Deanonymization techniques on Tor Hidden Services

Rahul Ravi Hulli
Graduate Student, CIISE
Concordia University
Montreal, Canada
Email: rahulravi.hulli@gmail.com
Student ID: 40234542

***Abstract*—This document is a part of three other documents submitted by my peers on this topic. The document has been written in a sequence of understanding the scenario, understanding the work process, and gathering results once these are executed. This document essentially introduces you to the concept of Tor hidden services, Protocol-Level Attack, the three types of attacks under this method, it's outcomes, challenges and improvements, some key findings, and mapping vulnerabilities to the key findings. The end goal is to give a general idea about the index of vulnerability of Tor hidden services.**

Keywords – Deanonymization, Tor Hidden links, Protocol level attacks, HTTP / HTTPS Response, SSL information gathering, HTTP Statistics.

I. Introduction

The Tor Browser, often known as "The Onion Router," is a fantastic resource for online privacy and anonymity. The Tor Browser [9], which was created on top of the Firefox platform, is intended to improve user security by masking the user's identity and location through the use of a network of computers run by volunteers, or nodes, to route internet traffic. Let us look at the design of the Tor Browser, its fundamental ideas, and the wide variety of uses that have elevated it to the status of a vital resource for people all around the world.

The underlying technology behind the Tor Browser is onion routing, which encrypts data in layers that mimic the layers of an onion. A user's data is routed through a number of nodes when they use the Tor Browser to browse the internet; each node removes a layer of encryption from the data. Enhancing anonymity, this multi-layered technique makes sure that no single node has full visibility into the source and destination of the user's internet traffic.

The uses of Tor browser [10] are as below:

Anonymous surfing: Offering users a private and anonymous surfing experience is one of the main purposes of the Tor Browser. People who are worried about their privacy or who live in areas where internet access is limited can use the Tor Browser to visit websites without disclosing who they are or where they are.

Overcoming Censorship: The Tor Browser is an effective means of gaining access to content that is restricted in areas where internet censorship is common. Through the use of a decentralized network and encryption, users are able to get around limitations and access information without restriction.

Activism and Whistleblowing: The Tor Browser is essential in helping activists and whistleblowers by offering a safe communication environment. Users can exchange private information without worrying about being watched, guaranteeing that their identities are safe.

Safe Online Transactions: The Tor Browser offers an extra degree of secrecy for anyone who is worried about the security of online transactions. By stopping third parties from monitoring users' online activity, it protects users' financial and personal data.

Research and Journalism: To safeguard their sources and uphold confidentiality, researchers and journalists covering delicate subjects frequently utilize the Tor Browser. This is especially important when doing investigative journalism or researching potentially politically sensitive subjects.

On a webpage, hyperlinks that are not immediately visible to the user are referred to as hidden links. Usually, these links are hidden using a combination of styles, such making the font smaller so it's almost undetectable or making it the same color as the backdrop. Although hidden links were originally used for good reasons, such as optimizing user experience or for SEO strategies, they are now frequently connected to search engine manipulation techniques. Websites that use hidden links [11] risk being penalized by search engines since it may be interpreted as an attempt to manipulate the ranking algorithms. To preserve transparency and steer clear of any potential drawbacks from the usage of hidden links, webmasters and SEO experts should apply caution and ethical standards.

The act of disclosing the true identity or location of a user or service using the Tor network is known as "deanonymization of onion links." [12] Onion links are connected to dark web domains that are intended to offer anonymity by directing internet traffic via a number of servers run by volunteers. Nonetheless, deanonymization could result from specific flaws or clever attacks, jeopardizing user security and privacy. Methods like traffic analysis, timing attacks, or taking advantage of software flaws may reveal a user's true IP address or location. There are significant concerns associated with deanonymization, particularly in areas where access to specific information may be prohibited or unlawful. The Tor network's ability to provide a private and secure online environment depends on ongoing efforts to patch any flaws and keep ahead of any new threats.

II. Background

Let us lightly grace over the platform on which this project has been centered around.

This specialized web browser called the Tor Browser was created to improve online anonymity and privacy. Based on the Firefox platform, it is set up to reroute internet traffic via the volunteer-run Tor network, which bounces a user's connection around a number of servers run by volunteers in an effort to improve security and privacy. Users that value online privacy and want to visit websites without disclosing their location or identity are especially fond of the Tor Browser.

Conversely, hidden links are usually URLs or hyperlinks that are not readily apparent on a webpage; in order to find and access them, users may need to work with the source code or utilize specialized tools. Although hidden links are not always harmful, they may be connected to services that prioritize privacy or, in certain situations, content that is difficult to obtain through traditional channels. It's crucial to remember that not all hidden links are connected to bad or unlawful activity [13]; in fact, some of them could have valid uses in the context of web development or privacy-related projects.

No matter how safe this platform seems, there always have been away by which this browser and its hidden services were attacked and deanonymized, thus defeating the purpose [14]. We will explore one such attack method and try it out to see how effective it is and what could be the solutions against it.

A. *Protocol Level Attack*

Exploiting weaknesses in the communication protocols utilized by the Tor network constitutes a protocol attack [1] to deanonymize Tor links. Targeting Tor, which routes traffic through a network of encrypted nodes to protect user anonymity, can be done in a few different ways:

Adversaries may use traffic analysis [2] to determine the origin or destination of encrypted traffic by examining patterns, timing, and packet size.

Attackers can correlate timings and possibly determine the source or destination by monitoring how long it takes for data to move through the Tor network. This technique is known as end-to-end timing attacks [3].

Sybil Attacks [4]: By managing several malevolent nodes, an assailant can analyze traffic patterns and is especially successful when coupled with traffic analysis.

Protocol Vulnerabilities [5]: An attacker can directly obtain information about the traffic flowing over the network by taking advantage of flaws in the Tor protocols.

Confirmation Attacks [6]: In order to link user behavior with observed traffic patterns and maybe verify user identification, attackers insert malicious code into websites or services that are accessed.

Intersection Attacks [7]: In order to obtain information about the source and destination, attackers try to correlate traffic patterns at both entry and exit nodes by manipulating several nodes.

The Tor team is constantly working to strengthen protocol security and fix vulnerabilities in order to mitigate protocol attacks. By staying up to date, avoiding nefarious exit nodes, and using extra encryption layers within the Tor network, users can improve their security [8]. Encouraging cybersecurity behaviors, such staying away from personally identifying information and updating software often, helps to make users safer overall on the Tor network.

B. *Utilization of this protocol in this project*

This project will undergo three methods under this protocol to gather information and deanonymize the server hosts, namely HTTP / HTTPS Response [15], SSL information gathering [16] and “Downloading” the websites [18]. The details of these methods are given in their respective sections.

III. Report Organization

The rest of the report is briefly outlined, providing a roadmap for readers to navigate through the subsequent sections. The structure encompasses an exploration of methodologies and approaches, with dedicated sections on dataset acquisition, pre-processing steps, and the implementation of machine learning algorithms. The report concludes with an extensive discussion on achieved results, insights from the confusion matrix, and the overall performance of the employed machine learning models.

IV. Main Text

The main text of this report is structured into distinct sections to provide a comprehensive overview of the project's development, execution, and outcomes.

V. Contribution and Disclaimer

I would want to stress that, as one of three team members working on a collaborative research project, every piece of information in this project report is derived from my own personal experience and research. The other team members and I made an equal contribution to the project, ensuring that duties and responsibilities were distributed fairly. The outcomes we achieved as a team were the product of our combined efforts, and they demonstrated effective teamwork.

VI. Methodology

A. *Method 1: Analysis of values obtained by HTTP / HTTPS Responses.*

The "Deanonymizing techniques of Tor links" project tackles the important problem of assessing the anonymity and security of Tor services, which are intended to give users more privacy and secrecy when they browse the internet. An essential part of this project's analysis of Tor links' resilience to possible deanonymization methods is the Python code that was written for it. Finding holes or other weak spots in the Tor network that can jeopardize user anonymity is the main goal.

The algorithm accomplishes this by putting into practice a number of thoughtfully chosen commands, each intended to mimic different situations and possible points of attack. These instructions make use of programs like curl [19], nc (netcat) [25], ssh [26], and wrappers specific to Tor, such torsocks [27] and torify. The wide range of commands investigates several facets of interactions across Tor links, such as header-manipulated HTTP requests, direct network connections, SSH connections, and attempts to access resources that would be considered sensitive.

Because torsocks and torify allow network traffic to be routed through the Tor network, ensuring that the analysis is carried out within the context of the Tor environment, their use is especially important. This is necessary to evaluate how well Tor services manage different kinds of connections and requests while maintaining user anonymity.

The code attempts to find potential flaws that might be used to deanonymize Tor users by methodically carrying out these commands and examining the results. In addition to helping developers and security professionals improve the privacy and security features of the Tor network, this thorough approach to security assessment provides insightful information about how well Tor services work.

Overall, the project and its associated Python code contribute significantly to ongoing efforts to strengthen Tor's anonymity and security provisions, a critical tool for users seeking online privacy in an era of increasing digital surveillance.

Among the commands employed in the analysis [28] are:

1. torsocks curl -X GARBAGE -H "Host: {link}" {link}: This command connects to the designated Tor link by using torsocks and the curl tool. One way to test how the Tor service responds to unusual requests is to use "GARBAGE" in the HTTP request and modify the "Host" header.
2. torsocks curl -X GARBAGE -H "Host: " {link}: Like the previous command, this one uses torsocks to inspect the Tor link's response to a request that contains an empty "Host" header in order to investigate any potential behaviors or vulnerabilities.
3. torsocks nc {link} 80: This command starts a network connection (nc) to the given Tor link on port 80 using torsocks. Examining how the Tor service reacts to direct network connections is made easier with the use of netcat (nc).
4. torify nc {link} 80: This command, like the previous one, establishes a network connection to the Tor link using torify. Torify is a wrapper for Torifying programs—in this example, netcat—that are not natively compatible with Tor.
5. torsocks ssh -o FingerprintHash=md5 {link}: This command connects via SSH to the given Tor link using torsocks, and it also has the option to use MD5 as the fingerprint hash. This investigates any potential flaws or vulnerabilities in SSH connections established via Tor.
6. torsocks curl {link}/private_key: This command requests a resource—possibly a private key—from the given Tor link using torsocks and curl. This evaluates the way in which requests for private or restricted resources are handled by the Tor service.

1) *Improvisation of this algorithm*

We added threading to the Python codebase to simplify and accelerate the evaluation of Tor links in the "Deanonymizing techniques of Tor links" project. By analyzing multiple Tor links at once, this threading [29] implementation significantly increases overall efficiency by enabling the concurrent execution of multiple instances of the program. As a result of this concurrency, controlling the program's output becomes more difficult. When several threads run simultaneously, the output produced by each instance may become mixed together, which could result in a confusing and possibly chaotic output.

Further steps to methodically arrange the output are needed to lessen this problem and preserve the results' clarity. The output from each thread can be gathered and arranged using thread-safe data structures like queues. Locks and other synchronization techniques can also be used to manage access to shared resources, avoiding conflicts and guaranteeing that each thread produces output that makes sense.

This refinement makes sure that the final product is organized and understandable while also maximizing the analysis's speed. In addition to speeding up the deanonymization assessment, efficiently controlling the simultaneous execution of multiple instances helps create a more reliable and approachable tool for assessing the security and anonymity of Tor links. This threading improvement shows a clever strategy for striking a balance between output organization and speed, increasing the overall efficacy of the deanonymization methods used in the project.

The "Deanonymizing techniques of Tor links" project presented a management and organization challenge. To address this, an initial approach was taken in which each Onion link was given its own folder, and the corresponding outputs were saved inside of these individual folders. Although this method offered a methodical and well-structured way to keep track of the results for every link, it unintentionally caused major delays in the analysis process. The program's efficiency was thus hampered, and it could take up to 5 or 6 hours to execute up to 30 links.

The overhead of creating and maintaining numerous folders at once most likely caused the delay. Performance bottlenecks and longer execution times may arise from the directory creation process, particularly in situations where a large number of links are being processed concurrently. This resulted in the choice to forgo the method of making a new folder for every link in favor of a simpler fix.

Rearranging the output structure is probably part of the updated plan to reduce delays. One way to do this would be to use a flat organizational structure, storing all outputs in a single directory rather than making subdirectories for every link. As an alternative, to achieve a compromise between organization and performance, batching links together or using a more efficient folder creation mechanism might be used. With these changes, the deanonymization assessment should run more quickly and effectively, handling more links in a reasonable amount of time.

2) Algorithm of this method

It should be noted that the structure of this algorithm resembles the actions an adversary might take in a traffic analysis attack to obtain details about the Tor links, thus jeopardizing the network users' anonymity. The algorithm as it is currently presented is a tool for network analysis and security assessment, not a real attack. For this reason, it is important to remember that it is analysis and falls well within ethical bounds.

