewall, this could be related to a targeted attack.

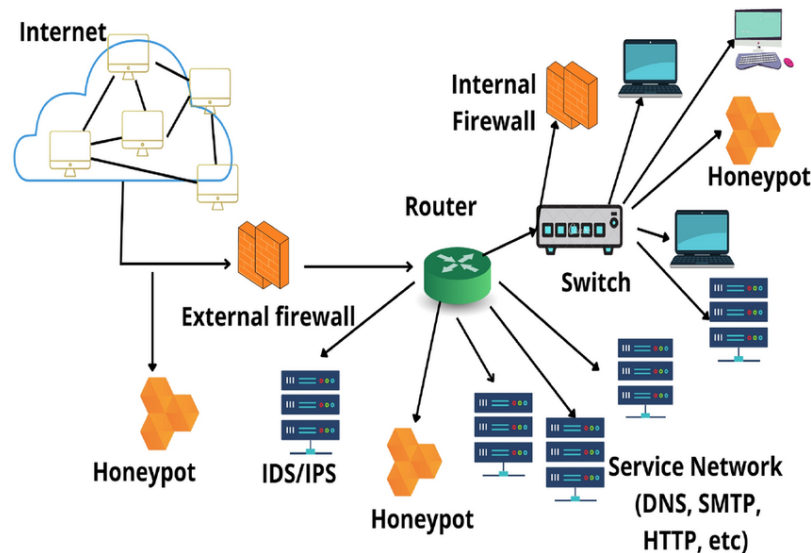


Figure 2.1: “Honeypot system placement in an organizational network” [71].

Another possibility is to deploy a client-side honeypot, also commonly referred to as honeyclients. In the case of a web-based honeyclient, a user’s normal navigation through the web is mimicked, and the interaction of potentially malicious servers with the client is logged so that it can be later analysed. Figure 2.2 represents a basic instance of a honeyclient in which the server performs an attack. For this kind of honeypot to work as intended, there are three areas that must be covered [80]:

1. Interaction with the server.

2.2. Online Advertisement

2. Generation of requests.
3. Analysis of the communication with the server or of the potential changes of the system.

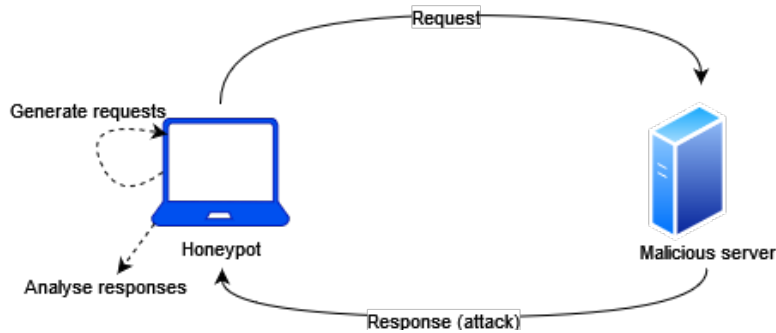


Figure 2.2: Basic client-side honeypot. The honeypot initiates a communication with a potentially malicious server and waits for the server’s response to analyse it.

The classification of client honeypots includes two main categories: high-interaction and low-interaction. The former allows the attacker to interact with real operating systems and applications rather than emulated services. These services often include web browsers and web crawlers [75]. This type facilitates in-depth analysis of attacker behaviour and enables the detection of 0-day exploits with the inevitable downside of increased risk.

On the other hand, low-interaction client honeypots take a more cautious approach by emulating specific components of the client environment, such as the web browser, thus limiting the scope of interaction with potential threats. While these honeypots provide rapid results and ease of deployment, their drawback lies in their inability to detect 0-day exploits due to their detection algorithm depending on implementing signatures for known attacks [75].

2.2 Online Advertisement

The presence of advertisements on the Internet is widely recognised. Specifically, in the United States, the increase in advertising revenue from 2020 to 2021 was 35.4% [76]. This change in the market revenue is a reflection of the changes in the industry itself: since the first online banner was shown in 1994 [43], the methods of displaying ads have evolved significantly. For instance, the prevalence of external *pop-up ads*, which would open in new tabs, has decreased as modern browsers now typically block them by default.

Within this evolving framework, the predominant forms of advertisement include Search Engine Advertisement (SEA), video advertising, and banner advertising [85].

- **SEA:** These ads appear near the search bar when a user conducts an online search, typically providing relevant options for the query. [78]
- **Video advertising:** This category encompasses two distinct types: ads delivered in video format and ads displayed during video playback on platforms such as YouTube. [70]
- **Banner advertising:** This format of ads make use of “attractive images, shapes, and sizes of displays” [25] to engage potential consumers.

Although for the final user, the way in which ads are displayed might be very simple, it requires the combined work of different entities, from which the following stand out: [69]

- The term **advertiser** refers to the part that is interested in promoting their product on the Internet and has to pay for that.
- The **publisher** is used to refer to the part that earns money by showing the ads to their users. An example could be an online magazine.
- To fill the role of a broker between the last two parts, and **advertising network** is required. This is the case of Google AdSense [36].
- An **ad exchange** is used to facilitate more effective use of the ad space. A common example of this is DoubleClick, which has now been integrated into Google Marketing Platform [34].

As expressed in [69], the main difference between traditional ads and online ones, lies in how personalised these can be. While traditional ads have been broadcasted, online ads are based on the user’s interests. To enable this, the use of cookies is essential.

2.2.1 Ad Blockers

Essentially, ad blockers serve as digital gatekeepers that block requests to download ads. This technology has become increasingly popular in recent years due to the exponential growth of online advertising. The 2023 eyeo Ad-Filtering Report (previously known as the PageFair Adblock Report) revealed that there were 912 million users worldwide actively blocking ads, which is an 11% increase from 2021 [27].

Ad blockers rely on filter rules that contain information on what elements to block, hide, and allow to appear on the websites the user is visiting [3]. Filter rules can range from basic to a more personalised and complex approach; for example, a basic filter can refer to one specific address or a set that should be blocked. In

2.2. Online Advertisement

this case, the rule “`||example.org`” would block `hxxp://example.org/ad1.gif` and `hxxps://ads.example.org:8000/`, but not `hxxp://ads.example.org.us/ad1.gif`.

On the other hand, cosmetic filter rules make use of the fact that the DOM allows content to be accessed and modified, making this document the container of all the elements of the website. These rules, based on CSS, append specific CSS styles to the webpage, effectively concealing targeted elements [6]. Thus, the ad blocker will use these filter rules to identify the parts of the websites containing ads [5]. As an example, the rule “`example.org##.banner`” hides an element with the class *banner* at `example.org` and all subdomains.

Additionally, in cases where basic and cosmetic filter rules are insufficient, ad blockers may employ HTML filtering rules to modify the HTML code of web pages before the browser loads them. These HTML filtering rules enable the specification of HTML elements to be removed or modified, effectively blocking ads at the structural level of the webpage. By implementing such filtering rules, ad blockers can provide users with a more comprehensive defence against intrusive advertisements and enhance their browsing experience. For example, the rule “`example.org$$script[data-src="banner"]`” removes all script elements with the attribute *data-src* containing the substring *banner*. [6]

Some well-known commercial ad blockers are *AdBlock*, *AdGuard* and *uBlock origin*. They compare the HTTP requests to the filter lists the user is subscribed to and any custom filters added. The users of these tools can subscribe to different public filter lists or add custom ones. Some examples of these filter lists are *EasyList*, *Peter Lowe’s Ad and tracking server list*, *AdGuard – Mobile Ads*, or the regional *Dandelion Sprouts nordiske filtrer*. Their items are then compared with the HTTP requests. If the URL of the request matches one of the filters, the request is blocked and the resource is not downloaded [3]. However, it is important to note that despite their effectiveness in blocking ads, ad blockers may not always eliminate all traces of the ad content. In some cases, remnants such as blank spaces or hidden URLs may still be visible on the site.

Ad blockers may consider various methods used to integrate ads into a website. These methods include using JavaScript to load an external script from the ad server for displaying the advertisement, embedding Iframes, utilising server-side scripting to include ad content during page rendering, and hard-coding ads directly into the website, among others [70].

2.2.2 EasyList

Among the aforementioned filter lists, EasyList [28] stands out. Originally created in 2005 for the now deprecated Adblock browser extension [4], EasyList merged in 2013 with the renowned Fanboy’s List [29]. It is currently used by some of the most popular ad blockers, such as AdBlock Plus, AdBlock, uBlock Origin and AdGuard

[28].

EasyList’s approach is to create a set of filters that balance simplicity and effectiveness. To achieve this, the list is organised into several categories, namely:

1. General advert blocking filters.
2. General element hiding rules.
3. Third-party advertisers.
4. Third-party adverts.
5. Specific advert blocking filters.
6. Specific element hiding rules.
7. Allowlists to fix broken sites.

These categories are delimited by comments in the format “! - - - - -” followed by the category name, making the list easy to navigate and maintain. In addition, EasyList includes specific subcategories to address particular areas, such as adult content and pop-up ads. Another interesting subcategory is focused on recognising anti-adblocks in the sites. The concept of anti-adblocks is explored in the following subsection 2.2.3.

2.2.3 Anti-Ad Blockers

In response to the widespread adoption of ad blockers, advertisers and website owners have adopted countermeasures in the shape of anti-ad blockers. These tools are designed to detect and evade ad blockers, thus ensuring that ads are displayed even when ad-blocking software is installed in the browser. Whilst this tool can be used for legitimate purposes, it can also be exploited by attackers who may take advantage of the anti-ad blocker technology to deliver malicious or deceptive ads.

Anti-ad blockers can follow different approaches, but they generally operate by analysing the website content and comparing the expected page layout with the rendered version. If there are differences, it suggests that an ad blocker is disrupting the content [73]. Another technique utilises “bait content”, which are typically invisible elements designed to resemble ads, serving as a trap for ad blockers. If the bait has been hidden after loading the page, the detection code assumes the presence of an ad blocker and can act accordingly [47].

Countermeasures against ad blockers include code obfuscation techniques, which conceal ad-related code to prevent ad blockers from identifying ads. Dynamic ad insertion is another strategy employed, where ads are loaded dynamically after the page loads, making it challenging for ad blockers to react fast and block the

2.3. Web Browsers

ads effectively. [47] However, these techniques could be branded as unethical since they circumvent ad blockers and, thus, users' wishes.

Therefore, a popular technique is to ask users to disable their ad blocker or whitelist the site. The softer version of this is followed by Forbes, as seen in Figure 2.3, where it asks to whitelist the site or create an account. Even so, this message can be ignored, and the website with blocked ads will still be available. On the other hand, pages like El País (Spanish newspaper) additionally to present a similar message, as seen in Figure 2.4, they limit the use of the website until the ad blocker is disabled or the user pays for a subscription.

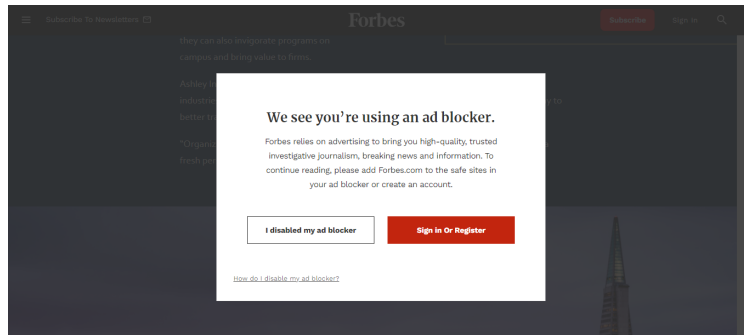


Figure 2.3: Message from the anti-ad blocker of Forbes.

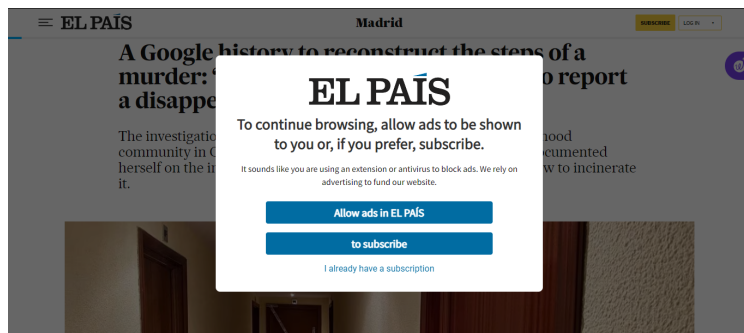


Figure 2.4: Message from the anti-ad blocker of El País.

Understanding the strategies employed by anti-ad blockers and the potential risks of their misuse is crucial for developing effective ad-blocking solutions.

2.3 Web Browsers

Web browsers, operating within the client/server model, are software applications that facilitate users' interaction with the World Wide Web. Understanding their architecture and functionality is crucial for comprehending how client-side attacks

exploit these platforms. In the context of this thesis, web browsers serve as the primary interface through which users encounter malicious advertisements, making them a focal point for studying and mitigating such threats.

Web browsers retrieve, interpret and render content from the World Wide Web. In this client/server model, the browser acts as the client, executed on a computer (or other Internet-enabled device supporting browsers), which initiates communication with web servers to request information [60]. Web browsers use protocols such as Hypertext Transfer Protocol (HTTP) to request and receive data, facilitating this exchange between the client and server, therefore connecting users with online resources. Through the interpretation of languages like HTML, CSS, and JavaScript, browsers present the content in a visually understandable format.

Given the importance of web browsers in accessing and displaying web content, they are often targeted by client-side attacks. Malicious actors may exploit vulnerabilities within the browser or through malicious content delivered via advertisements. Examples of such attacks include drive-by downloads, which exploit browser vulnerabilities to install malware without user consent, and social engineering attacks, where users are tricked into revealing sensitive information or downloading malicious software.

2.3.1 Reference Browser Architecture

Understanding web browsers and their architecture is essential for developing effective countermeasures against client-side attacks. The following diagram (Figure 2.5) [60, 23] illustrates the reference architecture that developers adhere to when designing browsers. Understanding this architecture will help in identifying potential attack vectors and vulnerabilities used in attacks on the client.

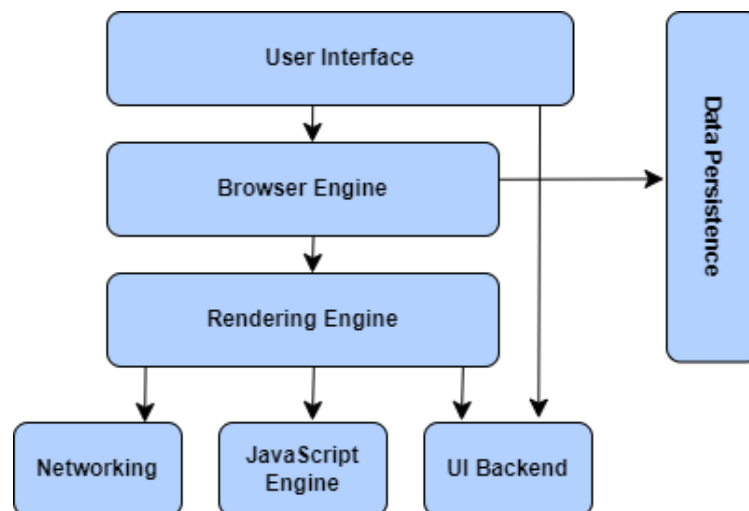


Figure 2.5: Web browser architecture.

2.3. Web Browsers

- **User Interface (UI).** It serves as the gateway for user interaction with the browser. It encompasses features like the address bar, navigation buttons, bookmarks, and refresh options. Understanding the UI is fundamental to developing a client honeypot in the shape of a browser extension, as it will serve as the main interface for users to interact with.
- **Browser Engine.** It handles the interaction between the UI and the rendering engine and other components of the browser. It translates user actions, such as pressing a button or typing a URL, into commands for the rendering engine. This component is essential for processing user interactions that may trigger suspicious scripts.
- **Rendering Engine.** It is responsible for interpreting and rendering web content. It creates the DOM tree by interpreting HTML while it keeps parsing other elements, such as XML documents and images formatted with CSS, to generate the layout displayed in the User Interface. The rendering engine and its understanding are important as it can be exploited by malicious advertisements to manipulate the DOM and execute harmful scripts.
- **Networking.** This component communicates with the server via HTTP and FTP to fetch resources, such as loading the requested web page. It also handles cookies and cache.
- **JavaScript Engine.** It parses and executes all the JavaScript code of the document. First, it transforms the code into a data structure called Abstract Syntax Tree (AST). Once the AST has been built, it is converted to machine code, executed, and the results are sent to the rendering engine [13]. Every major browser has one JavaScript engine; for example, Chrome uses the V8 engine, and Firefox uses SpiderMonkey. The JavaScript engine is especially important when it comes to developing a client honeypot, as malicious advertisements may involve having JavaScript code embedded in them.
- **UI Backend.** This subsystem is responsible for adding colours, textures, fonts, and other graphical elements to the website, enhancing the overall functionality and performance of the browser. This component helps ensure that the browser extension developed can present functionalities or alerts in a visually understandable manner.
- **Data Persistence/Storage.** It consists of a database on the local drive that manages data associated with the browsing session, like cache, cookies, preferences, security certificates, and browsing history. This is relevant to logging detected suspicious activity and maintaining a history of detected advertisements.

By understanding web browser architecture, the client honeypot can be effectively designed and implemented within the web browser environment, ensuring it can intercept and analyse client-side attacks delivered through malicious advertisements.

2.3.2 Security

As the primary gateway for internet access, attacks on web browsers pose a significant security risk, especially to users' sensitive information, including passwords and browsing habits. Exploiting vulnerabilities in web browsers allows attackers to access classified data stored on users' devices. These software applications are susceptible to vulnerabilities such as insecure configurations, exposure to malicious websites and applications, and risky browsing behaviours resulting from insufficient user awareness or training [20]. Thus, browser security measures are essential for defending against cyber attacks. Some of these measures are the following:

- **HTTPS and SSL/TLS.** HTTPS is the secure version of HTTP, and it is in charge of encrypting the communication between the browser and web servers to increase the security of data transfer. The protocol used in this encryption is the Transport Layer Security (TLS), formerly known as Secure Sockets Layer (SSL). This protocol secures communications by using asymmetric public key infrastructure [18], which ensures the confidentiality and integrity of transmitted data.
- **Content Security Policy (CSP).** This policy enhances security within a web page by detecting and mitigating specific attack types, such as Cross-Site Scripting (XSS) (see Section 2.4.2) and data injection attacks. CSP allows defining the domains that the browser should consider valid sources of executable scripts, thus restricting unauthorised execution of malicious code injected into web pages. The server can also specify which protocols are permitted when loading content; for example, a server can specify that HTTPS must be used for all content loading to provide an additional defence against packet sniffing. The Content-Security-Policy HTTP header is used to configure this policy [49].
- **Same-Origin Policy (SOP).** This security mechanism restricts interactions between documents or scripts from different origins. Enforcing strict domain-based interactions helps reduce malicious websites accessing sensitive data from other sites. Two URLs have the *same origin* if they both share protocol, port (if specified) and hostname. The SOP can serve as a mitigation against Cross-Site Request Forgery (CSRF) attacks (see Section 2.4.2). [55]

2.3. Web Browsers

- **Cross-Origin Resource Sharing (CORS).** This is an HTTP header-based mechanism that relaxes the consequences of SOP by letting a server specify third-party origins (domains, schemes, or ports) from which a browser should allow resources to be loaded [51]. Rather than controlling the sources as CSP does, CORS focuses on regulating cross-origin requests and resource access.
- **Phishing and Malware Protection.** Web browsers have robust defences against phishing attempts and malicious websites. Through heuristics, blacklists, and real-time analysis, browsers can identify and warn users about suspicious URLs and phishing scams. For example, the Safe Browsing Service by Google, also used by Firefox and Safari, is a popular and trusted database that helps protect devices when users attempt to navigate to dangerous sites or download dangerous files [37].
- **Sandboxing.** This is a technique that most browsers have integrated to enhance protection. Browser sandboxing [32] is the concept of having a safe and isolated environment where code execution is restricted from interacting with the operating system, files, or other critical components of the computer. Each browser tab or process operates separately, limiting code execution and resource access to that specific process. Additionally, within the sandbox, websites or scripts operate with significantly reduced rights.

