CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Uncovering Hidden Links with Malicious Non-Interference

Master's thesis in Computer science and engineering

JONATHAN LINDQVIST, ANTON LUNDH

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2025

Master's thesis 2025

Uncovering Hidden Links with Malicious Non-Interference

JONATHAN LINDQVIST
ANTON LUNDH

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2025

Uncovering Hidden Links with Malicious Non-Interference

JONATHAN LINDQVIST, ANTON LUNDH

Supervisors: Benjamin Lundblad and David Sands, Chalmers University of Technology
Examiner: David Sands, Chalmers University of Technology

Master's Thesis 2025
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Image of hidden link

Typeset in LaTeX
Gothenburg, Sweden 2025

Uncovering Hidden Links with Malicious Non-Interference

JONATHAN LINDQVIST, ANTON LUNDH
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Every day, billions of searches are made on search engines such as Google. The results shown are ranked using proprietary algorithms by the search engine providers. Some of these algorithms consider the number of backlinks, links from other websites, as an indicator of popularity and relevance. This ranking mechanism is widely known and, in some cases, exploited by attackers who inject links onto other websites to improve their own search engine rankings. While normal backlinks are visible to users and search engines alike, some attackers use hidden links, links that are not meant to be seen by users but are still indexed by search engines. In an attempt to artificially boost rankings in a search engine. In this thesis, we focus on links that are invisible to users but visible to search engines. We call these "hidden links".

Because of the malicious behavior of some websites, methods for detecting these hidden links have been developed previously. The key concept for this thesis is non-interference, a semantic condition that defines when a system is well-behaved, such as ensuring that a system does not leak secrets or preserve the integrity of certain data. In our thesis, the concept of non-interference is used in reverse, and is therefore referred to as malicious non-interference. The fundamental idea of our approach for detecting hidden links is to make changes to the website's code and make a visual inspection to see if the changes are visible.

The tool developed by applying malicious non-interference in this thesis is called Malicious Non-interference Scanner (MANIS), and it shows promising results, as it is capable of detecting hiding methods that other scanning tools, such as Sucuri SiteCheck, are unable to detect. MANIS shows promising results when tested on two different datasets: one randomly sampled from the latest Tranco domain ranking list, and a dataset collected by us during the development of MANIS with websites that we suspect to contain hidden links. From these datasets, MANIS is capable of detecting hidden links with accuracies of 86% and 97% respectively. With the majority of the false positives coming from an inability to interact with the website.

Keywords: Computer science, engineering, master thesis, MANIS, malicious non-interference, web scanning, search engine optimization.

Acknowledgements

We want to thank our supervisors, Benjamin Lundblad and David Sands at Chalmers University of Technology, for their valuable guidance, continuous support, and feedback. Benjamin Lundblad agreed to be a supervisor even though he was already busy with multiple other projects, and David Sands took on two roles, being both our supervisor and our examiner. We would also like to thank Daniel Gillblad at Recorded Future for his valuable input from an industry perspective and his genuine interest in the thesis.

A big thanks to Chalmers for making this thesis possible.

Jonathan Lindqvist, Anton Lundh, Gothenburg, 2025-06-22

Contents

List of Figures

List of Tables

Terminology

SEO	Search Engine Optimization
SE	Search Engine
URL	Uniform Resource Locator, another name for a link to a website
Signature	A predefined pattern or set of characteristics linked to a known method of hiding a link, based on traits that have been previously observed and catalogued.
Backlink	Link on a webpage that links to another domain
Hidden Link	A link on a webpage that cannot be seen by a user, but is visible in the page source.
Injected site	Site that is injected onto others in an attempt to boost SEO
Source domain	Domain where the URL is found
Target domain	Domain to which a source domain points
Internal link	Link to the same domain as the one it was found on
External link	Link to a different domain than the one it was found on
Cloaking	A website that attempts to conceal its true nature, by varying the content based on the user
Non-interference	Security property that ensures that a certain information source has no influence on a certain information channel
Webpage	A specific site on a website, such as an about page
Website	A collection of webpages
Threat actor	A person or group that carries cyberattacks

1

Introduction

To navigate the web, we use Search Engines (SE) to help us locate the information we need. These SEs index the Web and rank results based on various factors, one of the prominent factors being backlinks. Backlinks are links present on other websites that point to a specific website. The number of backlinks is one of the measures used to determine the ranking position on the SE [1]. Due to the value of backlinks, hackers inject hidden links on websites in an attempt to boost the ranking of their website. The website that has links injected into it can either be legitimate but hacked or owned by threat actors for the sole purpose of providing backlinks.

The process of improving a website's ranking in the SE is known as Search Engine Optimization (SEO). SEO involves strategies and techniques aimed at increasing a website's visibility in the search engine results. The goal of this increased visibility is to attract more organic (non-paid) traffic from search engine users. Effective SEO can significantly enhance a website's presence, credibility, and profitability. Through various optimizations, such as improving content, strategic use of keywords, and building quality backlinks. These optimizations may lead to higher web traffic, ultimately leading to increased revenue.

Hiding URLs can be achieved in various ways, some of which will be elaborated upon in Section 3.2, but the purpose is always the same, to make the links invisible to a user who is simply browsing the page. The goal of this thesis is to develop a tool to detect these links and some of the methods that hide them. Work within the field of detecting malicious SEO has been done before, as it is important for companies such as Google, as seen by their patent [2]. There has also been academic research done, as seen in the work done by Junjie Xiong, et al. [3]. However, the methods used previously are vastly different from this thesis's approach.

For determining whether a link is hidden, a black box method based on non-interference is established. That is to say, instead of performing an analysis directly on the code, it will instead be focused on what is visible to the user on the webpage. As detailed in Section 2.2.1, non-interference is traditionally used to show that a system is well behaved and secure by not being able to detect change when the inputs are modified.

But in this thesis, a reverse non-interference method is used. The method for detecting hidden links is based on the idea that by modifying the HTML element containing a link, there should be a visual change on the website. By comparing

an image of the page before and after modification, a conclusion can be drawn. If no change is detected, the link is hidden, and if a change is detected, the link is visible. This can be seen as the opposite of the non-interference security property, which ensures that certain inputs have no observable impact on outputs. As we are using an opposite definition, we want to see a change, the term "malicious non-interference" is coined to keep them separate. The benefit of this approach over some signature-based approach, which refers to a predefined pattern or set of characteristics that correspond to a known method of hiding a link with specific traits that have been previously observed and cataloged, is that it can detect new methods without needing to add new signatures to the database.

1.1 Aim of this thesis

The aim of this thesis is to show that the application of a black box method based on malicious non-interference is an effective concept for detecting hidden links. This can be divided into three parts: one is research on how different hiding methods work, a second on how to handle the specific hidden case with malicious non-interference, and a third is implementation into a novel tool for testing the concepts presented in the thesis.

In summary, the thesis will focus on the following research questions:

1. How effectively can a black box method based on malicious non-interference detect hidden links?
2. What factors need to be considered for accurate detection of hidden links with malicious non-interference?
3. Can malicious non-interference be used to discover new signatures?

1.2 Problem statement

Malicious threat actors use various techniques to inject hidden URLs into legitimate webpages. Some of the most common methods of making links hidden include various CSS attributes such as position, color, and display. These can be statically detected, such as done by Sucuri SiteCheck[1], which, from our understanding, checks for known signatures. But the problem with this method comes from how easy it is to evade detection. It can be as simple as changing some digits, which means that an approach that should be able to find new ones cannot rely fully on signatures. Some simple examples of methods used for the hiding of links found during the thesis can be seen below:

[1] https://sitecheck.sucuri.net

1.2.1 Example 1: Color matching

The example below is using a CSS styling attribute to match the color of the text and the background. Thereby making it impossible for a visitor to spot the hidden links, unless they accidentally hover over them or mark the text.

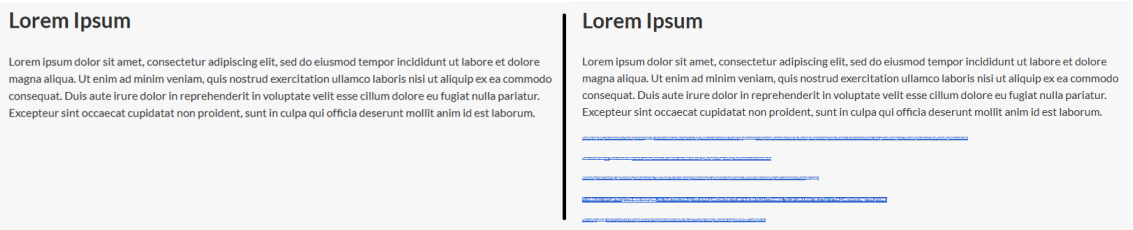


Figure 1.1: Anonymized example where text and background color match.

As seen in Figure 1.1, the links are impossible to spot when just glancing over them quickly, but when marked, it is quite obvious that there are some hidden links. The CSS for this attack is trivial and can be seen below:

```
<p style="color: \#f7f7f7; font-size: 3px;">
```

To note is that the links were not only concealed by setting the color but also by minimizing the font size. Thereby decreasing the chance for human detection, as there is an even smaller region that is clickable.

1.2.2 Example 2: Positioning

Another method for hiding links is by placing the links outside of the visible area. Either by placing it to the left/right or above/below the screen, when placed below or to the right, negative values are used so that they are actually placed above and to the left. The CSS code for achieving this can be seen below:

```
<div style="position:absolute;left:-519px;top:-394px;">
```

This CSS ensures that it is placed way outside the visible area and does not affect any of the other elements on the webpage, due to the use of absolute positioning. In Figure 1.2, an attempt at visualizing how a hidden link placed with negative left and top coordinates can be seen. The red "Hidden Link" in the top left of the figure represents the link that is hidden, and the computer monitor shows the visible area of the website that contains the hidden link.

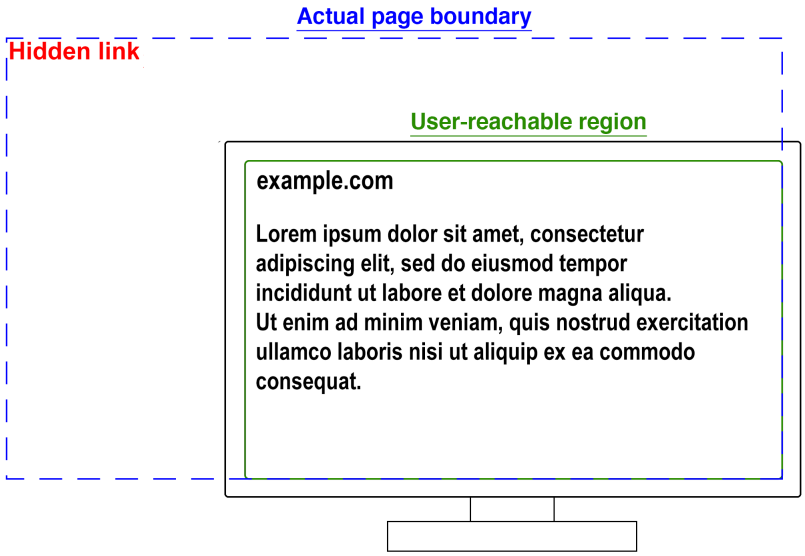


Figure 1.2: Example of how a position-based hidden link can be visualized.

1.3 Scope of this thesis

This thesis will focus on the detection of hidden links using an approach based on malicious non-interference, and the problems related to the application and development of a tool to demonstrate the capabilities of the approach. To limit the scope of the thesis, it will not focus on specifics of web development, SEO, or speculations regarding the inner workings of search engines. The reader is expected to have some basic understanding of how the Web works. Although several SEs are available, most explanations in this thesis are based on Google due to its dominant market share.

1.4 Ethical considerations

In computer science, especially when conducting research within computer security, ethical considerations must be taken into account, as the results of such research can theoretically have devastating effects if misused. In this thesis, however, no vulnerabilities will be disclosed. Though there will be data presented taken from websites that have been affected by hidden links. These websites will be masked to remove identifiable information to protect the privacy of the affected sites.

Although we would have preferred to contact every website identified as containing hidden links, this was not feasible given that over 20 000 different website owners would have needed to be contacted. This would have required extensive manual work, both for verification to not scare anyone, but also to locate appropriate contact information. Moreover, finding an email service capable of sending such high volumes of emails without being flagged as spam would have demanded significant time and likely substantial monetary resources.

Scanning the Web always carries some risk, as it is possible to be too aggressive. Sending too many requests in a short timespan can lead to excessive loads on the target website, potentially slowing it down, or in extreme cases, causing it to crash. The scanning done in this thesis is not deemed to cause any problems, as most of it will only be one request per website, and otherwise, requests will be sent sequentially. Meaning that we at most have one active request for a single site, which we can assume that any website should be able to handle.

The tool developed in this thesis, named Malicious Non-interference Scanner (MANIS), could technically be misused by malicious users to benchmark their ability to conceal their backlinks. However, this problem is not unique to this thesis, it is a common consideration when doing any kind of research, especially within computer security. Publishing this thesis and MANIS will mainly raise awareness of the situation of malicious practices within SEO, allowing website owners to investigate their websites for hidden links. Another benefit is that other developers and researchers can also contribute to MANIS, improving it even further, thereby making it even harder to hide links. Ultimately, the potential benefits of the research outweigh the negatives, as this tool will also allow website owners to find out whether they are affected or not.

Regarding data collection, real-world websites were scanned during the thesis, both throughout development and during evaluation. This raises the question of whether MANIS could unintentionally cause denial-of-service attacks by generating more traffic than a website can handle. However, this risk was considered from the outset. The number of requests sent to each website was kept to a minimum and is comparable to that of a regular user visiting the site. The only potential difference from a typical user could be the timing between some requests, which is only noticeable at the start of a scan for a given website. Ultimately, if a website can handle the traffic of a normal user, MANIS should not impact its availability, as it does not generate any significant load.

1.5 Contribution

This thesis contributes to a field with relatively little research by addressing the relatively unexplored concept of hidden links on the web. Its primary contribution is the creation of a tool based on a security concept dubbed malicious non-interference. The benefit of this is that it does not rely on any database of signatures or previous information.

1.6 Outline

The rest of the report is structured as follows. Chapter 2 will describe the key concepts necessary for understanding the thesis as well as related works. Chapter 3 will describe how the research was conducted and the capabilities needed for MANIS to function properly. Chapter 4 presents how everything was implemented in MANIS. Chapter 5 and Chapter 6 present and discuss the results and insights gained from running MANIS, as well as suggestions on how to continue the research. Chapter 7 concludes everything found during the thesis.

2

Concepts and tools

This chapter provides the theoretical and practical background for MANIS. Section 2.1 presents the core concepts: SEO, the Web, and backlinks that motivate the design of MANIS. Section 2.2 details the specific techniques and tools used to implement MANIS. Section 2.3 describes the publicly available datasets and tools primarily used for the evaluation phase of MANIS. Finally, Section 2.4 reviews the related work to the thesis.

2.1 Background to the Web and SEO

The internet is ever-growing, with the number of websites growing rapidly, resulting in manually navigating the web becoming impractical. Search engines such as Google, Bing, and DuckDuckGo serve as essential tools by traversing the web, indexing it by varying variables, all with their own proprietary algorithms. Even though their algorithms are proprietary, some variables have been made public or reverse-engineered. One such variable is the number of backlinks to a website, the more websites that contain a link to a specific site, the higher that specific site is likely to be ranked [4]. Given that Google has around a 90% market share in the search engine market, websites primarily focus their SEO efforts on the factors known or believed to influence Google's ranking algorithm [5].

Crawling forms the technical backbone of search indexing and related applications. Web crawling is the term commonly used for an application designed to download webpages, it can also be known as a robot, a spider, or simply as a crawler [6]. It is used in retrieving data for usage in search engines, web archiving, and data mining. A crawler works by visiting webpages, so to start, it is given a URL or set of URLs to visit, and based on all the external links present on those websites in continues to crawl either to a set depth or until an end is reached, whilst crawling the Web it continuously downloads the pages for further processing.

The Web is a complex place with ever-changing standards and technologies. One such newer approach is Single-page Applications (SPAs), the goal of SPAs is to make the user experience (UX) more fluid and seamless by using JavaScript to render content instead of redirecting the user to new pages [7], [8]. However, as the content is rendered dynamically, it is harder for search engines to index the content, impacting the SEO results. This is relevant to the thesis because if companies such as Google struggle to view the content with their bots, then we are likely to encounter similar problems with MANIS.

However, as SPAs are quite popular today, both developers and search engines have adapted to mitigate the negative impact of the technology. Developers use isomorphic JavaScript to render content better. As isomorphic JavaScript first renders the page on the server and then sends it to the client, where it behaves like a normal SPA [8]. This has made it easier for crawlers to render sites as they are partially static.

Understanding how search engines operate is essential to grasp how SEO and search manipulation work. The first step is "URL Discovery", which is the process of finding websites to crawl [9]. This is achieved either by user-submitted sitemaps or by finding links on websites pointing to a previously unknown website, the latter part is most relevant for this thesis. When the information is retrieved, the search engine gathers information about the content on the website by analyzing the elements found on the page. The information is then indexed and stored, and when a user makes a search request, the search engine tries to match the search to the most relevant website.

Because of this, there is a whole industry that specializes in optimizing websites for search engines so the website will be listed higher in search results, known as SEO [10]. The goal is to make the websites easier to understand for search engines, which is done in various ways. Google has published some basic recommendations such as using descriptive URLs, including useful alt-text to images, and avoiding spelling mistakes [10]. However, as stated by Google, it is important to get your website discovered, which is primarily done by Google finding links to your website on already discovered websites.

The reason for SEO and wanting to get ranked higher in search results is that users often trust and pick websites listed on the first page of the search engine [1]. SEO is especially important for small and medium-sized businesses as visibility for these companies is vital [11], therefore, if a malicious website can manipulate search engines to get listed above honest websites, it will benefit them financially. One thing that has a higher weight in search engine ranking than SEO is paid search marketing (PSM), in other words, advertisements, which allow websites to purchase a specific spot on the search result page for specific keywords [12].

The concept of maliciously injecting links to enhance SEO is not new. As it was discussed as early as 2008 by Kerry Dye in an article titled "Website abuse for search engine optimisation", where she talks about how SEO serves as free advertising and how attackers can try to exploit it by invisibly injecting their poorly ranked page into the higher ranked pages to boost their search ranking [13]. There have been attempts by J. Xiong, et al. at solving these issues by feeding the pages into common large language models (LLMs) like OpenLM OpenLLaMA, Google Gemma, and Microsoft

Phi, but this did not yield good results, as their research showed detection rates of less than 20% of hidden links being discovered [3]. Xiong also proposed a framework based on static code analysis and visual inspection to improve the detection rate, however, they did not provide proper metrics for the results, thus the effectiveness of the proposed framework is hard to grasp. Xiong's framework is explored further in Section 2.4.