Algorithm 1: Analysis of HTTP / HTTPS Responses

```
1. Initialize input_file = "onion_links.txt"
   Initialize output_folder = "output"
2. Define a function scan_link(link, output_dir):
      a. Define a list of commands to be executed for
         scanning the given link.
      b. Create the output directory for the link
         (link_output_dir) inside the output folder.
      c. Iterate through each command:
            i. Execute the command using
               subprocess.run with a timeout of 600
               seconds (about 10 minutes). (This is
               interchangeable)
           ii. Create an output file (output_file) for the
               command's result.
          iii. Write command details, status, and
               output/error to the output file.
           iv. Handle exceptions for timeout or
               general errors, writing corresponding
               information.
3. Define the main function:
      a. Read the list of Tor links from the input file.
      b. Create the output directory.
      c. Iterate through each link, calling scan_link(link,
         output_folder) for analysis.
4. If the script is executed directly (not imported as a
   module):
      a. Run the main function.
```

The output of this algorithm is provided in the appendix [Image 1]

This deanonymization method yields information that offers a thorough understanding of many facets of the targeted Onion services. The results are arranged in categories according to how frequently specific types of information appear in the examined data. Let's go into more detail about each category of data:

1. **Deanonymized Web addresses or IP addresses: (Frequency: Very Low)**
   - IP addresses connected to the examined Onion services are made visible through the deanonymization procedure. But the frequency is very low, meaning that getting the real IP address or website address is not a common result. Tor services, which are

intended to conceal the underlying infrastructure for increased anonymity, may be to blame for this.

2. **Available Ports for Communication: (Frequency: Low)**
   - Data regarding open ports used for communication is included in the analysis. A low frequency indicates that it is less common to find information about the particular ports that are used for communication. Although it might not always be present in the results, this information is crucial for comprehending the Onion services' network configuration.
3. **Identification of the web server (such as WordPress, Nginx, or Apache): (Frequency: High)**
   - The Onion services' web server type is identified, and the frequency is classified as medium. Determining whether a service is run on servers such as Nginx, Apache, or even content management systems like WordPress can reveal information about the technology stack that the operators of the Onion sites use. Understanding the web server's infrastructure and possible vulnerabilities is made easier with the help of this information.
4. **Operating system of the host (such as Windows, Linux, or Debian): (Frequently: Moderate)**
   - The deanonymization method performs exceptionally well in identifying the host operating system; a high frequency suggests that the underlying OS is routinely extracted. Determining whether the host is running Windows, Linux, or Debian is essential for evaluating security setups, potential vulnerabilities, and the general security posture of the Onion services.

In conclusion, the technique shows a noteworthy success rate in identifying the web server and host operating system, even though it might not always deanonymize IP addresses or provide information about open ports. The aforementioned insights hold significant value in security assessments as they facilitate a more profound comprehension of the Onion services' technology stack, possible points of attack, and overall security ramifications.

3) *What data was revealed using this method?*

This method worked, but at the same time, it did not seem enough. The method did dig in and reveal some Clearnet websites and some IPs, but we were very quick to realize something interesting about these results.

1. **These were well established and legal sites**: Websites such as Klecker [37] and Tor Project leaked their Clearnet web addresses [38]. To be real, this is enough to prove that this method does work, but it also shows that there is a method to halt the working of this method and the darknet web hosting agencies know it. The idea is simple, hide your configuration file. We will learn this shortly.
2. **Not all IPs revealed are trace-able**: We could very well go ahead and check these Clearnet address IPs using services such as Shodan, but you would realize that some of these addresses are no longer active to give you any results. It simply would be a waiting game for the activity of these addresses online here on.
3. **In almost every case, you would not really get much information from these Clearnet addresses**: Note, that these are not addresses that co-link dangerous and illegal activities, but simply, in most cases, a product advertisement. So, even if you get details out of them, you really would not be able to do anything with them anyways or, you will not see a point doing them.

A sample image of information revealed using this method is provided in the appendix [Image 2], [Image 3].

The results of this method, the information revealed, the dataset used, the graphs and so on, will be provided in the results section.

There is one very important thing that needs to be highlighted about the results. Almost every time, it's Apache that leaks the address. But we also mentioned that it's not exactly useful, but simply a case of leaked information. The reason for this argument relates more to the manner in which Apache works.

If you take a look at the apache server version 2.4 documentation [17], it clearly says that the server configuration file is written in such a way that an exception gets handled. If in any case the website cannot be accessed with errors (such as 400 Bad Request), the apache server leaks the address written in the configuration file at the most priority (At the top). Suppose we find a server host that has not configured this file properly and has placed a private address as its priority, then apache would leak this address. Hence, if every onion site server has this file configured well, then the output of this method has no real use.

4) *Challenges with this method*

Deanonymizing Onion sites and collecting related IPs with the given Python code is recognized as a useful but labor-intensive task. Although the approach might produce results, there are some intrinsic difficulties that affect its effectiveness and success rate.

1. **Time-consuming**: Several commands, such as network requests and security scans, must be carried out for every Onion link in the analysis. It naturally takes time to use this multifaceted approach, especially when working with numerous links. This could cause the process to take longer than expected and result in outputs that are later than expected.
2. **Offline Onion Sites**: The dynamic and transient nature of onion services is a noteworthy challenge. It is possible for an Onion site to go offline during the analysis process, making the data collected outdated. There is a chance of learning that the targeted Onion site is inaccessible due to the delay between starting the analysis and receiving results.
3. **Limited Success Rate**: A number of factors, such as the particular configurations and defenses put in place by Onion service operators, have a significant impact on the deanonymization process's success. In addition, the overall success rate is influenced by the stability of the targeted site, server responsiveness, and network conditions. With these things considered, the likelihood of successfully deanonymizing an onion site and collecting IPs may be minimal.
4. **Resource-intensive**: Several network requests and security scans for every Onion link are made during the process. This could require a lot of computational power and network bandwidth, which could cause bottlenecks and increase the execution time overall.

In light of these difficulties, it is critical to recognize the shortcomings of the current strategy and investigate possible improvements or substitute techniques in order to raise the effectiveness and success rate of the deanonymization procedure. The overall efficacy of the deanonymization efforts could be increased by investigating options like parallelization, more focused scanning methods, and adaptive strategies that consider the dynamic nature of onion services. Furthermore, continuous observation of Onion sites for variations in responsiveness and availability could lead to faster and more accurate results.

5) Improvement Scope

To increase the efficiency, accuracy, and speed of the analysis, the algorithm must be improved, and alternative approaches must be investigated in order to deanonymize Tor links and obtain more information quickly. The use of asynchronous and parallel processing to shorten processing times overall, output structure optimization for better organization, dynamic timeout adjustments based on link responsiveness, selective command execution based on relevance, integration of additional network analysis tools, application of machine learning and pattern recognition techniques, implementation of real-time monitoring for link status changes, user interaction for unclear cases, integration with threat intelligence feeds for context, and ongoing research and development are some suggestions for improvements. These enhancements collectively aim to make the deanonymization process more adaptive, efficient, and resilient to challenges associated with analyzing Tor links.

6) Does "Server-Status" reveal valuable information?

You can try adding "/server-status" [39] to the end of the onion link in the Tor Browser address bar in order to access the "server-status" page or endpoint if the onion site you are trying to monitor has one. As an illustration:

```
http://your-onion-site.onion/server-status
```

But remember that many websites disable their "server-status" pages for security reasons, and that the enabling one is a server configuration. A "404 Not Found" error message might appear if the website doesn't have a "server-status" page when trying to access it in this manner.

We tested the server status feature on the Tor Browser using a range of links, and we consistently saw that a significant amount of information was not returned in detail. When information was released, which was rare, it was usually about servers that are already well-known in the Tor community. These servers are known to be lawful, and they even have a clear net version that can be accessed from outside the Tor network. This minimal disclosure is consistent with the core values of the Tor Project, which prioritize user anonymity and privacy. The purposeful opacity of information, particularly with regard to concealed services or nodes attempting to maintain anonymity, highlights the Tor Project's dedication to preserving a private and safe browsing environment for its users. This deliberate design choice contributes to the network's reputation as a robust tool for safeguarding online privacy.

Server status sample of one such server has been provided in the appendix. The link to this server and certain details are also provided [Image 5].

B. *Method 2. Gather Secure Socket Layer Details*

This approach adds a few more commands to the list in the hopes that they will disclose some personal information. "SSLScan" [40] and "TestSSL" are these.

As an invaluable tool for vulnerability assessment, security configuration, and compliance checks pertaining to SSL/TLS implementations, SSLScan is essential to the upkeep of server security. Security experts use SSLScan to carefully assess how reliable SSL/TLS connections are on particular servers. This entails a thorough examination of protocols and supported ciphers, as well as the detection of any potential flaws in the SSL/TLS architecture.

The capacity of SSLScan to perform vulnerability assessments is one of its main advantages. It helps security experts identify vulnerabilities in SSL/TLS configurations by means of methodical scanning. This is necessary to proactively detect and close possible security holes and make sure that servers are protected from being exploited by bad actors.

SSLScan is also very helpful when it comes to security configuration. The tool is used by system administrators to ensure that their servers are set up securely, following industry best practices, and staying away from known vulnerabilities or inadequate encryption algorithms. This feature of SSLScan reduces the possibility of unauthorized access or data compromise by enabling administrators to proactively create and maintain strong security postures for their servers.

Additionally, SSLScan aids in compliance checks by acting as a crucial tool for determining if servers adhere to accepted security guidelines and recommended practices for SSL/TLS configurations [41]. This is especially crucial for sectors of the economy where following compliance rules is required. Businesses can reduce the risk of regulatory violations and related repercussions by using SSLScan for compliance checks to make sure their servers meet the necessary security requirements.

Essentially, SSLScan is a flexible tool that performs a tripartite function in security configuration validation, vulnerability assessment, and compliance checks, all of which work together to improve servers' overall security posture and the integrity of SSL/TLS implementations.

As a command-line tool that makes security audits easier, TestSSL plays a crucial part in maintaining the security of SSL/TLS servers. Its ability to thoroughly inspect and evaluate different security-related SSL/TLS server configurations emphasizes how important it is. This thorough examination includes examining the supported cipher suites, protocol versions, and finding any potential vulnerabilities in the SSL/TLS infrastructure of the server.

One of the main aspects of TestSSL's significance is that it helps with comprehensive security audits. It enables security experts to explore the nuances of SSL/TLS servers because it is a tool made just for this purpose. It assists in assessing how strong a server's security posture is, spotting potential flaws, and enabling preventative actions to strengthen against possible security threats by closely examining supported cipher suites.

TestSSL is also essential for evaluating cipher suites. It gives administrators and security experts the ability to evaluate how strong a server's cipher suites are. Making decisions regarding the server's security configuration requires careful consideration of the results of this evaluation. Administrators can ensure a robust defense against potential security breaches by implementing configurations that comply with industry best practices by assessing the strength of cipher suites.

Additionally, TestSSL is essential for confirming support for the SSL/TLS protocol. It helps make sure that obsolete and unsafe protocols are disabled by identifying which versions of the protocols are supported by a server. Disabling dated and weak protocols helps reduce potential security risks, which makes it a crucial part of keeping a secure server environment.

All things considered, TestSSL proves to be a flexible and invaluable instrument for protocol support verification, cipher suite assessment, and security audits. Its many features greatly improve SSL/TLS server security by giving administrators important information and resources to help them make well-informed decisions about security configurations.

1) *SSLScan does reveal certain private details, but…*

After adding SSLScan and TestSSL to the current program, we reran the code. This time, it did take longer because SSLScan takes a while to produce results. The process was sped up by at least four to a maximum of thirty times with the help of threading, and SSLScan did provide some intriguing information.

Certain onion link IP addresses were made public by SSLScan. This shouldn't have happened, but as soon as the IP addresses were made public, we began researching them on websites that provide information about IP addresses, like Geolocation. "Shodan" [42] is one reputable website that does this. Details were provided, but this appeared differently. Since the information was consistently related to the United States and featured specific cities more frequently than others, we began to doubt the veracity of the material being provided to us. As a result, we chose to avoid Shodan even though it is well known that many cyber security experts use it as their first option.

We decided to go ahead with a method which would have details offline. And one such amazing option was geoip (geoip2) [20].

2) What did the geoip method reveal?

Let us first understand what this tool is and how it works.

In order to ascertain an IP address's approximate location, the GeoIP2 database links IP addresses with geographical data. The developer, MaxMind, gathers and aggregates information from multiple sources, using the data at hand to map IP addresses to precise geographic locations. Because of the database's hierarchical structure, information can be stored and retrieved quickly. Country, region, city, postal code, latitude, and longitude are examples of common data fields. GeoIP2 data varies in precision and accuracy; generally speaking, data at the national level is more accurate than data at the city level. In order to improve location accuracy and reflect changes in IP address allocations, MaxMind updates the databases on a regular basis.

Apart from databases that can be downloaded, MaxMind offers libraries and APIs (such as the `geoip2} library for Python) that facilitate the integration of geolocation functionality into applications and allow real-time IP address lookups. Users use GeoIP2 for a variety of things, including fraud prevention, content personalization, targeted advertising, and security. It is crucial to understand that although GeoIP2 databases provide useful approximations, their accuracy may differ, and they have limitations. Users ought to take into account these elements as well as the particular use cases that fit their applications. Applications needing extremely accurate location data might not be suitable for IP-based geolocation alone.

Below is the pseudocode of our code using geoip2

Algorithm 2: Using Geoip2

```
1. import the maxmind db
2. define lookup_ip_geolocation(arg ip_address, arg database_path):
    a. try:
        i. Open MaxMind database
        ii. Perform IP lookup: reader.get(ip_address)
        iii. if response:
                Extract geolocation information – country, city, latitude, and longitude
                Display geolocation information
        iv. else:
                Mention IP address not found
        v. except maxminddb.errors.InvalidDatabaseError:
                Handle invalid database file exception
           except Exception as e:
                Handle unexpected errors
3. Main:
    a. Get MaxMind DB file path and IP address from user input
    b. Store input to ip_to_lookup
    c. Call the lookup function: lookup_ip_geolocation(ip_to_lookup, database_path)
```

The output of this is provided in the appendix [Image 6].

Geoip2 now operates in an intriguing way. Every time it runs, it also offers a link to another location where it performs a lookup; we chose to click on this link to view additional information. At this point, the reason Shodan withheld reliable information becomes clear.