2.3.3 Browser Extensions

Extensions are software that adds custom functionalities to the core browser. They are developed using web-based technologies such as HTML, CSS and JavaScript. One of the aspects that enhances the power of extensions is their ability to leverage the same web APIs as JavaScript on a web page, in addition to accessing their own set of JavaScript APIs, extending the browser's functionality beyond traditional web browsing capabilities [58].

Other important aspects to mention within extensions involve security implications. Extensions require specific permissions to access browser features and user data. By following the principle of least privilege, developers ensure that the extension does not access data that is not necessary for its functionality. Another important concept to consider is the Content Security Policy, explained earlier in this Chapter. Extensions built using WebExtension APIs are automatically subject to a CSP, which limits the origins from which they can fetch code, including `<script>`, and prohibits risky practices like evaluating strings as JavaScript like `eval()` does. However, this policy can be modified by the extension developer [50].

2.4 Web-Based Attacks

Although both browsers and websites frequently enhance their security measures, malicious actors continue to find ways to breach these defences. Typically, their objectives fall into two broad, non-mutually exclusive categories: compromising end-users, for example, by stealing sensitive information; or disrupting the website host, for instance, by diminishing its capacity to handle visitors. Consequently, the classification of web attacks is diverse. Some classifications focus on the technical aspects of attacks, as outlined by the OWASP Top Ten [66], while others concentrate on strategies targeting end-users, as discussed in [16]. Taking the list available at [79] as a guide, four primary types of web attacks are considered here:

- **Cross-Site Scripting (XSS):** This technique uses malicious scripts to exploit vulnerabilities. There are three main types [41]:
 - *Stored XSS* involves malicious code stored on the server, which executes when the user loads the affected content.
 - *Reflected XSS* occurs when the malicious script is part of the HTTP request and is reflected in the response, often via a malicious link.
 - *DOM-based XSS* modifies the DOM structure without sending the script to the server.
- **Cross-Site Request Forgery (CSRF):** This approach induces a user to perform unintended actions within an authenticated application context. [67]
- **Injection Attacks:** This category includes well-known methods like SQL injection, where the goal is to inject malicious code into a web application.
- **Distributed Denial-of-Service :** This well-known method targets the availability of a website by overwhelming it with traffic. A common defence is using Content Distribution Networks (CDNs) to distribute and manage traffic load.

Although the complexity of these threats might seem high for common users, it is relatively easy to get started with them using widely accessible tools. An example that was deeply explored to better understand this thesis is the Browser Exploitation Framework (BeEF) [11].

2.4.1 Browser Exploitation Framework

BeEF is an open-source project available since 2005. It takes inspiration from both XSS and protocol communication weaknesses research [15]. The intent of BeEF is

2.4. Web-Based Attacks

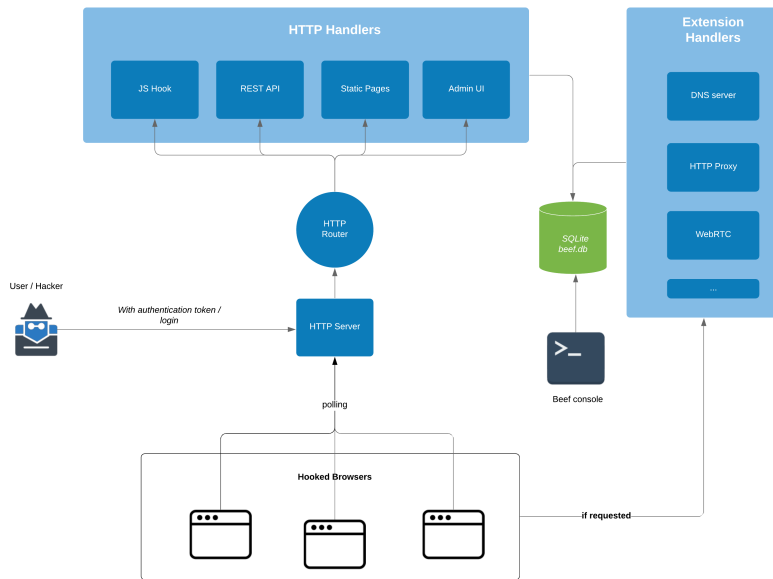


Figure 2.6: Browser Exploitation Framework architecture diagram [10]

to provide a way for a “professional penetration tester to assess the actual security posture of a target environment” [11]. Despite this, the truth is that anyone with malicious intent can access it. As shown in Figure 2.6, the intended use of BeEF involves the installation on an HTTP server, to which the hacker can connect through either the Admin UI or through an API. From there, the attacker can control all the hooked browsers and access logs and send different attacks to each of the browsers. The list of attacks is extensive, but it is divided into twelve main modules: browsers (such as accessing the visited URLs), Chrome extensions, debug, exploits, hosts (like getting the physical location), Inter-protocol Exploitation/Communication (IPEC), miscellaneous, network, persistence, PhoneGap, and social engineering.

In order for the hacker to make use of these modules, the key component of this framework is the `hook.js` script. This JavaScript code is embedded in the website the victim visits and connects back to the attacker’s control panel, maintaining continuous communication. Even though the frequency of the communication can be configured, if this is not done correctly, it can be a flag to detect the use of BeEF, especially in contexts where high volumes of traffic are not to be expected.

The complexity of the use of BeEF is easily changeable by its users. For instance, Figure 2.6 depicts a scenario where different hooked browsers are communicating directly with the HTTP server, but it is also possible to have only one browser that the `hook.js` is utilising to communicate with the server, and that at the same time

is gathering information from other browsers in the same network.

2.4.2 Maliciousness in ads

Within the realm of web-based attacks, malicious advertisements, or "malvertising," represent a significant threat to both users and websites. Due to the widespread nature of online advertisements, the possibility of them being malicious can pose a great risk for online users. Malicious ads can exploit vulnerabilities in websites, browsers, and plugins, compromising the security of users' systems. Therefore, it is relevant to understand and identify malicious behaviour within advertisements to protect users.

The definition of maliciousness in ads followed by this report includes anything that can compromise the Confidentiality, Integrity and Availability (CIA) of web content, as seen in Table 2.1. The effects of said compromise affect end-users, both in their static display when they do not interact by clicking on the ad, and when there is interaction through users' clicks.

Security Property	Description (if violated)
Confidentiality	Collection of sensitive information or exposing confidential content to unauthorised parties.
Integrity	Alteration or manipulation of the appearance and functionality of webpages without the consent or knowledge of the user.
Availability	Overload servers or injection of malicious code that causes unavailability of the services.

Table 2.1: Security properties involved in the definition of malicious ads.

A widely used term in ads security is *malvertising*. The definitions used by different researchers for this term vary widely. For instance, in the paper "Knowing Your Enemy: Understanding and Detecting Malicious Web Advertising" [42], malvertising is defined as any ad-related malicious activities such as propagating malware, scamming, click fraud, etc. It can target any link on an ad-delivery chain, including publishers, advertising networks, and advertisers. However, in the study conducted by Pooranian et al., [69] malvertising is defined as a platform for distributing malware by injecting malicious code into legitimate ad networks, where it only affects the advertiser and the user.

There are various types of malicious ads. Some of these ads alter the content of websites by injecting new functionalities or changing their appearance. This is done by inserting or manipulating scripts or redirecting users to unexpected destinations, often without any interaction with the ad. Such modifications can result in unauthorised access to sensitive information and degrade the quality of web content.

2.4. Web-Based Attacks

Another type of malicious ad occurs when interaction with the ad triggers the automatic download and installation of malware onto the device without the user's consent or knowledge. This is known as a *drive-by download* and is achieved through the exploitation of web browsers or plugin vulnerabilities. This attack vector is particularly dangerous as it requires minimal user interaction and can result in significant system compromise.

Social engineering is another significant attack vector in the context of malicious ads. This technique involves manipulating users into performing actions or revealing confidential information. Malicious ads might use deceptive messages to trick users into believing they are interacting with legitimate content, thereby gaining their trust and extracting sensitive information such as credentials. This type of attack relies on human psychology rather than technical vulnerabilities, making it an ever-present threat. Additionally, malicious ads may employ redirects and *phishing* attempts, redirecting users to fraudulent websites or phishing pages designed to steal sensitive information such as login credentials, financial details, or personal data. These redirects can occur upon clicking the ad. Phishing through ads uses the user's trust in legitimate-looking content to deceive them into revealing critical information.

A different kind of attack is *click fraud*. The final goal of this attack is to generate revenue through the use of fake clicks on ads. Although at first glance this kind of attack does not necessarily affect the end-user, one of the subcategories included here is *badvertising*. As defined by Gandhi et al. [46], badvertising refers to a type of click fraud attack that silently generates click-throughs on advertisements, deceiving advertisers and artificially increasing revenue by making the end user's machine run "extra malicious scripts to automatically deploy clicks" [69]. This can degrade the performance of the user's device and expose it to further malicious scripts.

The impacts on clients can be significant and multifaceted. First, there is a risk in terms of user data and privacy. Users can inadvertently expose sensitive information by clicking on or just visualising malicious ads. Second, these interactions can compromise the system and make the installation of malware without the user's knowledge possible, potentially leading to data loss and money. Third, there is a performance degradation aspect, as click fraud and badvertising can cause devices to run additional unwanted scripts, slowing down the system. Finally, social engineering attacks can deceive users into taking actions that compromise their security and privacy.

Chapter 3

Methodology

This chapter provides a comprehensive overview of our approach to capturing suspicious advertisements, facilitating their analysis, and increasing awareness about them, aligned with RQ3 introduced in Section 1.1. Our methodology involves creating a client-side honeypot in the form of a browser extension. This innovative approach serves as a deception technique to identify malicious advertisements, leveraging the fact that browser extensions have not been previously employed for this specific purpose.

The chapter begins by detailing the overall structure of *MaiBee*, illustrating how its various components interact. Following this, we delve into the design decisions that support *MaiBee*. Building on the foundational knowledge of web extensions discussed in Section 2.3.3, we expand on the technical requirements necessary for the architecture to function effectively. Additionally, this section examines the design of the honeypot’s deceptive capabilities.

Next, we outline the various aspects that the extension monitors to produce meaningful results. We then describe how these identified functionalities are integrated into a cohesive workflow, enabling *MaiBee* to operate effectively. This sets the stage for a practical guide on interacting with *MaiBee* from a user’s perspective, demonstrating its real-world application.

Furthermore, we introduce a scoring model designed to interpret and label the collected results. Finally, we present a comparative analysis between *MaiBee* and Thug, one of the honeypots discussed in the state-of-the-art review in Section 1.2.

3.1 Architecture of *MaiBee*

As identified in Chapter 1, this thesis explores the concept of honeyclients through the development of a browser extension named *MaiBee*. Unlike traditional client-side honeypots, which require emulating a system or browser to interact with malicious servers, *MaiBee* operates directly within the user’s browser environment.

Moreover, the extension focuses on ads, leveraging their widespread presence on the internet as potential vectors for malicious activity. The architecture of *MaiBee* is divided into two main modules: the user interface and the back-end.

3.1.1 User Interface

The user interface provides interactive options to control and utilise the extension's features, mostly in the form of buttons. The key components are:

- **Pop-up Menu (popup.html).** This menu allows users to activate or deactivate the extension's functionality. It includes buttons to start the extension in the current tab ("BEES"), stop it ("REMOVE BEES"), and apply it across all open tabs ("BEES in All Tabs"). This menu communicates the chosen options to the back-end's pop-up JavaScript.
- **Options Page (options.html).** This page allows users to visualise the data collected by the extension. Thus, it facilitates the download of the *results* in JSON format and the *shady urls* as a .txt. It also shows in real time the ads and their corresponding parent URL that have been found and are being analysed and the option to remove the detected URLs from storage. This menu communicates the actions to the options JavaScript.

3.1.2 Back-End

The back-end is responsible for the core processing tasks of the extension. It consists of several interconnected scripts and components:

- **Browser Action (popup.js).** Manages the button listeners and actions triggered from the pop-up menu. It controls the activation and deactivation of the background and content scripts for the different tabs.
- **Background Script (detect.js).** Operates continuously in the background, listening for browser activity from the designated tabs and detecting ads using EasyList.
- **Options Page Script (options.js).** Handles the button actions and triggers the analysis of the detected URLs.
- **Content Scripts (content.js and beesify.js).** Detects ad elements and modifies the page layout. The script content.js detects the ad selectors and tries to find URL sources in them that are different from the ones found in the background script. At the same time, beesify.js offers a visual indication of the activation of the extension's functionalities, displaying images of the bees for five seconds.

extensions in its shop, unlike Chrome [33], where you have to register as a CWS developer and pay a one-off registration fee.

The core element of extensions is the `manifest.json` file. This **Manifest** file is the only file that must be present in every extension to define its behaviour and functionality. It contains metadata such as the name, version, description, icons, permissions, and pointers to other files in the extension. The manifest consists of mainly the component seen in Figure 3.2 [48].

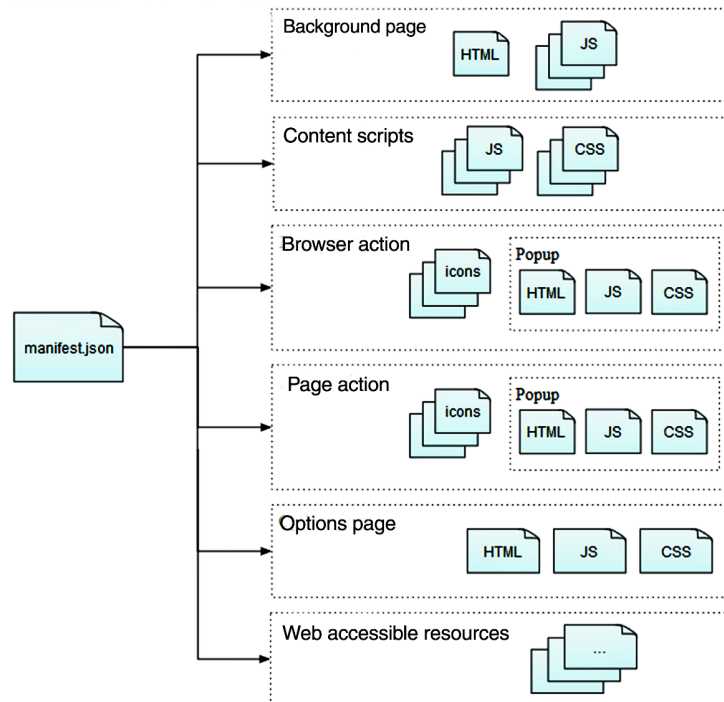


Figure 3.2: Overall anatomy of a web browser extension [58].

1. **Background scripts.** These scripts, as the name suggests, run in the background and have access to browser APIs but cannot directly access the content of web pages. They handle events in the browser, such as navigating to a new page or closing a tab; they also communicate with servers, manage data, and coordinate interactions between different parts of an extension. Background scripts can be persistent, making them useful for tasks that require continuous execution.
2. **Content scripts.** These scripts, as opposed to the background ones, execute within the context of a particular web page and can access their content through the DOM. This access makes it possible to manipulate the appearance and behaviour of web pages. Content scripts can also communicate

3.2. Design choices

with background scripts, thus indirectly accessing the WebExtension APIs, a cross-browser API for creating extensions.

3. **Browser actions.** These provide quick access to extension functionalities by adding buttons in the browser’s toolbar. They can display popup windows, execute JavaScript code, or trigger other actions when clicked by the user.
4. **Page actions.** These are similar to browser actions but are specific to the current web page, such as “bookmark the current tab”.
5. **Options page.** It allows users to configure extension settings and preferences, such as adjusting permissions, behaviour, or appearance.
6. **Web accessible resources.** It involves packaging resources like images, HTML, CSS, or JavaScript within the extension and making them available to web pages or other extensions.

Specifically to the development of *MaiBee*, the keys listed in Table 3.1 were utilised, apart from the “manifest_version”, “name” of the extension, “version” of the extension and “description”.

Key	Description
permissions	For <i>MaiBee</i> , access is needed to activeTab, all_urls, webRequest, webNavigation, storage, tabs, and dns.
browser_action	Used to add a button to the toolbar with which the user can interact. This activates a popup with buttons including the main functionalities of <i>MaiBee</i> .
icons	Specifies the icon of the extension.
background	In the case of <i>MaiBee</i> , the main functionality is coded in a background script. It reads and processes Easylist entries, listens for web requests and headers for checking matches and manages the state of ad detection in different tabs.
options_ui	Points to the HTML used to load the available options for the user. This HTML presents the options for exporting data gathered from the ad detection and analysis.
content_scripts	<i>MaiBee</i> includes two content scripts. One of them monitors and matches CSS selectors to detect specific ads and add functionality to distract from the honeypot. The other one adds a styling layer to the extension.
web_accessible_resources	An image is added as an accessible resource as it is needed for the styling layer.

Table 3.1: Key manifest elements used in *MaiBee*.

3.2.2 Deception Approach

Section 1.1 examines various honeyclients and their deception techniques designed to evade detection by the servers they investigate. This evasion is crucial because if malicious servers recognise they are being monitored, they may alter their behaviour or block the honeyclient. Consequently, honeypots must convincingly emulate a browser, simulating genuine user activity to avoid raising suspicion.

This thesis introduces a novel approach that diverges from traditional methods of deception. Instead of relying on complex emulations, we employ a browser extension to analyse sites through a legitimate browser, thereby reducing the likelihood of detection. While concerns exist that techniques similar to those used for detecting ad-blockers could be applied to identify this web extension, our research identifies three primary detection methods: changes in site layout (such as blocked ads), known extension IDs, and calls to well-known ad lists.

MaiBee, the browser extension developed for this study, addresses these deception methods as follows:

1. ***MaiBee* does not block ads**; it merely analyses them, thus maintaining the expected site layout. Unlike traditional ad blockers that remove ads from the web pages, our browser extension allows ads to be displayed as usual. This approach ensures that the site appears unchanged to both the user and the server, making it less likely to be detected.
2. ***MaiBee* is developed for Firefox**, which assigns random UUIDs to extensions [56], preventing detection through fetching the `web_accessible_resources` added to the `manifest.json` file. This randomisation is specific to each browser instance and differs from Chrome, where extensions can be detected via static internal UUIDs. This randomisation prevents websites from fingerprinting the browser by examining the extensions it has installed [89].
3. ***MaiBee* uses EasyList but incorporates measures to obscure its use**, evading detection based on well-known ad lists. While *MaiBee* leverages EasyList to identify ad-related content, it implements a simple yet effective technique to mask it: string obfuscation. Instead of directly referencing well-known strings, *MaiBee* concatenates parts of the string. This obfuscation makes it harder for malicious servers to detect the presence of EasyList, as the string is not directly visible in the source code. More sophisticated obfuscations, such as encoding, hashing, or using functions like `eval()`, were avoided to prevent raising suspicion and to maintain the transparency and accessibility of the code for developers.

Additionally, the extension is initially presented as a tool for placing virtual bees on the user's visited websites. This innocuous feature disguises its true purpose, with the detector activating only when the bees are enabled.

3.3. Extension’s functionalities

Another method to delay the detection of *MaiBee* involves obfuscating the JavaScript, making it more challenging for malicious servers to analyse. However, the JavaScript has been kept in a regular format to ensure developers interested in the project can understand it.

3.3 Extension’s functionalities

In this section, we delve into the multifaceted functionalities of *MaiBee*, designed to detect and analyse how suspicious online ads are. We present its capabilities: detecting ads, filtering suspicious characteristics such as the absence of trusted ad server providers or presence in malware databases, conducting DNS lookups and reverse lookups, analysing the Content Security Policy, and delving into VirusTotal reports. Each feature is designed to detect, filter and provide a better understanding of potential threats in ads.