There is a problem for crawlers, as there are some websites that are actively trying to hide their content from crawlers, by showing different content compared to what they show normal users [14]. These attempts to cloak a website's true intentions are done by checking parameters related to the connection, like the IP addresses, user-agent, and JavaScript capabilities. The research done in the paper [14] indicates that 11.7% of the top 100 search results related to searches frequently targeted by malicious actors, such as weight loss supplements, utilize cloaking against Google's crawler. Which means they are trying to conceal their website in some manner, against being properly indexed.

2.1.1 Purchasing backlinks

As part of search engine rankings are based on the amount of backlinks, there exists a prominent market in purchasing and reselling backlinks [15]. Looking into a service selling backlinks called backlinks.com they advertise themselves as [16]:

> "As one of the first, large-scale link automation platforms, our software has helped 100,000+ professionals buy and sell links. Now, over 10 years later, we are still the most trusted brand in the industry."

They offer users the ability to purchase backlinks on websites, with pricing being based on which page the link will be placed on, with a 75% lower price if the link is not on the front page. Pricing is based on the domain authority (DA) score calculated by Moz[1], with a higher price for a higher Moz DA score. For example, a backlink on a homepage with a Moz DA score of 10-14 costs one credit, while one on a site with a score of 45+ costs 30 credits [17]. With one credit being equal to one USD, with a discount of 10% for purchases over 500 USD [18]. If you instead want to sell placement for links on your website, they offer pricing based in USD, with backlinks.com taking 60% of the purchase price. Leaving you as the seller with 40% of the purchase price. For example, selling a link on a page with a Moz DA score of 10-14 gives you 0.4 USD, or 12 USD if you have a Moz DA score of 45-100 [19]. The link insertion is done automatically by a script that you add to your website, the script then automatically fetches the links from a database and adds them to your site when a user visits [20].

There are websites with other models, such as a subscription-based model as sold by linkbuilder.io. For a monthly cost of between 2 999 USD and 19 999 USD, you can get between 8 and 58+ high-quality backlinks (domain reputation 50-90 as calculated by Ahrefs) [21], [22].

[1] https://moz.com/domain-analysis

There does not seem to be any explicit mentions of websites that sell hidden backlinks, but there are people on SEO forums looking for it, and wanting to purchase it [23]. Likewise, there are people who mention they know of services that offer it, but do not want to tell which due to not agreeing with the practice [24]. Meaning that there should be services offering it, but perhaps not publicly disclosing it.

2.2 Concepts and tools used in MANIS

MANIS is a complex tool built in parts on concepts created by other researchers and developers. The following sections describe these to give a better overview of MANIS. The section is structured as follows: First, the two ground concepts for the detection method in MANIS, which is built upon a flipped definition of non-interference, named by us malicious non-interference, are explained. Then, methods for attempting to simulate human vision are explained as a way to try and detect if a text and background are of similar color. Lastly, the methods for programmatically controlling a web browser, as required to apply the concepts discussed in the previous section, are detailed.

2.2.1 Non-interference

Non-interference is a fundamental concept in information-flow control, which monitors how information moves through a program [25]. The idea behind this is to keep the information flow secure in both confidentiality and integrity aspects. The confidentiality aspect is that any information that was secret before should still be secret after running the program. Integrity can be defined in multiple ways, however, the one most related to the concepts of non-interference is as a dual to confidentiality, meaning that a trusted output should remain unaffected by an untrusted input.

Non-interference was first introduced in 1982 by Goguen and Meseguer [26] to formalize the property of security of information flow for programs. The property can be explained easily through an example: if an input to a system from a process has any effect on the output from the system to another process, then the interference implies that information flows from the first process to the other process. It is non-interfering if there is no effect from the input, and then there is no information flowing from the first process to the second. The property is specifically designed for deterministic systems and is quite strict [26], as just one small change in the output, such as a single bit, is deemed as a violation of the non-interference property [27]. However, some variants relax the definition and work with non-deterministic systems [26].

By relaxing the definition of non-interference, it can be applied to the hiding of links. Let W_1 and W_2 be two webpages that differ only in the CSS attributes of two otherwise identical links. Like Examples 1 and 2 from Section 3.2. Let V_1 and V_2 denote what a user can see when visiting the two websites W_1 and W_2. Then the definition of hidden links is that V_1 and V_2 are indistinguishable to the user, even if they are not bitwise identical. That is, changes to the links will not be reflected on the webpage. In this way, the traditional non-interference property, normally used to reason about integrity or confidentiality, is now used to define when the link is hidden.

Malicious non-interference inverts the concept of non-interference. Here, we apply modifications to an element and examine whether the alteration becomes visible. If the change is observable, the element is deemed visible, and if the change has no observable effect, the element is deemed hidden. Hence, the definition is reversed compared to classic non-interference, as it is desired to see a change. Therefore, the term malicious non-interference is used to keep the concepts separate.

2.2.2 Image perception

In order to assess the similarity between two images, several methods can be employed to produce a metric. A similarity metric enables a computer to compare images in a manner that approximates human visual perception. This functionality is important in MANIS, as it allows for the detection of elements whose color matches the background. Such a technique could be used to conceal content, such as text, from the user while still keeping it visible to machines. For an example, see Figure 6.1.

SSIM

Structural Similarity Index Measure (SSIM) is a metric that gives a percentage between $0 - 100\%$ of how similar two images are [28]. It does this by working within a window of pixels of a fixed size, typically eight by eight pixels. Within this window, there are three different calculations and comparisons done: a luminance, a contrast, and a structure [29]. These are then combined to create the similarity measure.

pHash

pHash is a perceptual image hashing algorithm [30], which means that instead of doing a normal hash that is not resistant to change like SHA256 and MD5, it instead calculates a hash based on the details in the image. As a result, instead of getting a completely different hash as a normal hashing algorithm would result in, perceptual hashing produces a hash that is still similar to the original hash for two similar images. This allows measurements for determining how similar two images are.

pHash first reduces the image size to 32x32 pixels, converts it to grayscale, and then applies a discrete cosine transform, saving only the lowest frequencies [30]. An average discrete cosine transform (DCT) is calculated on part of the result from the DCT calculation, which is used in conjunction with the results from the lowest DCT frequencies. To generate the hash, the average DCT is compared to the saved

DCT frequencies, a frequency above the average results in a 1, and below results in a 0. The resulting bits are concatenated to form the final hash. To compare two hashes, the Hamming distance is calculated, which counts the number of differing bits between the two hashes, indicating the degree of similarity.

2.2.3 Browser automation

Browser automation refers to the process of controlling a web browser with a programming language. It is commonly used for performing tasks in the browser, such as navigating pages, interacting with elements, and extracting data. Typically, browser automation is used to scrape data from websites or automate testing of web applications. With browser automation, it is able to perform actions on the website based on code, thereby simulating human behavior without having to rely on a human for manual input.

There exists a standard defined by the World Wide Web Consortium (W3C) for browser automation, however, there are also alternatives that do not follow the standard [31], [32]. B. Carcía et al. provide a comparison of four of the more popular alternatives for automating browser control in [32], in which they compare the functions of Selenium, Cypress, Puppeteer, and Playwright. A short comparison of the most basic functionality of these can be seen below in Table 2.1.

	Cypress	Playwright	Puppeteer	Selenium
Programming Language	JavaScript	Python, Java, and more	Node.js	Python, Java, and more
W3C Compliant				✓
Chromium	✓	✓	✓	✓
Firefox	✓	✓		✓
WebKit	✓	✓	✓	✓
Internet Explorer				✓

Table 2.1: Comparison of different browser automations.

In Table 2.1, there is a short comparison between the different browser automation tools. An interesting thing to note is that Selenium is the only one that follows the standard set by W3C.

The standard defined by W3C is called WebDriver, which defines a control interface designed to inspect and control user agents [31]. It allows interactions with a webpage's DOM, enabling actions such as clicking buttons and entering text. Additionally, it supports remote browser automation via a language-neutral wire protocol. Thereby giving a user the possibility to automate testing and interactions with a browser with the use of a normal programming language.

2.3 Publicly accessible data and tools

Both for the evaluation phase and for the development of MANIS, publicly accessible data and tools aided the work. This section describes them and their usage, in the following order: First, CommonCrawl and its two primary data resources, that was used both for information gathering and the evaluation. Next, the Tranco list is introduced, a research-oriented website popularity ranking, used both as an evaluation dataset and as a popularity metric in a filter for MANIS. Finally, Sucuri SiteCheck, a website scanner capable of detecting hidden links, is outlined, which is used as a baseline for comparing the results from MANIS.

2.3.1 Common Crawl

Common Crawl is a nonprofit that crawls the internet and indexes websites in various ways. In the context of the thesis, the most relevant dataset is their Web Graphs[2], but also the WARCs[3]. The benefit of Common Crawl is that they release their data for free, meaning it is possible to use it in research such as the one done in this thesis.

The Web ARChive

The Web ARChive (WARC) format is a file format commonly used to store crawl data [33]. It is the raw data that the crawler generates while crawling the web, meaning it includes both the metadata from the crawler, the HTTP header, and the HTTP response. This means that it is essentially a screenshot of the webpage without any external files such as images, CSS, and JavaScript.

Web graphs

Common Crawl releases Web Graphs every month, which is a dataset of the last three months of crawl data on how websites are connected together [34]. This data consists of close to 270 million websites and 2.7 billion connections between these websites [35]. These are separated into two files, one in which the websites are stored in reverse domain name notation, that is to say, subdomain.example.com is stored as com.example.subdomain together with an ID assigned to the website. The other file stores the connections between all the websites in the form of the two IDs being on the same row. This indicates that there is a link present on the website linking to the other website.

An example of this can be seen below. First, an excerpt was taken from the edges file, where each row contains two IDs. The left ID represents the website that contains a link, and the right ID represents the domain being linked to, as seen in Listing 1.

As shown in Listing 1, the numerical IDs do not give any directly valuable information. But if they are interpreted by looking them up in the vertices file, they can be translated to domain names, as demonstrated in Listing 2.

[2]https://commoncrawl.org/web-graphs
[3]https://commoncrawl.org/get-started

```
7134    85449358
7136    28631629
7137    4878317
7137    25871614
7137    32001750
7137    56001141
7137    76430226
7137    90387799
7137    92346054
7138    24376017
```

Listing 1: Excerpt from CommonGraph edges file

```
7134 academy.bazarsaz 1
7136 academy.bazino 1
7137 academy.bazium 1
7138 academy.bbb 1
4878317 bz.bzm 63
24376017 com.educational-bbb-academy 1
25871614 com.facebook 3700
28631629 com.google 13715
32001750 com.instagram 784
56001141 com.vk 2527
76430226 me.t 1187
85449358 org.gmpg 2
90387799 ru.bazium 310
92346054 ru.yandex 992
```

Listing 2: Lookup of the domain names of the excerpt from Lisitng 1 in the vertices file

By translating the IDs in Listing 1 to domain names using the vertices file (Listing 2), the actual domain names are obtained. Note that, as previously mentioned, they are in reverse domain notation, meaning one has to reverse them to get a functioning domain name. The third column in the listing represents the number of websites that have the link to that specific domain present on them [35].

2.3.2 Tranco list

Tranco is a domain ranking system that orders websites based on their popularity, designed to be more robust against manipulation and to provide a better list for research purposes through reliable and reproducible rankings [36]. The research done by V. Le Pochat et al. in [36] showed how the most common domain rankings, including Alexa, Cisco Umbrella, Majestic, and Quantacast, are all vulnerable to manipulation, in some cases as simple as making requests to a domain. This led to the creation of the Tranco list, which is an aggregation of Cisco Umbrella, Majestic, Farsight, Chrome User Experience Report, and Cloudflare Radar, with the different lists being weighted differently to create the Tranco list [36], [37]. This results in a

more stable list that is less prone to malicious outside influence. In contrast to many of the aforementioned lists, the Tranco list is permanently accessible, allowing for reproducibility [36].

2.3.3 Sucuri SiteCheck

Sucuri SiteCheck is a malware scanner developed by Sucuri, a subsidiary of the internet giant GoDaddy [38]. By inputting a URL, they scan the inputted website and give input on its security. Such as IDN support, TLS recommendations, defacements, and spam detection [39]. The most interesting for this thesis is spam injection, as this includes the detection of SEO injections that are done invisibly [40].

2.4 Related work

In this section we will dissect a scanning tool said to be capable of detecting hidden links developed by Junjie Xiong, et al. in the paper "Assessing the effectiveness of crawlers and large language models in detecting adversarial hidden link threats in meta computing", as well as briefly discussing a patent held by Google in the area.

2.4.1 Detection of hidden links using LLMs and a custom framework

One of the related works found is an article written by Junjie Xiong, et al. titled "Assessing the effectiveness of crawlers and large language models in detecting adversarial hidden link threats in meta computing" [3]. Their article specifies two different approaches for finding hidden links, one by having LLM models analyze the HTML code for webpages. However, this method did not yield high detection rates, with the highest detection for a specific hiding method of only 19.9%, an average of 13.4%, and a lowest of 7.2%. Due to this, the authors of the article instead developed a framework that works with static code analysis and visual inspection, which, according to their results, improved the detection ratios. To ease the explanations going forward, we have chosen to give the framework the name Meta Computing Hidden Links (MCHL).

Although the MCHL framework appears to share similarities with the method presented in this thesis, it is only described at a high level in the article and lacks important technical details. The authors claim that the MCHL framework can detect hidden links on major, reputable websites, yet they do not provide any concrete examples or evidence to support this. To gain further insights, we contacted Junjie Xiong, et al. and were granted access to the Python-based tool implementing the MCHL framework. However, access to the dataset was not provided, despite explicitly asking for it multiple times, making it impossible to verify their claim of identifying 84 hidden links on the 200 top-ranked websites, in areas such as shopping, lending, loans, credit, and gambling.

Upon reviewing the code for the MCHL tool, several flaws in its detection method became evident. The methodology of the MCHL tool is as follows, first it checks if

an element has the CSS attributes font-size or opacity set to zero, then it checks if it is an internal link, checks for characters inside of <a> tags, and finally if none are found, it captures a screenshot to perform a standard deviation analysis of the pixel colors, in an attempt to infer visibility based on a threshold.

However, this approach has several limitations. For instance, it fails on the deprecated but still occasionally used elements like the <marquee> tag with extreme speed, as the example below:

```
<marquee style="width:3px; height:1px" scrollamount="2571">
    <a href="https://example.com/marquee">Marqueeeeeiiii</a>
</marquee>
```

On top of this, the MCHL tool does not handle GIFs, images, or SVGs separately, resulting in these elements being processed by the same checks used for text. This can lead to false positives, depending on the complexity of the images, as a single color or a simple pattern may trigger the standard deviation analysis. Additionally, the tool lacks support for handling submenus or pages that render different HTML content based on screen size, such as mobile views, which contributes to more false positives.

Another issue lies in the internal link filtering logic, as the MCHL tool for distinguishing internal and external links is flawed, as it only filters out exact matches to the source URL or URI fragments (such as example.com#contact_us), thereby failing to account for subpages (such as example.com/about_us).

Most critically, however, are the checks for font-size and opacity, as the creators of the MCHL tool incorrectly wrote the if-statements. The if-statements expect the browser to return the CSS-properties in the format "font-size:0", but it returns them with a space after the colon "font-size: 0". As a result, these are never executed. Even if they were to be correctly executed, it would still result in issues, as they are using the "in" condition for checking the properties. Meaning a decimal fraction like "0.875rem", or "0.9px", etc., would be classified as hidden due to it containing the string. Moreover, the script checks for font-size two times, in two separate if-statements, as well as still running the standard deviation analysis even if it matches one of the opacity or font-size checks, leading to the possibility of a single <a> tag being marked as both visible and hidden.

2.4.2 Google patents

Google has and is most likely doing work within the same field as this thesis, as seen by the patent [2]. The patent explains that websites can contain hidden text and hidden links to boost the ranking of the website in the search results. It then proceeds to describe a framework to detect these hidden elements, if anything is deemed to be hidden, the elements will be ignored when indexing for ranking. The patent does not include any technical implementation, but describes aspects considered and the general evaluation method for said aspects. The described method is to create a tree structure of all the elements on a webpage, then iterate over all the elements starting at leaf nodes, and traversing upwards to, for example, compare color contrast.

3

Method

To determine if malicious non-interference is a suitable method for detection, this thesis outlines and creates a tool based on malicious non-interference for detecting hidden links on webpages. Based on our research on different hiding methods, this chapter defines the different detection modules required for successfully detecting and filtering out links on a webpage to determine whether it is a hidden link or not. The modules found by us to be necessary are text-based detection, image-based detection, and SVG-based detection.

To find methods of hiding, we started with a small initial dataset of around ten websites containing hidden links, provided by our supervisors. The links present on these websites were extracted. To expand this dataset, we developed a tool that used Common Crawl Web Graphs to find additional websites where the same links appeared. MANIS was evaluated by comparing its performance against other scanners that can detect hidden links. This evaluation was carried out by running MANIS on two distinct datasets, with a sample of domains from both datasets manually verified.

In the following sections, the method for expanding the first initial dataset with Common Crawl Web Graphs is explained in Section 3.1. Thereafter, the information gathering process is described together with some examples of hiding methods found during the information gathering phase, in Section 3.2. Subsequently, the selection process behind selecting Selenium as the WebDriver to control MANIS is described in Section 3.3. Following this, in Section 3.4, a description of the outlines for MANIS' detection modules is provided. Afterwards, in Section 3.5, the process and datasets used for gathering the results presented in Chapter 5 are introduced. Finally, in Section 3.6, the computers used for running the tests are presented for transparency.

3.1 Backlink finder

Backlink finder is a tool that utilizes Common Crawls Web Graphs[1], to form a set of links, and find other websites that have the same links on them. This is useful for finding more occurrences of a specific hidden link, as the hiding methods used might be different from website to website. There is also the possibility of finding new hidden links, as there might be new hidden links on another domain.

[1] https://commoncrawl.org/web-graphs

The outline of the backlink finder:

1. Load vertices and edge datasets from Web Graphs.
2. Find the IDs of all the target domains through the vertices file.
3. Check for all occurrences of the target domain ID in the edges file, and note all the matches.
4. Find the domain from the IDs of the matches through the vertices file.

The output from the backlink finder, as visually represented in Figure 3.1, is all the domains where the searched links appear. This can be used both as new websites to check for hidden links on, but also as a way to determine how often these URLs appear as backlinks. The downside of the backlink finder is that it does not give the exact path to the site containing the URL, it only gives the base domain. This limit is caused by Common Crawl and how they have chosen to store the data. This means you have to find the full path on your own, either through crawling or by looking through the WARCs Common Crawls saved at the same time.

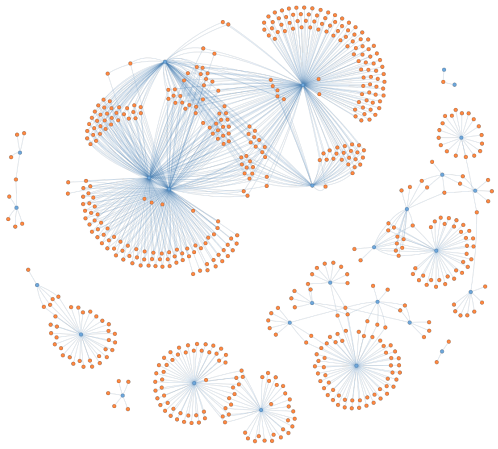


Figure 3.1: Output from backlink finder visualized. The blue nodes (at the center of the blue lines) are websites that contain links to the orange nodes.