3) What is Bogon IP Address?

IP addresses that shouldn't be listed in the Internet's public routing table are referred to as bogus addresses [43]. They are either left unallocated or reserved for particular purposes, such as multicast addresses or private networks. In network security, bogus addresses are frequently filtered out to stop the use of erroneous or unapproved IP addresses. Examples include addresses set aside for testing or documentation (e.g., 192.0.2.0 to 192.0.2.255) and addresses reserved for private networks (e.g., 10.0.0.0 to 10.255.255.255). By adding filters to firewalls or routers, network administrators can increase security by using lists of Bogon addresses.

4) Why does SSLScan reveal Bogon IP Address?

There are several situations in which SSLScan may identify Bogon IP addresses. This includes instances in which improperly configured servers unintentionally reveal private or internal IP addresses. When conducting an SSLScan, network misconfigurations like routing issues or improper security system settings may reveal Bogon IP addresses. If SSL services are set up to listen on interfaces connected to private networks, dual-homed servers—which have multiple network interfaces—may unintentionally reveal internal IP addresses.

When there are network routing errors or attempts at IP spoofing or tampering, SSLScan can also identify Bogon IP addresses. Additionally, as SSLScan might have trouble differentiating between public and private IP addresses, tool limitations could be a factor in the inclusion of Bogon IPs in scan results. All things considered, the discovery of Bogon IP addresses during SSLScan draws attention to possible security vulnerabilities or configuration errors that need to be looked into and fixed by network managers and security experts. A secure and properly configured network requires regular security audits and monitoring.

5) Can this method be improved?

This approach has really reached its limit and cannot be improved. Why are we saying this now? These essential points become clear when you delve deeply into the operation of the Tor browser and onion links.

End-to-End Encryption (HTTPS): Tor does not by default encrypt the content of your communications, even though it offers anonymity and privacy regarding your IP address. To protect the content of your communications, it is advised that you use end-to-end encryption, like HTTPS.

Onion Routing: This is the fundamental idea behind Tor. When you join the Tor network, your internet traffic is redirected via a network of relays, which are servers (nodes) run by volunteers.

Many Levels of Encryption: Each time your data travels through a relay, a layer of encryption is peeled away to reveal the address of the relay that comes after it. Until the data reaches its intended destination, this process is repeated. This layered encryption is the reason for the term "onion routing."

Now, let us also focus on one specific disadvantage of SSLScan which closes this deal completely.

...Additionally, as SSLScan might have trouble differentiating between public and private IP addresses, tool limitations could be a factor in the inclusion of Bogon IPs in scan results....

Yes, SSLScan is horribly poor when it comes to recognizing private and public IPs, which means it fetches any IP addresses it sees during communication, which it potentially thinks it has discovered the system and its real address, which it really has not.

This shortcoming of SSLScan was not evident to us, and hence this method had to be rejected, but it did provide some valuable information. It is not that every time SSLScan runs, it will return a bogon address, but every time SSLScan returns an IP address, there is a high chance that it will be a bogon address, and this does not eliminate the possibility of deanonymization.

It is important to know that the revealing of bogon addresses indicates potential security risks or misconfigurations that should be investigated and addressed. Network administrators and security professionals should review scan results carefully, identify the root causes of any issues, and take appropriate measures to secure the network infrastructure. Regular security audits and monitoring are essential to maintaining a secure and well-configured network. Hence, at a corporate level, SSLScan can potentially catch systems that use tools such as TOR Browsers, hence this is a slight improvement over the last method. But on a broader scale such as the real world, this tool provides little to no help.

Sample SSLScan output has been provided in the appendix [Image 4].

C. Method 3: "Download" the Websites

The head topic here is slightly vague, hence this needs some explanation.

It is very difficult to access Tor websites using traditional download methods. Using wget is one solution, but this adds a layer of complexity. To guarantee proper functionality and privacy, wget must be used in conjunction with torsocks [44]; it is not sufficient to use it alone. To perform this combined operation in Linux, you prefix the wget command with torsocks, which routes the traffic via the Tor network. In order to access content from Tor websites securely and anonymously, you must follow these two steps. The combination of wget and torsocks allows for a more secure and private method of downloading files over the Tor network. The following is the command to utilize this service:

torsocks wget <url>

Where <url> is the url of the onion website.

In most cases, using torsocks in conjunction with the wget method is a dependable method; the only significant exception is when the server linked to a particular link is unavailable or experiencing downtime. Wget effectively downloads the server site's contents in most cases, enabling users to access the requested content safely and anonymously via the Tor network. But something interesting happens during this process: wget frequently downloads a single file, which is frequently called "index.html." This pattern points to a common default behavior whereby wget retrieves a website's default index file when it is not instructed to do otherwise. This realization emphasizes how crucial it is to comprehend and set up wget correctly in order to get thorough and precise downloads, particularly when working with Tor websites. It highlights that when using wget in conjunction with torsocks for private and secure content retrieval, users must be aware of the possible limitations and behaviors related to it.

We were initially perplexed when wget appeared to download a single file. We nearly believed that the approach might not be successful. Nevertheless, we later discovered an explanation on a Reddit post [21]. Important information was mentioned on the page: "Search engines must abide by a text file named 'robots.txt,' which specifies what can and cannot be indexed." A lot was made clear by this short line. It basically states that websites have a "robots.txt" file that instructs search engines on what content to ignore and what can be checked. We were able to decipher why wget was frequently retrieving the "index.html" file—possibly because it was adhering to the directives in "robots.txt." This discovery highlighted the need to pay attention to how websites are set up when using wget with torsocks for safe and private downloading.

What would happen if we stopped using "robots.txt" as a search engine and still downloaded the files? was our next query. And that's how this software was developed. This exception was handled by the program; if the robots.txt file was discovered during the wget process, use robots=off when running the command. This was the solution. This pseudocode can be found below.

Algorithm 3: Wget usage with torsocks and robots flag

```
1. import subprocess and os
2. def download_links(file_path):
    a. Downloads links from a file using Tor for
       enhanced privacy
    b. Arg: - file_path (str): The path to the file
       containing the list of links
    c. try:
         i. Read links from the file
        ii. Process each link
       iii. Check if robots.txt exists
              1. If robots.txt does not exist, run
                 wget with torsocks as it is
              2. If robots,txt exists, run wget
                 with torsocks with robots=on
    d. Handle file not found exception
3. Main
    a. Provide the path to the file containing the list of
       links
    b. Call the download_links function ->
       download_links(file_path)
```

We made a case by offering a file containing links. We fed a file containing 4740 links to wget after compiling it. We realized there was a problem right away.

Wget executes on each link in an incredibly long time. Depending on the size of the website and the strength of the internet connection, this can occasionally continue for hours. We made the hasty decision to add one more dispute to it. Call a timeout.

Although the actual time savings here depended on the timeout, setting a timeout would make it much faster in terms of execution. For how long is a period of time considered to be just right? Let me establish the following to ascertain this.

We had a set number of goals with this method. If you can deanonymize? Fantastic! If you cannot, I want other details that will help me. (These details will be explained later). And for the set parameters, after trial and error, we figured that 30 seconds it’s the perfect timeframe for this to work effectively and quickly. Note that 4700 links would still take about 1 and a half days to complete execution, but this was a far more affordable timeframe.

This returned some amazing results. The results will be discussed in the results section.

1) What parameters did we include?

Just as I previously stated, " If you can deanonymize? Fantastic! If you cannot, I want other details that will help me.:” What specifics did I mean by this? Let's examine each of them individually now:

a) Bitcoin Address

Bitcoin addresses are important pieces of data that can be gleaned from wget details. They indicate active participation in the cryptocurrency network and can provide insightful information about current transactions. It is well known that user privacy is a top priority for Bitcoin transactions, with addresses created to hide the identity of the owner and transaction details. There is a "sort of" workaround, though, that presents a more complex viewpoint. Even though Bitcoin addresses by themselves do not immediately reveal the identity of the owner or the specifics of a transaction, a certain amount of transparency becomes apparent when examining the public blockchain.

As an open ledger, the Bitcoin blockchain makes it possible to view transaction histories associated with individual addresses. This means that information about the flow of funds, the size of transactions, and the frequency of transactions connected to a specific Bitcoin address can be obtained by examining the blockchain. Blockchain analysis tools have been developed as a result of this transparency, making it possible to find patterns and connections between various addresses.

Blockchain forensics: These tools can identify patterns of behavior and address relationships, even though they might not reveal real-life identities [45]. Blockchain forensics is the term for this kind of analysis that is used for investigations, security, and compliance. As the cryptocurrency landscape is always changing, it is important for users to be aware of potential analytical techniques that could infer certain information. It's still difficult to find a balance in digital financial systems between privacy and transparency. Users are better equipped to navigate the cryptocurrency space with greater awareness when they are aware of the subtleties of Bitcoin addresses and transaction privacy. We have a combination of two websites that help us perform some level of forensics, and the best part is that they are free.

The official blockchain website [22]: One tool provided by the well-known cryptocurrency platform Blockchain.com is called Explorer. Users can examine and examine transactions that have taken place on the Bitcoin blockchain with this platform. By entering a transaction ID or Bitcoin address into the Explorer, users can examine specific transaction details, such as inputs, outputs, transaction size, and more. The Explorer also lets users examine individual blocks in the Bitcoin blockchain, providing details like the block's size, hash, timestamp, and list of transactions. You may also be able to access real-time network statistics, which include metrics such as total hash rate and transaction count. Blockchain.com, well-known for its bitcoin wallet services, might incorporate wallet functionality into the Explorer. Users can also visualize a variety of blockchain-related metrics with the help of the platform's charts and graphs, which help them understand trends and patterns in the cryptocurrency market. In general, Blockchain.com Explorer is essential for offering visibility and transparency into the operations of the Bitcoin blockchain, serving users, researchers, and enthusiasts who are interested in learning more about the larger cryptocurrency ecosystem.

Chainabuse: A website called Chainabuse [23] reports nefarious cryptocurrency transactions. If you examine each cryptocurrency transaction individually, you might ultimately come across a malicious one. Essentially, this software lets users examine specific bitcoin transactions to find any acts that might be deemed damaging or fraudulent.

2) How does this work?

We restricted our investigation to these two techniques since further analysis of blockchain relationships would be outside the purview of this study, even if using these two tools could aid in identifying the criminal.

Algorithm 4: Gathering Bitcoin Address details

```
1. Input:
     • Receive the folder_path representing the directory to be searched for files containing potential cryptocurrency addresses.
2. Define Regular Expression:
     • Define a regular expression (crypto_address_pattern) to identify common cryptocurrency addresses, covering both legacy and SegWit Bitcoin addresses.
3. Initialize Unique Addresses Set:
     • Initialize an empty set (unique_addresses) to store unique cryptocurrency addresses found during the search.
4. File Search and Processing:
     • Traverse the specified directory and its subdirectories using os.walk.
     • each file encountered, open and read its content, handling any decoding errors with errors='ignore'.
     • Utilize the regular expression to identify cryptocurrency addresses within the file content.
     • Add the identified addresses to the unique_addresses set.
5. Write to Output File:
     • Create an output file named "BTC.txt" to store the unique cryptocurrency addresses.
     • Write each unique address from the set to the output file.
6. Output:
     • The "BTC.txt" file contains a compiled list of unique cryptocurrency addresses found within the specified directory and its subdirectories.
7. Exception Handling:
     • Implement robust exception handling to capture and log any errors encountered during file processing, ensuring the algorithm gracefully handles unexpected issues.
8. Execution:
     • Execute the script when it is directly run (__name__ == "__main__"), utilizing the provided folder_path.
```

We tried to locate Bitcoin addresses in certain data, but it was challenging because the information we discovered might not have been a genuine Bitcoin address at all—rather, it might have been merely a random string of characters used as a key or in code. Verifying that the Bitcoin address we had discovered was legitimate had been vital. At first, we attempted to use a tool called blockchain.info, but it was not very effective and frequently produced unfavorable results. As a result, we had to personally review every address on the official Bitcoin website. It had taken a while and been a little annoying to do this manual testing process.

a) Wordpress / PHP / Drupal / Django Statistics

Wordpress / PHP statistics can be obtained through a method described in the below pseudocode of the code we used:

Algorithm 5: CMS Details

```
1. Include beautifulsoup library. This helps to parse html files [46].
2. Function Definition:
    • Define a function (e.g., get_wordpress_version) that takes a website URL as input.
3. HTTP Request:
    • Use the requests.get() method to make an HTTP GET request to the specified website URL. This retrieves the HTML content of the webpage.
4. HTML Parsing:
    • Utilize BeautifulSoup to parse the HTML content and create a BeautifulSoup object. This object allows you to navigate and search the HTML structure.
5. WordPress Version Detection:
    • Look for the WordPress generator meta tag in HTML. This tag often contains information about the WordPress version used by the website.
6. Output:
    • Return the detected WordPress version or an appropriate message based on the presence or absence of the generator meta tag.
7. Error Handling:
    • Implement error handling to capture and handle potential issues, such as failed HTTP requests or missing WordPress version information.
8. Script Execution:
    • In the main part of the script (if __name__ == "__main__":), specify the URL of the WordPress site you want to check, and call the defined function.
9. Print Result:
    • Print the result, which could be the detected WordPress version or an error message.
```

The issue with the method we discovered is that it depends on other factors, such the website's uptime and internet stability. Even if we had to disregard that and continue, there was little to no consistency in the output. Therefore, the HTTP / HTTPS Response method performs fairly well for these details. The grep command could provide the specifics.