3.3.1 Ad Detection

To address research question RQ3, outlined in Section 1.1, and capture server-side maliciousness, an ad detector is developed in the form of a web browser extension. Figure 3.3 illustrates the process of ad detection implemented in *MaiBee*

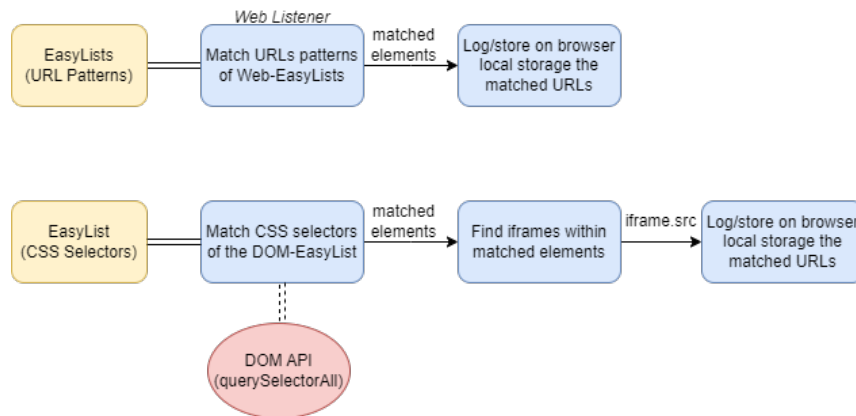


Figure 3.3: Overall functionality of the ad detection phase.

The ad detector comprises two independent modules:

1. **URL Request Detection Module:** This module uses a background script to monitor web requests and identify URLs matching ad-related patterns using EasyList rules. During initialisation, the module fetches and processes EasyList files from specified URLs, preparing a list of patterns stored in an array named `urls`. The background script runs persistently to ensure the detection

process is permanently active. When the extension is enabled for a tab, it activates a listener to inspect web requests using the `setupRequestListener` function. If a request URL matches one of the patterns in the `urls` list, the listener triggers the `alertURL` function to log the detected URL and store it for analysis.

To function effectively, the extension requires permissions in the `manifest.json` for `webRequest`, `activeTab`, `tabs`, and `storage`.

2. **DOM Element Detection Module:** This module identifies DOM elements on web pages that match parameters defined by EasyList. To identify sources that might have been overlooked by the URL request detection module, the related iframe of the detected element is inspected for the `src` attribute.

3.3.2 Filtering Suspicious Ads

An essential part of the ad detection process entails comparing the detected ad-related URLs against a whitelist containing domains of trusted ad server providers like Google, Amazon, Yahoo, and Meta [61]. These companies have been selected due to their strict ad policies and brand reputation. For example, Google requires that advertisers adhere to their ad policies [35] and applicable laws and regulations, ensuring ads are relevant and safe for users. Amazon Ads must also follow certain guidelines and acceptance policies [8], otherwise they may suspend or terminate the violating account. In the case of Yahoo, they ensure the parties involved, Yahoo and the advertisers, follow good and strict practices [88]. Meta and their advertising standards [45] promote transparency protecting users from, among others, fraud or harmful ads.

This comparison serves as our initial filter, distinguishing between legitimate and potentially suspicious ads. After the initial filtering process, URLs that do not meet the criteria are saved for further analysis and labelled as originating from “suspicious” ad providers. This updated list of ad-related URLs passes a secondary filter which involves a comparison with the URLhaus [1] database dedicated exclusively to malware. This step aims to identify and label any URLs associated with malware distribution.

3.3.3 DNS Lookup and Reverse Lookup

Domain Name System (DNS) is a fundamental feature of the Internet infrastructure in response to its continual growth and changes. The most widely known feature of DNS is the translation of domain names into IP addresses that computers use to identify each other on the network and vice versa. A DNS lookup helps identify the IP addresses associated with specific domains, while a reverse lookup determines domains hosted on a given IP address.

3.3. Extension's functionalities

Using these two concepts, we aim to detect patterns of malicious activity like domain flux or fast flux, which are often linked to distributing malware through ad networks or botnets.

Fast flux network is a tactic used by cybercriminals to hide malicious activities behind a constantly changing network of compromised computers (bots), where the domain names associated with malicious websites frequently change their IP addresses to avoid detection by using a network of bots as proxies, constantly swapping IP addresses while keeping the domain names static. This technique is designed to evade traditional security measures like IP blacklisting and make it difficult for authorities to take down malicious websites. [62]

Moreover, understanding the nature of IP-domain relationships is required due to the many-to-many associations between them. A single domain can map to multiple IP addresses, for example, when CDNs are being used to balance the traffic. Conversely, many domains can also share a single IP address, which is typical for shared hosting services.

As part of the analysis, the tool initially examines URLs to identify their corresponding IP address(es) through a DNS lookup. This is done thanks to JavaScript's `dns` module, which provides a straightforward way to resolve DNS queries. The IP addresses obtained are later used to perform a reverse lookup with Shodan InternetDB API [83]. This reverse lookup offers information about which services are being hosted and gives insights into related tags and CPEs that can help identify the system.

The analysis of this data can be complex; however, researchers can effectively identify malicious patterns by aggregating diverse data points.

For instance, consider a detected URL like `hxtps://aorta.clickagy.com/pixel.gif...`. This URL is associated with several IP addresses, as can be inspected in Listing 3.1. Each of these IPs falls within a distinct range, potentially raising suspicions. Nevertheless, a deeper investigation of the data retrieved from Shodan reveals that three IP addresses are linked to `amazonaws.com` and receive a "cloud" tag as expected. Based on this information, it can be deduced that the likelihood of the behaviour being malicious is low.

By analysing DNS lookups and reverse lookups results, patterns of potentially malicious activity can be discerned, helping to flag suspicious domains or IP addresses more effectively.

```
1 {
2   "parentURL": "https://info.contently.com/l/791483/2023-01-2
3     5/42rwn2?utm_source=website&utm_medium=homepage&
4     utm_campaign=2023-demo-request",
5   "detectedURL": "https://aorta.clickagy.com/pixel.gif?
6     clkgyv=pxl&ch=120&cm=",
7   ...
8   "ipAddresses": [
9     "44.213.166.231",
10    "3.224.185.20",
11    "34.237.212.171",
12    "35.171.209.41"
13  ],
14  "shodanResults": [
15    {
16      "ip": "44.213.166.231",
17      "hostnames": [],
18      "tags": [],
19      "cpes": []
20    },
21    {
22      "ip": "3.224.185.20",
23      "hostnames": [
24        "ec2-3-224-185-20.compute-1.amazonaws.com"
25      ],
26      "tags": [
27        "cloud"
28      ],
29      "cpes": []
30    }
31  ]
32  ...
33  ]
34  ...
35 }
```

Listing 3.1: Extract of JSON results for aorta.clickagy.com

3.3.4 Content Security Policy

As mentioned in Subsection 2.3.2, the CSP is a security mechanism that allows web developers to “detect and mitigate” various web-based attacks like XSS [49]. CSP allows developers to specify which content sources are considered trusted within their web applications. Doing so adds a robust second layer of defence against numerous vulnerabilities. Although CSP does not prevent web applications from

3.3. Extension's functionalities

containing inherent vulnerabilities, it significantly complicates an attacker's ability to exploit these weaknesses.

Developers can precisely define CSPs using over 20 directives. For instance, they can differentiate between image sources with the `img-src` directive and stylesheet sources with the `style-src` directive. The only mandatory directive is `default-src`, which sets the fallback source for most types of content.

Some special directive sources within CSP provide specific functionality [65] and are as follows.

- **'none'**. This directive disallows all sources, meaning no URLs match, effectively blocking any external content from being loaded.
- **'self'**. This directive allows resources to be loaded only from the origin site, matching the same scheme and port number as the site itself.
- **'unsafe-inline'**. This directive permits using inline source elements such as `style` attribute, `onclick`, or `script` tag bodies [19], which can introduce significant security vulnerabilities and is generally discouraged.
- **'unsafe-eval'**. This directive allows the usage of `eval()` in scripts, a practice that is typically avoided as it allows unsafe dynamic code evaluation such as JavaScript.

Typical examples of CSPs look like the following [49]:

- **Content-Security-Policy: default-src 'self'**. This directive ensures that all content comes from the site's origin.
- **Content-Security-Policy: default-src 'self'; img-src *; media-src example.org example.net; script-src userscripts.example.com**. This policy allows loading images from any source, media files from `example.org` and `example.net`, and executable scripts are only allowed from `userscripts.example.com`, while all other content must come from the site's origin.

If data is flagged as suspicious by other filters, the CSP of the parent URL or the visited website can help prevent potential attacks by restricting where resources can be loaded from. This proactive measure is crucial in maintaining web applications' integrity and security, especially in preventing XSS and other web-based attacks.

3.3.5 VirusTotal Analysis

To analyse content, VirusTotal [86] uses over 70 antivirus scanners and URL/domain blocklisting services, along with various tools to extract signals. VirusTotal

relies not only on signatures but also on its community, where users can contribute with comments and votes on whether particular content is harmful. Some suggestions by VirusTotal on comments ideas are describing malware propagation strategies, offering disinfection procedures, sharing reverse engineering reports, and notifying others of false positives [87].

Integrating the VirusTotal's API into our pipeline enriches the results obtained by thoroughly analysing the suspicious URLs flagged by our first filter. Their payloads are also analysed against a vast database of known threats, providing an additional layer of protection against suspicious or malicious advertisements.

The workflow followed in this step of our ad suspiciousness detection involves analysing through VirusTotal's API every detected URL flagged as suspicious by our extension MaiBee. It first requests the URL analysis report and saves meaningful fields of the response such as the *severity* value of the *crowdsourced_context* attribute, and the values of *last_analysis_stats* like *malicious*, *suspicious*, *harmless*, *undetected*, and *type-unsupported*. To make a request of the payload of each URL, we also save the value of *last_http_response_content_sha256* from the first response.

Having the hash value of the payload, we can request file reports to the API. In this case, the saved fields are the same as for the URL report, except for the severity field, which is not provided for this kind of analysis.

After doing some testing, we found that in many cases file reports provided more insightful responses, so when logging the final values, we kept the path that provides the highest summing of maliciousness and suspiciousness (either from the URL report or the file report).

3.4 Workflow in MaiBee

This section will discuss the flow of the present study as implemented in our web browser extension. The primary function of the extension is to detect and analyse advertisements, assessing their potential for suspicious activity. The diagram below (Figure 3.4) illustrates the workflow of our system:

3.4. Workflow in MaiBee

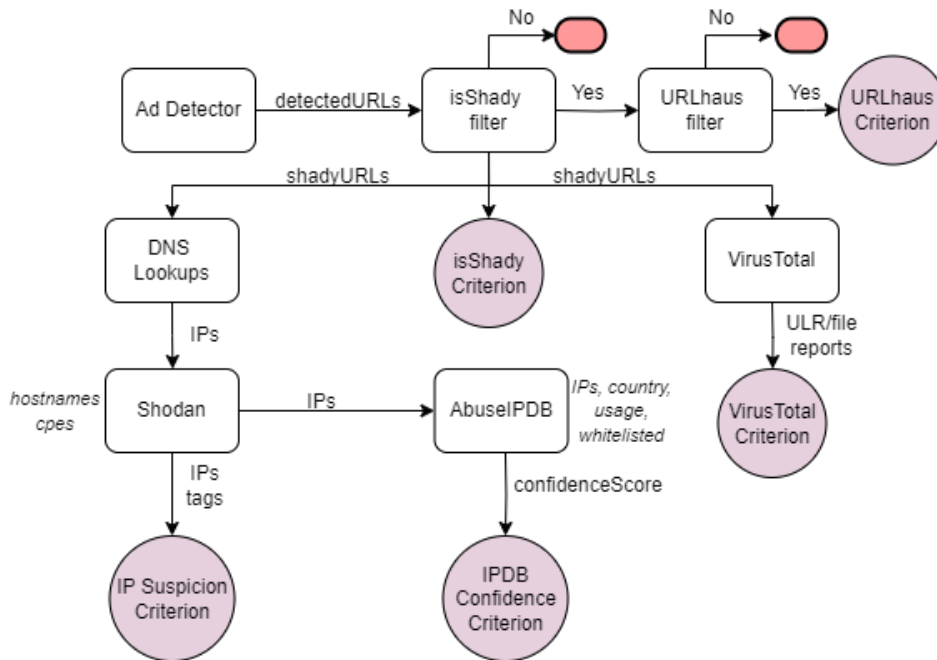


Figure 3.4: Overview of the workflow of *MaiBee*.

The flow begins with the **Ad Detector**, which identifies ad URLs and forwards them to the **isShady filter**. The role of this filter, as seen in Section 3.3.2, is to classify URLs based on a whitelist of ad server providers as potentially suspicious or benign.

If the URLs are flagged as suspicious, they are sent to several subsequent modules for further analysis. One of the modules is the **inInURLhaus filter**, which checks if the suspicious URLs match any entries in the URLhaus database. If a match is found, the URL is also classified under the “URLhaus Criterion,” which will be further explained in Section 3.6 together with the rest of the criteria. On the other hand, URLs that do not match this filter but are still considered suspicious proceed to the subsequent modules.

Suspicious URLs are also sent to VirusTotal, an external service that provides detailed reports on URLs and files, helping to validate their suspiciousness further. The information gathered in this step also contributes to the “VirusTotal Criterion”.

Another important module is the **DNS Lookup**, where URLs classified as “shady” are sent to extract their IP addresses. These IPs are then forwarded to the **Shodan** module for reverse DNS lookup and extracting relevant information. Shodan provides details about the IPs, including hostnames, CPEs, and tags. This information helps form the “IP Suspicion Criterion” and offers valuable insights for further analysis by security analysts.

The IP addresses are also sent to **AbuseIPDB**, a database that evaluates IP ad-

dresses for malicious activity and assigns an abuse confidence score. This contributes to the “IPDB Criterion”. Additional information this module provides includes the IP’s country of origin, usage, and whether it is whitelisted by them, offering further context for analysis.

Throughout the process, each module progressively refines the assessment of the detected ads, from initial detection to detailed analysis using multiple sources and criteria. This approach ensures a thorough evaluation, giving users and analysts information to determine whether the displayed web page ads are suspicious.

3.5 MaiBee Use Case

This section will demonstrate the practical use of the developed browser extension, specifically designed for Firefox. The first step is to install the extension, which can be done through the Firefox debugging interface. This can be accessed via the `about:debugging` page, where you navigate to “This Firefox” and select “Load Temporary Add-on”, as illustrated in Figure 3.5. Loading the extension can be done by selecting any of the files inside the extension’s directory or by choosing the packaged extension (.zip file). Once the extension is successfully loaded, it is ready for immediate use.

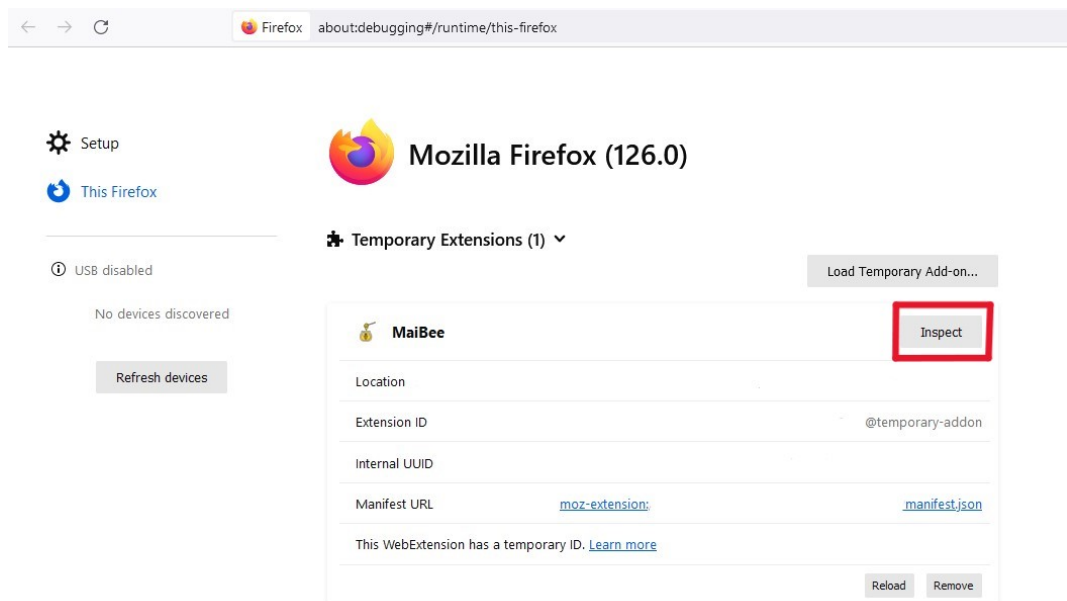


Figure 3.5: Extension already installed temporarily on Firefox.

To monitor *MaiBee*'s activity, the Developer Tools Toolbox can be accessed by clicking the “Inspect” button on the `about:debugging` page. This toolbox logs

3.5. MaiBee Use Case

messages from various extension scripts, including background scripts, the options page, the popup, and sidebars. These messages are displayed in the console, providing insight into the extension's operations. It is important to note that content script log messages appear in the developer tools of the specific tab where the content script is running, rather than in the *about: debugging* console.

Upon loading *MaiBee*, you will see three buttons in its popup: *BEES*, *REMOVE BEES*, and *BEES in All Tabs*, as illustrated in Figure 3.6. These functionalities are as follows:

- **BEES:** This functionality displays some images for five seconds to let the user know that the detection function is being activated for that tab, and in the background runs the ad detector for the current tab.
- **REMOVE BEES:** This functionality removes the images and stops the ad detector for the current tab.
- **BEES in All Tabs:** This functionality acts as the BEES button for all the opened tabs in the browser. It enables a faster analysis of the visited websites.

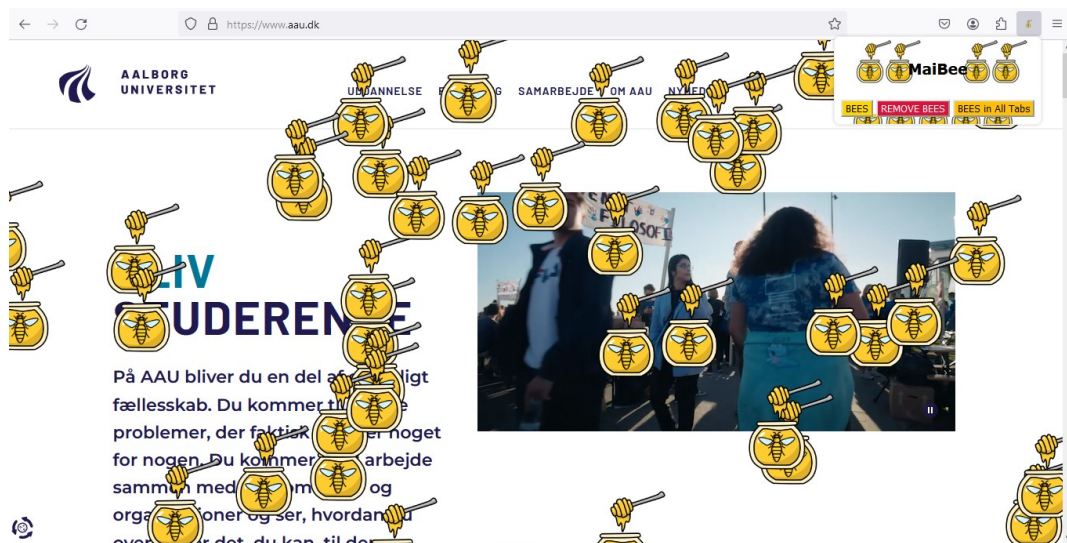


Figure 3.6: How *MaiBee* looks when running the ad detector (“BEES”)

After the desired pages have been inspected, the detected URLs are stored in the local storage of the browser as well as displayed along with their parent URL on the options page of the extension. This page can be accessed on *about: addons* by clicking on “Extensions”, then in the extension itself “*MaiBee*” and finally in the “Options” button.