Figure 3.1 visualizes a small part of the results from the backlink finder. The blue nodes, that is to say, the center of all the blue lines, are the nodes containing links to the orange nodes. As seen in the figure, there are multiple clusters of websites, some blue nodes have links to many websites, whilst some contain links to fewer than ten other websites.

3.2 Information gathering

The first step of the thesis focused on building a large set of unique hiding methods for hidden links. This included expanding the dataset that was presented as a starting point of the thesis from around ten links to a couple of thousand. The expansion was achieved by identifying hidden links on the original webpages and using a backlink

finder to discover related sites. This process was repeated for some of the newly found sites that appeared promising. Throughout this work, unique patterns were recorded to help guide the development of methods for detecting hidden links.

Examples of signatures found:

Example 1:

```
<div style="height: 0px; width: 0px; overflow: hidden;">
    <a href="example.com/ImHidden">Example</a>
</div>
```

Explanation: The styling attributes make the links invisible by setting the dimensions of the a-element to zero and preventing any eventual overflow from being shown.

Example 2:

```
<div style="display:none;">
    <a href="example.com/ImHidden">Example</a>
</div>
```

Explanation: The styling attribute removes the element from the page entirely, thereby making the a-element impossible to see.

Example 3:

```
<div style="position:absolute;left:-1919px;top:-4494px;">
        <a href="example.com/ImHidden">Example</a>
</div>
```

Explanation: The styling attributes position the element far outside of the user's browser window, making it impossible to see.

Example 4:

```
<marquee style="width:2px; height:3px" scrollamount="2939">
    <a href="example.com/ImHidden">Example</a>
</marquee>
```

Explanation: The marquee element continuously moves its contents horizontally, with the scrollamount attribute controlling the speed. However, the speed is so high that the element becomes invisible to the users. If some browser or device happens to make it visible, the small size makes it extremely hard to spot.

Example 5:

```
<div id="FooterLinks">
    <a href="example.com/ImHidden">Example</a>
</div>

<script type="text/javascript">
    if(!navigator.userAgent.match(/Google Web Preview/i))
        document.getElementById("Footer" + "Links")
            .style
            .display="no" + "ne";
</script>
```

Explanation: The JavaScript code hides the element depending on the user-agent of the visitor. For a normal user, the links are hidden due to "display=none", which removes the element from the layout. However, if a user with the user agent Google Web Preview visits, then the links are visible.

Example 6:

```
<div id="xingxie">
    <a href="example.com/ImHidden">Example</a>
</div>

<script>
    eval(function(p, a, c, k, e, d) {LONG OBFUSCATED JS CODE})
</script>
```

Explanation: The JavaScript code is obfuscated but can be deobfuscated using UnPacker[2]. This reveals that it sets the display attribute to none for id xingxie, which removes the element from the layout.

3.3 Selection of WebDriver

To accurately represent what a user sees when visiting a website, a WebDriver is needed. It should also be possible to interact with the WebDriver using a programming language to be able to add the necessary logic to work with malicious non-interference. For this, there are four alternatives available: Playwright[3], Selenium[4], Puppeteer[5], and Cypress[6].

[3]https://playwright.dev
[4]https://www.selenium.dev
[5]https://pptr.dev
[6]https://www.cypress.io

Although a comparison between these tools could have been conducted to determine which would be most beneficial for this thesis, it was quickly decided that integration with existing projects developed at Chalmers Security & Privacy Lab[7] was of higher priority. Therefore, Selenium was selected. Previous projects that have utilized Selenium include Black Window, a web crawler that tries to mimic user behavior to detect stored XSS [41], Black Ostrich a string constraint solver built ontop of the Black Widow crawler [42], and Blackmap, a SQL injection detector built on the crawler from Black Widow and sqlpmap [43].

Even though the selection of Selenium was made based on what is used for other research projects at Chalmers Security & Privacy Lab, the choice can also be motivated by it being one of the oldest tools for web testing, ensuring a wide range of documentation was available to support the work [32]. Additionally, it supports multiple browser drivers, allowing for easy switching if any limitations were to arise with any of the drivers. Furthermore, Selenium also supported programming languages that both authors of the thesis had previous experience with, in this case, Python.

3.4 Malicous Non-interference Scanner (MANIS)

The information gathering resulted in around 30 unique signatures being found, all of which would need to be handled. Most of the ones found were quite straightforward CSS styling attributes that made them completely hidden, but there were also signatures that matched the color of the link to the color of the background. JavaScript was also used in some cases to change attributes dynamically.

Even though none of these were for images or SVGs, it is technically possible for many of the hiding methods used for hiding text to be used for image and SVG tags. For this reason, separate handling of both image tags and SVG tags are necessary. As they require altered approaches to be successfully handled. The general outline for MANIS, can be seen in Figure 3.2.

In the following sections below, the general idea for the modification of the different elements are described.

3.4.1 Text detection

For detecting hidden links present in text, the initial research showed a couple of different scenarios that would need handling. As there are benign (visible) links present, the method will need to be able to identify whether something is visible or not.

As the method covered in this thesis is malicious non-interference, the idea is that the link should be changed to something new, and then the results before and after are compared. Therefore, a suitable approach was thought to be scrambling the text, as swapping the text to something new might result in the page layout being slightly

[7] https://www.cse.chalmers.se/research/group/security/

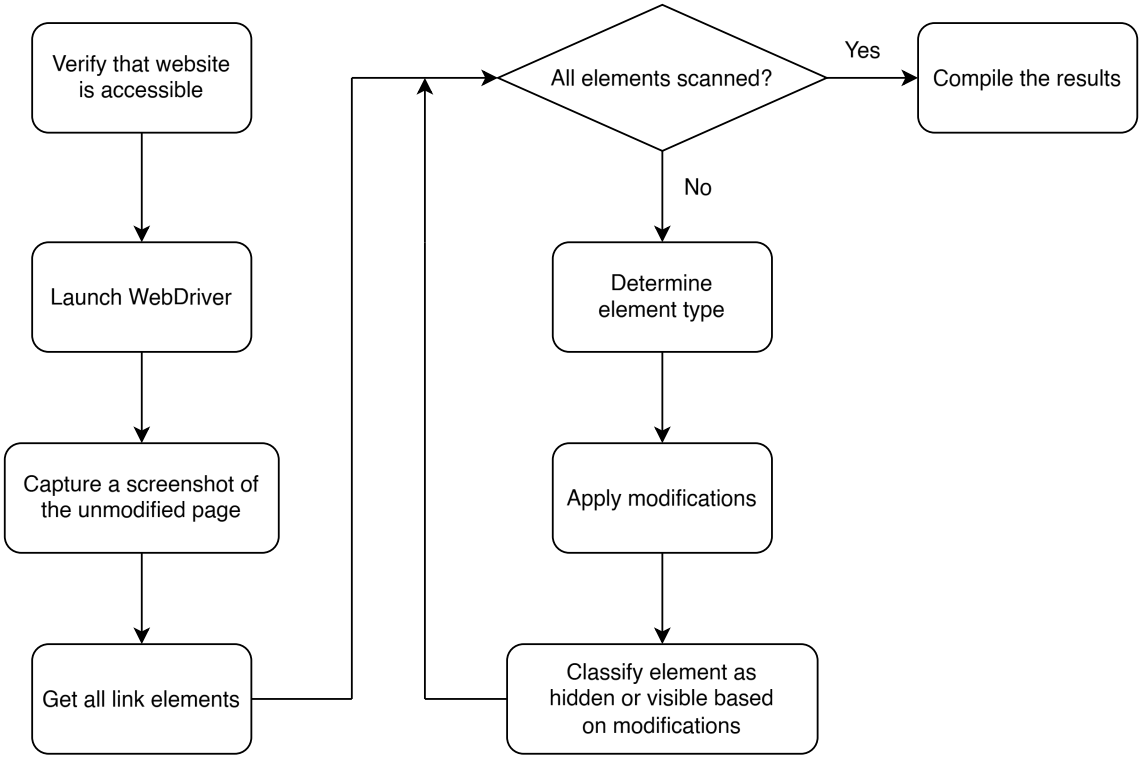


Figure 3.2: Flowchart of the general outline of MANIS.

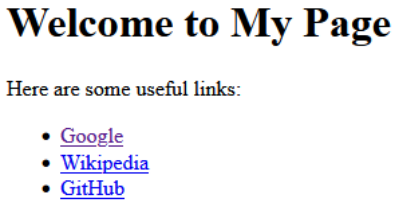
altered, thereby affecting detection. But this turned out to result in false positives for icon packs, which is why this was changed to modifying the opacity in the final version, which is demonstrated in Figure 3.3.

The information gathering also found some cases where the links where completely invisible to the human eye, but it was still possible to discern them by looking closely or comparing the pixels' color values. This meant that the program needed to be able to have some kind of human-like vision capabilities and not just go based on pure pixel values, as that could result in both false positives and false negatives.

3.4.2 Image detection

Images are common on websites and can be used as links as well. Theoretically, images can be created in a manner to hide links, and evidently, the images cannot be detected through text-based analysis. This makes them a potential vector for hidden links. Web images are commonly rendered in formats such as GIF, PNG, and JPEG. Each format has unique properties that need to be accounted for: JPEG is optimized for photographs and lacks transparency support, PNG supports full transparency, and GIF supports transparency and basic animations. These properties affect how images can be manipulated and detected during analysis.

To apply the concept of malicious non-interference, the images must be modified and analyzed. A naive approach would be to apply simple transformations only, such as flipping the image or shuffling the pixels, but this is unreliable as images can be symmetric or a plain color, which could lead to incorrect classification. Instead, our



(a) Webpage with links from a test case.

(b) Screenshot of visible link.

(c) Screenshot of visible link with opacity modifed.

(d) Screenshot of hidden link

(e) Screenshot of hidden link with opacity modfied.

Figure 3.3: Text detection method visualized from a test case in MANIS with both visible and hidden links.

method replaces the original image with a completely generated image with identical dimensions. When the original image is replaced by the generated image, a check can be performed to detect changes.

An additional complication arises with transparent images. A fully transparent PNG is obviously not visible to a user, but when replaced with the generated opaque image, the image is visible and therefore classified as such. To detect such cases, a fully transparent image is generated and inserted in the place of the original image. If the substituted transparent image yields no change in appearance, it is likely that the original was also fully transparent. This helps uncover instances where links are hidden behind non-visible images.

There are also cases of partially transparent images, commonly seen with products on shopping websites, which are subject to the same checks and will therefore be labelled accordingly. Finally, in the scenario of GIFs, which is a collection of images. Each image is checked the same way as separate images are, then if a strong majority of the images are hidden, the GIF is deemed hidden.

The multi-step approach used for image detection is robust and adaptable to a wide range of scenarios. An example of this detection process is illustrated in Figure 3.4. In Subfigure 3.4 (a), a webpage is shown where an image functions as a link. This image is then extracted, as shown in Subfigure 3.4 (b). Based on the properties of the original image, a replacement image is generated with the same dimensions, as seen in Subfigure 3.4 (c). Similarly, a transparent image is generated and inserted,

as seen in Subfigure 3.4 (d). In our experience developing MANIS, this approach effectively handles the majority of cases where images are used to conceal or embed links.

(a) Webpage with image from a testcase.

(b) Original image extracted from the webpage.

(c) Opaque generated image inserted into the webpage.

(d) Transparent generated image inserted into the webpage.

Figure 3.4: Image detection method visualized from a test case in MANIS.

3.4.3 SVG detection

SVGs are based on our research, primarily used for logos on websites. The most common being logos for social media such as Facebook, Instagram, X (formerly known as Twitter), and LinkedIn.

To apply the concept of malicious non-interference on SVGs, a similar method to the one used for images will be used. However, as SVGs can be built up of multiple paths with different colors, the idea is to apply different colors to different paths to minimize the risk of color overlap with anything else, both in the SVG tag, but also on the site.

As with images, an SVG can utilize transparent colors, meaning there is also a necessity to check whether the SVG is fully transparent or equal to the background. To do this, the color of the paths is changed to be transparent. Thereby being able to tell if the SVG was used to hide a link.

An example of this process can be seen in Figure 3.5. In Subfigure 3.5 (a), a webpage containing a link clickable by pressing on the house is shown. The house is then extracted to be handled by the checks described above, as seen in Subfigure 3.5 (b). First, the colors are randomized and inserted into the SVG as seen in Subfigure 3.5 (c). Then the colors are changed to transparent, as seen in Subfigure 3.5 (d). In our experience, these should effectively handle the majority of the cases where SVGs are involved.

(a) Webpage with SVG from a test case.

(b) Original SVG extracted from the webpage.

(c) Randomized colored SVG inserted into the webpage.

(d) Transparent colored SVG inserted into the webpage.

Figure 3.5: SVG detection method visualized from a test case in MANIS.

3.5 Evaluation

To answer the research question in the thesis, the developed tool is run on two different datasets to gain an understanding of the accuracy of MANIS, and how common hidden links are on the web. The result from MANIS is compared to Sucuri SiteCheck.

The two datasets used for evaluation come from two different sources. One by randomly selecting a set amount of links from the latest Tranco list, and one from websites found during the initial manual evaluation at the start of the thesis, known to contain maliciously hidden links. From these, a percentage of the evaluated websites

are manually checked to verify the reported number of visible and hidden links to determine the ground truth. The following definitions are used for classification:

- **True Positive (TP)**: A link marked as hidden that is not visibly found anywhere on the website.
- **False Positive (FP)**: A link marked as hidden that is visible on the website.
- **True Negative (TN)**: A link marked as visible that is indeed visible on the website.
- **False Negative (FN)**: A link marked as visible that cannot be found visibly on the website.

The definition of visible above means that it should be possible to find the element. Meaning that a link is counted as visible even if it is five levels deep in a menu. These classifications are then used to evaluate the accuracy of the developed tool.

Comparing the developed tool to Sucuri SiteCheck is done in a simplified manner. It is compared in a binary fashion, meaning if it detects hidden links on the website or not, it gives a 0 or 1. Meaning there could theoretically be a difference in the detection capabilities, not noted, as there can be a discrepancy in the number of found links.

3.5.1 Dataset - Tranco random

The dataset from Tranco random was generated from the full Tranco list on the 28th of April 2025[8]. From this, a set of 20 000 links was randomly sampled, links were then verified to be alive by performing an HTTP request and DNS lookup, and verifying if a 20X or 30X response code was given and that an IP was returned in the DNS lookup. The verification was performed until 10 000 websites are deemed to be alive. These 10 000 websites were then scanned with MANIS. Subsequently, 100 (1%) websites were manually checked to verify the number of hidden and visible links found by MANIS, as described previously. If any of the sampled 1% were dead, they were rerun a few days later to account for possible crashes caused by the mass testing and websites temporarily being down.

3.5.2 Dataset - suspected hidden

The suspected hidden links dataset was a dataset gathered and expanded during the development of the thesis. The initial version of the dataset was manually collected from websites that contained hidden links. The links hidden on those websites were extracted and used in the backlink finder to find more victim domains where these links have at some point existed. As the backlink finder only finds the domain of the website and not the specific page with the hidden links, a shallow crawl is performed in an attempt to find the specific webpage where the link is present.

Shallow crawling refers to the process of web crawling with a depth of one, where the crawler analyzes the first page, and then a small predefined number of directly

[8]https://tranco-list.eu/list/PN46J/full

linked subpages, in this case, just one. This approach was chosen over using WARCs, as mentioned earlier, due to limitations in the Common Crawl infrastructure. Based on first-hand experience, accessing WARCs through Common Crawl can lead to throttled traffic, making specific WARCs intermittently unavailable even across identical executions. Another benefit of shallow crawling is that it allows the analysis of the live version of a website, rather than relying on an archived copy.

This dataset started out as around 56 000 links, which was reduced down to around 22 000 after removing all the non-unique ones. After shallow crawling for looking for specific links on the 22 000 websites, around 6 500 websites remained. This dataset, consisting of 6 500 links, is then used for evaluating MANIS' detection capability when knowing there is a high probability of there being hidden links on the page. From these, a 1% sample (66 links) was manually evaluated, to determine the detection capabilities. If any of the 1% are dead, a new attempt at scanning them with MANIS was done a few days later, to handle possible issues caused by computer issues, networking issues, or just the website being temporarily down.

3.6 Experimental setup

In Table 3.6, the specifications for the computers used to run the evaluation datasets are presented.

Table 3.1: Computer specifications.

	Computer 1	Computer 2
CPU	Intel i7-14700k	AMD Ryzen 7 5800X3D
RAM	DDR5 32 GB	DDR4 32 GB
OS	Windows 11	Windows 10
Python	3.13.1	3.13.1

As seen in Table 3.6, the specifications for the two computers used are presented. Important to note is that both are running Windows, however, the operating systems differ, but this is deemed to be of no importance as Windows 11 is basically just a skin on top of Windows 10. More importantly, the Python versions are the same, which should mean that the conditions were as similar as possible. Ensuring at least that the OS and Python versions are the same should increase the likelihood of similar results if the research is to be replicated.

The computers ran one dataset each, with Computer 1 used to scan the suspected hidden dataset from Section 3.5.2 and Computer 2 used to scan the Tranco random dataset from Section 3.5.1.

4

Implementation

To reiterate the main goal of this thesis, the aim is to evaluate a reversed implementation of non-interference for detecting hidden links dubbed malicious non-interference. Therefore, the detection of hidden links described in this thesis can be achieved using other technologies and methods. Below, an example of an implementation of this is described in a pseudocode fashion to facilitate the idea of how the process works and what is needed for successfully detecting hidden links. Though, as it is done at a pseudocode level, parts of the implementation and thresholds are left out for the sake of simplicity. For the complete code, please check the project's GitHub repository[1].

This chapter is structured as follows: First, in Section 4.1, the pseudocode for all the different modules in MANIS is described. Afterwards, certain challenges related to the vastness of the Web are mentioned to provide context for the implementation limitations described in Section 4.2. Finally, some of the heuristics that are used to improve the results have their implementations shown in Section 4.3.

4.1 Scanning a website

Before scanning a page, it is first verified to be alive to avoid unnecessary operations being performed and to prevent Selenium from crashing due to an unavailable website. Once the page is confirmed to be accessible, Selenium is started and used to fetch the page.

Since Selenium does not automatically throw any errors if it has been redirected, it is necessary to verify that it always knows the URL of the page it is visiting. From this state, the page is screenshotted in its entirety by first scrolling the entire length of the page to make sure that everything is properly loaded before taking the screenshot. Then, all the elements that contain HREFs are fetched and returned in an array that will be iterated over to process all the elements one by one.