The grep command [47] can also be used to find details about other CMS (Content Management Systems), such as Drupal and Django. After wget, locate a folder that might contain the CMS information for the aforementioned onion file, then run grep on it:

```
grep –iRn home/test
```

Here, home/test is the folder which contains the wget details. The information this gives does come cluttered, but it is still a reliable method.

b) Web Server Details

Finding details about web servers is usually straightforward, but the wget option might not work in this scenario. Instead, the HTTP / HTTPS response method is needed to retrieve information about the web server. Common web servers include Apache, nginx, and others.

c) HTTP Statistics

Navigating the intricacies of the Tor network is necessary to retrieve HTTP statistics from onion links [48]. In order to access these URLs, you must have a browser that supports Tor, such as the Tor Browser, which ensures anonymity through a network of encrypted connections.

Because Tor traffic is encrypted and anonymized, traditional methods for HTTP statistics may not be as effective. Within the Tor network, there is a service called Tor Metrics that provides statistics on the network. Tools such as OnionScan examine privacy and security issues and provide information on web servers and any security holes.

Information about the web server software may be obtained by examining HTTP headers and server fingerprints. Statistics such as request counts and unique visitors can be obtained by gaining access to web server logs.

However, due to the privacy-centric design of Tor, obtaining granular HTTP statistics may pose challenges. Respecting privacy expectations and ethical considerations is paramount when dealing with hidden services on the Tor network.

The algorithm of how this works is as below:

Algorithm 6: Retrieving HTTP / HTTPS Statistics

```
1. Input:
    • Accept the paths to the input file (input_file_path) containing a list of onion URLs and the output file (output_file_path) to store the HTTPS statistics.
2. Define Function for HTTPS Statistics:
    • Create a function get_https_statistics that takes an onion URL as input.
    • Use the torsocks command with wget to make a quiet connection to the onion URL via Tor.
    • Capture the output and error streams from wget.
```

```
        • Check if the wget process was successful (return code 0).
        • Count the occurrences of 'https://' in the HTML content obtained from the onion URL.
        • Return the HTTPS count.
3.  Process Onion URLs:
        • Create a function process_onion_urls that takes the input and output file paths.
        • Read the onion URLs from the input file into a list.
        • Initialize an empty list (https_statistics) to store results.
4.  Retrieve HTTPS Statistics:
        • Iterate through each onion URL in the list.
        • Call the get_https_statistics function for each onion URL.
        • If HTTPS statistics are obtained, append a formatted string to the https_statistics list.
        • Print an error message if no output is obtained for an onion URL.
5.  Write Results to Output File:
        • Open the output file in write mode.
        • Write the HTTPS statistics to the output file, with each line containing the onion URL and its corresponding HTTPS count.
6.  Execution:
        • In the main part of the script (if __name__ == "__main__":), specify the paths for the input and output files.
        • Call the process_onion_urls function with the input and output file paths.
```

d) *Email address and Phone Number*

To minimize code overhead and maximize efficiency during the email address and password retrieval procedure, a Bash file has been optimized for execution. Overall, the email address recovery has been successful, most likely because the Bash file contains well-written scripts or instructions. In a similar vein, extracting passwords seems to be a part of the job and is likely made easier by customized instructions.

But there are problems when you try to get phone numbers back. Phone number extraction is a more complex operation, even though it works well with emails. This challenge may be caused by discrepancies in phone number storage practices, variations in data structures, and even missing or obscured data.

Using a Bash file instead of a more complicated Python script indicates a preference for command-line operations that are more efficient [49]. Refining the phone number retrieval inside the Bash file is advised to improve the extraction procedure. To do this, a detailed examination of data structures must be done, extraction commands must be modified to correctly match various phone number formats, and regular expressions must be added for accurate pattern matching.

The phone number extraction process can be improved by identifying specific execution issues by incorporating logging and debugging mechanisms into the Bash file. Because the task involves personal details, it is imperative to ensure data integrity and handle sensitive information ethically. Through the resolution of these issues and the improvement of extraction techniques, the Bash file can be better optimized to manage different kinds of data.

The pseudocode to this is as below:

Algorithm 7: Gathering contact details

```
1.  Input:
        • Accept the path to the main folder (folder_path) containing subfolders with potential contact information.
2.  Initialize Output File:
        • Set the output file name (output_file) as "contact_info.txt".
        • Remove the existing output file if it already exists.
3.  Loop Through Subfolders:
        • For each folder in the main folder (folder_path):
                ▪ Check if the item is a directory.
        • If it is a directory:
                • Append folder information to the output file, including the folder name.
                • Open a section for email addresses in the output file.
4.  Find and Store Email Addresses:
        • Use grep to search for unique email addresses within the current folder.
        • Append the found email addresses to the output file or indicate if none were found.
5.  Find and Store Phone Numbers:
        • Open a new section for phone numbers in the output file.
        • Use grep to search for unique phone numbers within the current folder.
        • Append the found phone numbers to the output file or indicate if none were found.
6.  Section Separators:
        • Add separators to distinguish sections in the output file.
7.  Completion Message:
        • Display a message indicating that the results have been saved in the specified output file.
```

VII. Maintaining and working on outputs:

We used these ways to gather different information pieces. An excel sheet was created and all information that relates to the corresponding onion links were pasted. The reason we choose to make an excel sheet is so that we can easily gather and work on the results. This worksheet has been uploaded to our github site [24].

VIII. Results

Below are the results of our finding using these three methods. You would see a stark contrast between the kind of information all of these methods reveal. The problems with these results and methodology have already been discussed above.

A. *Results obtained using Method 1: HTTP / HTTPS Response.*

This methodology did reveal some interesting results. The results were specifically under three different categories. The frequency, and at times, the count will also be provided.

Leaked Clearnet addresses or IP addresses:
Frequency: Extremely low
Count: 36 off 4740 websites (0.76% of all websites)

It was very difficult to retrieve information of this sort. Almost every onion website would conceal their Clearnet addresses or IPs very well. The manner with which they do this is to simply hide the website configuration file, as established above.

Below is a table that provides information on the leaked addresses.

Clearnet website	**Number of onion websites that have leaked this information**	**Server Software**
klecker.debian.org	29	Apache
archive-01.torproject.org	1	Apache
stpetezone.com	1	Apache
tornull.org	1	Apache (v2.4.18) on Ubuntu
172.24.0.2	1	Apache (v2.4.38) on Debian
web-fsn-01.torproject.org	1	Apache
default.mediaslibres.org	1	Apache
https://fgn.cdn.serverable.com	1	Nginx

Table 1: Clearnet addresses found and their frequencies.

Notice how 35 out of 36 times, it's Apache that leaked this information. But also, these websites are well known ones. Why does this happen, is provided above.

Available ports for communication:
Frequency: Low
The onion sites did at times leak the ports that were open for communication. These were between the ports: 80, 243, 8080. We did not bother counting this information as it would not really be of much use to us.

Identification of the web server:
Frequency: High
Count: 2965 off 4740 websites (62.55% of all websites)
Web server details were leaked at times, and this information is of utmost importance to us, since there were some versions that were known to be vulnerable. Below is a table of the webservers found using this method and their frequencies.

Web servers	**Frequency**	**Percentage***
Nginx	2307	48.67%
Apache	617	13.01%
simpleHTTP/0.6	11	0.23%
Caddy	11	0.23%
Microsoft-IIS	6	0.12%
Miscellaneous	13	0.27%

Table 2: Web Servers found and their frequencies.
**Percentage: Calculated out of the total number of onion sites worked on.*

Nginx is the most common webserver found in our research. Though many times, it does not leak the version that is installed. Apache is second, but still pretty common, and has a similar problem too. The rest have a very low frequency.

Specific versions of the servers were common too. Below is a table that demonstrates this:

Server	**Version**	**Frequency**
Nginx	1.22.1	18
Nginx	1.18.0	114
Nginx	1.16.0	16

Nginx	1.14.1	20
Nginx	1.20.2	40
Nginx	1.25.0	3
Nginx	1.24.0	6
Nginx	1.20.1	8
Nginx	1.21.0	2
Nginx	1.9.2	1
Apache	2.4.38	4
Apache	2.4.54	30
Apache	2.4.55	15
Apache	2.4.56	11
Apache	2.4.46	4
Apache	2.4.18	1
Apache	2.4.41	16
Apache	2.4.57	27
Apache	2.4.37	1
Microsoft-IIS	10.1	2
Microsoft-IIS	10	4
simpleHTTP/0.6	python 3.11.2	11

Table 3: Server Versions found and their frequencies.

Caddy never revealed the version number.

Operating system of the host
Frequency: Moderate
Count: 234 off 4740 websites (4.94% of all websites)

This method did do a decent job in recognizing the host operating systems, though it never mentioned the version. Below is the table of the host operating systems recognized and their frequency:

Host Operating System	**Frequency**	**Percentage***
Debian	91	1.91%
Ubuntu	95	2.00%
Win32	1	0.02%
Win64	23	0.48%
Unix	3	0.06%
Miscellaneous	21	0.44%

Table 4: Host Operating Systems found and their frequencies.
**Percentage: Calculated out of the total number of onion sites worked on.*

As we see, Ubuntu and Debian dominate this list, which also makes sense as these Operating Systems are ideal to host websites such as these.

B. *Results obtained using Method 2. Gather Secure Socket Layer Details*

This method revealed IP addresses in most cases, but all of these IP addresses were bogon addresses. We did not list the IPs and their related information here, but we did provide a sample output in the appendix. [Image 4].

This method was in most cases a failure since the information obtained was not really useful.

C. *Results obtained using Method 3: "Download" the Websites*

This method revealed quite a lot of details that could be very useful. The type of information and their frequencies are as below:

Bitcoin Addresses
Frequency: Moderate
Count: 285 off 4740 websites (6.01% of all websites)

This method did reveal bitcoin addresses in text format. The reason why we highlighted this, is because sometimes, blockchain addresses are stored as a QR code image, and this method does not always find that. It is interesting to note here that certain bitcoin addresses obtained were frequent across websites. Tracing these addresses and gathering more information exceeds the scope of this research.

Email addresses
Frequency: Moderate
Count: 437 off 4740 websites (9.21% of all websites)

This method also helped gather email addresses listed on the sites. Although this does not mean that these were email addresses of the host operator, it did reveal something interesting.

75.51% of these addresses were temporary. Which means that these addresses had expired, and perhaps hosted for limited communication and termination.

2.97% of these addresses were Gmail addresses. These could be addresses of legitimate websites.

Below is a table which shows the frequency of each email domain.

Email domain	**Frequency**	**Percentage***
cock.li	159	3.35%
dwmail.top	21	0.44%
Secmail.pro	8	0.16%

Skiff.com	58	1.22%
Bitemail.net	3	0.06%
Mail2tor.com	4	0.08%
Tuta.com	16	0.33%
Firemail.com	48	1.01%
Gmail	13	0.27%

Table 5: Email domains found and their frequencies.
**Percentage: Calculated out of the total number of onion sites worked on.*

Framework CMS of the website
Frequency: Moderate
Count: 251 off 4740 websites (5.29% of all websites)

This method revealed Framework CMS of numerous websites. The below table shows the common frameworks and their frequency.

CMS	**Frequency**	**Percentage***
Wordpress	198	4.17%
Joomla	4	0.08%
Django	1	0.02%
Drupal	8	0.16%
PHP	18	0.37%
Zeronet	1	0.02%

Table 6: CMS found and their frequencies.
**Percentage: Calculated out of the total number of onion sites worked on.*

Wordpress dominates as the most common CMS for these websites. Do note that not all of these details have versions and plugins revealed. Below is a table that shows specific versions and plugins found, and their frequencies.

CMS	**Versions**	**Frequency**
Wordpress	6.3.2	20
Wordpress	5.8.2	8
Wordpress	6.1.1	18
Wordpress	6.4.1	11
Wordpress	5.5.3	16
Wordpress	3.1.1	1
Wordpress	5.9.3	7
Wordpress	5.8.1	4
Wordpress	6.0.1	6
Wordpress	5.0.9	1
Wordpress	5.4.4	1
Wordpress	4.9.16	1
Wordpress	5.1.1	3
PHP	7.2.34	1
PHP	7.2.31	4
PHP	8.2.4	2
PHP	8.1.12	1

Table 7: CMS Versions found and their frequencies.

CMS	**Plugins**	**Versions**	**Frequency**
Wordpress	Yoast	21.0	4
Wordpress	Yoast	21.5	1
Wordpress	Yoast	14.5	1
Wordpress	Yoast	21.4	1
Wordpress	Yoast	14.7	11
Wordpress	Yoast	13.1	1
Wordpress	Yoast	14.9	4
Wordpress	Yoast	13.1	5
Wordpress	Yoast	v15	3
Wordpress	Rank Math SEO	-	16

Table 8: CMS Plugins and versions found and their frequencies.

One thing to notice here would be, it's only a specific version of wordpress (Yoast) that leaks the version numbers.

HTTP / HTTPS Statistics:
Frequency: High
Count: 3379 off 4740 websites (71.29% off all websites)

The two most important protocols for internet communication are HTTP (Hypertext Transfer Protocol) and HTTPS (Hypertext Transfer Protocol Secure). While HTTPS adds an extra layer of encryption for security, HTTP allows web browsers and servers to exchange data more easily. Adoption rates, encryption standards, security events, user preferences, online performance, browser support, and worldwide trends are all covered by statistics on HTTP/HTTPS. Decisions about user experience, security, and web development are informed by these findings.