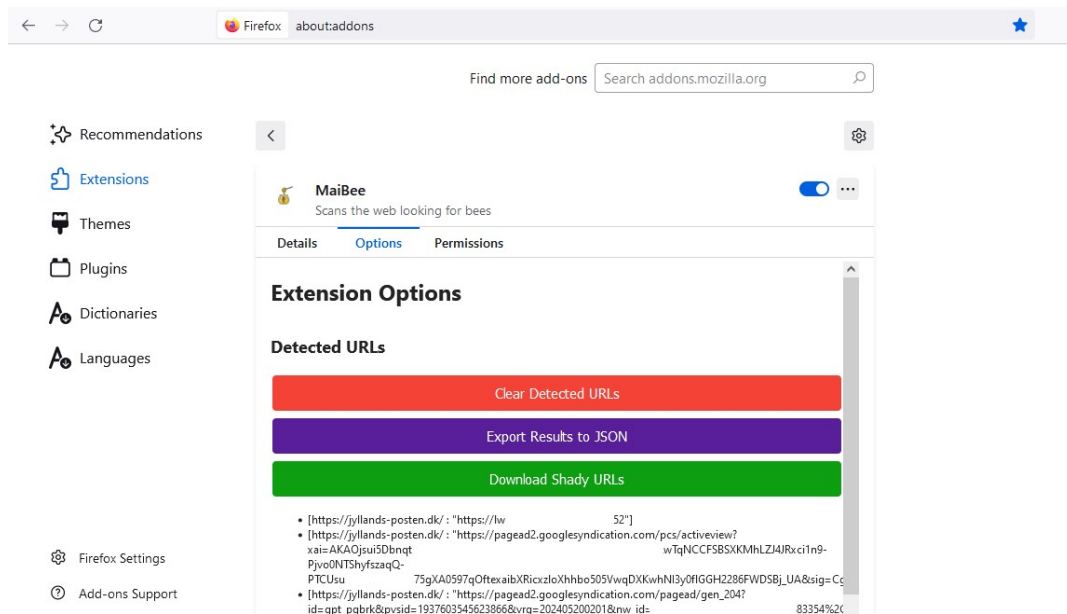


Figure 3.7: Options page of *MaiBee*.

As Figure 3.7 shows, the options page of *MaiBee* provides a user-friendly interface with three buttons to interact with: “Clear Detected URLs”, “Export Results to JSON” and “Download Shady URLs”. To avoid *MaiBee* exceeding the browser’s storage limit, the first button removes the detected URLs from it. As for the second button, it downloads a JSON file including all the relevant information related to the analysed detected ads. Finally, the third button downloads a text file including the detected “shady” URLs. An example of the keys of these results is as follows (Listing 3.2).

3.6. Weighted Scoring Model

```
1 [
2   {
3     "Summary": {
4       "totalParentURLs": ,
5       "totalAdDetectedURLs": ,
6       "totalShady": ,
7       "totalInUrlhaus": ,
8       "mostCommonIPs": [],
9       "mostCommonDomains": [],
10      "adsPerParentURL": [],
11      "cspSummary": {}
12    }
13  },
14  {
15    "parentURL": ,
16    "detectedURL": ,
17    "mdnClassification": {
18      "firstParty": [],
19      "thirdParty": []
20    },
21    "isShady": ,
22    "isInUrlhaus": ,
23    "ipAddresses": [],
24    "shodanResults": [],
25    "abuseIPDBData": [],
26    "cspHeader": ,
27    "cspHeaderValue":
28  },
29  ...
```

Listing 3.2: Overview of the final results JSON.

For a more detailed inspection of these results, Appendix A, B and C present an extract of the main analyses carried out in this thesis.

3.6 Weighted Scoring Model

In Section 3.3, the elements that the extension processes to generate the JSON output for each URL were explored. These elements include different criteria used to identify suspicious features in advertisements. While each element provides relevant data, the complexity and volume of this information can make it difficult for users of the extension to discern ads that may pose a security risk rapidly.

To improve the usability of this data and make it easier to interpret, the Weighted

Scoring Model (WSM) has been implemented. The WSM is a widely recognised model of multi-criteria decision-making used for evaluating, prioritising and selecting ideas, features, tools, or projects. Because the objective of this implementation is not to decide between ads, applying the WSM can effectively assess and score different criteria to provide valuable insights into suspicious advertisements. The following steps define the process for applying the weighted scoring model with the elements on the JSON [9]:

1. **Determine Criteria.** Identify the key criteria that are important for evaluating the advertisements in terms of suspiciousness.
2. **Assign Weights.** Assign weights to each criterion based on their importance. These weights reflect the significance of each factor in the overall assessment process, so they are dependent on each other.
3. **Develop a Scoring Rubric.** Create a scoring rubric to standardise the evaluation process and remove subjectivity. This rubric will define how each criterion is scored, independently of the others.
4. **Calculate the Weighted Scores.** Multiply the raw scores by their corresponding weights to obtain the weighted scores for each criterion for each advertisement.
5. **Calculate the Final Weighted Scores.** Add up the weighted scores for each advertisement to get a total score. This total score will indicate the level at which an ad is suspicious, with higher scores representing greater suspicion.

Several criteria are considered to ensure a comprehensive assessment when evaluating the suspicion of advertisements. Each criterion helps determine the overall risk associated with an ad. The following criteria are utilised in the evaluation process:

- **Shadiness (isShady).** Acting as the initial filter (explained in Section 3.3.2), this criterion determines whether an ad exhibits suspicious characteristics.
- **Appearance in URLhaus (isInURLhaus).** Serving as the secondary filter, this criterion assesses whether an ad URL is involved in malware distribution.
- **IPDB Confidence.** This criterion relies on the confidence score provided by AbuseIPDB [2], which, based on user reports, defines a rating of IP address in terms of maliciousness.
- **IP Suspicion.** This criterion evaluates the characteristics of the IPs extracted by Shodan from each suspicious detected URL. It includes the following sub-criteria: consistency of tags, geographical diversity and diversity of subnets.

3.7. Comparison with Thug

- **VirusTotal Reports.** Utilising analysis reports from VirusTotal, this criterion evaluates the potential threat posed by ads based on the detection results of over 70 antivirus scanners and URL/domain blocklisting services.

3.7 Comparison with Thug

This section provides a comprehensive overview of Thug's functionality and the specific parameters used during its operation. Thug is a Python-based low-interaction honeyclient (introduced in Section 1.2) designed to emulate web browsers and identify malicious content. In this thesis, Thug serves as a tool to compare with *MaiBee*, as this client honeypot aims to uncover malicious intent by simulating interactions with suspicious pages and thus tricking attackers into believing they are interacting with real users.

The specific parameters used when running Thug are as follows:

- **-u win10ie110.** Specifies the user agent as Mozilla version 5.0 running on the Windows 10 operating system, simulating a common browsing environment.
- **-n "output directory".** Sets the output directory for log files generated during the analysis, facilitating easy access and organisation of results.
- **-w 2000.** Defines a maximum delay of 2 seconds for methods such as `setTimeout` and `setInterval`, which introduce delays in executing functions. This setting optimises the analysis process for efficiency.
- **-v:** Enables verbose mode, providing detailed information about the execution process.
- **-F:** Activates text file logging mode, allowing for the generation of log files in text format for further analysis and review.
- **-Z:** Enables JSON logging mode, facilitating the creation of log files in JSON format for comprehensive analysis.

Upon completion of the analysis, Thug saves the information in a structured directory as shown in Figure 3.8. The files in this directory, except those in the "analysis" folder, are named using their MD5 hashes. These hashes can be cross-referenced with other tools, such as VirusTotal, to determine if they have been previously reported as malicious.

When using the JSON logging mode alongside file logging, both the analysis results and the downloaded resources are stored [12]. The JSON file of the analysis includes categories such as `awis`, `behavior`, `classifiers`, `code`, `connections`, `cookies`, `exploits`, `favicons`, `files`, `images`, `locations`, and `screenshots`. Table 3.2 provides a description of these fields [12].

```

malbeek@kali:~/Desktop/THUG_DETECTOR$ tree sitec
sitec
├── analysis
│   └── json
│       └── analysis.json
├── application
│   ├── javascript
│   │   ├── 5206989d4c2a22978f5e4c149942c16a
│   │   ├── 5236aa8b97987d2988a66da1dbacc239
│   │   └── d0ce4804a6c5858eacde1cb3da5e203d
│   ├── javascript: charset=UTF-8
│   │   ├── 07ccceb7ff11e2d2ebc59e88ba9fadf39
│   │   ├── 0bb5ceaf6c48b313be89547672bb1cd5
│   │   ├── 9e656f3afaf233290c0e2c06d5ea2543
│   │   └── d3b5336edaa0a53a3e0cd78066af3cdc
│   ├── octet-stream
│   │   ├── 09933dd22f64a19bd4faff78329139a5
│   │   ├── 5bf027ce612fdaf8e99f7c98ee41c1e
│   │   ├── 7d43a8166fe0254d8f02e47d0a996807
│   │   └── b57807860c1d1392528d015fd20d6ffa
│   └── x-javascript
│       └── c8c436ce448d743b9d2866a06b789b64
├── image
│   └── x-icon
│       └── 231067ac0dadd1f671a43e5e16cf9f55
└── text
    ├── css
    │   ├── 13f0d9f6eb57659dc6d861ab94fce23e
    │   └── 63712142376f518f39a3391fb0e2e996
    ├── html: charset=utf-8
    │   └── a1da6a56bbab30695ecf7fb253a80d05
    └── html: charset=UTF-8
        ├── dd324fda89aa7823d3e4f4aeca1bcde4
        └── df767d60fe93e530a90187258e95bbc5

13 directories, 19 files

```

Figure 3.8: Structure of the files provided by Thug.

Field	Description
awis	Used to store Alexa Web Information Service (AWIS) reports.
behaviour	Used to keep track of the suspicious and/or malicious behaviours observed during the analysis.
classifiers	Used to keep track of the Thug classifiers matches that fire during the analysis while visiting the URL.
code	Used to keep track of the (dynamic language) snippets of code identified during the analysis.
connections	Used to keep track of the redirections which could happen during the single analysis.
cookies	Used to store important information about the cookies of the URL analysed.
exploits	Used to keep track of the exploits which were successfully identified during the analysis while visiting the URL.
favicons	Used to store the dhashes of the favicons collected during the analysis.
files	Used to show each content downloaded during the analysis.
images	Used to store the results of the image processing analysis.
locations	Used to keep track of the content stored at each URL visited during the analysis.
screenshots	Used to store base-64 encoded screenshots (JPG format) of the analysed page.

Table 3.2: Explained fields provided in the JSON logging mode by Thug.

Chapter 4

Results

This chapter presents the results of our study on detecting suspicious ads using a client-side honeypot. We begin by detailing the data collection process, which involves gathering data from 10,000 websites and obtaining JavaScript files categorised into malicious and benign datasets. Next, we describe our methods for web navigation and the preliminary tests conducted to assess their performance and the impact of cookie acceptance.

In the data analysis section, we explore the characteristics of the detected ads and evaluate them using a WSM, considering various criteria such as not presence on a whitelist of ad server providers, URLhaus presence, IPDB confidence scores, suspicious characteristic of IPs, and VirusTotal reports. This provides a comprehensive view of the suspicious advertisement landscape.

4.1 Data Collection

The approach for data collection is divided into two different purposes. The first refers to the analysis of a large number of websites to conclude whether they contain suspicious ads. To do this, two different lists are utilised: the Cisco Umbrella Popularity List [17], and the Majestic Million dataset [44]. Cisco's list contains a vast collection of the most frequently accessed domains across the Umbrella global network. This network refers to more than 100 Billion daily requests from 65 million unique active users in more than 165 countries. Unlike conventional metrics like Alexa (deprecated), which focus primarily on browser-based HTTP requests, the Umbrella Popularity List takes into account the number of unique client IPs invoking a domain relative to the total sum of requests across all domains [17]. Similarly, Majestic offers a list of the top one million root domains, ranked by the number of unique referring subnets. This dataset utilises data from the *Fresh Index*, which is regularly updated with the latest information from web crawl [44]

The second purpose is to determine the maliciousness of the JavaScript files

embedded in the detected ads. For this, we collected both a malicious dataset and a benign dataset.

In terms of malicious samples, 38,738 JavaScript files (Table 4.1) were collected from HynekPetra [68] and MJDetector [63] dated from 2016 to 2018. We also fetched 4,353 JavaScript files by scraping URLs reported as phishing in 2024 on OpenPhish, a phishing intelligence platform [64].

Source	Number of JS
HynekPetra	38,443
OpenPhish	4,353
MJDetector	295
Total	43,091

Table 4.1: Overview of malicious JavaScript files.

As for benign samples, we collected JavaScript from the Majestic dataset. To retrieve enough data, the top 35k websites were selected. The JavaScript files were extracted by web scraping using BeautifulSoup [74], a Python library that creates a parse tree for documents, making extracting data from HTML easy. A total of 45,144 JavaScript files were retrieved, populating the benign dataset. As other studies did [31, 84, 14] (in their case with Alexa Top Sites, now deprecated), we assumed these websites to be trusted for the collection of JavaScript and as Fass et al. [31] explained, the method of parsing the web page statically to extract *script* and *src* tags, protects against dynamically generated elements and so presenting more confidence when labelling them as benign.

Both benign and malicious datasets contain files processed through obfuscation and minification. In benign code, these techniques protect code privacy and intellectual property, whereas, in malicious code, they evade static analysis and thus hide the malicious intent.

4.1.1 Web Navigation

As explained earlier, sufficient data for analysis of the current landscape of suspicious ads through the honeypot is collected following the Majestic dataset. To circumvent the need for manual execution, the present thesis analyses two options: incorporating custom code as a part of the extension that facilitates this work and using some popular tools.

Custom code can enable honeypot users to easily reproduce the data collection without the need to install code from external parties. The proposed approach is to add a supplementary button to the *MaiBee* pop-up to activate automatic search.

As showcased in Figure 4.1, in this approach, the underlying code receives a CSV file with the URLs that must be explored, and the availability of each is checked.

4.1. Data Collection

The accessible URLs are then loaded in a single tab. Every time a new site is loaded, a predefined waiting time is applied, after which the site is reloaded. After waiting for the predefined time again, the following site is loaded in the same tab.

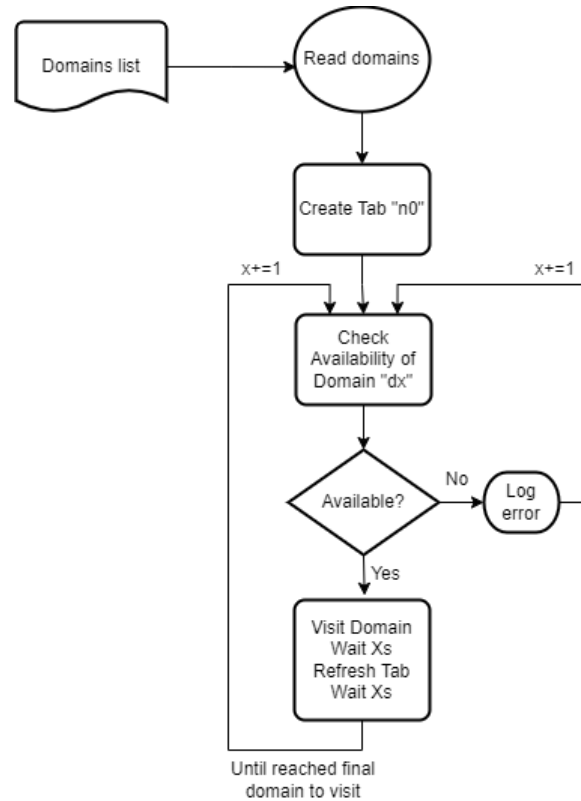


Figure 4.1: Overview of navigating multiple sites with an implementation inside *MaiBee*.

Alternatively, using existing tools to explore the dataset can provide data collection results that are more similar to those of a human visiting the sites. The tools typically used to navigate the Internet might have two purposes: web crawling or web scraping. The term web crawler applies to tools that visit web pages and analyse the related URLs present on the site. This is done, for example, by search engines like Google to index pages. Web scraper refers to those tools that visit sites and extract data from them through automation. It is usual to combine both concepts: first, by using crawling, a tool can crawl the URLs and then scrape data from those sites. Because of this, web scraping tools often offer crawling capabilities. Some widely recognised tools for the automation of web analysis are Selenium [81] and Scrapy [77].

- **Selenium** is an open-source suite to provide test environments for web-based applications. It supports all major browsers, including Firefox, and it is

formed by four components: Selenium IDE, Selenium RC, Selenium WebDriver, and Selenium Grid. From these, Selenium WebDriver, an API used to control the browsers “as a user would” [81], is often used for web scraping purposes. Each browser that Selenium supports has specific web driver implementations.

- **Scrapy** is an open-source framework designed initially for web scraping, but that can also be used as a general-purpose web crawler [77]. This tool is developed to parse static HTML content without needing a browser to maximise speed.

For the data collection using the browser extension - *MaiBee* - the more suitable solution among these last two options is using Selenium WebDriver, as, contrary to Scrapy, it can use instances of Firefox, where the extension can be loaded. As already mentioned, Selenium backs each browser with a specific driver. In the case of Firefox, this is called GeckoDriver [52], and it interacts with Marionette, which is “a remote protocol that lets out-of-process programs communicate with instrument and control Gecko-based browsers” [53], such as Firefox.

To support the use of Selenium for data collection, GeckoDriver and a Deb package version of Firefox must be installed. As opposed to the first approach of adding a supplementary button inside *MaiBee*, in this case, the implemented code (as shown in Figure 4.2) opens a Firefox instance with Selenium, loads *MaiBee* as a temporary extension, and opens a given number of tabs. In each of these tabs, a new site from the Majestic dataset is loaded. After this, Selenium scrolls down the page and moves on to repeat the same with the next tab. Once this is done for all specified tabs, there is a predefined waiting time, after which Selenium moves back to the first tab and starts reloading the content for each tab, waiting again in the last tab. Once this is done, the same process is repeated with the next set of sites.

4.1.2 Preliminary Tests

Following the implementation of the different web navigation options discussed in the previous section, conducting a preliminary test with a smaller dataset - specifically, the top 100 sites from the Cisco list - is deemed beneficial. This preliminary test serves two primary purposes:

1. Assess the performance of the various implementation options to select the most effective one for subsequent testing with Majestic’s dataset. Each approach has its advantages: incorporating a solution within the extension facilitates reproducing the data collection, whereas utilising Selenium offers a more human-like browsing experience, potentially influencing the results.

4.1. Data Collection

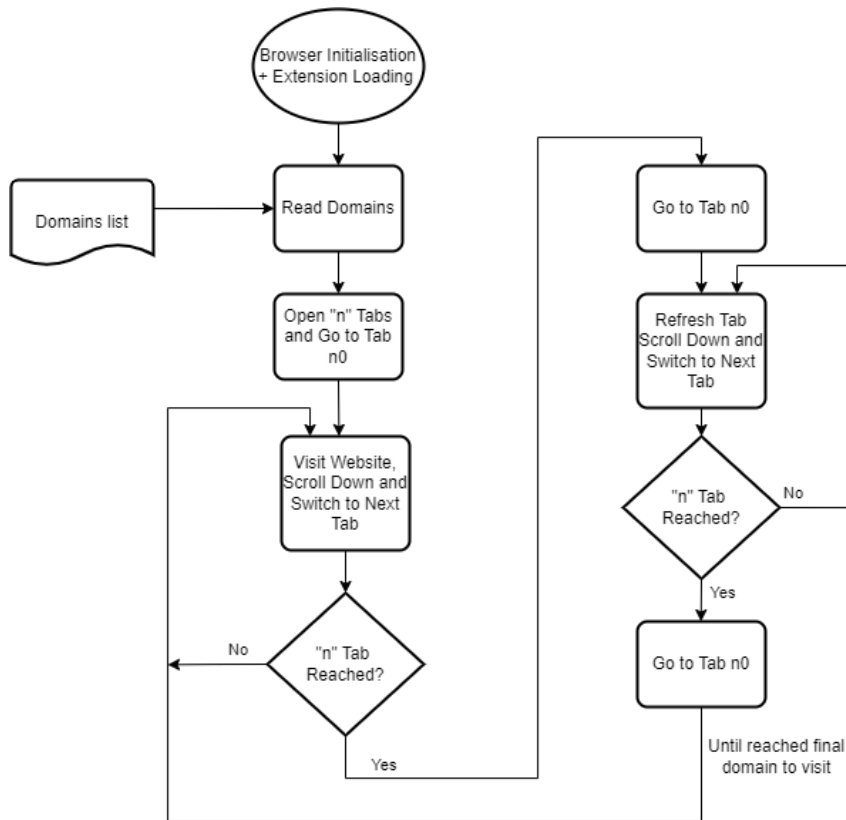


Figure 4.2: Overview of navigating multiple sites with Selenium.