The software iterates over the HREF elements, that is to say, all elements that contain links. There is a check first to verify that the link is an external link, which is a link that refers to a different domain than the one currently being processed. The element type is then checked to determine whether the element containing the

[1] https://github.com/jonathan-lindqvist/malicious-non-interference

HREF is one that MANIS is designed to handle. If it is a relevant element, it is then processed based on its type: text (Section 4.1.1), image (Section 4.1.2), or SVG (Section 4.1.3).

A pseudocode version of the given description can be found in Listing 3.

```
checkIfAlive(websiteToGet)
driver = driverSetup(websiteToGet)
driver.setUserAgent("Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
↪  (KHTML, like Gecko) Chrome/134.0.0.0 Safari/537.3")
websiteToGet = checkIfRedirected(websiteToGet)

scrollPage(driver)
fullpageScreenshot(driver)

elements = getAllHREFSElements(driver)

for element in elements
    if isInternalLink(element.href, site.url)
        continue

    elementType = getElementType(element)

    if elementType == "text"
        # See Section 4.1.1

    elif elementType = "image"
        # See Section 4.1.2

    elif elementType = "SVG"
        # See Section 4.1.3
```

Listing 3: High-level outline of the hidden link detection algorithm.

4.1.1 Scanning text elements

For applying malicious non-interference on text, there are a couple of checks necessary to minimize the risk of false positives. But before doing any of these, a baseline of the unmodified element must be established. To do this, the element is screenshotted by first retrieving the bounding box of the element, then capturing a full-page screenshot, and cropping it down to the coordinates from the bounding box, as shown in Subfigure 3.3 (b). It is also necessary to disable any eventual transitions being done to the text. As transitions make the changes to CSS attributes take longer to come into effect, meaning the screenshots may otherwise be incorrect. For example, the opacity may only have reached 50% at the time of screenshotting instead of the desired 0%.

The first step for applying malicious non-interference is by modifying the opacity of the text, which can be observed in Subfigure 3.3 (c). To do this, the opacity is first saved so that it can be restored after the modifications to allow for further testing. Then, the opacity is set to zero for the element and potential child

elements that are commonly associated with icon packs, such as Font-Awesome[2] and Material Icons[3], like span and i tags. After this, a screenshot is taken by using the same coordinates and method as for the baseline screenshot. The baseline and opacity screenshots are then compared to see if there are any differences on a pixel-by-pixel basis, if no difference is noted, the text element is marked as hidden.

The next step is to check if the text is nearly indistinguishable from the background, which is done with SSIM. The reason for using SSIM over other perceptual detection methods, such as a pHash, is that it is easier to determine how similar something is based on a percentage rather than an amount of bits.

First, the color of the text is saved and then set to be transparent, as seen in Subfigure 4.1 (b). From this, a screenshot of the element is taken using the same method as before. Since the bounding box for an element can be quite large, whilst the text inside is quite small in comparison (Subfigure 4.1 (a)), the resulting SSIM score will be quite high, as the area with change is so small.

To avoid this, the image needs to be cropped down to the modified area. This is done by comparing the blank screenshot to the base screenshot, which allows for the detection of the modified parts of the image. From this, the base and blank images are cropped down to the same dimensions of the modified area plus a small margin, as seen in Subfigure 4.1 (c, d). These are then used for the SSIM calculation of the similarity between them to determine if the color of the text is close to indistinguishable for human eyes (an example of this can be seen in Figure 6.1).

If it is indistinguishable, it is counted as hidden. The threshold for this is defined by what has been proven to work in our testing and not by a scientific report on human vision. The color is set to transparent rather than a random color to ensure that SSIM detection compares the text against a background as if the text were absent. Using transparency minimizes interference with any other attributes of the element. Setting the opacity to zero, which would also make any eventual text background colors disappear, which is undesired, as it would remove more than just the text.

(a) Before checks.

(b) Blanked.

(c) Base Cropped.

(d) Blanked Cropped.

Figure 4.1: Image indistinguishability steps visualized with images from MANIS.

If the cropping during the blank detection fails, it most likely means that no contours were detected. This commonly occurs with icon packs, such as Font Awesome. When

[2]https://fontawesome.com/
[3]https://fonts.google.com/icons

this happens, the system calculates the percentage of pixels that have been modified. If fewer than X percent of the pixels have changed, additional checks are done. The first check determines whether the base and blank images are identical. If they are not, the base image is then compared to the opacity image. If these two images differ, the image is likely an icon, and the element is marked as visible.

It is possible to apply opacity to an icon, but the color attribute (blank detection) does not work. However, if the opacity and blank images are identical, the element is marked as hidden. Since that means MANIS did not detect any changes between the three checks, meaning that most likely the element is hidden. If it passes all the checks, the text element is deemed visible, and MANIS moves on to the next element. The pseudocode version of the given description can be found in Listing 4.

```
boundingBox = getBoundingBox(driver, element)

basePNG = screenshotElement(driver, boundingBox)

transition = getTransition(driver, element)
setTransition(driver, element, 0)

opacity = getOpacity(driver, element)
setOpacity(driver, element, 0)
opacityPNG = screenshotElement(driver, boundingBox)
setOpacity(driver, element, opacity)
if areImagesIdentical(opacityPNG, basePNG)
    hidden += 1
    break

color = getColor(driver, element)
setColor(driver, element, transparent)
blankPNG = screenshotElement(driver, boundingBox)
setColor(driver, element, color)
setTransition(driver, element, transition)
ssimScore = calculateCroppedSSIM(blankPNG, basePNG)
if ssimScore <= 0.9 and ssimScore >= 0.7
    hidden += 1
    break
elif ssimScore == -1
    pixelDifference = calculatePixelDifference(basePNG, blankPNG)
    if pixelDifference <= 0.05
        if areImagesIdentical(blankPNG, basePNG)
            if areImagesIdentical(blankPNG, opacityPNG)
                hidden += 1
                break

            # Element is probably an icon
            visible += 1
            break

visible += 1
```

Listing 4: High-level outline of the text detection algorithm.

4.1.2 Scanning image elements

When working with images, there are a couple of things to take into consideration. First, different kinds of images may require different implementations to work correctly. To handle this, MANIS first retrieves the image, its type, and the corresponding image element by identifying which image element contains the tag with the HREF. Based on the image type, the appropriate handling method is selected. If the image is an SVG, it is first converted to a PNG. Then the image element is handled either as a GIF or using the default image handling module, both are explained later in this Section. A pseudocode version of the given description can be found in Listing 5 below.

```
img, imgType, imgElement = getImage(driver, element)

if imgType == "SVG"
    img = convertSVGToPNG(img)
    imgType = png

if imgType == "gif"
    # See Listing 6
else
    # See Listing 7
```

Listing 5: High-level description of the image type algorithm.

Handling GIFs

For handling GIFs, MANIS simply sees them as multiple images and handles them one by one. This means MANIS iterates over all the frames in the GIF, inserting them one by one into the image frame and running the same detection as the one described previously. Then, to determine if the GIF is hidden or not, it uses a threshold that if X amount of frames are detected as hidden, the element is marked as hidden, otherwise, it is marked as visible. A pseudocode version of the description given can be found below in Listing 6.

```
hiddenFrames = 0

boundingBox = getBoundingBox(driver, imgElement)

for frame in gifFrames
    insertImage(driver, imgElement, frame)
    framePNG = screenshotElement(driver, boundingBox)

    oneColoredImage = createOneColoredImage(boundingBox)
    insertImage(drive, imgElement, oneColoredImage)
    oneColoredPNG = screenshotElement(driver, boundingBox)
    if areImagesIdentical(framePNG, oneColoredImage)
        hiddenFrames += 1
        continue

    transparentImage = createTransparentImage(boundingBox)
    insertImage(drive, imgElement, transparentImage)
```

```
    transparentPNG = screenshotElement(driver, boundingBox)
    if areImagesIdentical(framePNG, transparentPNG)
        hiddenFrames += 1
        continue

if hiddenFrames >= totalFrames * 0.9
    hidden += 1
else
    visible += 1
```

Listing 6: High-level description of the GIF detection algorithm.

Handling Common Image Formats

Working with normal images, that is, those not filtered out by the preliminary checks in Section 4.1.2, MANIS uses the default image handling method. Examples of common image formats include PNGs and JPEGs. First, the image retrieved is inserted to ensure that the image in the image element is static, and to be certain which image is being shown in the browser. The image is then screenshotted by first getting the bounding box of the image, then taking a full-page screenshot and cropping it down to the correct dimensions to have a baseline before any modification is performed.

The image is first checked to see if it is visible or not. This is done by creating an image with the same dimensions as the original image, with a solid random color from a color space of 16.8 million colors, which is then inserted into the image element. This is then screenshotted using the same coordinates as before, and by comparing this new image to the baseline using pixel-by-pixel comparison, MANIS will determine if it can see a modification or not. If it cannot detect a modification, it means that the image is hidden and is marked as such.

The second check is to see if the image is transparent. This is done by inserting a fully transparent image generated by MANIS. By inserting the transparent image and taking a screenshot of it, and comparing it to the baseline, it is possible to determine whether the image present on the website is just a fully transparent image used to hide the link. If no change is detected between the transparent image and the baseline, it means that the image was transparent. It is therefore impossible for the user to see, and the element is marked as hidden. If the image instead passes all the checks, the image is marked as visible. A pseudocode version of the described checks can be found in Listing 7.

```
insertImage(driver, imgElement, frame)

boundingBox = getBoundingBox(driver, imgElement)
basePNG = screenshotElement(driver, boundingBox)

oneColoredImage = createOneColoredImage(boundingBox)
insertImage(drive, imgElement, oneColoredImage)
oneColoredPNG = screenshotElement(driver, boundingBox)
if areImagesIdentical(framePNG, oneColoredImage)
```

```
10      hidden += 1
11      continue
12
13  transparentImage = createTransparentImage(boundingBox)
14  insertImage(drive, imgElement, transparentImage)
15  transparentPNG = screenshotElement(driver, boundingBox)
16  if areImagesIdentical(framePNG, transparentPNG)
17      hidden += 1
18      continue
19
20  visible += 1
```

Listing 7: High-level description of the common image format detection algorithm.

4.1.3 Scanning and handling of SVGs

Applying the concept of malicious non-interference to SVGs is tricky, as there are many different ways SVGs can be used on websites and many different subtags that all require special handling, such as loading the SVG from external sources, having the necessary information directly inline, or having multiple SVGs in a single file with SVG sprites. The implementation below in Listing 8 covers some of the encountered cases, but due to the flexibility of the format, it is possible that it does not handle some edge cases.

It begins by first fetching the SVG, which is done in a couple of different ways due to the flexibility of the SVG tag in the HTML code. Either it can be stored directly inline, or in a use tag with a location, either on the same website or on an external website. If a use tag is used, the SVG must be fetched and inserted into the code to allow for processing in the same pipeline as an inline SVG. There is also a need to handle SVG sprites, which is a special format within an SVG that allows for storing multiple SVGs within a single SVG file, and displaying the desired one by appending a hashtag and the element ID to the file name. To work around the limitation, the desired SVG is extracted from the SVG sprite before inserting it back onto the website.

Once the SVG has been embedded in the website, the visual checks can be performed. First, the bounding box is fetched, and a screenshot is taken and cropped to the dimensions of the SVG. From the SVG tag, the content of the SVG is fetched to allow for modification. The first modification involves randomizing the colors of the SVG from a color space of 16.8 million colors. The modified SVG is then reinserted into the website, and a new screenshot is taken and compared to the original. If no visual change is detected, the element is marked as hidden.

The next check is to see if the SVG is fully transparent or has colors that match the background. This is done by setting the color of the SVG to be fully transparent. As this would make the SVG match the background, if the SVG happened to be transparent. A screenshot is taken and compared to the one taken before any modifications were made. If no change is detected, it is marked as hidden, otherwise, the element is marked as visible and the SVG is reset to the saved SVG.

```
svgElement = getSVG(driver, element)

useElement = getUseElement(driver, svgElement)
visualElements = getVisualSVGElements(driver, svgElement)

if useElement and len(visualElements) == 0
    svgLink = useElement.getLink(driver)

    # Handle internally stored SVG Sprites
    if svgLink.startswith("#")
        referencedElement = getElement(driver, useHREF[1:])
        referencedSVG = getOuterHTML(referencedElement)

        if referencedSVG
            replaceSVG(driver, referencedSVG)

    # Handle externally stored SVGs
    else
        # Handle external SVG Sprite
        if "#" in svgLink
            svgLink, elementID = svgLink.split("#", 1)
        else
            svgLink, elementId = svgLink, None

        response = httpRequest(svgLink)
        if response.code == 200
            svgContent = response.text
            xmlSVG = parseXML(svgContent)

            # SVG Sprite
            if elementID
                newSVG = findSVGinXML(xmlSVG, elementID)
                if newSVG
                    newSVG = convertToSVG(newSVG)

                    parentElement = findParentElement(driver, useElement)
                    removeElement(driver, useElement)
                    insertElement(driver, parentElement, newSVG)
                else
                    break
            # SVG "Normal"
            else
                newSVG = findSVGinXML("svg")
                if newSVG
                    parentElement = findParentElement(driver, useElement)
                    removeElement(driver, useElement)
                    insertElement(driver, parentElement, newSVG)

        svgElement = getSVG(driver, element)

boundingBox = getBoundingBox(driver, imgElement)
baseSVGPNG = screenshotElement(driver, boundingBox)

svgContent = getSVGContent(svgElement)

```

```
randomColoredSVG = randomizeSVGColors(svgContent)
updateSVG(driver, randomColoredSVG)

randomSVGPNG = screenshotElement(driver, boundingBox)
if areImagesIdentical(baseSVGPNG, randomSVGPNG)
    hidden += 1
    break

transparentSVG = transparentSVG(svgContent)
updateSVG(driver, transparentSVG)

transparentSVGPNG = screenshotElement(driver, boundingBox)
if areImagesIdentical(baseSVGPNG, transparentSVGPNG)
    hidden += 1
    break

updateSVG(driver, svgContent)
visible += 1
```

Listing 8: High-level description of the SVG detection algorithm.

4.2 Implementation limitations and design choices

In our experience and based on what we saw during the development of MANIS, the Web is constantly evolving, with new standards and technologies being adopted at a high pace. However, many websites lag behind, continuing to use outdated and deprecated approaches.

One such example is the marquee tag[4] as shown in Section 3.2 Example 4. This made us realize that covering all the corner cases would be an extremely time-consuming task, possibly never-ending. Therefore, we had to make our approach more generic and decide what cases were relevant to cover, as most of them were just a question of a couple of engineering hours needed to make it work. Therefore, it was decided that if we only find a few instances of a deprecated technology being used, it is not worth covering it with the implementation done in the thesis. But the decision might not always be that simple. However, with the case of the marquee tag, it turned out to work with the standard detection methods already implemented in MANIS.

There are also tags that are not deprecated but that are primarily used by older websites. An example of this is the area tag, which allows for the definition of a clickable area, like a link, on top of an image that has a user-defined shape. The current iteration of MANIS, leaves this as future work, as the handling of custom shapes means it will require a substantial implementation to be handled correctly.

[4]https://developer.mozilla.org/en-US/docs/Web/HTML/Element/marquee

4.3 Heuristics

To effectivise the scanning and increase the accuracy, a couple of heuristics are applied during the execution of MANIS, and some can be applied afterwards to filter the results. The reason why not all are applied afterwards is due to optimizing the execution time of MANIS. The primary heuristics that are used in MANIS are: filtering internal links, filtering out seen domains, and filtering top-ranked domains.

4.3.1 Filter internal links

Filtering out internal links is a heuristic that is applied to increase the speed of MANIS. A website should not be boosted in the SE based on how many times its own link appears on its own website, but rather how often it appears on other websites. Thereby filtering it out both increases the speed of MANIS, as it no longer has to work with irrelevant links, but it also removes any possible false positives caused by internal links. The implementation for filtering internal links can be seen in Listing 9 below. The function described is used in Listing 3.

```
isInternalLink(elementURL, siteURL):
    domainElement, tldElement = extractURL(elementURL)
    domainSite, tldSite = extractURL(siteURL)

    return domanElement == domainSite and tldElement == tldSite
```

Listing 9: High-level description of the filtering algorithm for internal links.

4.3.2 Filter seen domains

A filter is applied to exclude links that have previously been determined to be visible, if they later appear as hidden. This serves as a countermeasure against websites that display different layouts based on screen size. As it is not uncommon for websites to have different views, for example, one for mobile phones and one for desktop, resulting in duplicated links. This leads to one of the views never being visible to MANIS, and thereby, all the links are false positives unless filtered out. Listing 10 shows the implementation of the algorithm described.

```
visibleDomains = []

# Every time something is deemed as visible
visibleDomains.append(element.href)

# At the end, before writing logs, etc.
visibleDomains = normalizeURLs(visibleDomains)
element.url = normalizeURLs(element.url)

# This code should be placed at the end of Listing~3
for element in elements
    if element == hidden
        if element.url in visibileList
            hidden -= 1
            visible += 1
```

Listing 10: High-level description of the filtering algorithm for seen domains.

4.3.3 Filter most popular

The heuristic to filter out the most popular domains is applied afterwards. This is done to avoid influencing the baseline accuracy. Filtering them out at a later stage also enables an understanding of how frequently they are marked as hidden. The reason it is interesting to filter them out is that top domains should, in theory, never be invisibly injected onto websites as a way to boost SE rankings, as they are already among some of the most visited websites, meaning that most likely the reason behind why they are marked as hidden is more likely due to them being located in submenus, on mobile views, missing handling of a specific icon, or in some cases, simply left commented out with some CSS used to hide links.

An implementation to filter out the most popular domains is shown in Listing 11. The filtering process begins by reading the top X entries from a Tranco list, where X is a configurable number of domains (e.g., 100, 200, 1000, or 2000). The Tranco list provides a ranked list of the most popular domains. Next, the log files generated by MANIS are processed one by one. For each CSV file, the script examines the visibility column to identify links marked as hidden. If a hidden link is found, its URL is checked against the list of top domains. If the URL belongs to a top domain, it is counted as filtered; otherwise, it is counted as unfiltered. This allows us to measure how many hidden links originate from highly popular domains.

```
folderToFilter = "folder_name"
trancoCSV = "tranco_list.csv"
unmatchedDomains = "unmatchedDomains.csv"
matchedDomains = "matchedDomains.csv"

topDomains = readLinesFromCSV(trancoCSV, 100)

for folder in folderToFilter
    if csv in folder
        csvFile = csvReader(csv)
        for row in csvFile
            if row["visibility"] == "hidden"
                totalHidden += 1

                if row["URL"] in topDomains or row["URL].endswith("." +
                ↪  topDomains)
                    filteredHidden += 1
                else
                    unfilteredHidden += 1
```

Listing 11: High-level description of the filtering algorithm for seen domains.