Below is the table of certain specific statistics and their frequency:

HTTP / HTTPS Statistics	**Frequency**	**Percentage***
200	3000	63.29%
404	4	0.08%
301	9	0.19%
302	13	0.27%
403	353	7.44%

Table 9: HTTP / HTTPS Statistics found and their frequencies.
**Percentage: Calculated out of the total number of onion sites worked on.*

The meanings of these statistics are provided in the appendix [Information Table 2].

Private / Public Keys:
Frequency: Insignificantly low
Count: 1 off 4740 websites (0.021% of all websites)

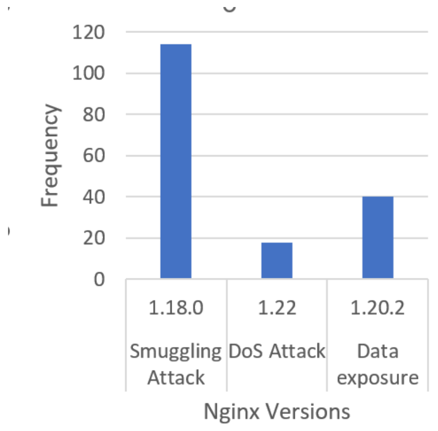
It was really interesting to know that we found one PGP Public key file on one of the websites. This could be a file that aids communication between systems [57].

A picture of the PGP key is provided in the appendix [Image 7]

D. *Correlation between Vulnerabilities and information obtained*

Information Table 1 in appendix describes all related vulnerabilities that would be helpful for this section. We have interlinked that table and the frequency tables above to come up with a set of graphs that give an idea about the vulnerabilities and their frequencies.

Graph 1 was obtained through correlation between Information Table 1 of appendix and (Frequency) Table 3 above (Nginx Versions only).



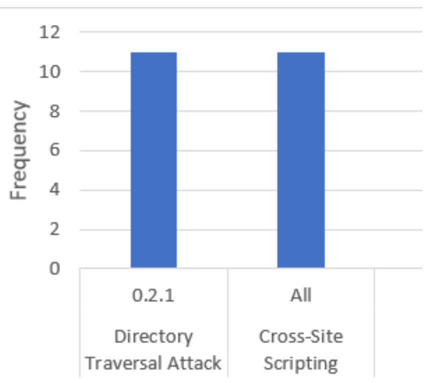
Graph 1: Nginx versions and corresponding Vulnerabilities.

Vulnerabilities in 118 cases of websites running particular Nginx web servers were found to have the potential to be used for HTTP Request Smuggling attacks [50]. This kind of attack includes tricking a web server into processing HTTP requests in a way that could compromise data or grant unauthorised access, among other security risks.

It was discovered that 18 websites running particular Nginx versions had security holes that could be exploited by Denial-of-Service (DoS) attacks [51]. DoS vulnerabilities in Nginx versions could be used to interfere with the web server's regular operation and make it unavailable to users for a short period of time or permanently.

There were found to be vulnerabilities in 40 cases of websites using particular Nginx versions that might expose data. Vulnerabilities in data exposure [52] could result in sensitive information being exposed or accessed without authorization, which would be a serious security risk.

Graph 2 was obtained through correlation between Information Table 1 of appendix and (Frequency) Table 3 above (SimpleHTTP Versions only).

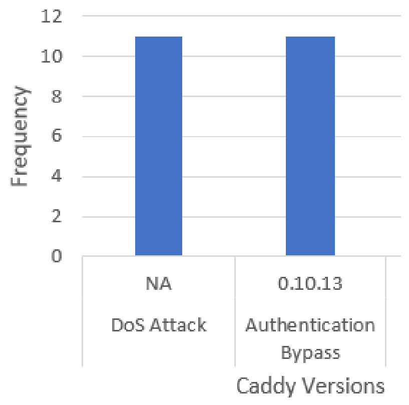


Graph 2: SimpleHTTP Versions and corresponding Vulnerabilities

Vulnerabilities that might be used for Directory Traversal attacks [53] were found in 11 cases of websites using particular versions of simpleHTTP. Vulnerabilities related to directory traversal give attackers the ability to access files and directories that are not intended, which may result in sensitive information being exposed or unauthorised access.

Similarly, it was discovered that 11 websites that used particular versions of simpleHTTP had security holes that made them vulnerable to Cross-Site Scripting (XSS) attacks [54]. SimpleHTTP versions may have XSS vulnerabilities that put websites at risk of malicious activities, unauthorised script injections, and compromised user data.

Graph 3 was obtained through correlation between Information Table 1 of appendix and Caddy Version values.

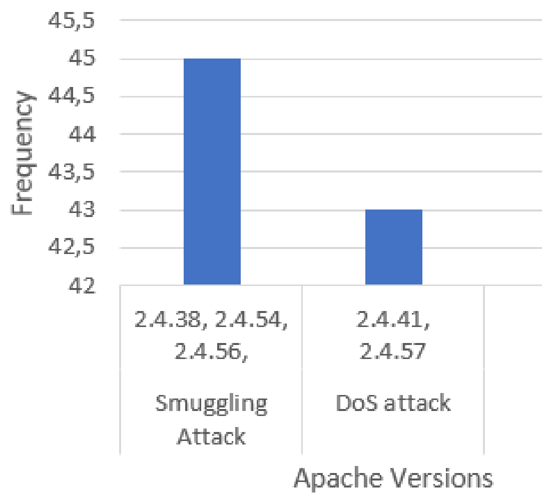


Graph 3: Caddy Versions and corresponding Vulnerabilities

Vulnerabilities in 11 cases of websites using particular Caddy web server versions were found that might be used to launch Denial-of-Service (DoS) attacks. DoS vulnerabilities in Caddy versions could be used to interfere with the web server's regular operations and make it unavailable to users for a short period of time or permanently.

Similarly, it was discovered that 11 websites that used particular Caddy web server versions had security holes that could be exploited by an Authentication Bypass attack [55]. Vulnerabilities in authentication bypass could pose a serious security risk by enabling unauthorised users to access resources that are restricted or take actions without the required authentication.

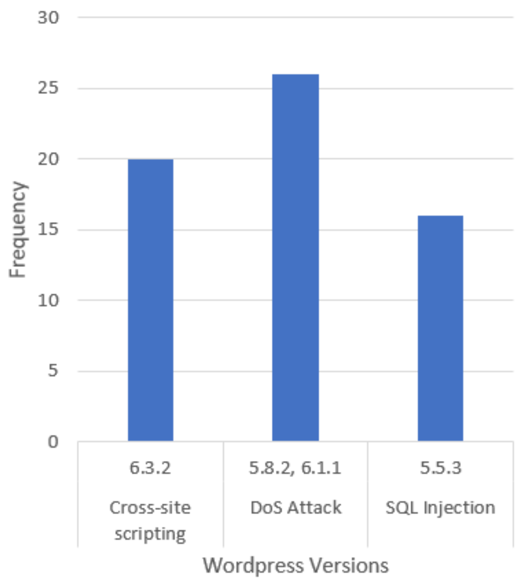
Graph 4 was obtained through correlation between Information Table 1 of appendix and (Frequency) Table 3 above (Apache Versions only).



Graph 4: Apache Versions and corresponding Vulnerabilities

We found vulnerabilities in 45 cases of websites using particular versions of the Apache web server that might be used for HTTP Request Smuggling attacks. This kind of attack includes tricking a web server into processing HTTP requests in a way that could jeopardize data or grant unauthorised access, among other security hazards. It was discovered that 43 websites that used particular versions of the Apache web server had security holes that might be exploited by DoS attacks. DoS vulnerabilities in Apache versions could be used to interfere with the web server's regular operation and make it unavailable to users for a short period of time or permanently.

Graph 5 was obtained through correlation between Information Table 1 of appendix and (Frequency) Table 7 above (Wordpress Versions only).



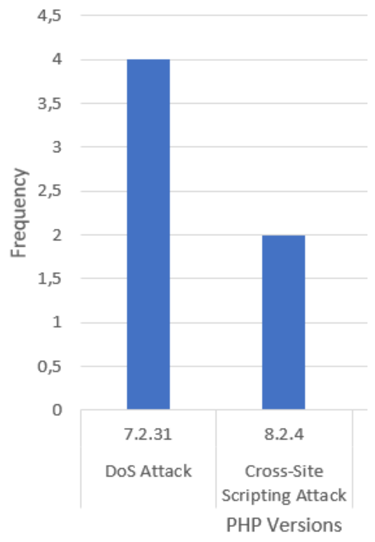
Graph 5: Wordpress Versions and corresponding Vulnerabilities

20 WordPress version instances were found to be vulnerable to cross-site scripting (XSS) attacks. One kind of security flaw called cross-site scripting (XSS) lets hackers insert dangerous scripts into websites that other users are seeing. These vulnerabilities have the potential to be used against WordPress in order to compromise user data, session information, or start new attacks.

It was discovered that 26 websites, particularly those that used WordPress, had security holes that might be used to launch Denial-of-Service (DoS) attacks. DoS attacks seek to interfere with a website's or online service's regular operation, rendering it unavailable to users for a short period of time or never. Ensuring the accessibility and dependability of the impacted websites depends on identifying and fixing these vulnerabilities.

There were 16 WordPress versions found to have vulnerabilities open to SQL injection attacks [56]. By inserting malicious SQL code, an attacker can influence a website's database through SQL injection attacks. This can result in sensitive data stored in the database being accessed, retrieved, or altered without authorization. Resolving SQL injection vulnerabilities is essential to stopping illegal access and preserving the accuracy of the data on the website.

Graph 6 was obtained through correlation between Information Table 1 of appendix and (Frequency) Table 7 above (PHP Versions only).

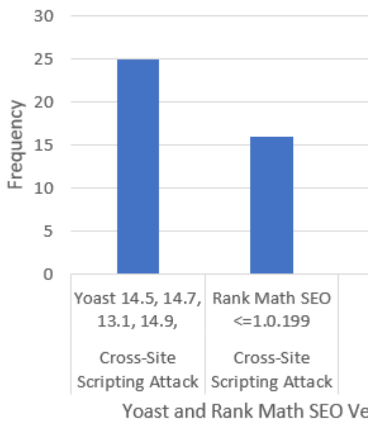


Graph 6: PHP Versions and corresponding Vulnerabilities.

We found vulnerabilities in two cases of websites using particular versions of PHP that might be used as a springboard for Cross-Site Scripting (XSS) attacks. PHP versions that contain XSS vulnerabilities run the danger of allowing malicious operations to take place, compromising user data, or allowing unauthorized script injections on websites.

It was discovered that four examples of websites using particular versions of PHP had security holes that might be exploited by DoS attacks. DoS vulnerabilities in PHP versions could be used to interfere with a website or online service's regular operation and make it unavailable to users for a short period of time or permanently.

Graph 7 was obtained through correlation between Information Table 1 of appendix and (Frequency) Table 8 above.



Graph 7: Yoast and Rank Math SEO and corresponding Vulnerabilities.

Vulnerabilities that might be used for Cross-Site Scripting (XSS) attacks were found in 25 cases of websites running the WordPress Yoast plugin. XSS vulnerabilities in plugins put websites at risk of malicious scripts being injected and executed by attackers, compromising user data or leading to other attacks.

It was discovered that 16 websites utilizing the Rank Math SEO plugin had security holes that may be exploited by Cross-Site Scripting (XSS) attacks. These incidents may put websites at risk of unwanted script injections and possible exploitation by malevolent actors, much like the Yoast plugin vulnerabilities.

The table below gives a full summary of the types of vulnerabilities possible, their frequencies and the significant percentage of websites that had this vulnerability.

Common Vulnerabilities	**Frequency**	**Percentage***
Smuggling Attacks	163	3.43%
Cross-Site Scripting Attack	74	1.56%
DoS Attack	102	2.15%
Data Exposure	40	0.84%
Directory Traversal Attack	11	0.23%
SQL Injection	16	0.34%
Authentication Bypass	11	0.23%

Table 10: Mapping of possible vulnerabilities to the tables mentioned above.
**Percentage: Calculated out of the total number of onion sites worked on.*

The most common type of vulnerability, with 163 occurrences and 3.43% of all websites, is smuggling attack. Attacks using Cross-Site Scripting (XSS) vulnerability come next, occurring 74 times (1.56%), and attacks using Denial of Service (DoS) vulnerability were found in 102 instances (2.15%). Of the cases, directory traversal attack vulnerabilities are relatively less common, found in 11 instances (0.23%) while data exposure vulnerability is found in 40 instances (0.84%). There are 16 instances of SQL injection and 11 instances of authentication bypass on all websites, or 0.34% and 0.23%, respectively.

To give a brief about Smuggling attack and its significance here, Smuggling attack is essentially when the malicious attacker is able to convince the traffic analysis system at the server admin end of the site being accessed, that it's malicious / manipulated packet is legitimate and should be addressed. Although the method with which this is done is not clear, once it is, this can determine the ease with which the attack can be done on these onion servers.

IX. Conclusion

The project's goal was to find possible dark web vulnerabilities and exposed onion addresses. The important findings are summarized below based on the results:

Out of all the domains analyzed, or 0.75%, 36 onion websites were found to have compromised IP addresses or Clearnet addresses. Remarkably, 35 of these leaks were linked to Apache servers, indicating a strong correlation between Apache servers and the unintentional disclosure of IP addresses or Clearnet addresses.

Rate of Incidence: Of all the websites the research looked into, the 36 leaks that were found to be present account for 0.75%. This percentage sheds light on how common leaks are on the onion websites that were sampled.

Server-Specific Leaks: There was a strong association found; 35 of the 36 leaks were from websites running on Apache servers. This demonstrates how Apache servers contribute to the inadvertent release of IP addresses or Clearnet addresses .