2. Examine the impact of accepting cookie banners. As mentioned in Section 2.2, cookies are pivotal in the current online advertising ecosystem, tailoring ads to the user’s interests. Therefore, it is crucial to analyse the differences in data collection outcomes between accepting and not accepting cookies.

For the first objective, both approaches are implemented as previously described and illustrated in Figures 4.1 and 4.2. For the second objective, a Firefox add-on, *I don’t care about cookies* [21], which automatically accepts cookies, is utilised. Although there are multiple add-ons available to handle cookies, the most popular ones are focused on denying acceptance. Thus, the chosen add-on is the one that meets our requirements with the highest number of users.

The comparative results are presented in Table 4.2, categorised into four scenarios: Test 1 involves running Selenium without the cookie acceptance add-on, Test 2 entails running the code from within the extension without accepting cookies, Test 3 includes running Selenium with the automatic cookie acceptance add-on, and Test 4 involves running the internal code with the add-on installed.

	Test 1	Test 2	Test 3	Test 4
Number of detected parent URLs	12	12	33	36
Number of detected ad URLs	115	128	263	254
Number of suspicious ad URLs	54	66	118	112

Table 4.2: Relevant results after testing Selenium and internal browsing functionality against one hundred sites

To accurately interpret these results, it is essential to acknowledge that websites are dynamic and continuously evolving, which can cause slight variations in the number of detected ads over short periods. Therefore, the differences observed between *Test 1* and *Test 2*, and between *Test 3* and *Test 4*, are not considered significant. However, the substantial increase in detected advertisements when cookies are accepted validates the decision to use the external add-on for comprehensive data collection.

In conclusion, despite minor variations in the results, the enhanced capabilities of Selenium, particularly its ability to interact with websites in future developments, make it the preferred choice. Thus, the final data collection will be conducted using Selenium WebDriver, with automated cookie acceptance facilitated by the *I don't care about cookies* add-on.

4.2 Data Analysis

This section presents a detailed examination of the data collected using Selenium, which loaded the top 10,000 sites from the Majestic dataset. Following the data collection, we applied a scoring procedure for synthesising this data and identifying suspicious ads. The initial overview of our analysed data, as shown in Table 4.3 provides a foundation for a deeper exploration. The characteristics of the detected ads, their distribution, and the scores derived from our analysis will be examined in the subsequent sections to offer insights into the presence of suspicious ads.

4.2. Data Analysis

	Top 10k Majestic
Number of detected parent URLs	8,732
Number of detected ad URLs	44,610
Number of suspicious ad URLs	14,039
Top tags distribution (IPs)	cloud: 18041, eol-product: 14,421, cdn: 12,617
Top country distribution (IPs)	US: 16,308, NL: 11,069, DK: 9,677

Table 4.3: Relevant results from the analysis on the Top 10k Majestic.

The dataset includes the detection of 8,732 unique parent URLs and a total of 44,610 detected ad-related URLs. Out of these, 14,039 ads were flagged as suspicious by the initial filter. The top tags distribution among IPs reveals a significant presence of tags such as “cloud” (18,041), “eol-product” (14,421), and “cdn” (12,617). Additionally, the top country distribution for IPs indicates the highest number of IPs in the US (16,308), followed by the Netherlands (11,069) and Denmark (9,677).

In the subsections that follow, we will analyse these results in detail, exploring the implications of the detected patterns and scores. In particular, we will examine the characteristics of the suspicious ads, their geographical distribution, and other relevant metrics to provide a comprehensive understanding of the data and its significance in identifying ad suspicion.

4.2.1 Characterisation of The Results

One of the outcomes of this data analysis aims at assessing the levels of suspicion of the detected ads. This assessment complements the overall analysis that directly addresses the third research question (RQ3) presented in Section 1.1: “How could we capture such malicious attempts on the client side for comprehensive analysis?”

This classification is first based on a series of filters, like the comparison with a whitelist of trusted ad server providers or the comparison with the URLhaus database. As for URLhaus, our analysis has not found any matches between the detected ads and malware URLs. The following pie chart summarises the proportion of ad URLs considered suspicious (or “shady” as it is called in the browser extension).

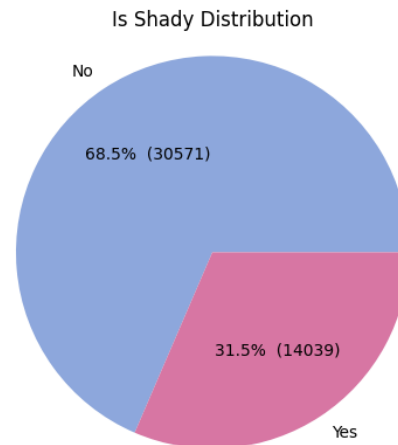


Figure 4.3: Proportion of suspicious URLs following the comparison with our Whitelist.

As we can see in Figure 4.3, 31.5% of the detected URLs by *MaiBee* do not match the whitelist of trusted ad providers, marked then as suspicious. This data is important as the following analyses are based solely on this percentage of suspicious ad URLs.

Another aspect of our analysis focuses on the distribution of tags assigned by Shodan for each IP. In the following graph (Figure 4.4), the distribution of the tags that appeared the most among the identified ads is illustrated. The top 5 tags in our results are: *cloud*, indicating that the IP is located in the network range of popular cloud hosting providers such as Amazon AWS, Microsoft Azure, and Google Cloud, which denotes a high level of trust. The tag *eol-product* indicates that the service is at the end of its life, no longer supported or maintained; thus, being susceptible to pose vulnerabilities, which is highly important to consider. *cdn* denotes content delivery networks used to distribute content, which might indicate that the purposes of these are legitimate; the tag *self-signed* is identifying services where the SSL/TLS certificate appears to be self-signed, this tag might indicate that the site associated with that IP is exposed to vulnerabilities and attacks such as Main-In-The-Middle or phishing schemes, as the communication is not secured. Lastly, the *database* tag indicates confirmed database instances.

4.2. Data Analysis

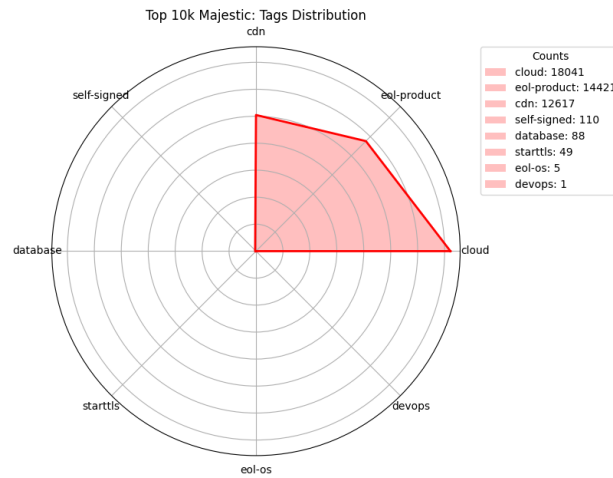


Figure 4.4: Overview of the tags discovered in the analysis.

Next, the geographical distribution of the IPs collected from the detected “shady” ads through AbuseIPDB is examined. The country distribution graph in Figure 4.5 shows the number of IPs associated with “shady” advertisements per country, with the top countries being the USA, the Netherlands, and Denmark. Various factors, including local internet regulations, the presence of cybercriminal groups, or the robustness of cyber security measures in place, could influence a higher prevalence of suspicious ads in certain regions. For instance, the high number of “shady” ads in the USA could be attributed to its large internet user base and the presence of numerous data centres and hosting services. The Netherlands, as it stands on several pages like [26, 22], is known to be a leading data centre hub across Europe, which might make it a target for hosting and distributing ads, both legitimate and malicious. The high prevalence in Denmark might be influenced by the fact that this analysis was conducted in Denmark. This could lead to a higher detection rate of local ads due to proximity and network routing efficiencies.

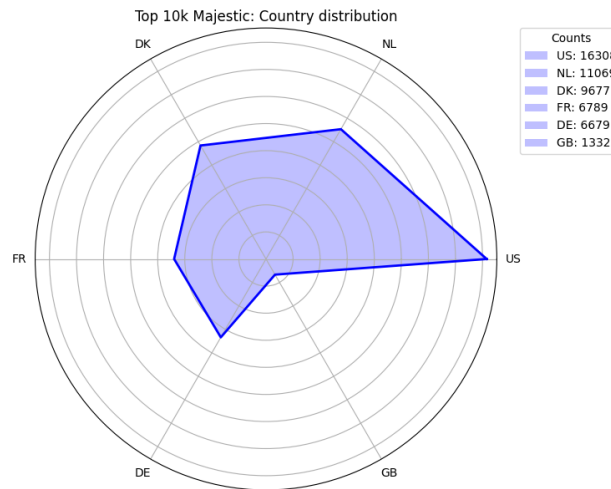


Figure 4.5: Overview of the countries discovered in the analysis.

4.2.2 Scoring of the Data

As defined in 3.6, a ranking done through the WSM depends mainly on three factors: the criteria, the weights given to each criterion, and the scoring rubric to analyse each criterion. The defined criteria for the ranking of suspicious ads are: *Shadiness*, *Appearance in URLhaus*, *IPDB Confidence*, *IP Suspicion*, *VirusTotal Reports*.

To synthesise the significant volume of data collected through navigating the web with Selenium, the weights on Table 4.4 have been proposed. The reasoning behind these weights is each criterion's reliability and significance in determining advertisements' overall suspicion. The **Shadiness** criterion has a low-medium weight. Although this criterion is used for the subsequent steps in identifying suspicious ads, we only include large companies with strict policies and great reputations in both the ad and non-ad domains in the whitelist. This limitation might imply not considering other reputable companies specialising in providing ad servers or typically benign companies that have been affected by malware.

In the case of **Appearance in URLhaus**, while it is effective at flagging malware-associated URLs, it primarily focuses on malware distribution and may miss other types of malvertising. Therefore, although it is a valuable tool, its scope of detection is limited.

IPDB Confidence also receives a medium weight because although it is a known tool for reporting IPs, it is based solely on user reports. This could mean that an IP could get falsely marked as suspicious through a coordinated attack.

As for the **IP Suspicion** criterion, it carries a lower weight because, as seen

4.2. Data Analysis

in Section 3.3.3, assessing the suspicion of IPs can be challenging due to factors such as the dynamic nature of IP addresses and the possibility of shared hosting environments where multiple websites share a single IP address. This complexity makes it difficult to draw definitive conclusions based solely on IP information.

Lastly, due to its extensive coverage, the **VirusTotal** criterion has the highest weight. It utilises over 70 antivirus scanners and URL/domain blocklisting services. This multitude of data sources ensures high reliability and accuracy in detecting malicious content, making it the most critical criterion in our evaluation process.

Criteria	Weight (0-100)
Shadiness	15
Appearance in URLhaus	20
IPDB Confidence	20
IP Suspicion	5
VirusTotal	40

Table 4.4: Weights for our given criteria.

As mentioned, an essential step in the Weighted Scoring Model is developing and applying a scoring rubric. Each criterion is assigned a score that reflects its potential risk level. The scores are then multiplied by their respective weights (Table 4.4) to calculate the total score for each advertisement. This total score helps us determine the likelihood of an ad being suspicious or malicious. Each score for the different criteria is given between 0 and 1, ensuring that the final weighted score is between 0 and 100, making it more direct to understand what every final weighted score means.

- For the **Shadiness** criterion, the score can be either 0 or 1. This criterion is the initial filter to determine whether an ad is suspicious. Therefore, if the detected URL is not flagged as “shady” by this filter, it receives a score of 0. If it is flagged, it directly receives a score of 1.
- The **Appearance in URLhaus** criterion indicates whether the URL is listed in the URLhaus database, which tracks URLs known for distributing malware. Thus, if the URL is not found in URLhaus, it receives a score of 0. If it is listed, it gets a score of 1.
- The **IPDB Confidence** criterion is based on the confidence score provided by AbuseIPDB. The score is calculated by parsing the results given by the API, which can vary between 0 and 100, to their equivalent between 0 and 1.
- The **IP Suspicion** score depends on three sub-criteria and combines data extracted from the IP lookup and Shodan’s reverse lookup.

1. The first sub-criterion corresponds to the nature of the tags found by Shodan; this is how consistent the tags are across the different IPs and if they include any information that can be used to raise suspicion, such as “self-signed”.
2. The second sub-criterion is the geographical diversity, doing a rough estimation based on the first /8 from the IPs.
3. The third sub-criterion examines the diversity of subnets, attempting to identify how likely these are to be part of a CDN, or to be part of a bots network. Each sub-criterion adds a number between zero and two to the suspicion calculus.

Additionally, a last check on the nature of the tags, looking for indicators of CDNs, can lower the calculus. The final number is normalised to represent a score between 0 and 1. It is worth mentioning that this scoring decision applies to cases with a minimum of 3 IPs, as no information can be drawn related to groups lower than this number.

- Finally, the **VirusTotal** score is derived from calculating malicious and suspicious votes on each URL divided by the total effective checks. This number is then passed by a method to calculate the scores as follows: if the VirusTotal score is between 0 and 20, the given score is 0.2; a score of 0.4 is given if the value is between 20 and 40; when the value is between 40 and 60, the score is 0.6; in the range of 60 and 80, the given score is 0.8; a score of 1 is given for values above 80, and if there is no available rating, the score is 0.

Table 4.5 shows a summary of the scores, where “VT” stands for calculating the rating from VirusTotal reports. Notably, the IPDB Confidence score does not appear in this table as it follows a different scoring rubric due to the nature of the abuse confidence score provided by AbuseIPDB. This score ranges from 0 to 100, which can be easily converted to fit within a 0 to 1 scale.

Criteria	0.2	0.4	0.6	0.8	1
IP Suspicion	IPs have mixed types without 'Unknown', 'Private', or 'Self-signed' tags.	Presence of 'Unknown', 'Private', or 'Self-signed' tags.	IPs have mixed types without 'Unknown', 'Private', or 'Self-signed' tags AND diverse first octets.	Presence of 'Unknown', 'Private', or 'Self-signed' tags AND each IP has a unique first octet.	Presence of 'Unknown', 'Private', or 'Self-signed' tags AND each IP has a unique subnet.
Shadiness	-	-	-	-	When the ad is flagged by this filter.
Appearance in URLhaus	-	-	-	-	When the ad is flagged by this filter.
VirusTotal	$0 < VT < 20$	$20 \leq VT < 40$	$40 \leq VT < 60$	$60 \leq VT < 80$	$VT \geq 80$

Table 4.5: Scoring rubric applied to our given criteria.

4.2. Data Analysis

4.2.3 Scoring Evaluation Results

After analysing the collected data, the weights and scoring rubric proposed in the previous section are applied. As a result, the ads are ranked according to their suspicion level. This analysis not only highlights the ads flagged as suspicious by the isShady filter but also provides insights into the effectiveness of different detection criteria.

	URLHaus (%)	VirusTotal (%)	IPDB Confidence (%)	IP Suspicion (%)	Any criteria (%)
Shady sites with score higher than 0	0	0.65	6.67	40.95	44.20

Table 4.6: Percentage of sites that were marked as *Shady* by the initial filter that received scores from each criterion

Table 4.6 presents the percentage of sites flagged as “shady” by the initial filter that receives scores from each criterion. Notably, while no sites are flagged by URLhaus, a significant 44.20% of the ads receive some assessment by the remaining criteria, indicating the presence of potentially suspicious activity.

The **VirusTotal** criterion, which flagged only 0.65% of the ads, suggests that only a small fraction of the “shady” ads are also flagged as suspicious to some extent according to this criterion. Furthermore, as illustrated in Table 4.7, the highest score assigned by VirusTotal is 8 out of a maximum of 40 (as per the weights in Table 4.4). This suggests that VirusTotal’s scanning mechanisms are effective but stringent, maintaining high thresholds for flagging content.

The 6.67% score from **IPDB Confidence** reflects this tool’s capability to detect suspicious IP addresses but also highlights the need for complementary tools to cover a broader spectrum of threats. As shown in Table 4.7, the highest score for IPDB is 3.4, while the lowest score is 0.2 with an average score of 1.76 out of a maximum possible score of 20. This low-scoring range indicates that this criterion is useful but might be conservative or require more corroborative evidence from other criteria.

The 40.95% score for **IP Suspicion** reveals that over half of the sites flagged by the initial filter exhibited suspicious IP characteristics. As shown in Table 4.7, the highest score for this criterion is 4.00, and the lowest is 1.00, with an average score of 1.28 out of a maximum possible score of 5.00. This high percentage, compared with other criteria, underscores the importance of IP analysis in the detection process. It captures a wide range of potentially malicious activities that other criteria might not catch.

According to the insights already presented, Table 4.7 provides a detailed breakdown of the highest, lowest, and average scores for VirusTotal, IPDB Confidence, IP Suspicion, and Final Score. The maximum possible scores for each significant

criterion are 40 for VirusTotal, 20 for IPDB Confidence, and 5 for IP Suspicion. Analyzing these scores offers valuable insights into the detention capabilities of each criterion.

	Highest Score	Lowest Score	Average Score	Maximum Possible Score
VirusTotal	8.00	8.00	8.00	40
IPDB Confidence	3.40	0.20	1.76	20
IP Suspicion	4.00	1.00	1.28	5
Final Score	26.40	15.00	15.69	100

Table 4.7: Highest, lowest, and average scores for each criterion.

Overall, the scoring analysis demonstrates the effectiveness of the Weighted Scoring Model in evaluating the suspicion level of ads. The significant percentages and scores indicate that a combination of criteria, rather than reliance on a single source, is crucial for the comprehensive detection of potentially malicious advertisements.

Chapter 5

Discussion

In the discussion chapter, the obtained results are interpreted. We start with an overview of the final results, summarising the general findings and highlighting the top three most suspicious sites identified. This is followed by a detailed examination of the limitations encountered during the study. Additionally, we examine the effectiveness of detecting malicious JavaScript, presenting a proof of concept using JStap and Thug. This discussion aims to contextualise our findings within the broader cyber security landscape.

5.1 Overview of Final Results

This section discusses the final results obtained from *MaiBee*, our honeyclient browser extension designed to detect and analyse suspicious ads. The tool is tested on the top 10,000 domains of the Majestic dataset, evaluating the presence and characteristics of ads using multiple criteria. The primary objective of this section is to study potentially harmful ads and assess the overall effectiveness of the detection mechanism.

The analysis identified 14,039 “shady” ad-related URLs out of 44,610 detected ones (Table 4.3). Among these ads classified as suspicious by our first filter, 44.20% (6,205 ads) met at least one additional scoring criterion as developed in Section 4.2.3. This means that 13.90% of all detected ads from the top 10,000 sites are classified as suspicious, based on being flagged by at least two of our criteria. The variance in the levels of suspicion highlights the different techniques and methods used by potentially malicious actors in the realm of online advertisements. By employing a multi-criteria approach, *MaiBee* can detect ads that exhibit behaviours indicative of tracking, data collection, and other potentially suspicious activities. This section provides an overview of the general findings and a detailed examination of the top three most suspicious sites.

5.1.1 General Findings

Upon completion of the analysis, the WSM is employed to rank the ads based on their suspicion levels. The key metrics considered include URLhaus presence, VirusTotal reports, IPDB Confidence scores, and IP Suspicion. As highlighted in Section 4.2.3 the findings reveal varying degrees of suspicious activities across the analysed websites, with significant disparities in how each criterion flagged the ads.

An important observation is that the top three most suspicious sites explained in the following Section 5.1.2, are not strictly the top three highest-scoring sites identified by *MaiBee*. This is because the sites ranked as second (`hxxps://id5-sync.com/g/v2/369.json`), third (`hxxps://id5-sync.com/g/v2/692.json`), and fourth (`hxxps://id5-sync.com/g/v2/617.json`) are structurally very similar to the top-ranked site (`hxxps://id5-sync.com/g/v2/445.json`) and share the same scores. To provide a more diverse set of case studies, we present the analysis of the most suspicious sites that exhibit different behaviours and structures.