5
Results

In this chapter, we present the results obtained from our large-scale evaluation, executing MANIS on thousands of real-world websites, following the methodology described in Section 3.5. In Section 5.1, the run against a 10 000 random sample from Tranco as outlined by Section 3.5.1 is presented. Section 5.2 presents the results from running against around 6 500 websites suspected to contain hidden links. Then, in Section 5.3 we will present some interesting cases found and dissect them, and lastly some statistics from the scan are presented in 5.4.

5.1 Results from testing on the Tranco random dataset

The results from running MANIS on the 10 000 links randomly selected from Tranco, a total of 138 344 links were evaluated. Among them 73 808 links were labeled as hidden links, meaning around 53% of the links scanned were deemed as hidden. Figure 5.1 shows how many hidden links were present on the websites that contained the most hidden links, to give perspective into the distribution of hidden links among the websites in the dataset. For a graph that shows the entire distribution, please check Figure C.1 in Appendix C.

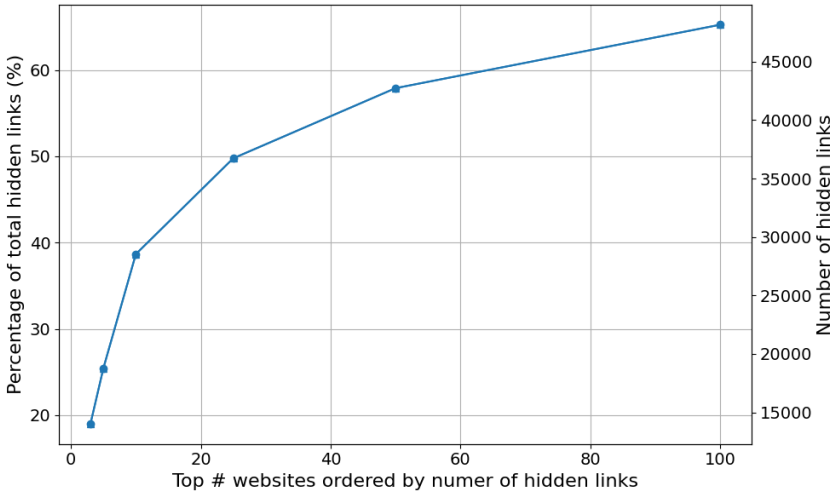


Figure 5.1: Distribution of hidden links from the Tranco random dataset.

As Figure 5.1 shows, a large percentage comes from the top websites. With close to 50% of the hidden links coming from the 25 websites that contained the most amount of hidden links. This highlights the fact that a lot of the found links come from a small number of websites. Looking at Figure C.1 in the appendix, one can note that somewhere around 3 500 websites contain hidden links, with the rest containing none.

When a filter to exclude popular websites is applied, aimed at minimizing false positives, as described in Section 4.3.3, the number of hidden links decreases, as shown in Table 5.1 below.

Table 5.1: Results of filtering the most popular domains from the links labeled as hidden in the Tranco random dataset.

Top # of domains	# of unique domains used to filter	# of filtered hidden	Percentage of total amount of hidden links
100	50	9 493	13.27%
200	91	10 510	14.24%
1 000	340	13 209	17.90%
2 000	440	13 787	18.68%

Table 5.1 shows that just 50 domains from the top 100 most popular domains are responsible for filtering 9 493 links. This accounts for approximately 69% of the total 13 787 links filtered when using the top 2 000 domains, meaning that a small set of popular domains contributes to the majority of the filtering effect. As the filter size increases, the number of filtered links continues to grow, but the average number of links filtered per domain decreases, from 189.86 with the top 100 to 31.33 with the top 2 000, indicating diminishing returns with larger filter sets.

As indicated by the applied filter, some domains are more common than others. Table 5.2 shows the top five domains used to filter, which are responsible for 8.4% of the total amount of hidden links.

As seen in Table 5.2, it is primarily the classic websites that appear, with Facebook, Google, and Instagram standing for the largest share. The reason for this is likely to be because these often occur as social media links for the website in question. From our observations, these often appear as SVGs, icons, or in menus, which we explore more in detail in Section 6.3.1.

Table 5.2: The top five domains used in the filter for the Tranco random dataset.

Domain	Number of times filtered
facebook.com	1 594
google.com	1 496
instagram.com	1 213
youtube.com	1 069
twitter.com	848

5.1.1 Results from Tranco random sample

From the original 10 000 websites, 1% were manually checked for TP, FP, TN, and FN. The results of the verification can be seen in Appendix B. In total, 969 links were manually checked, 434 of these were labeled as hidden, and 535 were labeled as visible. The 100 websites were also filtered the same way as the full sample, which can be seen in Tables 5.3 and 5.4.

To provide the most accurate result possible for the sample, any sites that were down on the original scanning day were rescanned a few days later. If the website were back online, its values were updated accordingly.

Table 5.3 shows a similar pattern to what was observed in Table 5.1, although on a smaller scale due to the reduced dataset size. When the top 2 000 most popular domains are used as a filter, 13 domains are responsible for about 79% of the filtering. However, it is important to remember that since it is a smaller set, the percentage is more sensitive.

Table 5.3: Filtering of top domains from the hidden links for the Tranco random 1% sample dataset.

Top # of domains	# of unique domains used to filter	# of filtered links
100	13	41
200	13	41
1 000	19	49
2 000	22	52

It is interesting to note that in Table 5.4, new domains are introduced as the top domains used, when compared to Table 5.2, which was the top domains for the entire dataset.

Table 5.4: The top five domains used in the filter for the Tranco random 1% sample dataset.

Domain	Number of times filtered
facebook.com	8
twitter.com	7
pinterest.com	6
instagram.com	4
microsoft.com	3

Figure 5.2 shows the results for the TP, FP, TN, FN in a confusion matrix. High values in the top left and bottom right quadrants indicate desirable outcomes, while low values are expected in the top right and bottom left quadrants. From these values, metrics such as true positive rate (TPR), false positive rate (FPR), false negative rate (FNR), true negative rate (TNR), and accuracy can be derived. The TNR for the sample is ≈ 0.79, the FPR is ≈ 0.21, the TPR and FNR are trivially 1 respectively 0, and the accuracy is ≈ 0.86.

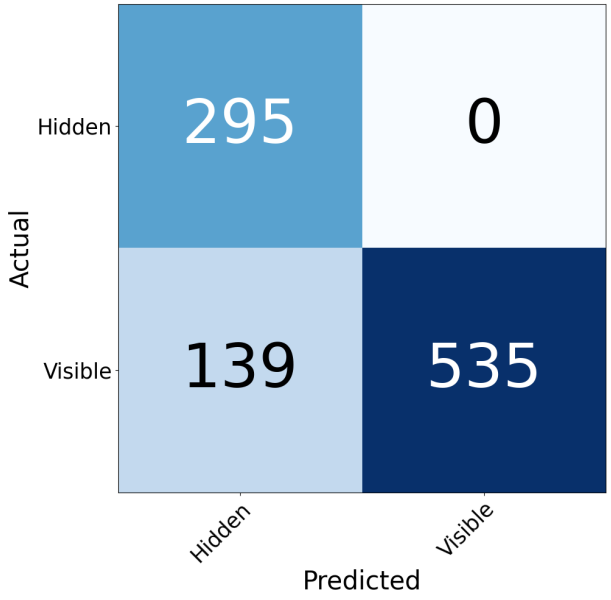


Figure 5.2: Confusion matrix from the Tranco random 1% sample dataset. Numbers are based on individual links.

Individual websites can influence specific metrics more than others when looking at individual links, for the simple reason that some websites contain more links. Therefore, it can be useful to look at the same data but per domain instead, i.e., FPs, TPs, TNs, FNs, as binary. The result of this can be seen in Figure 5.3. The derived values then become the following; the TNR is ≈ 0.76, the FPR becomes ≈ 0.24, the TPR and FNR remain the same, 1 and 0 respectively, and the accuracy becomes ≈ 0.77.

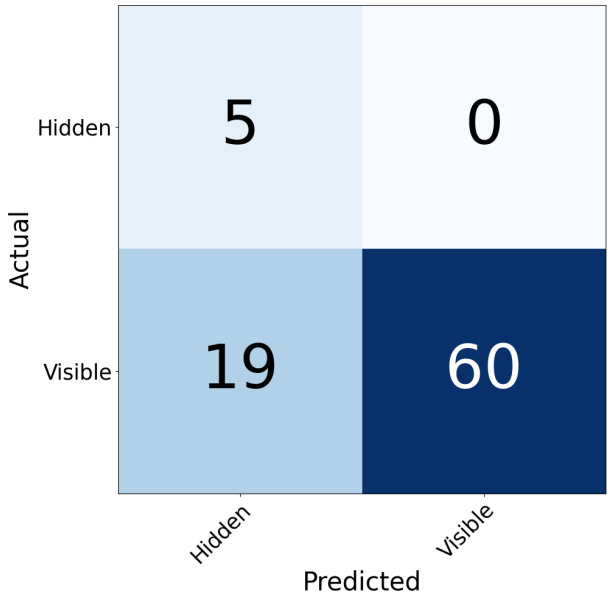


Figure 5.3: Confusion matrix from the Tranco random 1% sample dataset. Numbers are on the domain level.

Looking into the specific reasons why there are FPs is quite interesting, as it shows in which areas MANIS is lacking. Ideally, there should be no false positives, as that means a visible link is misclassified as hidden. In the Table 5.5 below, the FPs have been split into categories to indicate which areas of MANIS need further development.

In Table 5.5, the different reasons for FPs are split into categories, the most prominent area in which MANIS is lacking is in cases where some sort of interaction are required to find the links, such as drop-downs, elements that require hovering, and widgets with subpages. But there are also issues with the screenshotting, and not being able to set the checks as outlined in Section 3.4 on the element. Problems with SVGs and icons mostly stem from incorrect handling of SVGs or icon packs like Font Awesome, where MANIS is unable to apply the desired modifications as outlined in Section 4.1.

Table 5.5: Reasons for false positives for the Tranco random 1% sample dataset.

Reason	Domain-based occurences
SVG/Icons	4
Tool Failure	5
Line Break	2
Mobile View	2
Image Carousels	1
Interaction Required	10

5.1.2 Comparison of Sucuri versus MANIS with Tranco dataset

When compared to Sucuri SiteCheck, MANIS performs well, as seen in Figure 5.4. MANIS was able to detect hidden links in five cases, whilst Sucuri did not detect any in this dataset, as highlighted by the Venn diagram. In the diagram, each circle represents detections made by each tool, and the overlap in the center shows detections made by both. Since all detections lie solely within the MANIS circle, it means, as previously mentioned, that only MANIS detected occurrences of hidden links.

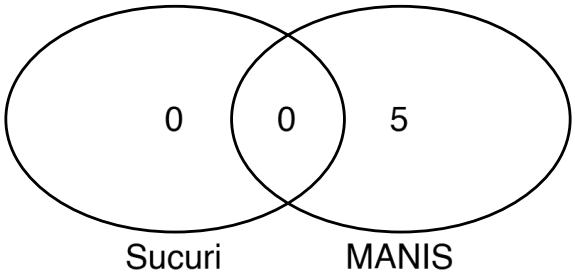


Figure 5.4: Domain-based detection rates of hidden links for MANIS compared to Sucuri SiteCheck on the Tranco random 1% sample dataset.

Most of the instances where MANIS is able to detect hidden links are straightforward cases, which can be seen below. Which makes the outcome from Sucuri a bit surprising, as there is no novel technique used. This will be discussed further in Section 6.1.1. It is worth noting that the styling was not necessarily placed on the link tag in all cases, but in a parent element containing the link tag.

```
<a style="display:none;" href="example.com">Hidden link</a>
```

One unique case was a webpage that seemed to have removed the interactive elements, such as menus. The elements are still present and can be clicked if you know where they are located, but nothing happens, and therefore, the links are hidden.

5.2 Results from testing on the suspected hidden dataset

The results of scanning 6 561 websites suspected to contain hidden links with MANIS, which evaluated 174 007 links and deemed 119 761 of those links to be hidden, and 54 246 as visible. Which means that about 69% of the links are hidden. For more insight, Figure 5.5 shows part of the distribution of hidden links in the data, by displaying how many of the hidden links reside in the websites that contain the most hidden links. For a graph that shows the entire distribution, see Figure C.2 in Appendix C. Looking at these two figures, it can be seen that 50% of the total number of hidden links is present on around 40 of the websites with the most hidden links on them, with 20% of the total coming from just the first three. From the figure in the Appendix it is also possible to note that the last 5% of the total number of hidden links come from a little over 2 000 websites, and around 2 000 websites containing no hidden links at all.

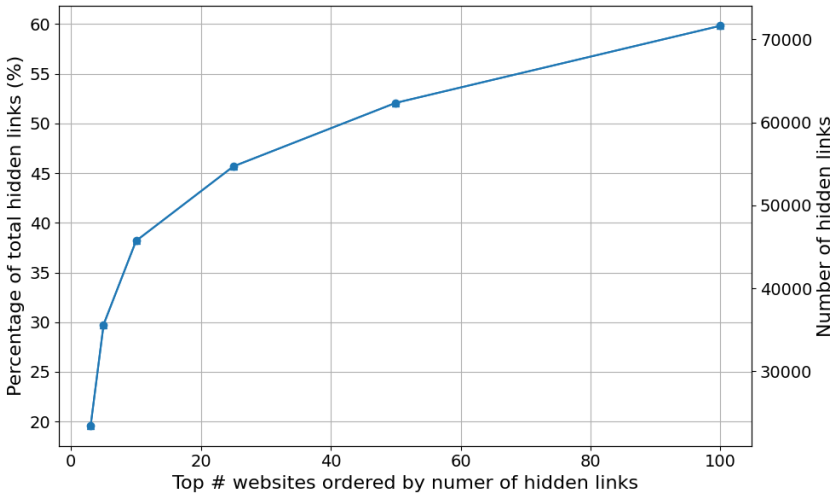


Figure 5.5: Distribution of hidden links from the top websites from the suspected hidden dataset.

Applying the same filter as with the Tranco dataset, to minimize the amount of FP, the number of hidden links decreases as desired. The results can be seen in Table 5.6. Once again, a subset of the top 100 domains accounts for a significant portion of the filtering. Specifically, 42 of these domains are responsible for approximately 65% of all filtered links when the top 2000 domains are used as the filter set. When looking at what domains are used as filters the most in the suspected hidden dataset, it becomes clear that some domains are more common than others, these can be seen in Table 5.7.

Table 5.6: Filtering of top domains from the hidden links of the suspected hidden dataset.

Top # of domains	# of unique domains used to filter	# of filtered hidden	Percentage of total amount of hidden links
100	42	9 724	8.12%
200	73	11 176	9.33%
1 000	260	13 265	11.08%
2 000	336	14 949	12.48%

Table 5.7: The top five domains used in the filter for the suspected hidden dataset.

Domain	Number of times filtered
youtube.com	2 512
twitter.com	1 338
facebook.com	984
google.com	870
instagram.com	512

5.2.1 Results from suspected hidden sample

A random sample of 66 websites was selected from the original set of 6 561 and manually evaluated to gather performance metrics such as TPs, FPs, TNs, FNs, and overall accuracy. The detailed results of this manual verification are presented in Appendix B. In total, 1 452 links were manually reviewed, of which 892 were classified as hidden and the remaining 560 as visible. The top domain filtering approach was also applied to the sample, as shown in Tables 5.8 and 5.9. Some websites that could not be scanned during the initial MANIS run were later rescanned to ensure a more accurate and complete evaluation.

Table 5.8: Filtering of top domains from the hidden links of the suspected hidden 1% sample dataset.

Top # of domains	# of unique domains used to filter	# of filtered hidden
100	4	17
200	5	18
1 000	7	20
2 000	7	20

Table 5.9: The top five domains used in the filter for the suspected hidden 1% sample dataset.

Domain	Number of times filtered
twitter.com	8
facebook.com	7
wordpress.org	1
google.com	1
archive.org[1]	1

Figure 5.6 is a confusion matrix displaying the TPs, FPs, TNs, and FNs based on the result from the manual verification. Same as before, in the top left and bottom right quadrants, a higher number is desirable, and in the bottom left and top right quadrants, a lower number is desirable. From this, the TPR, FPR, FNR, TNR, and accuracy can be extracted. The calculated values are the following TNR:≈ 0.93, FPR: ≈ 0.07, TPR: ≈ 0.997, FNR: ≈ 0.003, and an accuracy of ≈ 0.97.

Interestingly, there are three cases of FN, which means MANIS marks something as visible when it is invisible. These three cases come from two different sites, as can be seen in Figure 5.7, both related to dynamic loading or animations. The result from the website containing two FNs, can be explained by the fact that the website has images that move around, and the links happen to be placed behind them. This results in the image having a slightly changed position in screenshots for the opacity and the blank checks when compared to the base screenshot. The second site appears to have an FN related to an element that loads in dynamically after a few seconds, which pushes the content around on the website. Leading to an FN between when the baseline image was taken and the images for opacity and blank detection.

The numbers in Figure 5.6 run the risk of being inflated by specific websites for all values, as some websites might contain a lot of links that fall into one specific category. Therefore, Figure 5.7 instead shows the same data but translated to TPs, FPs, TNs, and FNs per website instead of per domain. The per website values are the following, TNR:≈ 0.79, FPR: ≈ 0.21, TPR: ≈ 0.96, FNR: ≈ 0.04, and an accuracy of ≈ 0.86.

[1] archive.org was first used as a filter when filtering with the top 200 domains.

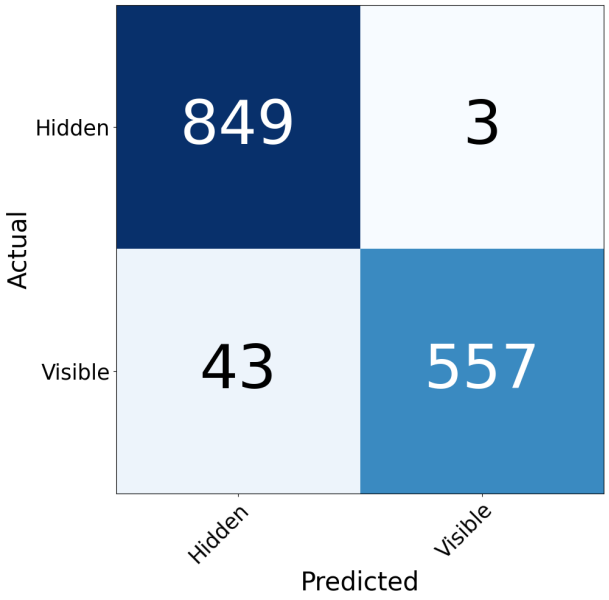


Figure 5.6: Confusion matrix from the suspected hidden 1% sample dataset. Numbers are based on individual links.