Risk Assessment for Apache Servers: Based on analysis, there is a 0.7383% risk that IP addresses or Clearnet addresses will be leaked from Apache servers. This statistical finding highlights how crucial it is to configure Apache servers correctly in order to reduce the possibility of disclosing private or sensitive data.

Proposed solutions here are twofold:

1. **Switch to a safer solution**, such as Nginx. You would not then have to bother as much about configuration files [58].
2. Write your configuration files extremely meticulously. This way, there will be no chance of Clearnet address leaks that would lead to the deanonymization of the server host.

A lot of the websites were found to have a configuration that would lead to a possible attack. 3.43% of all websites worked on had the risk of smuggling attacks, 2.15% of them had DoS attack risks and 1.56% of them had Cross-Site Scripting risks.

The solution here would be simply to upgrade the version of the servers being used [59].

Other solutions would be to not save personal email addresses on the website folders [60], do not save private keys as well as these can be downloaded using Wget.

With this, the goal of this project was met, and our research was able to provide concrete information in numbers for more clarity of the vulnerabilities and their frequencies.

Acknowledgment

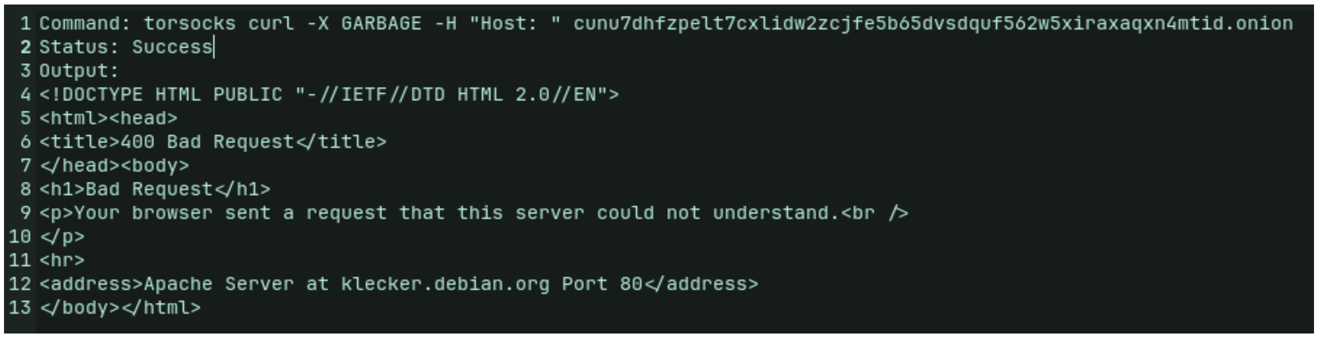
Our profound thanks and appreciation go out to Mr. Ivan Pustagarov, our project guide and professor, for his important advice and steadfast support during our research project. His knowledge, incisive criticism, and dedication to academic achievement have greatly influenced this endeavor's success.

Additionally, We are appreciative of our institute, Concordia University, Montreal, Canada, which offers a stimulating academic environment that promotes research and learning. The university's facilities and resources have been invaluable in helping us finish our project.

In addition, we would like to express our gratitude to the Department of Computer Science and Software Engineering (CIISE), particularly to the Information Systems Security department, for offering a platform that promotes research and development in the sector.

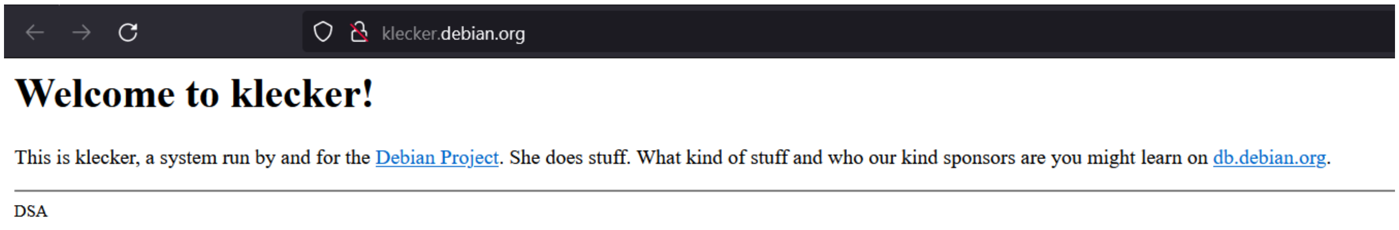
We have learned a great deal from this study endeavor, and we are grateful for the help of these kind people and organizations. Their assistance has been essential to the project's successful conclusion.

APPENDIX



```
Command: torsocks curl -X GARBAGE -H "Host: " cunu7dhfzpelt7cxlidw2zcjfe5b65dvsdquf562w5xiraxaqxn4mtid.onion
Status: Success
Output:
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>400 Bad Request</title>
</head><body>
<h1>Bad Request</h1>
<p>Your browser sent a request that this server could not understand.<br />
</p>
<hr>
<address>Apache Server at klecker.debian.org Port 80</address>
</body></html>
```

Image 1: Sample output retrieved through HTTP / HTTPS Response. Notice how this leaks the Clearnet address of the onion website.



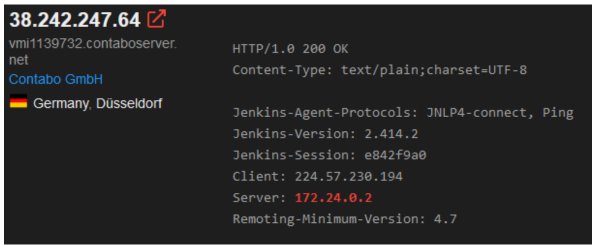
klecker.debian.org

Welcome to klecker!

This is klecker, a system run by and for the Debian Project. She does stuff. What kind of stuff and who our kind sponsors are you might learn on db.debian.org.

DSA

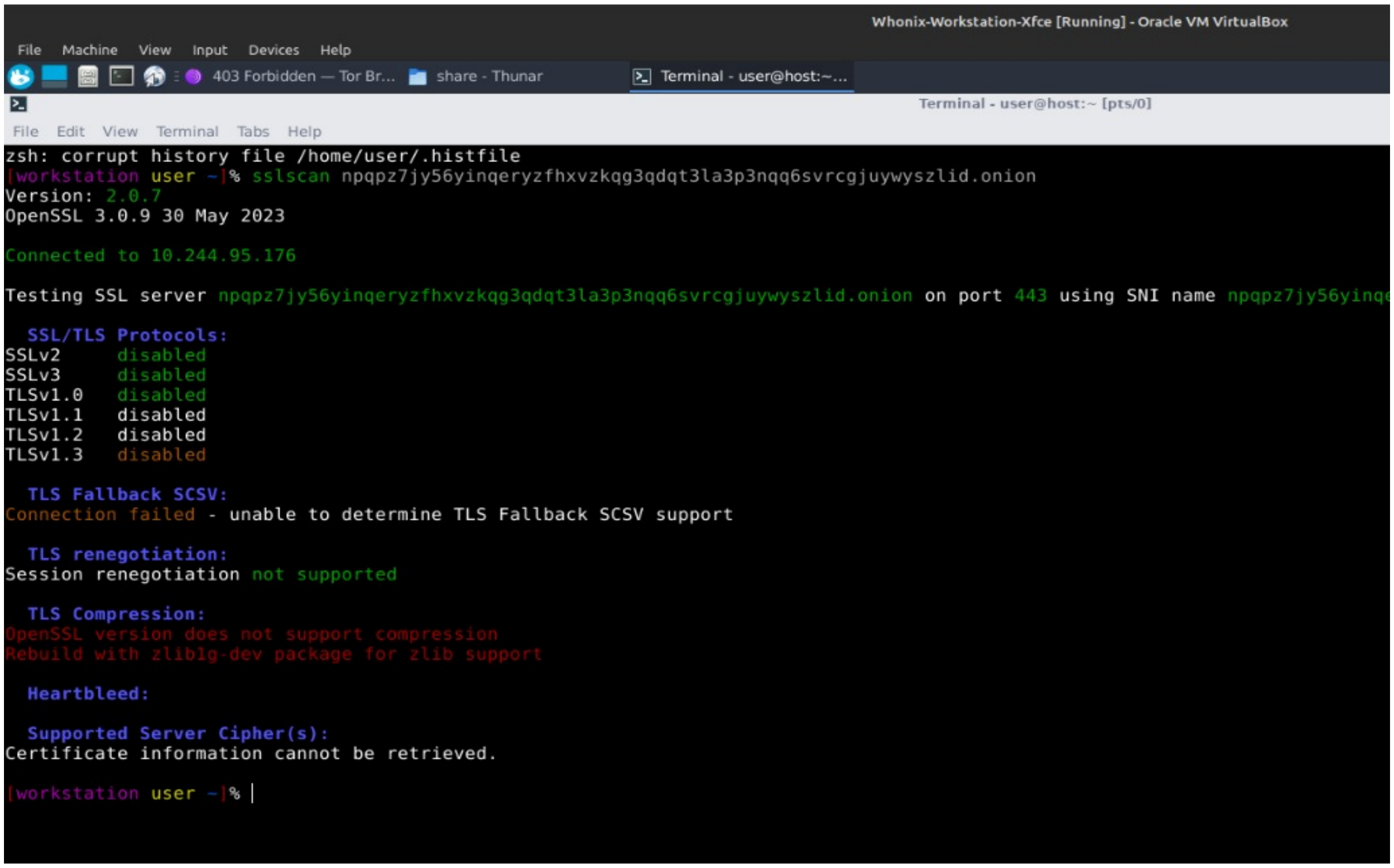
Image 2: Image of the Clearnet version of the leaked website klecker.debian.org. Clearly, this information simply appears to be a product ad.



38.242.247.64
vmi1139732.contaboserver.net
Contabo GmbH
Germany, Düsseldorf

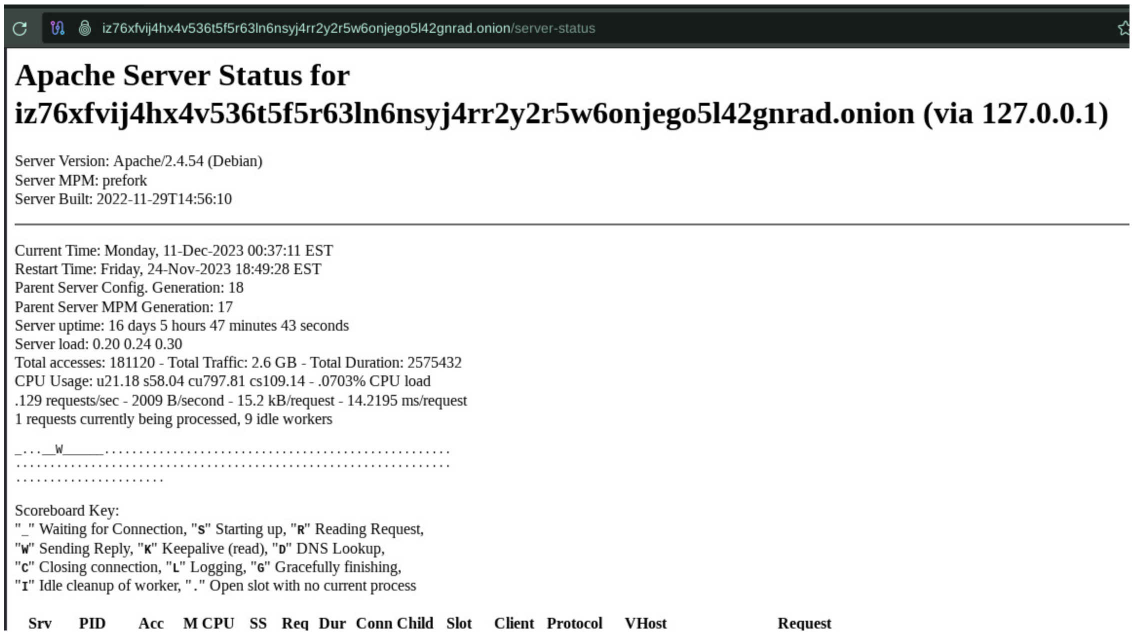
```
HTTP/1.0 200 OK
Content-Type: text/plain;charset=UTF-8

Jenkins-Agent-Protocols: JNLP4-connect, Ping
Jenkins-Version: 2.414.2
Jenkins-Session: e842f9a0
Client: 224.57.230.194
Server: 172.24.0.2
Remoting-Minimum-Version: 4.7
```

Image 3: Information from Shodan website on a leaked IP address. However, this is not the only information received, and in most cases, this IP address is a server, which hints at a possibility that this could be a VPN service. [30]


Whonix-Workstation-Xfce [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
403 Forbidden — Tor Br...
share - Thunar
Terminal - user@host:~...
Terminal - user@host:~ [pts/0]
File Edit View Terminal Tabs Help

```
zsh: corrupt history file /home/user/.histfile
[workstation user ~]% sslscan npqpz7jy56yinqeryzfhxvzkqg3qdqt3la3p3nqq6svrcgjuywyszlid.onion
Version: 2.0.7
OpenSSL 3.0.9 30 May 2023

Connected to 10.244.95.176

Testing SSL server npqpz7jy56yinqeryzfhxvzkqg3qdqt3la3p3nqq6svrcgjuywyszlid.onion on port 443 using SNI name npqpz7jy56yinqe

  SSL/TLS Protocols:
SSLv2     disabled
SSLv3     disabled
TLSv1.0   disabled
TLSv1.1   disabled
TLSv1.2   disabled
TLSv1.3   disabled

  TLS Fallback SCSV:
Connection failed - unable to determine TLS Fallback SCSV support

  TLS renegotiation:
Session renegotiation not supported

  TLS Compression:
OpenSSL version does not support compression
Rebuild with zlib1g-dev package for zlib support

  Heartbleed:

  Supported Server Cipher(s):
Certificate information cannot be retrieved.

[workstation user ~]% |
```