The analysis of these sites aims to provide valuable insights into the types of suspicious behaviour that can be detected using *MaiBee* and a multi-criteria scoring approach. The findings underscore the importance of vigilance and the need for continuous monitoring to protect users from potentially harmful ads.

5.1.2 Top 3 Most Suspicious Sites

The top three sites ranked as the most suspicious provide insightful case studies into the types of malicious behaviours that our tool aims to detect. These sites were flagged based on high aggregated scores from the weighted criteria, indicating substantial evidence of potentially malicious activities. Below, we provide a detailed analysis of each site, with the corresponding extracts of the JSON file detailed in Appendix A, Appendix B and Appendix C, respectively.

SiteA

Site A (`hxxps://id5-sync.com/g/v2/445.json`), which extract of the JSON results can be found in Appendix A, was identified across three websites: `apnews.com`, `dictionary.com`, and `irishnews.com`, ranked in the Majestic dataset as shown in Figure 5.1

Rank on Majestic	Website
349	<code>apnews.com</code>
601	<code>dictionary.com</code>
9,719	<code>irishnews.com</code>

Table 5.1: Ranking position on the Majestic dataset of affected domains by Site A.

5.1. Overview of Final Results

The site's overall performance in our analysis accumulated a final score of 26.4%. The following findings are presented by breaking down this score (illustrated in Table 5.2).

Criteria	Assessed Score
isShady	15
isInURLhaus	0
IPDB Confidence	2.4
IP Suspicion	1
VirusTotal	8
Final Score	26.4

Table 5.2: Final scores for Site A.

- The detected URL resolves to multiple IP addresses, primarily within the range of 141.95.33.120 to 162.19.138.120. These IP addresses are associated with the hostnames `id5-sync.com` and various subdomains, indicating a large network of servers. According to the Shodan analysis, these IPs do not have specific tags, indicating a lack of detailed information on their exact configurations.
- As **AbuseIPDB** reports, all IPs are hosted in the same country and receive abuse confidence score values ranging from 2 to 12 out of 100, with a majority being 12 and none being whitelisted. All the IPs are categorised under "Data Center/Web Hosting/Transit", which may serve as infrastructure for both legitimate and malicious purposes.
- The **IP Suspicion** score for Site A is 1 out of 5. This low score suggests minimal suspicion due to the lack of tags. Additionally, the fact that there are only two groups with the same initial octets and that over half of the IPs are within the same subnet further reduces the suspicion level for this criterion.
- The **CSP** headers for Site A exhibit varying degrees of permissiveness. The highlights are for `apnews.com` and `dictionary.com`, which CSP directives allow loading resources from any source (*) for default, script, connect, img, and style directives. Additionally, 'unsafe-inline' and 'unsafe-eval' are permitted for script-src and style-src, which allows inline JavaScript the use of `eval()`, potentially increasing the risk of XSS attacks. On the contrary, `irishnews.com` implements a highly restrictive CSP. It disallows loading any scripts (script-src 'none') and objects (object-src 'none') from external sources, essentially preventing the execution of any dynamic content or scripts from non-whitelisted sources. Despite the variation in the CSP headers, ranging

from highly permissive settings allowing all sources to restrictive policies that prevent loading external scripts and objects, the presence of the ad itself raises concerns. Such configurations might be a tactic to obfuscate the ad’s behaviour and avoid detection.

- Finally, the **VirusTotal** score for this site was 8 out of 40, indicating that only one out of 93 antivirus engines flagged the URL as malicious. This suggests that although VirusTotal detects some risk, it does not classify it as a high one.

When attempting to visit the URL, the server returns an error: “Required request body is missing”, suggesting that the endpoint expects a structured request body to process a POST request. This URL is not designed to be accessed directly on the browser but is meant for API calls, including a well-defined request body. Site A could be used for collecting or processing user data, which aligns with the given classification as a tracking ad.

In conclusion, Site A stands out as particularly suspicious due to its role as a tracking ad and the error message indicating a required request body. Combined with the varied CSPs and moderate VirusTotal score, these factors suggest that this URL is part of a complex tracking and data collection system that operates beyond basic web interactions. This makes it a significant point of concern in the context of ad tracking and potential user privacy violations.

Site B

Site B (<https://creativecommons.com/cm-notify?pi=taboola&tc=1>) was identified 81 times across similar parent URLs with structure: <https://am-match.taboola.com/sync?dast=V9...>. The site’s overall performance in our analysis yielded a final score of 25.8%. The following findings are presented by breaking down this score (illustrated in Table 5.3).

Criteria	Assessed Score
isShady	15
inInURLhaus	0
IPDB Confidence	2.8
IP Suspicion	0
VirusTotal	8
Final Score	25.8

Table 5.3: Final scores for Site B.

- **AbuseIPDB** data reports that the IP address associated with the ad URL, 185.184.8.90, has an abuse confidence score of 14. This suggests moderate

5.1. Overview of Final Results

reports of malicious activity. While Shodan cannot assign any tag to this IP, AbuseIPDB defines its usage type as “commercial” purposes, which can serve both legitimate and malicious activities.

- The **IP Suspicion** score for Site B is 0, indicating a lack of information due to the only IP assigned. This score suggests no direct evidence of malicious use from the IP itself.
- Regarding the **CSP** header value for Site B is extensive and includes a wide range of directives and allowed sources. Notably, it includes “data: ‘unsafe-inline’ ‘unsafe-hashes’ ‘unsafe-eval’”, serving as exceptions to the default policy (“default-src self”) by allowing data: URIs, inline scripts and styles (‘unsafe-inline’), inline hashes (‘unsafe-hashes’), and the use of eval() (‘unsafe-eval’). Allowing inline scripts and eval() poses security risks as attackers can exploit them to execute malicious code. Following the default-src directive, various specific domains are listed, each prefixed with (*) denoting a broad range of subdomains. These include domains associated with ad services, analytics, social media platforms, content delivery networks, and various other third-party services. Allowing such a wide range of external sources can enhance website functionality but also increase the attack surface and potential security risks.
- Finally, **VirusTotal** flagged the URL with a score of 8 out of 40, indicating that only one out of 87 antivirus engines detected the URL as potentially malicious. This suggests limited recognition as malicious, but still as it is flagged, it suggests being cautious.

In conclusion, Site B exhibits several indicators of potentially suspicious activity, notably through its integration with third-party tracking systems and specific security configurations. While its final score of 25.8% does not denote an immediate high risk, the presence of various red flags, particularly its tracking functionality and moderate abuse reports by AbuseIPDB, justify further scrutiny and monitoring.

Site C

Site C (hxxps://google-bidout-d.openx.net/w/1.0/pd?p1m=5) was identified across eight high-traffic websites: lavanguardia.com, antena3.com, la-croix.com, sport.es, winfuture.de, jeuxvideo.com, heraldo.es, and marieclaire.fr, ranked in the Majestic dataset as shown in Figure 5.4.

Rank on Majestic	Website
1,439	lavanguardia.com
4,735	la-croix.com
6,907	sport.es
7,078	antena3.com
8,981	winfuture.de
9,096	jeuxvideo.com
9,669	heraldo.es
9,721	marieclaire.fr

Table 5.4: Ranking position on the Majestic dataset of affected domains by Site C.

The site’s overall performance in our analysis yielded a final score of 25.4%, closely mirroring the risk level identified in Site B. By breaking down this score (illustrated in Figure 5.5), the following findings are presented.

Criteria	Assessed Score
isShady	15
inInURLhaus	0
IPDB Confidence	2.4
IP Suspicion	0
VirusTotal	8
Final Score	25.4

Table 5.5: Final scores for Site C.

- **AbuseIPDB** data for Site C reveals that the associated IP addresses had abuse confidence scores of 11 and 12 out of 100, respectively. The first IP address, 35.244.159.8, was reported 53 times for various malicious activities, including port scans, brute force attacks, exploited hosts, and hacking attempts. The second IP address, 34.98.64.218, was reported 55 times for similar activities. Both IPs are classified as belonging to data centres or web hosting services, which may serve as infrastructure for both legitimate and malicious purposes.
- The **IP Suspicion** score for Site C was 0 out of 5. This score reflects the lack of information due to only being associated with two IPs and being identified as “cloud” by Shodan.
- Despite the restrictive **CSP** header, which was set to “script-src ‘none’; object-src ‘none’”, the presence of the ad itself raises concerns. Such restrictive policies are often used to prevent loading external scripts and objects, which might be a tactic to obfuscate the ad’s behaviour and avoid detection.

5.2. Limitations

- **VirusTotal** flagged the URL with a score of 8 out of a possible 40, indicating that one out of 92 antivirus engines detected the URL as malicious. This suggests a moderate level of risk associated with the ad, as detected by VirusTotal’s extensive database.

A visit to the site confirmed its suspicious nature, as it is flagged by an adblocker extension, Malwarebytes, which marked it as “adware” as seen in Figure 5.1. This external validation aligns with our findings and reinforces the need for vigilance against such ads.

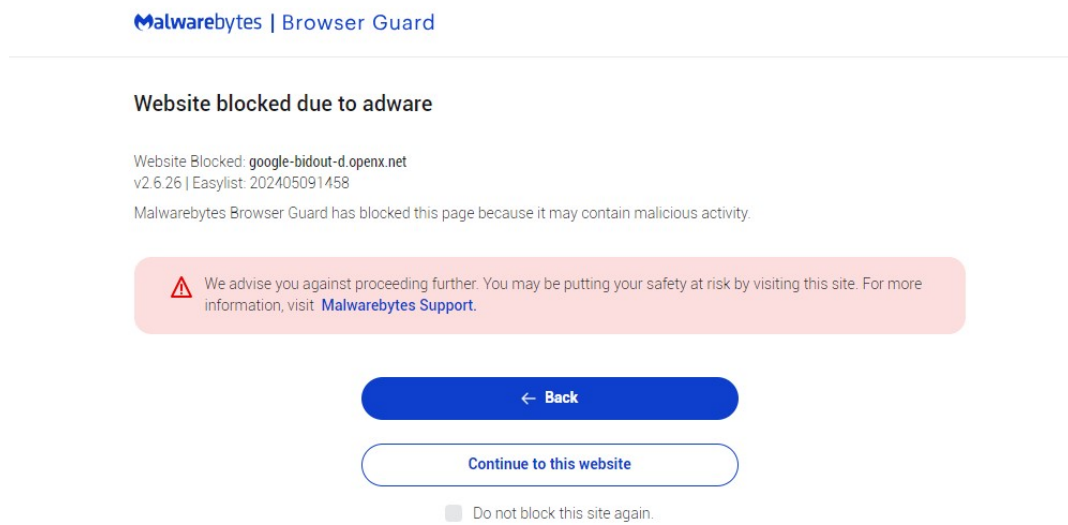


Figure 5.1: Malwarebytes Ad blocker showing adware detection on Site C.

In conclusion, Site C exhibits some suspicious activity indicators, including VirusTotal and AbuseIPDB reports. This highlights the importance of a systematic and multi-criteria approach to assessing suspicion of advertisements.

5.2 Limitations

While this study provides valuable insights into the methodology for classifying suspicious ads and presents the results derived from analysing 10,000 sites, several limitations must be acknowledged. These limitations, both influenced by external and internal factors, will be reviewed in the following subsections.

5.2.1 Browser Extension

The client-side honeypot presented in this paper, *MaiBee*, is designed as a Firefox extension. As discussed in Section 1.3 and Chapter 3, the decision to implement

the honeypot as a browser extension stemmed from the aim to explore innovative techniques for monitoring malicious behaviour. Nevertheless, as the development of the honeypot progressed, several limitations associated with this choice have emerged. While some of these limitations, such as the incompatibility of certain methods with browsers other than Firefox, have not directly impacted our current work, others have posed significant challenges.

In-browser Limitations

Section 2.3.3 explored the main ideas behind browser capabilities, security, and web extensions specifically. As mentioned, browsers employ multiple security strategies in terms of the communication they allow and their structural design. To ensure user safety, browsers sandbox the content of loaded sites in the renderer, while plugins operate in their own separate processes. Consequently, web extensions are not intended to have unrestricted access to the JavaScript present on loaded sites. Relating distinct pieces of code with the detected advertisement could give better insights into potentially malicious behaviour to the client-side honeypot.

Storage Limitations

To enable users to download the information found by the background and content scripts, the `storage.local` API methods [57] were utilised. However, in the case of Firefox, this storage is limited to 5 MiB, preventing the extension from running for over-extended periods. It is worth noting that this storage limitation does not apply to all data the user sees; information from VirusTotal, Shodan, IPDB, and IP extraction occurs outside of these scripts. Therefore, the limitation only affects the storage of the comparison whitelist of ad server providers, detected URLs, shady URLs, URLhaus data, and CSP data. Storage constraints would not be an issue if the honeypot ran outside the browser.

5.2.2 Selenium

As explained in Section 4.1.1, the extension was run against a portion of Majestic's dataset using Selenium's web crawling capabilities to check the status of suspicious ads on the web. However, several unforeseen challenges were encountered during its implementation.

Interactivity with Browser Extensions

The decision to use Selenium for data analysis was based on its ability to interact with websites, providing an experience comparable to a user interacting with the

5.2. Limitations

content. However, this interactivity does not extend to interactions with browser extensions. While Selenium can automatically load extensions in Firefox, including temporary ones like *MaiBee*, it cannot natively interact with these extensions [82]. Accordingly, interaction with the main functionalities - such as the buttons "Bees", clearing storage, and downloading content - had to be done manually. This manual interaction introduces some limitations typical of human involvement. For instance, although both downloading and clearing of stored information were done carefully, it is possible that minimal data was lost in the process as Selenium continued to load sites during that time.

Unexpected Errors and Troubleshooting

The use of Selenium, combined with the necessity of manual interaction with the extension, resulted in several errors during data collection. It was observed that interacting with the "Clear Detected URLs" button would occasionally disrupt the normal loading process for sites in the tabs. The cause of this issue remains unresolved, necessitating the interruption of Selenium's operation and its subsequent restart, resuming from the last successfully processed site. This method, however, may have introduced contamination into the original results.

Additionally, there were instances where Firefox would suddenly crash, causing Selenium to print an error related to the unexpected closure of the browser. The underlying reason for these sudden closures of Firefox remains unknown, and the only remedy was to employ the previously mentioned imperfect method of restarting the process from the last recorded site.

Detection of Non-human Behaviour

Due to Selenium's widespread use, some websites implement detection techniques to prevent bot usage or site scraping. These techniques can identify automated behaviours or signatures and block access accordingly. While various workarounds circumvent this detection, such methods are often neither straightforward to implement nor completely effective. One employed approach was to use non-headless browsers to mimic normal human browsing. However, this strategy was unsuccessful, as some sites still managed to detect and block the crawler. Thus, data could not be successfully extracted for all visited sites.

5.2.3 Usage of APIs

Different databases were used to collect information related to the detected ad URLs: URLhaus, Shodan InternetDB API, AbuseIPDB, and VirusTotal. In the case of the latter two, a paid subscription is required to use their full capabilities. Since the person running the extension is the one to provide the API keys, it is on their

side to choose a plan. In the case of the AbuseIPDB API, the free version only covers up to 1000 requests per day, while VirusTotal allows 4 requests a minute, or 500 requests a day.

It was possible to use a free trial of AbuseIPDB to navigate the first 10,000 sites in Majestic’s dataset, but this was not the case with VirusTotal. Consequently, VirusTotal needed to be externalised from the browser extension, which added a considerable amount of time to the site analysis.

5.3 Malicious JavaScript: Proof of Concept

In line with our study, another thread to explore is the analysis of JavaScript files originating from the detected advertisements. Our investigation begins with an exploration of JStap, a modular static JavaScript detection system distinguished for its ability to differentiate between malicious and benign samples. In the subsequent section, we delve into the analysis results conducted by Thug, comparing its effectiveness with our browser extension and addressing any limitations encountered during our testing process.

5.3.1 JStap

JStap, as their authors defined in [30], is “a modular static JavaScript detection system” capable of distinguishing between malicious and benign samples. It is a promising tool for achieving our objective. Therefore, we decided to train a model with our specific needs.

One of the primary motivations for creating a new model was that JStap’s original model did not account for phishing samples and relied on outdated data. Therefore, specific data collection was conducted for this purpose, as explained in Section 4.1. When testing this model, whose proportions are found in Table 5.7, the results were notably different as reported by the authors. As shown in Table 5.6, our experiments revealed a True Positive Rate (TPR) of 36.4% and a True Negative Rate (TNR) of 98.7%, which means that as the True Positive (TP) indicates, only 28 of the 79 malicious samples were correctly classified as malicious. This indicates a high capability to detect benign samples but a significantly low performance in identifying malicious ones.

5.3. Malicious JavaScript: Proof of Concept

	JStap model + JStap samples	MaiBee model + MaiBee samples	MaiBee model + JStap test samples
TP	977	28	733
FP	8	1	7
TN	992	77	993
FN	23	49	267
TPR	97.7%	36.4%	73.3%
TNR	99.8%	98.7%	99.4%

Table 5.6: Confusion matrix of JStap’s experiments.

Further testing involved using our novel model with JStap’s test data, which resulted in improved rates: TPR of 73.3% and TNR of 99.4%. Despite the improvements, JStap’s original performance was not met. This discrepancy highlights a critical insight: while developing a new model with contemporary and different data such as phishing samples is a promising approach, the size of our dataset and selection significantly impact the outcomes. JStap’s model benefits from over 100,000 samples, suggesting that the volume and diversity of training data play a crucial role in achieving high detection rates.

MaiBee samples (PDGs)	Training	Validation	Testing
Benign	15.676	7.804	78
Malicious	15.676	7.804	77

Table 5.7: Datasets size used for JStap experiments.

In conclusion, this part of the thesis serves as a proof of concept due to the constraints of time and data availability. Although the custom model did not fully reach the efficacy of JStap’s reported results, it demonstrates potential as an effective method for detecting malicious JavaScript files. This capability is particularly relevant for our project focused on identifying suspicious ads, indicating that the approach could be highly effective with more extensive and well-curated datasets.

5.3.2 Thug Discussion

The following section provides an overview of the analysis results conducted by Thug, comparing its effectiveness with our browser extension. Additionally, it addresses any limitations that were encountered during our testing process.