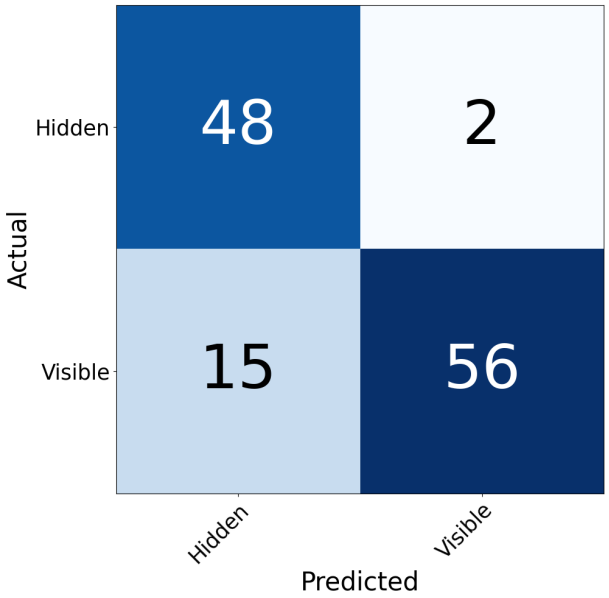


Figure 5.7: Confusion matrix from the suspected hidden 1% sample dataset. Numbers are on the domain level.

The FPs that occur have been categorized as seen in Table 5.10. This highlights the shortcomings of MANIS and where it can be improved upon. As of writing, MANIS mainly lacks in two categories. The first is tool failure, which is when MANIS, for example, fails to screenshot a link or start the scan. The other big category is required interaction, which more often than not is menus for navigation. This is explored further in Chapter 6.

Table 5.10: Reasons for false positives for the suspected hidden 1% sample dataset.

Reason	Domain-based occurences
SVG/Icons	1
Tool Failure	5
Line Break	0
Mobile View	0
Image Carousel	1
Interaction Required	8

5.2.2 Comparison of Sucuri versus MANIS with suspected hidden dataset

Comparing MANIS to Sucuri for the suspected hidden sampled dataset, MANIS outperforms Sucuri by quite a large margin, as seen in Figure 5.8. There is only one time that Sucuri finds a hidden link when MANIS fails to scan the website, this is due to MANIS crashing on the day of the tests. During the writing of the results and discussion, this case was rescanned, and MANIS successfully detected the hidden links. There are, however, 32 websites where MANIS is the only one able to detect hidden links. But both of the scanners can detect hidden links on 16 websites. These results will be explored further in Section 6.1.1.

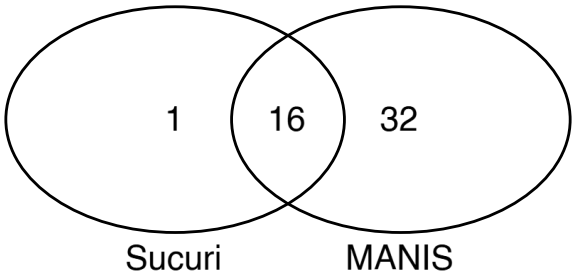


Figure 5.8: Domain-based detection rates of hidden links for MANIS compared to Sucuri SiteCheck on the suspected hidden 1% sample dataset.

Once again, the same hiding method is found as in the Tranco dataset, described in Section 5.1.2. But as there are more cases of hidden links, there are more unique hiding methods, as seen in the example below, which MANIS can detect but not Sucuri. This sets the strong element containing all the links to be small, and thus not visible to the human eye. It is interesting because Sucuri can detect other similar scenarios that differ in the tag used.

```
<strong style="display: block; overflow: hidden; height: 1px;
↪  width: 1px;"> Multiple hidden links </strong>
```

Another example that MANIS finds that Sucuri does not, which can be seen below. This hiding method has been explained previously in Section 3.2.

```
<p style="position:absolute;left:-19365px">Hidden link</p>
```

Looking at one of the examples that both Sucuri and MANIS can detect, as shown below. The hiding method is similar to the one that Sucuri was not able to detect, as explained above. This suggests that Sucuri is using a signature-based approach, which is also discussed in Section 6.1.1.

```
<div style="width:2px;height:2px;overflow:hidden;">Hidden
↪  link</div>
```

5.3 Interesting discoveries

During the thesis, we have found some interesting cases of hiding methods, the first is the discovery of the deprecated marquee tag as previously presented as examples in Sections 2.4 and 3.2. The marquee tag is typically used for scrolling text, but when used for hiding, the speed is set to a very high number. This effectively makes the text invisible as the browser stops displaying the text. This creative use of an HTML tag was discovered during the development of MANIS. Interestingly, Sucuri fails to detect links hidden with this method.

Another interesting discovery was that links might be hidden or visible depending on the user-agent, as seen in one of the examples Section 3.2, which hid the links if the user-agent was anything else than "Google Web Preview". This is a clear attempt at trying to boost SEO ranking whilst also avoiding any repercussions from Google, such as being labeled as spam. The hiding method was once again only found thanks to our work with MANIS, and Sucuri is not able to detect it.

Another notable finding was that some websites containing hidden links redirected users to a different site based on the client's operating system. In all observed cases, the redirection led to the same gambling website. This behavior occurred only when the client was Linux-based, whilst Windows-based clients were not redirected. The reasoning behind this behavior is unclear. If the goal were to avoid detection by search engines such as Google, it would also prevent the hidden links from being indexed, and thus get no benefit from the hidden links. As this falls outside the main scope of the thesis, no further investigation was conducted into other influencing parameters other than the operating system. It is possible that additional parameters contribute to the redirection behavior, which could better explain the reasons as to why.

A particularly interesting set of cases involves sites with interactable cookie banners that list all data sharing partners. One of the most prominent ones from the Tranco random dataset included around 1 000 partners, each linked twice within the banner, resulting in over 2 000 false positives. Although this particular case was not included in the 1% sample, it highlights the likelihood of other sites containing similar cases, but most likely not as extreme, perhaps containing about 100 partners. A possible solution to this is discussed in Section 6.3.1.

5.4 Scanning statistics

A summarization of the general scanning statistics from both the Tranco random and suspected hidden dataset is presented in Table 5.11 below. To show the true scale of the testing and the amount of data that it resulted in.

Table 5.11: Scanning statistics from testing.

Metric	Tranco	Suspected Hidden
Dataset size	10 000	6 561
Internal links	1 368 762	568 604
External links	138 932	174 081
Scanning time (core-hours)	≈ 102 h	≈ 77 h
Time per external link	≈ 0.50 s	≈ 0.47 s
File amount	≈ 335 000	≈ 342 000
Output folder size	26.7 GB	13.6 GB

As seen in Table 5.11, there was a lot of data collected during the testing phase of MANIS. Over 2.25 million links have passed through MANIS, which were either skipped because they were internal or analyzed as they were external. This resulted in over 40 gigabytes of data generated from screenshots and log files during the 180 core-hours required to scan the approximately 16 500 links. However, since the tool only runs on a single thread, multiple instances can be executed in parallel. As a result, the actual running time was much shorter, with 20 instances running simultaneously across two machines, as outlined in Section 3.6.

6
Discussion

In this chapter, the results will be interpreted and discussed to see if there are any interesting findings. The main finding is that malicious non-interference is a suitable detection method for hidden links. However, as the Web is created by a lot of people, it is hard to predict all the different ways a website can be coded, which leads to some drawbacks that influence the results and the reliability of the results produced by MANIS.

The structure for the discussion is as follows: Section 6.1 highlights findings in the results found during the evaluation of MANIS. Section 6.3 addresses some of the challenges faced during the thesis and explains how they were handled. It also deep dives into some of the problems encountered, and we give our understanding of how the problems could be solved, to increase the reliability and detection rate of MANIS. Section 6.4 explores potential applications of MANIS and how it should be used to produce valuable results. Finally, Section 6.5 outlines larger features found to be missing during development and testing, which should be considered for future implementation.

6.1 Evaluation and performance of MANIS

The results presented in Chapter 5 are promising, but they also highlight areas where MANIS could be improved. Out of the 2 421 manually reviewed links, only three were false negatives, indicating that MANIS is highly effective at avoiding the incorrect classification of hidden links as visible. However, the number of false positives is somewhat higher, with a total of 182 across both samples. This indicates that MANIS sometimes misclassifies visible links as hidden, which is not a desirable outcome. Many of these misclassifications arise from identifiable challenges that could be addressed with additional engineering effort, as previously discussed. As shown in Tables 5.5 and 5.10, most of these issues fall into clear categories and are related to typical web behavior, such as interacting with menus, handling SVGs, or adapting to specific design practices. These factors will be explored in more detail later in this chapter.

Additionally, during the manual evaluation, it was sometimes difficult to determine if a link was visible or hidden, as some may require a complex set of interactions to become visible or are technically visible but not for a human. While we did our best to classify all links accurately, there remains a small possibility that a few were incorrectly labeled, but this number is believed to be small.

The Tranco random dataset serves as a representation of real-world conditions, as it is unbiased with respect to the goals of this thesis. In contrast, the suspected hidden dataset is specifically intended to assess MANIS's performance in scenarios where hidden links are expected. When examining the metrics from the 1% samples, the results are promising, with both datasets showing high accuracy. Diving deeper into Tranco results, when looking at the per-link results, both the TPR and FPR show promising results, and the accuracy is satisfactory. This indicates that MANIS is quite reliable at determining the status of a link on a given website. With the accuracy, it is possible to estimate how many of the links deemed as hidden are hidden from the full Tranco random dataset, which is about 63 475 from the original 73 808 reportedly hidden links.

The perspective remains the same when examining the results on a per-domain basis. With the FPR increasing by four percentage points, and the TPR rate remaining the same. This suggests that MANIS is good both on a per-domain basis as well as on a link basis. However, the accuracy remains at an acceptable level in our opinion, as it is far better than simply guessing if a link is hidden or not. Applying the per-domain accuracy to the reportedly hidden links, it is more likely that 56 832 of them are hidden. But there is a catch to this number, a per-domain accuracy is used on a per-link basis, but while it works, it becomes slightly skewed. At the same time, it cannot be applied to the total number of websites containing at least one hidden link, as it is not how MANIS works or reports data. The two numbers should therefore be interpreted as a sort of interval for how many of the links were hidden, such that somewhere in between 63 475 - 56 832 is the correct number of hidden links.

Delving into the suspected hidden dataset performance instead, the performance is more consistent and even better. The trend for FPR and accuracy is still the same as for the Tranco dataset when going from a per-link basis to a per-domain basis, the FPR increases and the accuracy drops, but remains at decent levels. Applying the per-link accuracy from the suspected hidden dataset to the reported hidden links brings it down to 116 168 from 119 761, and with the per-domain accuracy, it becomes 102 994.

6.1.1 Comparative Analysis of MANIS and Sucuri

The conclusion that can be drawn from this is that MANIS performs well when given a website that most likely does not contain hidden links, and performs great when given a website that does contain hidden links. Which leads to the comparison between Sucuri, in which MANIS outperformed Sucuri in both datasets as seen by Figure 5.4 and 5.8.

In the Tranco dataset, Sucuri does not detect any hidden links, but MANIS can find five, and in the suspected hidden dataset, there are 32 instances where MANIS is the only tool able to detect hidden links on a webpage, whereas Sucuri only has one instance when it can detect hidden links when MANIS cannot. The reason for this case is that MANIS is unable to scan the website, this is because it cannot load the website.

However, at the time of writing the report, two weeks after the testing, MANIS can scan the website without any changes to the source code of MANIS and is able to detect the same hidden links as Sucuri. The reason for not including this update in the metrics is due to the time between the original scan and the new one, as it was weeks instead of days.

Looking into the specific case with "display:none" mentioned in Section 5.1.2, it is interesting to note that Sucuri does not detect such a basic case. As there are no specific numbers that Sucuri would not have in their database. This might, however, be because Sucuri has decided against marking "display:none" as a way to minimize FP. As we have seen it used on social media links and pop-ups, suggesting that benign ways to utilize the hiding method. However, this also means that Sucuri fails to detect many of the hidden links detected by MANIS.

The three examples as presented in Section 5.2.2, are quite interesting as they demonstrate our suspicion that Sucuri relies on predetermined signatures rather than analyzing the tags in detail. Cases one and three share the same characteristics: low width and height values, with overflow set to hidden. The difference is that in case one, it also sets display to block, and height and width to "1px" instead of "2px". However, Sucuri only detects case three as having hidden links.

Case two from Section 5.2.2 further supports our suspicion, as that is a very trivial case, as it uses "position:absolute;left:-19365px", which is not hard to tell that it will not be visible to the user as it is 19 365 pixels. Cases one and two are a bit hard to reason about why Sucuri fails to detect, as for case one, it might be due to the usage of the strong tag in combination with the specific styling, which may be something that Sucuri has not encountered previously. But with case two, we have seen previous occurrences where Sucuri successfully detects the position left, indicating that they most likely rely on exact matches from their database.

6.2 Dataset bias

The two datasets used to evaluate MANIS may contain some bias, particularly the suspected-hidden dataset, as it was partially used during the development of MANIS. However, this is unlikely to have a significant impact on the results, since only a small portion of the approximately 6 500 links were used during development. The potential for a bias would only be substantial if all hiding methods were the same across the dataset. This is unlikely, as our observations showed that many websites vary the properties of their hiding techniques.

For the Tranco random dataset, the likelihood of bias is slim, given that it was randomly sampled from the latest Tranco list, meaning we had no influence over which links were selected. Since the full Tranco list was used, the dataset should contain a wide variety of websites, both modern and old.

One aspect that could introduce a slight bias is the manually verified samples, as the samples represent a small portion of all evaluated links. This also means the samples may include websites that are beneficial for the results, but also disadvantageous. For instance, if any sample had included websites with the highest number of hidden links, the confusion matrices and overall accuracy would become skewed, especially considering that some of these websites contained around 8 000 hidden links.

Nevertheless, both datasets included a few cases that were beneficiary to our results, for example, the suspected hidden sample had 480 TP, which accounts for roughly half of the total TP in that sample. But, as suspected, there were also some cases that negatively impacted the results, like in the Tranco random dataset, one website accounted for a third of the FP. Given that there are cases of both positive and negative contributions to our results, the results can still be considered fair.

6.3 Encountered problems

As mentioned above in Section 6.1, there are a couple of issues with the current implementation, or rather lack of implementation. Many of them are deemed to be easily solvable by more engineering, while other requires more fundamental changes or deeper studies. The ones that are a bit more complex are discussed to give insight into our understanding of how they could be solved, but some of the solved ones are also mentioned to give insight into the development process. The benefit of solving these would be the decrease of FP, leading to an increase in the accuracy and thereby increasing the trustworthiness of the results produced by MANIS.

6.3.1 Unsolved issues

Even though we spent many hours developing MANIS, there are still issues, bugs, and opportunities for improvements in the code. Solving these currently unsolved issues would primarily lead to improved results, to aid any future research on the topic or future developments of MANIS. Below are some of the most prominent currently unsolved ones are detailed.

Interaction

One of the hardest problems to solve is that of links that are hidden by elements that require some form of interaction with the webpage. This was found to be caused by both menus and submenus, but also by links that required scrolling in separate objects, or ones hidden behind banners, such as cookie banners. It is, however, possible to implement interaction with the webpage through Selenium, as we know that there are projects working on interacting with elements at Chalmers, it is just a question regarding the time needed. It would require careful consideration of which elements should be interacted with, as in the case of our tool. It would waste a bunch of time screenshotting nothing if it were unable to activate the correct elements, which would increase the running time for analyzing a webpage even more.

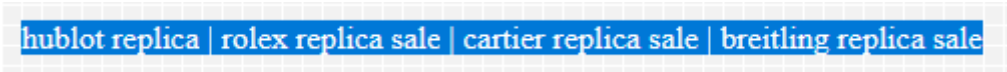
Thresholds

Another challenge encountered in this work involves defining appropriate thresholds in parts of the code. Especially those that need to reflect human perception in some fashion, as this thesis is conducted within the computer security field and not psychology. This means that assumptions had to be made based on what we thought should be deemed visible.

One difficulty is that human attention naturally shifts across different parts of a webpage. Some areas draw more focus than others, depending on layout, color contrast, and content. However, our system does not dynamically account for these varying "focus points." Instead, we had to define visibility thresholds statically, based on our own subjective assessment of how difficult it was to distinguish text from background color when focusing only on the element in question. An example of this can be seen in Figure 6.1, where the text technically differs from the background but is effectively indistinguishable to the human eye.

Consequently, the chosen thresholds may not always be optimal or universally applicable across all scenarios. A similar issue appears with smaller images or text elements. While MANIS treats screenshots smaller than three by three pixels as too small to analyze, it does not apply the same logic to slightly larger ones, such as a four by four pixel image. This is because SSIM calculations cannot be done on images that small, as it needs to work with a certain window size.

(a)



(b)

Figure 6.1: Real world example of hidden links text color matching well with the background.

Perhaps there should also be a threshold implemented for when images are too small, as a five by five image is almost just as hard as a three by three image to see, but there is no threshold implemented for this case. Still, the core challenge remains the same as with color perception, since it is difficult to determine what size an element needs to be in order to be visible to the human eye, as it depends on many factors.

Another such threshold is how many frames of a GIF should be invisible for an element to be considered hidden, as that is once again dependent on the context of the site. Right now, it is simply implemented as a static threshold that deems an element as hidden if a percentage of the total frames are hidden. This is not optimal in all scenarios, especially if it is a large website with time-consuming GIFs. It would theoretically be possible for the user to have scrolled past the GIF before the visible frames are shown. This is hard to work with, as it would require simulating typical user behavior, which falls outside the scope of this work. A more refined approach for future implementations could also involve taking into account the sequence of hidden and visible frames, rather than just the number of hidden frames.

SVGs and icons

SVGs and icons posed a big issue for minimizing FP, as they required special handling. Multiple implementations have been developed and implemented into MANIS, but in the end, the flexibility of both types is large. Meaning there are still multiple unhandled implementations, causing FP in MANIS. Advising on how to solve these is hard, as the flexibility of the formats requires consideration, but they should not be too complicated, as a lot of the basics required to handle them are already implemented. The suggested approach for the time being would most likely be to add specific edge case handling.

Line breaks

Line breaks present a particular challenge, as they tend to cause the SSIM calculation to fail as used in the text detection module (Section 4.1.1), due to the changed area of the image being very small. The line breaks cause the image to span two lines of text, resulting in the screenshot containing more text than just the changed area, as can be seen in Figure 6.2. An idea for how this could be solved is by locating all areas with changes in the image if the SSIM check fails, grouping them close to each

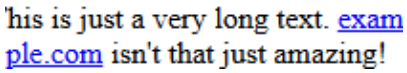


Figure 6.2: Example of text with line break.

other, removing the rest of the image, and then recalculating the SSIM score. This approach would remove all the unchanged text visible in the screenshot and allow for a more accurate SSIM score calculation by only keeping modified areas.

Mobile view

The issues related to mobile views were somewhat unexpected, as we had not initially considered that the same codebase might serve two distinct versions of a website. In retrospect, this should have been more apparent, given how many websites we use every day that have different layouts or content based on the device used. To mitigate some of the challenges with mobile view, there is a heuristic implemented in MANIS described in Section 4.3.2, which solves many of the problems caused by mobile views. However, we have found certain links to still differ between the two views. One possible way to solve this would be to extend MANIS with the functionality to scan two different screen sizes, one with a size typical for mobile phones, and one with a size typical for desktops. While this would, in theory, improve the scanning results, it would also significantly increase the total scanning time.