Image 4: Output of method 2: Gathering SSL details. Notice that the IP: 10.244.95.176 was leaked, but it is a bogon IP address


iz76xfvij4hx4v536t5f5r63ln6nsyj4rr2y2r5w6onjego5l42gnrad.onion/server-status

Apache Server Status for iz76xfvij4hx4v536t5f5r63ln6nsyj4rr2y2r5w6onjego5l42gnrad.onion (via 127.0.0.1)

Server Version: Apache/2.4.54 (Debian)
Server MPM: prefork
Server Built: 2022-11-29T14:56:10

Current Time: Monday, 11-Dec-2023 00:37:11 EST
Restart Time: Friday, 24-Nov-2023 18:49:28 EST
Parent Server Config. Generation: 18
Parent Server MPM Generation: 17
Server uptime: 16 days 5 hours 47 minutes 43 seconds
Server load: 0.20 0.24 0.30
Total accesses: 181120 - Total Traffic: 2.6 GB - Total Duration: 2575432
CPU Usage: u21.18 s58.04 cu797.81 cs109.14 - .0703% CPU load
.129 requests/sec - 2009 B/second - 15.2 kB/request - 14.2195 ms/request
1 requests currently being processed, 9 idle workers

```
_...__W______...................................................
................................................................
......................
```

Scoreboard Key:
"_" Waiting for Connection, "S" Starting up, "R" Reading Request,
"W" Sending Reply, "K" Keepalive (read), "D" DNS Lookup,
"C" Closing connection, "L" Logging, "G" Gracefully finishing,
"I" Idle cleanup of worker, "." Open slot with no current process

Srv	PID	Acc	M	CPU	SS	Req	Dur	Conn	Child	Slot	Client	Protocol	VHost	Request

Image 5: Sample of a server status page for a certain onion link

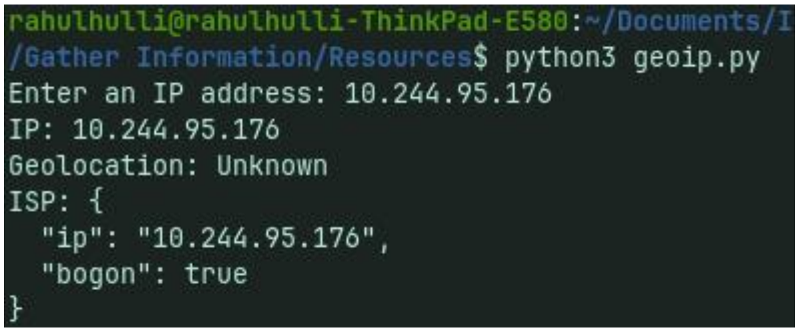
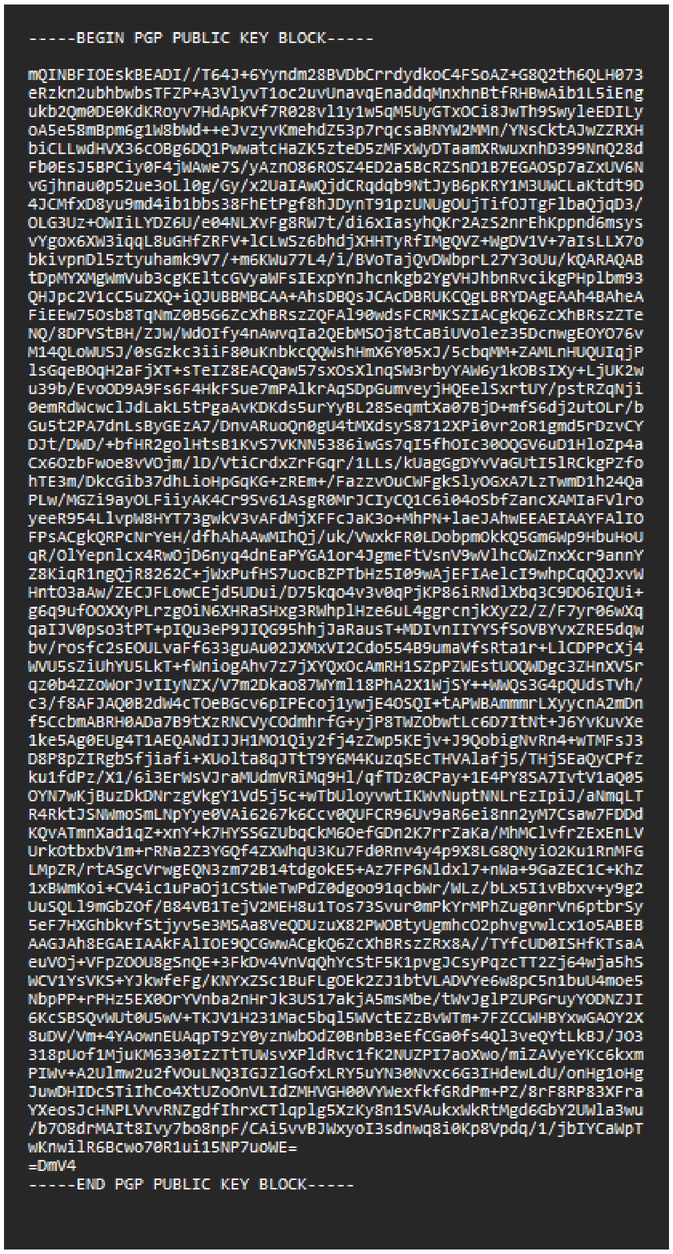

```
rahulhulli@rahulhulli-ThinkPad-E580:~/Documents/I
/Gather Information/Resources$ python3 geoip.py
Enter an IP address: 10.244.95.176
IP: 10.244.95.176
Geolocation: Unknown
ISP: {
  "ip": "10.244.95.176",
  "bogon": true
}
```

Image 6: Output under method 2, using geoip2 method



```
-----BEGIN PGP PUBLIC KEY BLOCK-----

mQINBFIOEskBEADI//T64J+6Yyndm28BVDbCrrdydkoC4FSoAZ+G8Q2th6QLH073
eRzkn2ubhbwbsTFZP+A3VlyvT1oc2uvUnavqEnaddqMnxhnBtfRHBwAib1L5iEng
ukb2Qm0DE0KdKRoyv7HdApKVf7R028vl1y1w5qM5UyGTxOCi8JwTh9SwyleEDILy
oA5e58mBpm6g1W8bWd++eJvzyvKmehdZ53p7rqcsaBNYW2MMn/YNsCktAJwZZRXH
biCLLwdHVX36cOBg6DQ1PwwatcHaZK5zteD5zMFxWyDTaamXRwuxnhD399NnQ28d
Fb0EsJ5BPCiy0F4jWAwe7S/yAznO86ROSZ4ED2a5BcRZSnD1B7EGAOSp7aZxUV6N
vGjhnau0p52ue3oLl0g/Gy/x2UaIAwQjdCRqdqb9NtJyB6pKRY1M3UWCLaKtdt9D
4JCMfxD8yu9md4ib1bbs38FhEtPgf8hJDynT91pzUNUgOUjTifOJTgFlbaQjqD3/
OLG3Uz+OWIiLYDZ6U/e04NLXvFg8RW7t/di6xIasyhQKr2AzS2nrEhKppnd6msys
vYgox6XW3iqqL8uGHfZRFV+lCLwSz6bhdjXHHTyRfIMgQVZ+WgDV1V+7aIsLLX7o
bkivpnDl5ztyuhamk9V7/+m6KWu77L4/i/BVoTajQvDWbprL27Y3oUu/kQARAQAB
tDpMYXMgWmVub3cgKEltcGVyaWFsIExpYnJhcnkgb2YgVHJhbnRvcikgPHplbm93
QHJpc2V1cC5uZXQ+iQJUBBMBCAA+AhsDBQsJCAcDBRUKCQgLBRYDAgEAAh4BAheA
FiEEw75Osb8TqNmZ0B5G6ZcXhBRszZQFAl90wdsFCRMKSZIACgkQ6ZcXhBRszZTe
NQ/8DPVStBH/ZJW/WdOIfy4nAwvqIa2QEbMSOj8tCaBiUVolez35DcnwgEOYO76v
M14QLoWUSJ/0sGzkc3iiF80uKnbkcQQWshHmX6Y05xJ/5cbqMM+ZAMLnHUQUIqjP
lsGqeBOqH2aFjXT+sTeIZ8EACQaw57sxOsXlnqSW3rbyYAW6y1kOBsIXy+LjUK2w
u39b/EvoOD9A9Fs6F4HkFSue7mPAlkrAqSDpGumveyjHQEelSxrtUY/pstRZqNji
0emRdWcwclJdLakL5tPgaAvKDKds5urYyBL28SeqmtXa07BjD+mfS6dj2utOLr/b
Gu5t2PA7dnLsByGEzA7/DnvARuoQn0gU4tMXdsyS8712XPi0vr2oR1gmd5rDzvCY
DJt/DWD/+bfHR2golHtsB1KvS7VKNN5386iwGs7qI5fhOIc30OQGV6uD1HloZp4a
Cx6OzbFwoe8vVOjm/lD/VtiCrdxZrFGqr/1LLs/kUagGgDYvVaGUtI5lRCkgPZfo
hTE3m/DkcGib37dhLioHpGqKG+zREm+/FazzvOuCWFgkSlyOGxA7LzTwmD1h24Qa
PLw/MGZi9ayOLFiiyAK4Cr9Sv61AsgR0MrJCIyCQ1C6i04oSbfZancXAMIaFVlro
yeeR954LlvpW8HYT73gwkV3vAFdMjXFFcJaK3o+MhPN+laeJAhwEEAEIAAYFAlIO
FPsACgkQRPcNrYeH/dfhAhAAwMIhQj/uk/VwxkFR0LDobpmOkkQ5Gm6Wp9HbuHoU
qR/OlYepnlcx4RwOjD6nyq4dnEaPYGA1or4JgmeFtVsnV9wVlhcOWZnxXcr9annY
Z8KiqR1ngQjR8262C+jWxPufHS7uocBZPTbHz5I09wAjEFIAelcI9whpCqQQJxvW
HntO3aAw/ZECJFLowCEjd5UDui/D75kqo4v3v0qPjKP86iRNdlXbq3C9DO6IQUi+
g6q9ufOOXXyPLrzgOiN6XHRaSHxg3RWhplHze6uL4ggrcnjkXyZ2/Z/F7yr06wXq
qaIJV0pso3tPT+pIQu3eP9JIQG95hhjJaRausT+MDIvnIIYYSfSoVBYvxZRE5dqw
bv/rosfc2sEOULvaFf633guAu02JXMxVI2Cdo554B9umaVfsRta1r+LlCDPPcXj4
WVU5sZiUhYU5LkT+fWniogAhv7z7jXYQxOcAmRH1SZpPZWEstUOQWDgc3ZHnXVSr
qz0b4ZZoWorJvIIyNZX/V7m2Dkao87WYml18PhA2X1WjSY++WWQs3G4pQUdsTVh/
c3/f8AFJAQ0B2dW4cTOeBGcv6pIPEcoj1ywjE4OSQI+tAPWBAmmmrLXyycnA2mDn
f5CcbmABRH0ADa7B9tXzRNCVyCOdmhrfG+yjP8TWZObwtLc6D7ItNt+J6YvKuvXe
1ke5Ag0EUg4T1AEQANdIJJH1MO1Qiy2fj4zZwp5KEjv+J9QobigNvRn4+wTMFsJ3
D8P8pZIRgbSfjiafi+XUolta8qJTtT9Y6M4KuzqSEcTHVAlafj5/THjSEaQyCPfz
ku1fdPz/X1/6i3ErWsVJraMUdmVRiMq9Hl/qfTDz0CPay+1E4PY8SA7IvtV1aQ05
OYN7wKjBuzDkDNrzgVkgY1Vd5j5c+wTbUloyvwtIKWvNuptNNLrEzIpiJ/aNmqLT
R4RktJSNWmoSmLNpYye0VAi6267k6Ccv0QUFCR96Uv9aR6ei8nn2yM7Csaw7FDDd
KQvATmnXad1qZ+xnY+k7HYSSGZUbqCkM6OefGDn2K7rrZaKa/MhMClvfrZExEnLV
UrkOtbxbV1m+rRNa2Z3YGQf4ZXWhqU3Ku7Fd0Rnv4y4p9X8LG8QNyiO2Ku1RnMFG
LMpZR/rtASgcVrwgEQN3zm72B14tdgokE5+Az7FP6Nldxl7+nWa+9GaZEC1C+KhZ
1xBWmKoi+CV4ic1uPaOj1CStWeTwPdZ0dgoo91qcbWr/WLz/bLx5I1vBbxv+y9g2
UuSQLl9mGbZOf/B84VB1TejV2MEH8u1Tos73Svur0mPkYrMPhZug0nrVn6ptbrSy
5eF7HXGhbkvfStjyv5e3MSAa8VeQDUzuX82PWOBtyUgmhcO2phvgvwlcx1o5ABEB
AAGJAh8EGAEIAAkFAlIOE9QCGwwACgkQ6ZcXhBRszZRx8A//TYfcUD0ISHfKTsaA
euVOj+VFpZOOU8gSnQE+3FkDv4VnVqQhYcStF5K1pvgJCsyPqzcTT2Zj64wja5hS
WCV1YsVKS+YJkwfeFg/KNYxZSc1BuFLgOEk2ZJ1btVLADVYe6w8pC5n1buU4moe5
NbpPP+rPHz5EX0OrYVnba2nHrJk3US17akjA5msMbe/tWvJglPZUPGruyYODNZJI
6KcSBSQvWUt0U5wV+TKJV1H231Mac5bql5WVctEZzBvWTm+7FZCCWHBYxwGAOY2X
8uDV/Vm+4YAownEUAqpT9zY0yznWbOdZ0BnbB3eEfCGa0fs4Ql3veQYtLkBJ/JO3
318pUof1MjuKM6330IzZTtTUWsvXPldRvc1fK2NUZPI7aoXwo/miZAVyeYKc6kxm
PIWv+A2Ulmw2u2fVOuLNQ3IGJZlGofxLRY5uYN30Nvxc6G3IHdewLdU/onHg1oHg
JuwDHIDcSTiIhCo4XtUZoOnVLIdZMHVGH00VYWexfkfGRdPm+PZ/8rF8RP83XFra
YXeosJcHNPLVvvRNZgdfIhrxCTlqplg5XzKy8n1SVAukxWkRtMgd6GbY2UWla3wu
/b7O8drMAIt8Ivy7bo8npF/CAi5vvBJWxyoI3sdnwq8i0Kp8Vpdq/1/jbIYCaWpT
wKnwilR6Bcwo70R1ui15NP7uoWE=
=DmV4
-----END PGP PUBLIC KEY BLOCK-----
```

Image 7: PGP Public Key sample

Information Table 1: Set of vulnerabilities and affected Server versions.