To further classify potentially malicious JavaScript and enhance our understanding of suspicious ads, Thug was selected as a complementary method of the assessment that *MaiBee* implements. Although Thug is designed to mimic the actions of

5.3. Malicious JavaScript: Proof of Concept

```
r c=null!=a?a:{pvsid:yk(window),Ma:"m202405220101",qf:"202405220101",Da:new Aw(3,"n202405220101",0),Ch:10,sg:1};try{oc(func
tion(ha){em(c,1190,ha)});var d=Wo();kf(!.zh(Rm).g);_y(Object,"assign").call(Object,Sm,d._vars_);d._vars_=Sm;if(d.evalScri
pts).d.evalScripts();else{HK();try{f1()}catch(ha){em(c,408,ha)}sr();var e=new g0;try{bl(e.I),jo(13,c),jo(3,c)}catch(ha){em(c
,408,ha)}var f=zw(c,e),g=null!=a?a:Dw(f,c),h=null!=b?b:new f0(g);Wl(g);$p("gpt_fifwin",function(ha){pp(ha,g)},d.fifwin?0:
0);var k=new HL,l=new MS(k,e),n=new vR(g),n=_.Tm(260),p=new wL(g),r=new wL(g),u=new wL(g),v=_.Tm(150),z=sB(),x=(_.I(KK)?Vu:
Uu)(g,window,l,on()),h,k,n,e,m,p,v),C=_.Tm(221),E=new rQ,D=new hP,L,P,R,aa=null!=(R=null==(L=x.Lb)?void 0:null==(P=L.Qd)?voi
d 0:P.Jb)?R:new yq,fa=new zS(g,l,h,k,n,z,e,p,n,C,E,D,x,aa);_I(oH)&&new 00(g,p,k,l);var ea=on().g;Bu(g,h,fa,ea,l,r,u,e,D,aa
);jp(g,d,h);window.setTimeout(function(){for(var ha=window.document.scripts,ua=0,sa=0,Da=0;Da<ha.length;Da++){ha[Da].src.mat
ch("securepubads.g.doubleclick.net/tag/js/gpt.js")?ua++;ha[Da].src.match("www.googletagservices.com/tag/js/gpt.js")&&sa++;1
<ua&&0==sa||1<sa&&0==ua?0(h,$M()):0<sa&&0<ua&&h.error(aN()),1E3);hu();if(!.I(oH)||.zh(cm).g)rw(),vw();mp(g)}catch(ha){
em(c,106,ha)}});_05=_.t.requestAnimationFrame|_.t.webkitRequestAnimationFrame;_P5=!(!_05&&!/'iPhone'/test(_.t.navigat
or.userAgent);_Q5=function(a,b,c){_W.call(this);var d=this;this.j=a;this.l=b;this.g=c;this.U=null;_Np(this,function(){re
turn d.U=null});_U(_Q5,_W);}).call(this,{});}
[2024-05-24 17:02:19] [Timer] Scheduler error:
[2024-05-24 17:02:19] [document.write] Deobfuscated argument: <script src="https://securepubads.g.doubleclick.net/pagead/ma
naged/js/gpt/m202405220101/pubads_impl.js" id="gpt-impl-0.3586153714904239"></script>
[2024-05-24 17:02:19] [script src redirection] https://motor-cdn.prensaiberica.es/widget?portal=sp&pagina=portada&v1 -> htt
ps://securepubads.g.doubleclick.net/pagead/managed/js/gpt/m202405220101/pubads_impl.js
[2024-05-24 17:02:19] [HTTP] URL: https://securepubads.g.doubleclick.net/pagead/managed/js/gpt/m202405220101/pubads_impl.js
(Status: 200, Referer: https://motor-cdn.prensaiberica.es/widget?portal=sp&pagina=portada&v1)
[2024-05-24 17:02:19] [HTTP] URL: https://securepubads.g.doubleclick.net/pagead/managed/js/gpt/m202405220101/pubads_impl.js
(Content-type: text/javascript; charset=UTF-8, MD5: 6d282a1e65cc9b56f4ce1d5a91d66322)
[2024-05-24 17:02:20] ActiveXObject: microsoft.xmlhttp
```

Figure 5.3: Terminal view of one of the tests conducted by Thug (part 2).

Another attempt at using Thug led us to run the honeyclient against the ad URL. Yet the results were even smaller, as it could only provide general information about that URL as seen in Listing 5.1.

```
1
2
3 "locations": [
4   {
5     "url": "https://google-bidout-d.openx.net/w/1.0/pd?
        plm=5",
6     "content": "<html><head><title>Pixels</title></head>
        <body><script>if(\\\"browsingTopics\\\"in document
        &&document.featurePolicy.allowsFeature(\\\"
        browsing-topics\\\"))document.browsingTopics()</
        script></body></html>",
7     "status": 200,
8     "content-type": "text/html",
9     "md5": "34216f487875f43b5fdd24c077eec1f2",
10    "sha256": "de259eb7ba7a0e45575deb33946f1fbc695c97c3
        3145ae4e49af0069d010868e",
11    "ssdeep": "6:qFzLIMQHXLxkGX16DVQ8ouqxdLaYwrtL1
        wVVIBq4QL:2ebxkC6DVU4dxhwVVI/QL",
12    "flags": {},
13    "size": 199,
14    "mimetype": "text/html"
15  }
16 ]
```

Listing 5.1: JSON extract of Thug analysis on ad suspicious URL.

This observation underscores our tool's effectiveness while highlighting areas where Thug could improve to offer a more comprehensive analysis. Although Thug generates detailed logs, it does not add significant value in the context of our specific needs: detecting potentially suspicious ads.

The limitations observed with Thug in the context of analysing ad URLs reflect the inherent challenges in detecting sophisticated malicious activities. Overall, while Thug remains a promising tool for dynamic analysis, our findings suggest that our developed browser extension is more capable of detecting suspicious ads. This emphasises the need for continuous improvement and integration of multiple analytical approaches in cyber security.

Chapter 6

Conclusion

In this thesis, we addressed the problem of malicious advertisements exploiting online users by developing a client honeypot system in the form of a browser extension. Three key research questions guided this study:

- **RQ1:** What strategies are used by malicious servers in online advertising today?
- **RQ2:** What are the attack vectors originating from the above strategies, and what is the impact on clients?
- **RQ3:** How could we capture such malicious attempts on the client side for comprehensive analysis?

Regarding **RQ1** and **RQ2**, our investigation examines malicious actors who compromise the user's CIA, for example, by stealing sensitive information or disrupting the website host's capacity to handle visitors. This is achieved through techniques such as Cross-Site Scripting, Cross-Site Request Forgery, injection attacks, and DDoS.

Our study revealed that malicious ads leverage multiple attack vectors, including drive-by downloads, social engineering, and phishing. These vectors execute specific attacks like click fraud, badvertising, and other exploits. These attacks can lead to severe consequences for users, including unauthorised access to sensitive information, financial loss, and system compromise by inadvertently installing malware. By dissecting these vectors, we presented malicious ads' impact on end-users.

To address the practical aspect of capturing and analysing malicious attempts on the client side (**RQ3**), we developed *MaiBee*, a Firefox browser extension functioning as a client honeypot. We demonstrated that *MaiBee* effectively detects and analyses ads, assessing if they are suspicious based on multiple criteria, such as URLhaus presence, IPDB confidence scores, VirusTotal reports, and suspicious

characteristics of IPs. This approach allowed us to capture and explore suspicious attempts related to online advertising, providing a novel approach to identifying harmful ads.

Among the main findings of this thesis, our analysis of the top 10,000 domains from the Majestic dataset identified 44,610 ad-related URLs, of which 14,039 were flagged as “shady” by an initial filter based on a whitelist of trusted ad server providers. Furthermore, 44.20% (6,205 ads) of these “shady” ads were flagged by at least one additional criterion for assessing suspiciousness. This means that *MaiBee* detects and assesses that 13.90% of all the detected ads of the top 10,000 visited are classified as suspicious based on being flagged by at least two of our criteria. Among these ads, behaviours such as ad tracking, data collection and other potentially suspicious activities were exhibited.

6.1 Future Work

This thesis has provided a thorough study of suspicious ads, however, the limitations presented in Section 5.2 require further investigation. In this section, we suggest areas for future research to address them and explore new questions.

- **EasyList:** Throughout this thesis, EasyList and its variants have been referenced for filtering online advertisements. However, their use has been rather basic and could be improved in several ways:
 1. Allow users to select their countries of interest to access information from relevant lists. Currently, this selection is hardcoded in the application.
 2. Enhance list usage by creating dictionaries for search effectiveness.
 3. Reduce the number of requests to the lists. Presently, the extension calls EasyList to download information each time it is loaded. Since these lists have a defined lifetime, implementing caching mechanisms should be considered to retain information.
- **Maliciousness analysis:** As discussed in Section 5.2, one challenge of using an add-on as a honeypot is the difficulty in investigating potentially malicious JavaScript from the extension. Some options could be researched to address this issue.
 1. Wrappers: In JavaScript, a wrapper is a function designed to call other functions, either for convenience or to modify their behaviour [54]. Some browser add-ons, such as JShelter [40], use wrappers to safeguard users’ security.

6.1. Future Work

It is worth investigating whether a method can be developed to identify which JavaScript code belongs to ads when suspicious functions are called. This approach could help distinguish and analyse potentially malicious scripts embedded within advertisements.

2. **Mutation Observers:** Mutation Observers can detect changes in the DOM tree and have been used to study browser extensions that inject ads into websites [90]. Exploring the possibility of identifying the scripts causing these changes could allow for individual analysis of potentially malicious scripts outside the extension.
- **Selenium Interaction Level:** Future work can improve Selenium’s web automation by better mimicking user behaviour. Introducing randomised delays between actions and incorporating sophisticated mouse movements can make interactions appear more human-like. This could produce more realistic results and help evade website detection mechanisms that block automated traffic.
 - **Suspicious ads categorisation:** It could be studied how to further categorise ads marked as suspicious in terms of the attack vectors they use. This could be done based on the JavaScript content or by exploring external databases.

Appendix A

Suspicious Site A

```
1 {
2   "parentURL": "https://apnews.com/",
3   "detectedURL": "https://id5-sync.com/g/v2/445.json",
4   "mdnClassification": {
5     "firstParty": [],
6     "thirdParty": [
7       "tracking_ad",
8       "any_basic_tracking",
9       "any_strict_tracking"
10    ]
11  },
12  "isShady": "Yes",
13  "isInUrlhaus": "No",
14  "ipAddresses": [
15    "162.19.138.119",
16    "141.95.33.120",
17    "141.95.98.64",
18    "162.19.138.120",
19    "162.19.138.118",
20    "141.95.98.65",
21    "162.19.138.117",
22    "162.19.138.82",
23    "162.19.138.83",
24    "162.19.138.116"
25  ],
26  "shodanResults": [
27    {
28      "ip": "162.19.138.119",
29      "hostnames": [
30        "id5-sync.com",
31        "ns31533570.ip-162-19-138.eu"
32      ]
33    }
34  ]
35 }
```

```

33     "tags": [],
34     "cpes": []
35 },
36 {
37     "ip": "141.95.33.120",
38     "hostnames": [
39         "id5-sync.com",
40         "ns3203256.ip-141-95-33.eu"
41     ],
42     "tags": [],
43     "cpes": []
44 },
45 {
46     "ip": "141.95.98.64",
47     "hostnames": [
48         "id5-sync.com",
49         "ns3216658.ip-141-95-98.eu"
50     ],
51     "tags": [],
52     "cpes": []
53 },
54 {
55     "ip": "162.19.138.120",
56     "hostnames": [
57         "ns31533571.ip-162-19-138.eu",
58         "id5-sync.com"
59     ],
60     "tags": [],
61     "cpes": []
62 },
63 {
64     "ip": "162.19.138.118",
65     "hostnames": [
66         "id5-sync.com",
67         "ns31533569.ip-162-19-138.eu"
68     ],
69     "tags": [],
70     "cpes": []
71 },
72 {
73     "ip": "141.95.98.65",
74     "hostnames": [
75         "ns3216659.ip-141-95-98.eu",
76         "eu-1-id5-sync.com"
77     ],
78     "tags": [],
79     "cpes": []

```

```

80     },
81     {
82         "ip": "162.19.138.117",
83         "hostnames": [
84             "ns31533568.ip-162-19-138.eu",
85             "id5-sync.com"
86         ],
87         "tags": [],
88         "cpes": []
89     },
90     {
91         "ip": "162.19.138.82",
92         "hostnames": [
93             "eu-1-id5-sync.com",
94             "ns31532337.ip-162-19-138.eu"
95         ],
96         "tags": [],
97         "cpes": []
98     },
99     {
100        "ip": "162.19.138.83",
101        "hostnames": [
102            "ns31532338.ip-162-19-138.eu",
103            "id5-sync.com"
104        ],
105        "tags": [],
106        "cpes": []
107    },
108    {
109        "ip": "162.19.138.116",
110        "hostnames": [
111            "ns31533567.ip-162-19-138.eu",
112            "id5-sync.com"
113        ],
114        "tags": [],
115        "cpes": []
116    }
117 ],
118 "abuseIPDBData": [
119     {
120         "ipAddress": "162.19.138.119",
121         "countryCode": "FR",
122         "isWhitelisted": false,
123         "abuseConfidenceScore": 10,
124         "usageType": "Data Center/Web Hosting/Transit"
125     },
126     {

```

```
127     "ipAddress": "141.95.33.120",
128     "countryCode": "FR",
129     "isWhitelisted": false,
130     "abuseConfidenceScore": 12,
131     "usageType": "Data Center/Web Hosting/Transit"
132   },
133   {
134     "ipAddress": "141.95.98.64",
135     "countryCode": "FR",
136     "isWhitelisted": false,
137     "abuseConfidenceScore": 12,
138     "usageType": "Data Center/Web Hosting/Transit"
139   },
140   {
141     "ipAddress": "162.19.138.120",
142     "countryCode": "FR",
143     "isWhitelisted": false,
144     "abuseConfidenceScore": 10,
145     "usageType": "Data Center/Web Hosting/Transit"
146   },
147   {
148     "ipAddress": "162.19.138.118",
149     "countryCode": "FR",
150     "isWhitelisted": false,
151     "abuseConfidenceScore": 2,
152     "usageType": "Data Center/Web Hosting/Transit"
153   },
154   {
155     "ipAddress": "141.95.98.65",
156     "countryCode": "FR",
157     "isWhitelisted": false,
158     "abuseConfidenceScore": 12,
159     "usageType": "Data Center/Web Hosting/Transit"
160   },
161   {
162     "ipAddress": "162.19.138.117",
163     "countryCode": "FR",
164     "isWhitelisted": false,
165     "abuseConfidenceScore": 12,
166     "usageType": "Data Center/Web Hosting/Transit"
167   },
168   {
169     "ipAddress": "162.19.138.82",
170     "countryCode": "FR",
171     "isWhitelisted": false,
172     "abuseConfidenceScore": 10,
173     "usageType": "Data Center/Web Hosting/Transit"
```

```

174     },
175     {
176         "ipAddress": "162.19.138.83",
177         "countryCode": "FR",
178         "isWhitelisted": false,
179         "abuseConfidenceScore": 12,
180         "usageType": "Data Center/Web Hosting/Transit"
181     },
182     {
183         "ipAddress": "162.19.138.116",
184         "countryCode": "FR",
185         "isWhitelisted": false,
186         "abuseConfidenceScore": 10,
187         "usageType": "Data Center/Web Hosting/Transit"
188     }
189 ],
190 "cspHeader": "Content-Security-Policy",
191 "cspHeaderValue": "default-src * 'unsafe-inline' 'unsafe-
eval'; script-src * 'unsafe-inline' 'unsafe-eval';
connect-src * 'unsafe-inline'; img-src * data: blob: '
unsafe-inline'; frame-src *; style-src * 'unsafe-inline
';",
192 "VirusTotal": {
193     "detectedURL": "https://id5-sync.com/g/v2/445.json",
194     "PayloadSHA256": "2e5b50c64fcc5f4e0ad3e6cdf3d28655a271de2
8f234b26f365ebf6b1a4a0f48",
195     "attributes": {
196         "crowdsourced_context": {
197             "severity": "null"
198         },
199         "last_analysis_stats": {
200             "malicious": "1",
201             "suspicious": "0",
202             "undetected": "9",
203             "total_checks": "93"
204         },
205         "popular_threat_classification": {
206             "popular_threat_category": "",
207             "suggested_threat_label": "null"
208         }
209     }
210 }
211 }

```

Listing A.1: Most suspicious ad based on our criteria.

Appendix B

Suspicious Site B

```
1 {
2   "parentURL": "https://am-match.taboola.com/sync?dast=V9cQ
3     ...",
4   "detectedURL": "https://creativecdn.com/cm-notify?pi=
5     taboola&tc=1",
6   "mdnClassification": {
7     "firstParty": [],
8     "thirdParty": [
9       "tracking_ad",
10      "any_basic_tracking",
11      "any_strict_tracking"
12    ]
13  },
14  "isShady": "Yes",
15  "isInUrlhaus": "No",
16  "ipAddresses": [
17    "185.184.8.90"
18  ],
19  "shodanResults": [
20    {
21      "ip": "185.184.8.90",
22      "hostnames": [
23        "ip-185-184-8-90.rtbhouse.net",
24        "adscdn.com",
25        "www.adscdn.com"
26      ],
27      "tags": [],
28      "cpes": []
29    }
30  ],
31  "abuseIPDBData": [
32    {
```

```

31     "countryCode": "NL",
32     "isWhitelisted": false,
33     "abuseConfidenceScore": 14,
34     "usageType": "Commercial"
35   }
36 ],
37 "cspHeader": "content-security-policy",
38 "cspHeaderValue": "default-src 'self' data: 'unsafe-inline'
    'unsafe-hashe' 'unsafe-eval' getcody.ai trinketsofcody
    .com *.adsecurity.com *.qbigads.com *.mitgame.com *.
    mobmio.com *.univibes.ru *.admitad-connect.com *.bing.
    com *.clarity.ms *.ttwstatic.com *.w.org *.tapaffiliate.
    com *.convertsocial.net *.qbigtech.com *.admitad.ru *.
    stage.monetize *.tinkoff.ru *.smartredirect.de mtusgate.
    de linkitten.com mtusing.de convertlink.com pmf.tech *.
    pmf.tech fairsavings.com *.fairsavings.com *.admitad.com
    *.admit.ad *.admitad.academy mitgo.com *.mitgo.com
    takeads.com *.takeads.com univibes.org *.univibes.org *.
    ads-twitter.com *.trustpilot.com *.zopim.io *.zopim.com
    *.smooch.io *.zdassets.com *.zendesk.com *.
    consentmanager.net *.mindbox.cloud *.popmechanic.ru *.
    gravatar.com *.facebook.net *.facebook.com *.fb.com *.
    consensu.org *.amazonaws.com *.twitter.com *.instagram.
    com *.tiktok.com *.webvisor.org *.quizyworld.tech *.
    linkedin.com *.ampproject.org yastatic.net *.yandex.com
    *.yandex.net *.yandex.ru *.ya.ru *.mail.ru vk.com *.
    scriptcdn.net *.typekit.net *.google.net *.google.io *.
    google.eu *.google.su *.gooogle.com *.gogle.com *.com.
    google *.google.be *.google.by *.google.ca *.google.cn
    *.g.cn *.google.dk *.google.ee *.google.fi *.google.gg
    *.google.gr *.google.hr *.google.hu *.google.is *.google
    .it *.google.co.jp *.google.kz *.google.lv *.google.md
    *.google.com.mx *.google.nl *.google.pl *.google.pt *.
    google.ro *.google.rs *.google.ru *.google.se *.google.
    com.ua *.google.ua *.google.us *.google.co.uz *.google.
    de *.google.cz *.google.at *.google.ae *.google.com.co
    *.google.com.do *.google.jo *.google.gl *.google.sc *.
    google.co.ve *.google.com.uv *.google.co.ao *.google.co.
    in *.google.bg *.google.com *.googleapis.com *.translate
    .goog *.gstatic.com *.google.co.uk *.google.com.tr *.
    google.fr *.google.es *.google.com.eg *.google.com.cy *.
    google.ge *.google.co.id *.googleusercontent.com *.
    googletagmanager.com *.google-analytics.com *.adwords.
    com *.adwords.ru *.adsense.com *.adsense.ru *.feedburner
    .com *.doubleclick.com *.doubleclick.net *.igoogle.com
    *.youtu.be *.youtube.com *.youtube.ru *.blogger.com *.
    chromium.com *.setka.io *.google.com.gh ymetrica1.com *.

```

```

    google.com.pk *.google.com.br *.google.co.th *.google.
    com.vn *.google.lt;";
39
40 "VirusTotal": {
41   "detectedURL": "https://creativecdn.com/cm-notify?pi=
      taboola&tc=1",
42   "PayloadSHA256": "ef1955ae757c8b966c83248350331bd3a30f658
      ced11f387f8ebf05ab3368629",
43   "attributes": {
44     "crowdsourced_context": {
45       "severity": "null"
46     },
47     "last_analysis_stats": {
48       "malicious": "1",
49       "suspicious": "0",
50       "undetected": "9",
51       "total_checks": "87"
52     },
53     "popular_threat_classification": {
54       "popular_threat_category": "",
55       "suggested_threat_label": "null"
56     }
57   }
58 }
59 },

```

Listing B.1: Most second suspicious ad based on our criteria.