Image carousel

Image carousels were something that were occasionally encountered on the web, where the image changes either based on user input or based on an interval. The tricky part with this and what causes false positives is that the image might change since the element was first fetched, but also because there are images that are hidden that cannot be properly checked. Solving this is tricky, given that it either requires interaction or waiting for the correct image to appear. The hardest challenge with this is the large number of ways it can be implemented. However, the easiest way to solve this would most likely be by getting MANIS to perform user interactions.

Tool failures

There are still some issues with the foundational functions of MANIS as highlighted by the presence of tool failure in Table 5.5 and 5.10. Some of these come from the screenshotting function not working correctly on some elements, meaning that elements are missed, or only partially screenshotted, leading to incorrect classifications. Solving this is quite hard, as it most likely stems from SPAs or the page being improperly loaded by the browser. One possible solution would be to implement a fallback to Selenium's basic screenshot function instead of Chrome Developer Tools as used by MANIS.

The motivation for using a more advanced screenshotting method was to ensure greater accuracy in capturing what is actually displayed by the browser, avoiding potential shortcuts implemented by Selenium to speed up the screenshot process. We believed such shortcuts could result in hidden elements being captured in screenshots even though they would not be visible during normal browsing. Our method did, however, lead to some issues with websites that were longer than what Chrome wanted to render, leading to MANIS being unable to handle those.

6.3.2 Solved issues

During the thesis, a multitude of issues were encountered and solved, some of which had a larger impact on the final results than others. Below, some of them are described with the benefits they brought to the scanning capabilities of MANIS.

Filter - internal links

The heuristic to filter out internal links as described in Section 4.3.1, is hands down the heuristic that improved the scanning time the most. As internal links stood for approximately 86% out of all links, meaning that if also scanning internal links, it would most likely have increased the scanning time by around seven times. Scanning internal links would also have increased the amount of false positives, as dropdown menus primarily contain internal links. The application of this heuristic in MANIS is believed to have no downside when it comes to SEO, as internal links should not affect the SE ranking. However, if one, for some reason, wants to let MANIS scan internal links, the heuristic can always be disabled with the usage of an already integrated calling argument.

Filter - seen links

Changing the label of links that have been marked as hidden to visible if the same link appears again but is visible this time, as described in Section 4.3.2. The heuristic was primarily implemented for pages with two different versions present in the same HTML code, such as one for mobile phones and one for desktops. It did, however, turn out to be useful for other cases as well, as in some cases, there were social media buttons that MANIS could not handle that were saved from becoming false positives by links to social media appearing multiple times in formats that MANIS could handle. Ideally, this should not be necessary as MANIS should work for those, but as the Web is tricky, it is always good to have fallback methods.

Filter - most popular websites

As described by the heuristic in Section 4.3.3, a filter was applied after the initial detection to remove the most popular domains. The effect of this filtering is presented in Tables 5.1 and 5.6. One can see that filtering out the top 100 domains provides by far the largest improvement to the FPR by filtering out close to 9 500 and 9 750 entries marked as hidden for the two datasets, meaning that the total amount of FP could minimized by the same amount. Thereby leading to higher accuracy for MANIS, this was not implemented into the results, as to give a more accurate view of the raw performance of MANIS.

While the top 100 domains provide the largest benefit in terms of filtering efficiency, relying solely on this small subset may not be optimal. It is important to recognize that larger filters, such as the top 2000 domains, still represent a relatively small portion of the web, yet they consist of highly popular and generally trustworthy websites. The filter could be increased to the top 10 000 as it still represents a small percentage of all available websites, which reasonably do not act maliciously and hide their links on other vulnerable websites, reducing the FPs even further. The primary trade-off of increasing the filter size lies in performance, but with appropriate data structures, the additional lookup cost remains negligible.

Improvements to text detection

The current method for text detection, as mentioned in Section 4.1.1, is based on changing the opacity, but in the beginning of the development of MANIS, it instead relied on manipulating the characters. The first thought was to just fill it up with a single random character that was not the one used in the string, this was determined to not be a good idea as it would be hard to control the physical length of the string, as ten "I" would be a lot smaller than ten "W". Due to this, it was decided to scramble the characters in the string until they were in a different position than before, and if a single kind of character was used, just substitute it with another character known to be large. It was undesirable to have any other text potentially enter the bounding box if the text length was shorter. Though this would technically have a chance to be longer or shorter due to kerning[1], this was accepted as a risk as the difference would be minimal.

This problem has since been eliminated completely by instead of swapping characters, just changing the opacity to zero, which achieves a modification that can be detected. An added benefit of this, and the reason why it was changed, was due to the handling of icon packs. As they are processed in the text detection module due to their HTML implementation, but as there is no text available to modify, scrambling the text did not work. But changing the opacity allowed MANIS to successfully detect and handle icons.

[1] https://en.wikipedia.org/wiki/Kerning

Changes on the web

As the Web is user-controlled, there is always the possibility of the website changing between visits. We noted this during the thesis, with some websites removing their hidden links. But most of the time, when there were changes, it was due to the website dynamically injecting hidden links, meaning they changed between visits, or adding new content on their website. This could have caused a lot of issues during the manual sample verification, as it would have caused an increase in the scoring, as the links would have been missing. But this was mitigated as MANIS saves the source code of the websites it scans.

One example of this was a website that, once we did the manual verification of the samples from the two datasets, contained 32 new hidden links when comparing the source code and the active website. There was also a set of websites with a similar design that had dynamically injected hidden links, which meant that they changed every reload of the page.

Handling timeouts

Website speeds tend to vary greatly, some load quickly, others are slow, some contain hundreds of links, whilst others contain very few. This can cause issues for MANIS, such as getting stuck processing websites for hours, whilst not giving any valuable data. This can be caused by websites that are running on very slow servers, leading to very slow loading times. To counteract this, there is a timeout set in MANIS to make sure it does not try to fetch a site that is too slow. Without any timeouts, Selenium times out somewhere in the range of five minutes, and the request library, as used for the deadsite filter and shallow crawling, has no default timeout, meaning that it essentially waits forever. We did, however, observe that it eventually times out, but it took a long time.

Another issue related to timeouts is the possibility of overwhelming MANIS with external links, as scanning a single link takes a couple of seconds. Filling a website with thousands of external links would therefore work like a denial-of-service attack. To counteract the possibility of this, a scanning timeout is implemented and set during the testing to five minutes, giving MANIS plenty of time to scan ordinary websites. As in our testing, it seldom reaches that high.

6.3.3 Rescanning failed sites

As mentioned in Section 3.5, any of the websites that were unreachable during the scanning of the 1% samples were rescanned a few days later with MANIS, to give a new opportunity for eventual temporary issues to be resolved. In the end, seven websites were rescanned, and only four turned out to have been temporarily not working. The reason for only rechecking the manual sample and not the entire dataset comes down to the sample being used to approximate the overall correctness of MANIS, meaning any missing datapoint could seriously skew the data in both a positive and a negative way. Whilst dead websites in the large dataset would have a minimal impact due to the dataset being so large.

6.3.4 Revisiting the research questions

Having presented and analyzed the results, we now consider how these findings respond to the research question and their implications. The first one is the most important in the context of the thesis, whether malicious non-interference can be used to detect hidden links on the Web or not. Based on the results and the fact that MANIS is the first of its kind, the answer is yes. While malicious non-interference does not produce perfect results, as indicated by the false positives, this is not a flaw of the method itself but rather our implementation. The concept works, and even with limited development, it yields highly promising results.

As for the research question regarding what factors need to be considered to accurately detect hidden links using malicious non-interference, it is difficult to give a precise answer. There were many unforeseen challenges, which have already been discussed above. The two most important factors when developing tools similar to MANIS are, first, the need to account for failure. No website is the same, so the tool must be able to handle a wide range of unique scenarios and include multiple checks for the same type of elements, such as with SVGs. The second important factor is human perception, which was not thoroughly considered at the start of the development of MANIS. The thresholds used in MANIS to determine whether a link is hidden are somewhat arbitrary, as discussed earlier. In hindsight, it is more important than initially deemed, the realization came when more and more hiding methods were encountered that needed a good threshold. The thresholds in MANIS still produce a good result, but could be improved and supported with actual testing and data in the future.

Looking into the last research question, RQ3, whether malicious non-interference can be used to find new signatures or not, Section 5.3 shows clear examples that malicious non-interference is flexible and can discover new hiding methods. The reason for this is due to the core idea that relies on a generic approach, rather than a specific heuristic. This is not to say that heuristics have not helped, as MANIS has seen great results from using the heuristics outlined in Section 4.3. MANIS was never designed to look for specific HTML tags when scanning for hidden links, instead, it looks for anything that could potentially be a link and tries to evaluate it.

Given that MANIS can discover new signatures and demonstrates reliable performance, with potential for further improvement, the concept of malicious non-interference shows very promising results and should be considered for application in real-world scenarios. The following two sections will explore potential use cases and directions for future work.

6.4 Real world usage of MANIS

MANIS should not be used standalone but rather integrated into a larger pipeline. It can be used to flag certain websites for further investigation. Also, it can be combined with other tools that provide other metrics to determine the trust of a certain website, through this, false positives can be detected.

Otherwise, a service can be hosted for any user to check a website and see if it has any hidden links in it. Thereby, the user can verify any false positives on their own. This does, however, come with its own risks, such as injection attacks, but these are outside of the scope of this discussion.

Another perspective on MANIS is as a tool for discovering new hiding methods, as running MANIS in a pipeline all the time is expensive, given that it renders the websites and performs the checks on the links one by one. The signatures extracted from the identified hiding methods could be used to train a machine learning model or to build a database of hiding methods suitable for static analysis. Based on our findings, there does not appear to be a significant number of entirely new hiding methods. However, it is important to acknowledge that our analysis has been limited to a very small part of the internet, and an even smaller part had the hiding methods manually examined.

6.5 Future work

There are several aspects that could have been included in the thesis if we had more time, but that will always be the case. The following sections outline broader areas of improvement for MANIS, complementing the more detailed suggestions discussed in Section 6.3.1.

6.5.1 Code

MANIS codebase could use restructuring to help with readability, clarity, and to better follow the single responsibility principle. The goal from the beginning was also to keep the code modular, to be able to swap and add new detection modules, following Liskov's substitution principle. However, this was not entirely achieved, since it was hard to outline the code structure as it changed as the thesis progressed, and it was also partly ignored due to the time constraint.

As previously mentioned, the Web is ever-changing, trends and technology change often. Therefore, it contains a lot of unique and deprecated approaches to building websites, and as seen, it has caused the developed tool to give false positives and false negatives in certain situations as mentioned in Table 5.5 and 5.10, as well as Section 5.2.1. With more engineering hours, these false results could be reduced.

Another thing we noted during the development of MANIS was that some websites redirect based on the operating system. Where Linux machines that visited certain websites were redirected to websites that did not contain any hidden links, Windows machines remained on the website, which coincidentally contained hidden links. Meaning that there are attempts from websites to conceal their true identity by means of cloaking. However, their means were very limited in this specific case. It was solved by setting the user-agent to a generic Windows user-agent. But there are probably other cases out there with more sophisticated methods that would be even harder to detect and properly handle.

6.5.2 Report and data

The specific method of malicious non-interference has not been evaluated before. Thus, it was hard beforehand to determine what metrics could be collected and was needed to properly asses the method. During the development, more and more data were collected. First, only the most basic information was saved, such as the screenshots. This then progressed to hiding methods and, finally, the page source.

As it is a niche field, there are not a lot of similar tools available for comparison. The only publicly available tool as of this moment is Sucuri's SiteCheck, which is not open-source. This made us realize that more data was needed since the focus shifted towards an internal evaluation. One such metric that could be improved upon is the FPR. For this report, it was simply sampled from an already sampled dataset, which then becomes an indicator instead of a calculated metric.

7
Conclusion

In this thesis, we have developed and evaluated a tool named MANIS to explore whether non-interference principles can be applied to the Web as a way to combat malicious SEO. Specifically, MANIS was designed to classify whether links on websites are visible or hidden from the user, which, to our knowledge, is the first tool with this focus.

In short, MANIS was successful, and we were able to answer our research questions. It outperformed similar tools and produced strong results. The evaluation was done on two datasets. The first was randomly selected from the Tranco web ranking list to reflect real-world conditions, where MANIS managed to detect actual maliciously hidden links. The second dataset was created by us during development to push MANIS to its limits. In both cases, MANIS performed well, although results were slightly weaker on the Tranco dataset due to the lower presence of hidden links.

Despite this, there are still some areas where MANIS can improve. Its weaknesses usually come from how websites are structured, such as links placed inside menus or within dynamic elements like image carousels. Still, MANIS shows solid performance and has the potential to become even better with improvements focused on these specific challenges encountered on the web.

Bibliography

[1] J. Killoran, "How to Use Search Engine Optimization Techniques to Increase Website Visibility," *IEEE Transactions on Professional Communication*, vol. 56, no. 1, pp. 50–66, 2013. DOI: 10.1109/TPC.2012.2237255.

[2] F. Schneider and M. Cutts, "Systems and Methods for Detecting Hidden Text and Hidden Links," Utility Patent US8392823B1, Filed: August 25, 2009, Mar. 2013. [Online]. Available: https://patents.google.com/patent/US8392823B1.

[3] J. Xiong, M. Wei, Z. Lu, and Y. Liu, "Assessing the effectiveness of crawlers and large language models in detecting adversarial hidden link threats in meta computing," *High-Confidence Computing*, p. 100 292, 2024, ISSN: 2667-2952. DOI: 10.1016/j.hcc.2024.100292. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2667295224000953.

[4] B. Dean. "What Are Backlinks in SEO & Why You Need Them." Accessed: 2025-05-11. (2025), [Online]. Available: https://backlinko.com/hub/seo/backlinks.

[5] StatCounter Global Stats. "Search Engine Market Share Worldwide." Accessed: 2025-01-22. (2025), [Online]. Available: https://gs.statcounter.com/search-engine-market-share.

[6] C. Olston and M. Najork, "Web crawling," *Foundations and Trends® in Information Retrieval*, vol. 4, no. 3, pp. 175–246, 2010.

[7] K. Phan, "A Comprehensive Study on Single-Page Applications: Pros, Cons, and Practical Guidelines," M.S. thesis, Haaga-Helia University of Applied Sciences, 2024. [Online]. Available: https://www.theseus.fi/handle/10024/871030.

[8] A. Huotala, M. Luukkainen, and T. Mikkonen, "Benefits and Challenges of Isomorphism in Single-page Applications: Case Study and Review of Gray Literature," *Journal of Web Engineering*, Mar. 2023. DOI: 10.13052/jwe1540-9589.2186.

[9] Google. "How Search Works." Accessed: 2025-01-23. (2024), [Online]. Available: https://developers.google.com/search/docs/fundamentals/how-search-works.

[10] Google. "Explore Google Search documentation to improve your site's SEO." Accessed: 2025-01-23. (2024), [Online]. Available: https://developers.google.com/search/docs.

[11] T. D. Le, T. Le-Dinh, and S. Uwizeyemungu, "Search engine optimization poisoning: A cybersecurity threat analysis and mitigation strategies for small and medium-sized enterprises," *Technology in Society*, vol. 76, p. 102 470, 2024,

ISSN: 0160-791X. DOI: https://doi.org/10.1016/j.techsoc.2024.102470. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0160791X24000186.

[12] K. Li, M. Lin, Z. Lin, and B. Xing, "Running and Chasing – The Competition between Paid Search Marketing and Search Engine Optimization," in *2014 47th Hawaii International Conference on System Sciences*, 2014, pp. 3110–3119. DOI: 10.1109/HICSS.2014.640.

[13] K. Dye, "Website abuse for search engine optimisation," *Network Security*, vol. 2008, no. 3, pp. 4–6, 2008, ISSN: 1353-4858. DOI: https://doi.org/10.1016/S1353-4858(08)70028-X. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S135348580870028X.

[14] L. Invernizzi, K. Thomas, A. Kapravelos, O. Comanescu, J. Picod, and E. Bursztein, "Cloak of Visibility: Detecting When Machines Browse a Different Web," in *2016 IEEE Symposium on Security and Privacy (SP)*, 2016, pp. 743–758. DOI: 10.1109/SP.2016.50.

[15] A. Barysevich. "Buying Backlinks For SEO: Yes, This Is Still A Thing." Accessed: 2025-02-04. (Oct. 2022), [Online]. Available: https://www.searchenginejournal.com/buying-backlinks-for-seo/207510/.

[16] Backlinks.com. "Backlinks.com." Accessed: 2025-04-01. (n.d.), [Online]. Available: https://www.backlinks.com.

[17] Backlinks.com. "Buy One-Way Text Links." Accessed: 2025-04-01. (n.d.), [Online]. Available: https://www.backlinks.com/buy-backlinks/.

[18] Backlinks.com. "Link Buyer FAQ." Accessed: 2025-04-01. (n.d.), [Online]. Available: https://www.backlinks.com/link-buyer-faq/.

[19] Backlinks.com. "Sell BackLinks." Accessed: 2025-04-01. (n.d.), [Online]. Available: https://www.backlinks.com/sell-backlinks/.

[20] Backlinks.com. "Link Seller FAQ." Accessed: 2025-04-01. (n.d.), [Online]. Available: https://www.backlinks.com/link-seller-faq/.

[21] Linkbuilder.io. "LinkBuilder.io | Simple, Transparent Pricing | LinkBuilder.io." Accessed: 2025-04-02. (n.d.), [Online]. Available: https://linkbuilder.io/pricing/.

[22] Linkbuilder.io. "DA Vs. DR: Link Building Metrics Uncovered." Accessed: 2025-04-02. (n.d.), [Online]. Available: https://linkbuilder.io/da-vs-dr-seo-metrics/.

[23] BlackHatWorld Forum Users. "Looking for hidden backlinks." Accessed: 2025-05-11. (2024), [Online]. Available: https://www.blackhatworld.com/seo/looking-for-hidden-backlinks.1711965/.

[24] BlackHatWorld Forum Users. "Where to buy these types of backlinks? (Hidden backlinks, kind of SAPE)." Accessed: 2025-05-11. (2018), [Online]. Available: https://www.blackhatworld.com/seo/where-to-buy-these-types-of-backlinks-hidden-backlinks-kind-of-sape.993201/.

[25] D. Hedin and A. Sabelfeld, "A perspective on information-flow control," in *Software safety and security*, IOS Press, 2012, pp. 319–347.

[26] H. Mantel, "Information Flow and Noninterference," in *Encyclopedia of Cryptography and Security*, H. van Tilborg and S. Jajodia, Eds. Boston, MA: Springer US, 2011, pp. 605–607, ISBN: 978-1-4419-5906-5. DOI: 10.1007/978-1-4419-

5906-5_874. [Online]. Available: https://doi.org/10.1007/978-1-4419-5906-5_874.