CVE number	Type of Vulnerability	Details	Server Affected	Specific version
CVE-2018-16478	Directory Traversal Attack	A Path Traversal in simplehttpserver versions <=0.2.1 allows to list any file in another folder of web root.	simpleHTTP /0.6	<=0.2.1
CVE-2018-3716	Cross-Site Scripting	simplehttpserver node module suffers from a Cross-Site Scripting vulnerability to a lack of validation of file names.	simpleHTTP /0.6	All
CVE-2023-44487	DoS Attack	The HTTP/2 protocol allows a denial of service (server resource consumption) because request cancellation can reset many streams quickly, as exploited in the wild in August through October 2023.	Caddy	NA
CVE-2018-21246	Authentication Bypass	Caddy before 0.10.13 mishandles TLS client authentication, as demonstrated by an authentication bypass caused by the lack of the StrictHostMatching mode.	Caddy	<=0.10.13
CVE-2022-34037	DoS Attack	An out-of-bounds read in the rewrite function at /modules/caddyhttp/rewrite/rewrite.go in Caddy v2.5.1 allows attackers to cause a Denial of Service (DoS) via a crafted URI.	Caddy	2.5.1
CVE-2013-0337	DoS Attack	nginx 1.22 debian-bullseye CVE-2013-0337 nginx 1.22.0-1~bullseye The default configuration of nginx, possibly 1.3.13 and earlier, uses world-readable permissions for the (1) access.log and (2) error.log files, which allows local users to obtain sensitive information by reading the files.	Nginx	1.22
CVE-2020-12440	Smuggling Attack	Nginx allows an HTTP request smuggling attack that can lead to cache poisoning, credential hijacking, or security bypass.	Nginx	1.18.0
Not specified Link [31]	Data exposure	When curl < 7.84.0 saves cookies, alt-svc and hsts data to local files, it makes the operation atomic by finalizing the operation with a rename from a temporary name to the final target file name. In that rename operation, it might accidentally widen the permissions for the target file, leaving the updated file accessible to more users than intended.	Nginx alpine-perl	1.20.2
CVE-2023-27522	Smuggling Attack	HTTP Response Smuggling vulnerability in Apache HTTP Server via mod_proxy_uwsgi. This issue affects Apache HTTP Server: from 2.4.30 through 2.4.55.	Apache	2.4.38, 2.4.54, 2.4.56, 2.4.46
CVE-2019-9517, CVE-2023-43622	DoS attack	A malicious client could perform a DoS attack by flooding a connection with requests and basically never reading responses on the TCP connection. Depending on h2 worker dimensioning, it was possible to block those with relatively few connections.	Apache	2.4.41, 2.4.57

Information Table 2: Set of vulnerabilities and affected CMS versions.

CVE number	**Type of Vulnerability**	**Details**	**CMS Affected**	**Specific version**
Not Specified Link [32]	Cross-site scripting	Cross-site scripting (XSS) vulnerability in the post link navigation block	Wordpress	6.3.2
Not Specified Link [33], [34]	DoS Attack	A Denial of Service could occur via Cache Poisoning when the X-HTTP-Method-Override header is sent in a request to the REST API in a heavily cached configuration	Wordpress	5.8.2, 6.1.1
Not Specified Link [35]	SQL Injection	Due to lack of proper sanitization in WP_Meta_Query, there's potential for blind SQL Injection.	Wordpress	5.5.3
CVE-2019-11048	DoS Attack	According to its self-reported version number, the version of PHP running on the remote web server is 7.2.x prior to 7.2.31, 7.3.x prior to 7.3.18 or 7.4.x prior to 7.4.6. It is, therefore, affected by a denial of service (DoS) vulnerability in its HTTP file upload component due to a failure to clean up temporary files created during the file upload process. An unauthenticated, remote attacker can exploit this issue, by repeatedly submitting uploads with long file or field names, to exhaust disk space and cause a DoS condition.	PHP	7.2.31
Not Specified Link [36]	Cross-Site Scripting Attack	A File Inclusion and Reflected Cross Site Scripting vulnerability was discovered during the testing of Sojobo, Static Analysis Tool.	PHP	8.2.4

Information Table 2: Set of vulnerabilities and affected CMS versions.

CVE number	Type of Vulnerability	Details	CMS Plugin Version	Specific Version
CVE-2023-40680	Cross-Site Scripting Attack	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') vulnerability in Team Yoast Yoast SEO allows Stored XSS.This issue affects Yoast SEO: from n/a through 21.0.	Wordpress Yoast	14.5, 14.7, 13.1, 14.9, 13.1, 15
CVE-2023-32600	Cross-Site Scripting Attack	The Rank Math SEO plugin for WordPress is vulnerable to Stored Cross-Site Scripting via the plugin's shortcode(s) in versions up to, and including, 1.0.119 due to insufficient input sanitization and output escaping on user supplied attributes	Wordpress Rank Math SEO	<=1.0.199

Information Table 2: Meanings of HTTP Statistics values

Values	**Code**	**Description**

200	OK	The server successfully processed the request, and the response contains the requested data or confirmation of the action.
404	Not Found	The requested resource could not be found on the server, indicating that the URL or endpoint is invalid, or the resource has been removed
301	Moved Permanently	The requested URL has been permanently moved to another location, and clients should update their bookmarks or links to the new URL.
302	Moved Temporarily	The requested URL has been temporarily moved to another location, and clients should continue to use the original URL for future requests.
403	Forbidden	The server understood the request, but it refuses to authorize it. This status code is typically returned when the user does not have the necessary permissions for the requested resource.

REFERENCES

[1] Protocol-level attacks against Tor http://www.cs.ucf.edu/~xinwenfu/paper/Journals/14_CN_57_4_Protocol-levelAttacksagainstTor.pdf

[2] De-anonymization attacks on Tor: A Survey https://arxiv.org/pdf/2009.13018.pdf

[3] Timing Attack https://ropesec.com/articles/timing-attacks/

[4] Sybil Attack https://www.imperva.com/learn/application-security/sybil-attack/

[5] Common Protocol Vulnerabilities https://cqr.company/web-vulnerabilities/common-protocol-vulnerabilities/

[6] Tor security advisory: "relay early" traffic confirmation attack https://blog.torproject.org/tor-security-advisory-relay-early-traffic-confirmation-attack/

[7] Mitigating Intersection Attacks in Anonymous Microblogging https://arxiv.org/abs/2307.09069

[8] Attacks on Tor https://github.com/Attacks-on-Tor/Attacks-on-Tor

[9] Tor browser: https://www.torproject.org/

[10] How to use Tor and why should you use it: https://www.zdnet.com/article/how-to-use-tor-browser-and-why-you-should/

[11] Onion Hidden links : https://tb-manual.torproject.org/onion-services/

[12] Deanonymization of Tor links: https://www.bitdefender.com.au/blog/hotforsecurity/de-anonymization-of-tor-hidden-services-with-88-percent-certainty-researchers-say/

[13] The Trouble of Tor: https://blog.cloudflare.com/the-trouble-with-tor/

[14] Thirteen years of Tor Attacks: https://github.com/Attacks-on-Tor/Attacks-on-Tor

[15] What is Traffic Analysis Attack? https://www.netreo.com/blog/traffic-analysis-attack/

[16] How to gather SSL details from any website? https://securitytrails.com/blog/extract-ssl-data

[17] Apache HTTP Server Version 2.4 Documentation : https://httpd.apache.org/docs/2.4/vhosts/examples.html

[18] What is Wget and how to use it? https://www.hostinger.com/tutorials/wget-command-examples/

[19] What is cURL? https://blog.hubspot.com/website/curl-command

[20] Geoip: https://www.maxmind.com/en/geoip-demo

[21] ELI5: The deep web, onion routing , and TOR: https://www.reddit.com/r/explainlikeimfive/comments/19twt9/eli5_the_deep_web_onion_routing_and_tor/

[22] Blockchain website: https://www.blockchain.com/explorer

[23] Chainabuse website: https://www.chainabuse.com/

[24] Worksheet of our project on github: https://github.com/RahulRaviHulli/ENGR-6991

[25] What is Netcat (nc)? https://nordvpn.com/fr/cybersecurity/glossary/netcat/

[26] What is SSH and how to use it? https://www.ucl.ac.uk/isd/what-ssh-and-how-do-i-use-it

[27] Torsocks: https://linux.die.net/man/8/torsocks

[28] Deanonymization of Tor HTTP Hidden Services: https://www.youtube.com/watch?v=v45_tkKCJ54

[29] What is a thread? https://www.iitk.ac.in/esc101/05Aug/tutorial/essential/threads/definition.html

[30] Shodan search of a leaked IP: https://www.shodan.io/search?query=172.24.0.2

[31] Docker nginx:1.20.2-alpine-perl: https://snyk.io/test/docker/nginx%3A1.20.2-alpine-perl

[32] WordPress 6.3.2 Security Update For 8 Vulnerabilities: https://www.searchenginejournal.com/wordpress-6-3-2-security-update-for-8-vulnerabilities/498398/

[33] WordPress 5.8.2 Vulnerabilities https://wpscan.com/wordpress/582/

[34] Wordpress 6.1.1 Vulnerabilities https://wpscan.com/wordpress/611/

[35] Wordpress 5.5.3 Vulnerabilities https://wpscan.com/wordpress/553/

[36] PHP 8.2.4 Vulnerabilities: https://www.exploit-db.com/exploits/29733

[37] Klecker Debian: https://lists.debian.org/debian-devel-announce/2000/10/msg00010.html

[38] Using Clearnet Websites using Tor browser: https://tor.stackexchange.com/questions/19704/visiting-clearnet-websites-using-tor-browser

[39] Server-Status: https://docs.oracle.com/cd/E89529_01/doc.124/wireless/concepts/c_cmp_cable_configuring_pm_topology_server_status.html

[40] SSLScan: https://github.com/rbsec/sslscan

[41] SSL / TLS Configurations: https://www.ibm.com/docs/en/external-auth-server/6.0.0?topic=login-create-ssltls-configuration

[42] Shodan Search Engine: https://www.shodan.io/

[43] What is bogon IP address: https://www.apnic.net/manage-ip/apnic-services/registration-services/resource-quality-assurance/what-is-a-bogon-address/

[44] How to use wget with Tor bundle in Linux: https://superuser.com/questions/404732/how-to-use-wget-with-tor-bundle-in-linux

[45] Blockchain Forensics: https://cipherblade.com/blockchain-forensics/

[46] Beautiful Soup – Python: https://realpython.com/beautiful-soup-web-scraper-python/

[47] The Grep Command : https://www.cyberciti.biz/faq/howto-use-grep-command-in-linux-unix/

[48] HTTP Statistics: https://www.wireshark.org/docs/wsug_html_chunked/ChStatHTTP.html

[49] Bash vs Python: https://dev.to/husseinalamutu/bash-vs-python-scripting-a-simple-practical-guide-16in

[50] Smuggling Attack: https://portswigger.net/web-security/request-smuggling

[51] DoS Attack: https://www.paloaltonetworks.com/cyberpedia/what-is-a-denial-of-service-attack-dos

[52] What is Sensitive Data Exposure? https://securiti.ai/blog/sensitive-data-exposure/

[53] Path Traversal Attack: https://owasp.org/www-community/attacks/Path_Traversal

[54] Cross-Site Scripting Attack: https://owasp.org/www-community/attacks/xss/

[55] Authentication Bypass Attack: https://www.automox.com/blog/vulnerability-definition-authentication-bypass

[56] SQL Injection Attack: https://owasp.org/www-community/attacks/SQL_Injection

[57] Private and public keys: https://www.ibm.com/docs/en/zos/2.1.0?topic=keys-rsa-private-public

[58] Best securiy practices to avoid web server attacks: https://www.indusface.com/blog/what-are-the-best-security-practices-to-protect-against-the-main-types-of-attacks-on-web-applications/

[59] Headless CMS Security: https://hygraph.com/blog/cms-security

[60] List of Secure Dark Web Email Providers in 2023: https://freedomhacker.net/list-of-secure-dark-web-email-providers-in-2016-4946/