Appendix C

Suspicious Site C

```
1
2   "detectedURL": "https://google-bidout-d.openx.net/w/1.0/pd?
      plm=5",
3
4   "mdnClassification": {
5     "firstParty": [],
6     "thirdParty": [
7       "tracking_ad",
8       "any_basic_tracking",
9       "any_strict_tracking"
10    ]
11  },
12  "isShady": "Yes",
13  "isInUrlhaus": "No",
14  "ipAddresses": [
15    "35.244.159.8",
16    "34.98.64.218"
17  ],
18  "shodanResults": [
19    {
20      "ip": "35.244.159.8",
21      "hostnames": [
22        "openx.net",
23        "8.159.244.35.bc.googleusercontent.com"
24      ],
25      "tags": [
26        "cloud"
27      ],
28      "cpes": []
29    },
30    {
31      "ip": "34.98.64.218",
```

```

32     "hostnames": [
33         "218.64.98.34.bc.googleusercontent.com",
34         "openx.net"
35     ],
36     "tags": [
37         "cloud"
38     ],
39     "cpes": []
40 }
41 ],
42 "abuseIPDBData": [
43     {
44         "countryCode": "US",
45         "isWhitelisted": false,
46         "abuseConfidenceScore": 11,
47         "usageType": "Data Center/Web Hosting/Transit"
48     },
49     {
50         "countryCode": "US",
51         "isWhitelisted": false,
52         "abuseConfidenceScore": 12,
53         "usageType": "Data Center/Web Hosting/Transit"
54     }
55 ],
56 "cspHeader": "content-security-policy",
57 "cspHeaderValue": "script-src 'none'; object-src 'none'",
58 "VirusTotal": {
59     "detectedURL": "https://google-bidout-d.openx.net/w/1.0/
60         pd?plm=5",
61     "PayloadSHA256": "de259eb7ba7a0e45575deb33946f1fbc695c97c
62         33145ae4e49af0069d010868e",
63     "attributes": {
64         "crowdsourced_context": {
65             "severity": "null"
66         },
67         "last_analysis_stats": {
68             "malicious": "1",
69             "suspicious": "0",
70             "undetected": "21",
71             "total_checks": "92"
72         },
73         "popular_threat_classification": {
74             "popular_threat_category": "",
75             "suggested_threat_label": "null"
76         }
77     }
78 }

```

Listing C.1: Most third suspicious ad based on our criteria.

Glossary

AST Abstract Syntax Tree.

BeEF Browser Exploitation Framework.

CDN Content Delivery Network.

CIA Confidentiality, Integrity and Availability.

CORS Cross-Origin Resource Sharing.

CPE Common Platform Enumeration.

CSP Content Security Policy.

CSRF Cross-Site Request Forgery.

DNS Domain Name System.

DOM Document Object Model.

FTP File Transfer Protocol.

HTTP Hypertext Transfer Protocol.

HTTPS Hypertext Transfer Protocol Secure.

JS JavaScript.

SEA Search Engine Advertisement.

SOP Same-Origin Policy.

SSL Secure Sockets Layer.

TLS Transport Layer Security.

TNR True Negative Rate.

TPR True Positive Rate.

UI User Interface.

UUID Universally Unique Identifier.

WSM Weighted Scoring Model.

XSS Cross-Site Scripting.

Bibliography

- [1] abuse.ch. *URLhaus*. 2024. URL: <https://urlhaus.abuse.ch/>.
- [2] AbuseIPDB. *About AbuseIPDB*. 2024. URL: <https://www.abuseipdb.com/about>.
- [3] Adblock. *How does Adblock work*. 2024. URL: <https://helpcenter.getadblock.com/hc/en-us/articles/9738502775315-How-does-AdBlock-work>.
- [4] *Adblock Project Page*. <http://adblock.mozdev.org/>. Archived version accessed on 2024-04-01 via the Wayback Machine: <https://web.archive.org/web/20200430081315/http://adblock.mozdev.org/>. 2020.
- [5] AdGuard. *How ad blocking works*. 2024. URL: <https://adguard.com/kb/general/ad-filtering/how-ad-blocking-works/>.
- [6] AdGuard. *How to create your own ad filters*. 2024. URL: <https://adguard.com/kb/es/general/ad-filtering/create-own-filters/#html-filtering-rules>.
- [7] Yaser Alofer and Omer Rana. "Honeyware: A Web-Based Low Interaction Client Honeypot". In: *2010 Third International Conference on Software Testing, Verification, and Validation Workshops*. 2010, pp. 410–417. DOI: 10.1109/ICSTW.2010.41.
- [8] Amazon Advertising. *Amazon Advertising Acceptance Policies*. 2024. URL: <https://advertising.amazon.com/resources/ad-policy/creative-acceptance>.
- [9] Applied Management Centre. *Weighted Scoring Model*. Document available for download at the provided URL. 2024. URL: <https://appliedmanagement.com/downloads/>.
- [10] BeEF Project. *BeEF Architecture*. Accessed: 2024-05-16. 2024. URL: <https://github.com/beefproject/beef/wiki/Architecture>.
- [11] BeEF Project. *The Browser Exploitation Framework (BeEF)*. Accessed: 2024-02-16. URL: <https://beefproject.com/>.
- [12] Buffer. *Thug Documentation*. Section: JSON Logging Mode. 2024s. URL: <https://buffer.github.io/thug/doc/>.

- [13] Mathias Bynens. *JavaScript engine fundamentals: Shapes and Inline Caches*. 2024. URL: <https://mathiasbynens.be/notes/shapes-ics>.
- [14] Gerardo Canfora et al. "Detection of Malicious Web Pages Using System Calls Sequences". In: *International Cross-Domain Conference and Workshop on Availability, Reliability, and Security (CD-ARES)*. Ed. by Stephanie Teufel et al. Vol. LNCS-8708. Availability, Reliability, and Security in Information Systems. Part 2: 4th International Workshop on Security and Cognitive Informatics for Homeland Defense (SeCIHD 2014). Fribourg, Switzerland: Springer, Sept. 2014, pp. 226–238. DOI: 10.1007/978-3-319-10975-6\17. URL: <https://inria.hal.science/hal-01403998>.
- [15] Maxim Chernyshev and Peter Hannay. "The Zombies Strike Back: Towards Client-side BeEF Detection". In: *Proceedings of the 12th Australian Digital Forensics Conference*. Edith Cowan University. 2014. DOI: 10.4225/75/57b3de3dfb87a. URL: <https://ro.ecu.edu.au/adf/133/>.
- [16] Ericka Chickowski. "10 Web-Based Attacks Targeting Your End Users". In: *Dark Reading* (2013). Accessed: 2024-05-14. URL: <https://www.darkreading.com/cyber-risk/10-web-based-attacks-targeting-your-end-users>.
- [17] Cisco. *Cisco Cloud Security Documentation*. 2024. URL: <https://developer.cisco.com/docs/cloud-security/#!legacy-umbrella-apis-investigate-api-reference-api-umbrella-popularity-list-top-million-domains-get-top-most-seen-domains>.
- [18] Cloudflare. *What is HTTPS?* 2024. URL: <https://www.cloudflare.com/learning/ssl/what-is-https/>.
- [19] Content Security Policy Reference. *Content Security Policy Reference*. 2024. URL: <https://content-security-policy.com/>.
- [20] Cybersecurity and Infrastructure Security Agency (CISA). *CISA CEG Securing Web Browsers And Defending Against Malvertising*. Available at <https://www.cisa.gov/sites/default/files/2023-09/CISA%20CEG%20Securing%20Web%20Browsers%20And%20Defending%20Against%20Malvertising.pdf>. 2023.
- [21] Daniel Kladnik. *I Don't Care About Cookies*. <https://www.i-dont-care-about-cookies.eu/>. <https://www.i-dont-care-about-cookies.eu/>. 2024. (Visited on 05/17/2024).
- [22] DC Byte. *Data Centres in Amsterdam*. 2024. URL: <https://www.dcbyte.com/market-infographics/data-centres-in-amsterdam/#:~:text=Amsterdam%2C%20capital%20city%20of%20the,800MW%20of%20Live%20IT%20Capacity..>
- [23] Soumyajit Deb. *Understanding the architecture of the web browser*. 2024. URL: https://medium.com/@crypto_gaijin/understanding-the-architecture-of-the-web-browser-25cbf572fa1f.

Bibliography

- [24] Angelo Dell’Aera. *Thug*. Copyright (C) 2011–2024 Angelo Dell’Aera <angelo.dellaera@honeynet.org>. 2011-2024. URL: <https://github.com/buffer/thug>.
- [25] *Digital banner advertising spending growth in the United States from 2020 to 2028, by device*. <https://www.statista.com/forecasts/460827/banner-advertising-revenue-growth-device-digital-market-outlook-usa>. Accessed: Feb-2024. Nov. 2023.
- [26] Dutch Data Centers. *Dutch Data Centers 2019 Report*. 2024. URL: <https://www.dutchdatacenters.nl/en/nieuws/dutchdatacenters2019-2/>.
- [27] eyeo. *2023 eyeo Ad-Filtering Report*. 2024. URL: <https://info.eyeo.com/adfiltering-report/>.
- [28] fanboy et al. *EasyList*. <https://easylist.to/>. Accessed: 2024-04-01.
- [29] *Fanboy’s Annoyance List*. <https://fanboy.co.nz/>. Accessed: 2024-04-01.
- [30] Aurore Fass, Michael Backes, and Ben Stock. “JSTAP: A Static Pre-Filter for Malicious JavaScript Detection”. In: *Proceedings of the Annual Computer Security Applications Conference (ACSAC)*. 2019.
- [31] Aurore Fass et al. “JAS: Fully Syntactic Detection of Malicious (Obfuscated) JavaScript”. In: *Proceedings of the International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*. 2018.
- [32] GeeksforGeeks. *What is Browser Sandboxing?* 2024. URL: <https://www.geeksforgeeks.org/what-is-browser-sandboxing/>.
- [33] Google. *Chrome Web Store: Register*. 2024. URL: <https://developer.chrome.com/docs/webstore/register>.
- [34] Google. *DoubleClick by Google*. Accessed: 2024-02-15. 2024. URL: <https://marketingplatform.google.com/about/enterprise/>.
- [35] Google. *Google Ad Policies*. 2024. URL: <https://support.google.com/adspolicy/answer/6008942?hl=en>.
- [36] Google. *Google AdSense*. Accessed: 2024-02-15. 2024. URL: <https://www.google.com/adsense>.
- [37] Google. *Google Safe Browsing*. 2024. URL: <https://safebrowsing.google.com/>.
- [38] María Huerga Espino and Ana Isabel Asencio Martín. *MaiBee*. Version 1.0.0. May 2024. URL: <https://github.com/mhuergae/MaiBee>.
- [39] Instituto Nacional de Ciberseguridad (INCIBE). *Honeypot: una trampa para los ciberdelincuentes*. <https://www.incibe.es/empresas/blog/honeypot-una-trampa-para-los-ciberdelincuentes>. Accessed: 2024-03-28. June 2023.

- [40] JSHeilder Project. *JSHeilder: A Browser Security Tool*. <https://jshelter.org/>. Accessed: 2024-05-25. 2024.
- [41] Jitendra Kumar, A. Santhanavijayan, and Balaji Rajendran. "Cross Site Scripting Attacks Classification using Convolutional Neural Network". In: *2022 International Conference on Computer Communication and Informatics (ICCCI)*. 2022, pp. 1–6. DOI: 10.1109/ICCCI54379.2022.9740836.
- [42] Zhou Li et al. "Knowing Your Enemy: Understanding and Detecting Malicious Web Advertising". In: *Proceedings of the ACM Conference on Computer and Communications Security* (Oct. 2012). DOI: 10.1145/2382196.2382267.
- [43] Yuping Liu-Thompkins. "A Decade of Online Advertising Research: What We Learned and What We Need to Know". In: *Journal of Advertising* 48.1 (2019), pp. 1–13. DOI: 10.1080/00913367.2018.1556138. eprint: <https://doi.org/10.1080/00913367.2018.1556138>. URL: <https://doi.org/10.1080/00913367.2018.1556138>.
- [44] Majestic. *Majestic Million Report*. 2024. URL: <https://majestic.com/reports/majestic-million>.
- [45] Meta. *Meta Ad Standards*. 2024. URL: <https://transparency.meta.com/en-gb/policies/ad-standards/>.
- [46] Markus Jakobsson Mona Gandhi and Jacob Ratkiewicz. "Badvertisements: Stealthy Click-Fraud with Unwitting Accessories". In: *Journal of Digital Forensic Practice* 1.2 (2006), pp. 131–142. DOI: 10.1080/15567280601015598. URL: <https://doi.org/10.1080/15567280601015598>.
- [47] Aleesha Jacob MonetizaMore. *What's Anti-Adblock? STOP Losing Revenue To Adblock NOW*. 2024. URL: <https://www.monetizemore.com/blog/what-is-an-anti-ad-blocker/>.
- [48] Mozilla Developer Network. *Anatomy of a WebExtension*. 2024. URL: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/Anatomy_of_a_WebExtension.
- [49] Mozilla Developer Network. *Content Security Policy (CSP)*. 2024. URL: <https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP>.
- [50] Mozilla Developer Network. *Content Security Policy (CSP)*. 2024. URL: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/Content_Security_Policy.
- [51] Mozilla Developer Network. *Cross-Origin Resource Sharing (CORS)*. 2024. URL: <https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS>.
- [52] Mozilla Developer Network. *Geckodriver Documentation*. <https://firefox-source-docs.mozilla.org/testing/geckodriver/index.html>. 2024. (Visited on 05/17/2024).

Bibliography

- [53] Mozilla Developer Network. *Marionette*. <https://firefox-source-docs.mozilla.org/testing/marionette/index.html>. 2024. (Visited on 05/17/2024).
- [54] Mozilla Developer Network. *MDN Web Docs: Glossary: Wrapper*. <https://developer.mozilla.org/en-US/docs/Glossary/Wrapper>. Accessed: 2024-05-25. 2024.
- [55] Mozilla Developer Network. *Same-Origin Policy (SOP)*. 2024. URL: https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy.
- [56] Mozilla Developer Network. *Web Accessible Resources*. 2024. URL: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/manifest.json/web_accessible_resources.
- [57] Mozilla Developer Network. *WebExtensions API: storage.local*. <https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/API/storage/local>. 2024. (Visited on 05/17/2024).
- [58] Mozilla Developer Network. *What are WebExtensions?* 2024. URL: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/What_are_WebExtensions.
- [59] Jose Nazario. "Phoneyc: A virtual client honeypot". In: (May 2009).
- [60] Yadav Neelam. *How Does Web Browsers Work*. 2024. URL: <https://medium.com/@yaduvanshineelam09/how-does-web-browsers-work-ec0f171aedc6>.
- [61] Netify.ai. *Applications*. 2024. URL: <https://www.netify.ai/resources/applications>.
- [62] Palo Alto Networks. *What is a Fast Flux Network?* <https://www.paloaltonetworks.com/cyberpedia/what-is-a-fast-flux-network>. Accessed: 2024-05-05. 2021.
- [63] njuhxc. *MJDetector*. 2024. URL: <https://github.com/njuhxc/MJDetector>.
- [64] OpenPhish. *OpenPhish*. 2024. URL: <https://openphish.com/index.html>.
- [65] OWASP. *Content Security Policy Cheat Sheet*. 2024. URL: https://cheatsheetseries.owasp.org/cheatsheets/Content_Security_Policy_Cheat_Sheet.html#content-security-policy-cheat-sheet.
- [66] OWASP. *OWASP Top Ten*. Accessed: 2024-05-16. 2023. URL: <https://owasp.org/www-project-top-ten/>.
- [67] OWASP Foundation. *Cross-Site Request Forgery (CSRF)*. <https://owasp.org/www-community/attacks/csrf>. Accessed: 2024-05-15.
- [68] Hynek Petrak. *JavaScript Malware Collection*. 2024. URL: <https://github.com/HynekPetrak/javascript-malware-collection/tree/master>.

- [69] Zahra Pooranian et al. "Online Advertising Security: Issues, Taxonomy, and Future Directions". In: *IEEE Communications Surveys & Tutorials* 23.4 (2021), pp. 2494–2524. DOI: 10.1109/COMST.2021.3118271.
- [70] Elliott L. Post and Chandra N. Sekharan. "Comparative Study and Evaluation of Online Ad-Blockers". In: *2015 2nd International Conference on Information Science and Security (ICISS)*. 2015, pp. 1–4. DOI: 10.1109/ICISSEC.2015.7370988.
- [71] Devi Priya and Sibi Chakkaravarthy. *Containerized cloud-based honeypot deception for tracking attackers*. https://www.researchgate.net/figure/Honeypot-system-placement-in-an-organizational-network_fig2_367413267. Affiliation: VIT-AP University, Accessed: 28 Mar, 2024. 2023.
- [72] N. Provos and T. Holz. *Virtual Honeypots: From Botnet Tracking to Intrusion Detection*. Addison-Wesley, 2008. ISBN: 9780321336323. URL: <https://books.google.dk/books?id=QuHnPgAACAAJ>.
- [73] Brock Munro Publifit. *How To Detect Ad Blockers*. 2024. URL: <https://linguana.publifit.com/blog/ad-blockers>.
- [74] Python Software Foundation. *Beautiful Soup 4*. 2024. URL: <https://pypi.org/project/beautifulsoup4/>.
- [75] M T Qassrawi and Hongli Zhang. "Client honeypots: Approaches and challenges". eng. In: *4th International Conference on New Trends in Information Science and Service Science*. IEEE, 2010, pp. 19–25. ISBN: 9781424469826.
- [76] Mobile Marketing Reads. *Online advertising revenue in the United States from 2000 to 2022 (in billion U.S. dollars) [Graph]*. <https://www.statista.com/statistics/183816/us-online-advertising-revenue-since-2000/>. [Online]. Available: <https://www.statista.com/statistics/183816/us-online-advertising-revenue-since-2000/>. Apr. 2023.
- [77] Scrapy. *Scrapy: An open source and collaborative framework for extracting the data you need from websites*. <https://scrapy.org/>. Accessed: 2024-05-17.
- [78] *Search Advertising - Worldwide*. <https://www.statista.com/outlook/amo/advertising/search-advertising/worldwide?currency=EUR>. Accessed: Feb-2024. n.d.
- [79] Bright Security. *8 Types of Web Application Attacks and Protecting Your Organization*. Accessed: 2024-05-15. 2023. URL: <https://brightsec.com/blog/8-types-of-web-application-attacks-and-protecting-your-organization/>.
- [80] Christian Seifert, Ian Welch, and Peter Komisarczuk. "HoneyC - The Low-Interaction Client Honeypot". In: Jan. 2007.
- [81] Selenium. *Selenium: A browser automation framework and ecosystem*. <https://www.selenium.dev/>. Accessed: 2024-05-17.

Bibliography

- [82] SeleniumHQ. *Selenium Issue #7805: Add more detailed logging for driver start failures*. <https://github.com/seleniumhq/selenium-google-code-issue-archive/issues/7805>. 2016. (Visited on 05/17/2024).
- [83] Shodan. *Shodan InternetDB*. <https://internetdb.shodan.io/>. Accessed: 2024-05-16. 2024.
- [84] Xuyan Song et al. "Malicious JavaScript Detection Based on Bidirectional LSTM Model". In: *Applied Sciences* 10.10 (2020). ISSN: 2076-3417. DOI: 10.3390/app10103440. URL: <https://www.mdpi.com/2076-3417/10/10/3440>.
- [85] Statista. *Digital advertising spending in the United States from 2019 to 2028, by format (in billion U.S. dollars) [Graph]*. <https://www.statista.com/forecasts/455840/digital-advertising-revenue-format-digital-market-outlook-usa>. [Online]. Available: <https://www.statista.com/forecasts/455840/digital-advertising-revenue-format-digital-market-outlook-usa>. Nov. 2023.
- [86] VirusTotal. *VirusTotal*. 2024. URL: <https://www.virustotal.com/>.
- [87] VirusTotal. *VirusTotal*. 2024. URL: <https://docs.virustotal.com/docs/comments>.
- [88] Yahoo. *Yahoo Advertising Terms*. 2024. URL: <https://legal.yahoo.com/us/en/yahoo/terms/advertising-322/index.html>.
- [89] z0ccc. *Extension Detector*. <https://github.com/z0ccc/extension-detector>. GitHub repository. 2024.
- [90] Azreen Zaini and Anazida Zainal. "Exploiting DOM Mutation for the Detection of Ad-injecting Browser Extension". In: *Recent Trends in Data Science and Soft Computing*. Ed. by Faisal Saeed et al. Cham: Springer International Publishing, 2019, pp. 657–669. ISBN: 978-3-319-99007-1.