[27] J. McLean, J. Millen, and V. Gligor, "Non-interference, who needs it?" In *Proceedings of the IEEE Workshop on Computer Security Foundations*, 2001, pp. 237–238.

[28] scikit-image contributors. "Structural similarity index." Accessed: 2025-03-27. (2025), [Online]. Available: https://scikit-image.org/docs/stable/auto_examples/transform/plot_ssim.html.

[29] Z. Wang, A. Bovik, H. Sheikh, and E. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, 2004. DOI: 10.1109/TIP.2003.819861.

[30] N. Krawetz. "Looks Like It." Accessed: 2025-03-31. (2011), [Online]. Available: https://www.hackerfactor.com/blog/index.php?/archives/432-Looks-Like-It.html.

[31] S. Stewart and D. Burns, *WebDriver 2.0*, W3C Working Draft, 6 March 2025, Mar. 2025. [Online]. Available: https://www.w3.org/TR/2025/WD-webdriver2-20250306/.

[32] B. García, J. M. del Alamo, M. Leotta, and F. Ricca, "Exploring Browser Automation: A Comparative Study of Selenium, Cypress, Puppeteer, and Playwright," in *Quality of Information and Communications Technology*, Cham: Springer Nature Switzerland, 2024, pp. 142–149, ISBN: 978-3-031-70245-7.

[33] S. Merity. "Navigating the WARC File Format." Accessed: 2025-03-31. (Apr. 2014), [Online]. Available: https://commoncrawl.org/blog/navigating-the-warc-file-format.

[34] Common Crawl. "Common Crawl Web Graphs." (2025), [Online]. Available: https://commoncrawl.org/web-graphs.

[35] T. Vaughan. "Host- and Domain-Level Web Graphs December 2024 and January/February 2025." (Feb. 2025), [Online]. Available: https://commoncrawl.org/blog/host--and-domain-level-web-graphs-december-2024-and-january-february-2025.

[36] V. L. Pochat, T. V. Goethem, S. Tajalizadehkhoob, M. Korczyński, and W. Joosen, "Tranco: A research-oriented top sites ranking hardened against manipulation," *arXiv preprint arXiv:1806.01156*, 2018.

[37] P. Pochat, T. V. Goethem, and W. Joosen. "Tranco List Downloads." Accessed: 2025-03-31, Tranco: A Research-Oriented Top Sites Ranking. (2025), [Online]. Available: https://tranco-list.eu/#download.

[38] Sucuri Security. "Sucuri - Complete Website Security, Protection & Monitoring." Accessed: 2024-11-15. (2024), [Online]. Available: https://sucuri.net/.

[39] J. Andrijauskas. "Improvements to SiteCheck Website Scanner." Accessed: 2025-04-08. (Jan. 18, 2019), [Online]. Available: https://blog.sucuri.net/2019/01/improvements-to-sitecheck-website-scanner.html.

[40] Sucuri, *SiteCheck Mid-Year 2024 Report*, Accessed: 2025-04-08, 2024. [Online]. Available: https://sucuri.net/documentation/Sucuri-Mid-Year-2024.pdf.

[41] B. Eriksson, G. Pellegrino, and A. Sabelfeld, "Black widow: Blackbox data-driven web scanning," in *2021 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2021, pp. 1125–1142.

[42] B. Eriksson, A. Stjerna, R. D. Masellis, P. Rüemmer, and A. Sabelfeld, "Black Ostrich: Web application scanning with string solvers," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, 2023, pp. 549–563.

[43] M. Degerman and D. Dubrefjord, "Modular Blackbox SQL Injection Vulnerability Web Scanning," M.S. thesis, Chalmers University of Technology, Department of Computer Science and Engineering, Gothenburg, Sweden, 2021. [Online]. Available: `https://odr.chalmers.se/items/a76c863a-92cf-4811-8c01-7782d10c845c`.

A

Appendix 1 - AI usage

This thesis has been aided by multiple LLM models to speed up the work. Models used includes:

- OpenAI GPT-4o
- OpenAI GPT-4o mini
- OpenAI o3-mini
- Anthropic Claude 3.5 Sonnet
- GitHub Copilot - Anthropic Claude 3.7 Sonnet Thinking (preview)
- GitHub Copilot - OpenAI ChatGPT 4o
- NotebookLM - Gemini 1.5 Pro
- NotebookLM - Gemini 2.0 Flash

A.1 Coding

The usage has been done primarily to speed up coding and to find bugs causing scripts developed to malfunction. Example of prompts used include:

- What is causing this "ERROR MESSAGE" in my code?
- Is there any way to improve the logic to make the program run faster?
- Why doesn't this work?
- Explain this
- Give me a function that does X, Y, Z based on these requirements: a, b, c

A.2 Writing

LLM has been involved in the writing of the text, both to aid in the grammar and flow of the text. But also to get ideas on how to formulate sections. LLM has, however, never been used to generate entire sections, and careful consideration has always been given to any improvements suggested to make sure the original idea is still present. Example of prompts used include:

- Can you fix the grammar of this sentence
- How can I improve the readability?
- What is an appropriate way to start a section about X, given that I want it to contain, "LIST OF THINGS"
- How do I get X in LaTeX?
- What is causing this "ERROR MESSAGE" in LaTeX?
- Why does LaTeX hate me, and what is causing this error?

B

Appendix 2 - Sampled data

The sampled data of both the suspected hidden dataset and the tranco dataset, used to draw statistical conclusions from. The URLs are masked so as not to expose any vulnerable sites, subpages are indicated with Xs as to show the depth but not the actual path.

Table B.1: Tranco 1% sample.

URL	Hidden	Visible	TP	FP	TN	FN	Sucuri SPAM	Sucuri Hidden	Sucuri Score
b***a.nb.ca	0	1	0	0	1	0	0	0	0
v***n.com.my	1	3	0	1	3	0	0	0	0
s***a.us	0	2	0	0	2	0	0	0	0
o***e.com	0	41	0	0	41	0	0	0	0
b***e.tools	0	0	0	0	0	0	0	0	0
n***n.com	0	0	0	0	0	0	0	0	0
m***t.de	21	1	0	21	1	0	0	0	0
h***y.net	0	0					0	0	0
j***e.ru	0	15	0	0	15	0	0	0	0
u***e.com	5	28	0	5	28	0	0	0	0
l***g.com	0	0	0	0	0	0	0	0	0
g***s.com	1	0	0	1	0	0	0	0	0
c***a.top	0	0					0	0	0
g***p.cn	0	0					0	0	0
6***s.bet	0	0	0	0	0	0	0	0	0
o***s.com	0	4	0	0	4	0	0	0	0
a***n.com	0	0	0	0	0	0	0	0	0
p***d.com	0	0					0	0	0
b***o.app	0	0	0	0	0	0	0	0	0

a***g.com	0	0	0	0	0	0	0	0	0
s***o.de	0	4	0	0	4	0	0	0	0
i***m.icu	0	0	0	0	0	0	0	0	0
w***d.com	0	8	0	0	8	0	0	0	0
g***e.com	0	0					0	0	0
t***a.com	0	0	0	0	0	0	0	0	0
n***h.gov.vn	14	12	3	11	12	0	0	0	1
z***j.ru	0	21	0	0	21	0	0	0	0
m***r.com.my	0	0	0	0	0	0	0	0	0
d***a.co.in	0	1	0	0	1	0	0	0	0
c***a.com	0	3	0	0	3	0	0	0	0
b***a.gov	0	13	0	0	13	0	0	0	0
h***r.co.uk	3	4	0	3	4	0	0	0	0
t***b.com	0	7	0	0	7	0	0	0	0
z***n.ru	0	10	0	0	10	0	0	0	0
x***i.com	0	1	0	0	1	0	0	0	0
p***z.top	0	2	0	0	2	0	0	0	0
s***a.com	0	0	0	0	0	0	0	0	0
g***a.org	0	6	0	0	6	0	0	0	0
e***y.com	0	0	0	0	0	0	0	0	0
j***s.com	0	3	0	0	3	0	0	0	0
p****m.org	287	6	287	0	6	0	0	0	1
r***a.com	0	0	0	0	0	0	0	0	0
n***h.com.ua	0	1	0	0	1	0	0	0	0
p***2.online	0	3	0	0	3	0	0	0	0
i***k.com	3	14	0	3	14	0	0	0	0
n***4.com	0	6	0	0	6	0	0	0	0
c***p.co.jp	0	1	0	0	1	0	0	0	0
c***e.xyz	0	0					0	0	0
t***m.ua	0	2	0	0	2	0	0	0	0
c***x.jp	1	7	0	1	7	0	0	0	0

l***w.com	0	0					0	0	0
j***c.com	0	0	0	0	0	0	0	0	0
m***i.click	0	0	0	0	0	0	0	0	0
m***m.site	0	8	0	0	8	0	0	0	0
p***e.com	0	1	0	0	1	0	0	0	0
m***a.co	15	3	0	15	3	0	0	0	0
p***s.com	1	0	0	1	0	0	0	0	0
m***n.ru	0	0	0	0	0	0	0	0	0
u***d.org	0	5	0	0	5	0	0	0	0
p***p.online	1	1	1	0	1	0	0	0	1
m***i.co.uk	0	5	0	0	5	0	0	0	0
d***p.com	0	1	0	0	1	0	0	0	0
a***y.com	0	13	0	0	13	0	0	0	0
s***s.co.uk	0	0	0	0	0	0	0	0	0
e***s.ro	10	9	0	10	9	0	0	0	0
s***n.click	0	0	0	0	0	0	0	0	0
o***s.de	0	4	0	0	4	0	0	0	0
p***i.co.id	1	4	0	1	4	0	0	0	0
g***b.ru	0	0	0	0	0	0	0	0	0
h***e.com	0	26	0	0	26	0	0	0	0
a***s.com	0	0	0	0	0	0	0	0	0
r***t.org	1	5	0	1	5	0	0	0	0
b***s.com	0	3	0	0	3	0	0	0	0
l***i.com	0	0					0	0	0
e***o.com	5	32	3	2	32	0	0	0	1
h***d.kr	0	13	0	0	13	0	0	0	0
s***n.ink	0	0	0	0	0	0	0	0	0
s***a.ru	0	4	0	0	4	0	0	0	0
t***j.com	9	1	0	9	1	0	0	0	0
b***a.ru	0	3	0	0	3	0	0	0	0
c***l.com	0	7	0	0	7	0	0	0	0

l***d.com	0	2	0	0	2	0	0	0	0
g***t.com	0	1	0	0	1	0	0	0	0
w***o.org	3	10	0	3	10	0	0	0	0
c***f.com	0	0	0	0	0	0	0	0	0
l***y.space	0	0	0	0	0	0	0	0	0
v***l.com	0	0	0	0	0	0	0	0	0
a***d.ru	0	72	0	0	72	0	0	0	0
d***y.cn	0	0	0	0	0	0	0	0	0
f***e.net.au	2	17	1	1	17	0	0	0	1
l***2.com	0	0	0	0	0	0	0	0	0
s***s.com	49	22	0	49	22	0	0	0	0
t***a.co	0	0	0	0	0	0	0	0	0
b***t.net	0	8	0	0	8	0	0	0	0
r***n.in	0	12	0	0	12	0	0	0	0
j***s.xyz	0	0	0	0	0	0	0	0	0
j***c.com	1	11	0	1	11	0	0	0	0
k***r.com	0	0	0	0	0	0	0	0	0
t***p.com	0	0	0	0	0	0	0	0	0
g***d.live	0	2	0	0	2	0	0	0	0
			295	139	535	0			5

Table B.2: Suspected hidden 1% sample.

URL	**Hidden**	**Visible**	**TP**	**FP**	**TN**	**FN**	**Sucuri SPAM**	**Sucuri Hidden**	**Sucuri Score**
t***f.com/x	4	5	4	0	5	0	0	1	0
c***s.com/x/	1	9	1	0	9	0	0	0	1
r***k.co.uk	8	6	8	0	6	0	0	0	1
n***n.de/x/x/x/	1	10	1	0	10	0	0	0	1
w***e.pt	480	3	480	0	3	0	1	0	1
s***r.in/x	1	11	1	0	11	0	0	0	1
l***t.com/x	0	1	0	0	1	0	0	0	0

j***s.org/x	0	7	0	0	7	0	0	0	0
f***b.in/x	1	1	1	0	1	0	0	1	0
i***n.cz	4	7	4	0	7	0	1	0	1
l***s.com/x/x	1	11	1	0	11	0	0	0	1
d***l.com.au	6	4	6	0	2	2	0	1	0
z***a.pl	8	47	7	1	47	0	0	1	0
c***d.academy/x	1	2	1	0	2	0	0	1	0
y***2.ru	8	2	8	0	2	0	0	0	1
v***e.hu/x	0	0	0	0	0	0	0	0	0
c***a.fr	6	17	0	6	17	0	0	0	0
s***r.repair	5	24	3	2	24	0	0	0	1
l***n.net	1	3	1	0	3	0	0	0	1
s***s.tw/x	2	43	0	2	43	0	0	0	0
r***m.ru/x/x	4	4	3	1	4	0	0	0	1
o***o.com	10	1	10	0	1	0	0	0	1
b***s.com/x/x	0	0	0	0	0	0	0	0	0
w***s.com/x	0	0	0	0	0	0	0	0	0
g***e.com							0	0	0
p***e.ro	5	7	5	0	7	0	0	0	1
f***s.co.uk/x/x	1	1	0	1	1	0	0	0	0
t***e.com	4	1	4	0	0	1	0	1	0
a***n.com	9	4	9	0	4	0	0	0	1
s***s.com/x/x	1	1	1	0	1	0	0	0	1
z***m.sk/x	21	7	21	0	7	0	1	0	1
g***s.com	47	0	37	10	0	0	0	0	1
j***y.org	5	10	5	0	10	0	0	1	0
i***o.com	8	0	0	8	0	0	0	0	0
f***e.org/x/x	3	12	3	0	12	0	1	0	1
j***o.nl	3	7	3	0	7	0	0	0	1
h***e.com	1	2	1	0	2	0	0	0	1
a***f.com	0	74	0	0	74	0	0	0	0

d***s.com/x/x/x/x/x	1	1	1	0	1	0	0	0	1
l***o.com	4	1	4	0	1	0	0	0	1
c***n.com/x	40	12	40	0	12	0	0	1	0
m***o.at	1	6	1	0	6	0	1	0	1
s***e.com/x/x	3	64	0	3	64	0	0	0	0
p***o.fr/x	8	5	8	0	5	0	1	0	1
g*m.com	10	10	8	2	10	0	0	1	0
d***y.ru	9	1	8	1	1	0	0	0	1
a***l.com.tr	6	9	6	0	9	0	0	0	1
a***2.com/x	0	12	0	0	12	0	0	0	0
a***s.nl/x	11	5	9	2	5	0	0	1	0
e***a.com	9	0	7	2	0	0	0	1	0
d***a.cz/x	2	8	2	0	8	0	0	0	1
j***s.com/x	1	2	1	0	2	0	0	1	0
s***n.com/x	1	10	1	0	10	0	0	0	1
b***n.com.au	6	3	6	0	3	0	0	0	1
f***s.co.uk	5	7	5	0	7	0	0	1	0
s***a.com/x	5	10	5	0	10	0	1	0	1
a***l.in/x	0	2	0	0	2	0	0	0	0
t***c.com	97	2	97	0	2	0	0	1	0
p***y.com	0	2	0	0	2	0	1	0	0
a***a.in							0	0	0
b***y.com.au							0	1	-1
n***n.com	7	6	7	0	6	0	0	0	1
c***e.com/x/x	3	7	2	1	7	0	0	1	0
z***m.cz/x/x	1	12	1	0	12	0	1	0	1
a***e.ca	2	6	1	1	6	0	0	1	0
i***l.com/x/x	0	13	0	0	13	0	0	0	0
			849	43	557	3		SUM:	31

C

Appendix 3 - Full distribution graphs

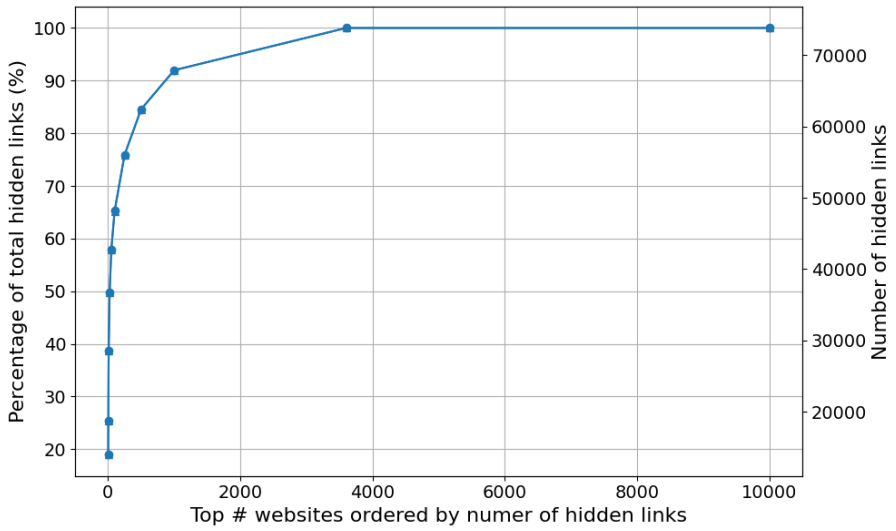


Figure C.1: Distribution of hidden links from the Tranco random dataset.

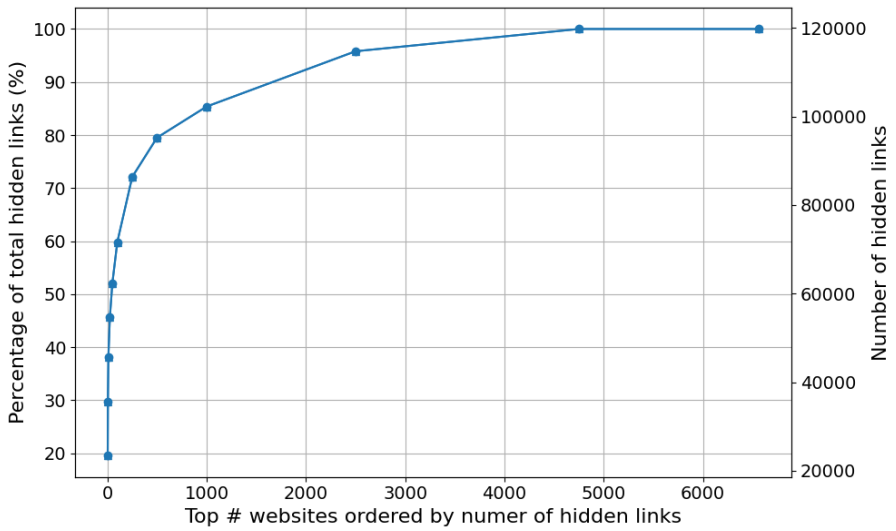


Figure C.2: Distribution of hidden links from the suspected hidden dataset.