HTTPFuzz: Web Server Fingerprinting with HTTP Request Fuzzing

Animesh Kar[a], Andrei Natadze[b], Enrico Branca[c] and Natalia Stakhanova[d]
Department of Computer Science, University of Saskatchewan, Saskatoon, Canada

Keywords: Web Server Fingerprinting, Protocol Fuzzing.

Abstract: Web server-based fingerprinting is a type of fingerprinting that allows security practitioners, penetration testers, and attackers to distinguish between servers based on the set of information these servers disclose. A common approach to hide this information is to apply fingerprinting mitigating techniques. In this work, we present a new approach for fingerprinting web server software irrespective of the applied fingerprinting mitigation techniques. The premise of our approach is based on the simple insight, i.e., web servers handle different types of HTTP requests differently. We use the fuzzing approach for intelligent and adaptive selection of HTTP requests that are able to provoke servers to disclose their service-level information.

1 INTRODUCTION

Web services are pervasive in the modern Internet, so are the attacks on web applications. The attacks on the underlying web server technology or web applications often leverage the server's misconfigurations or security flaws of web application software. The presence and the extent of security misconfigurations are typically uncovered through *web server fingerprinting* process that allows to determine type, version of server software, used libraries, and application-related vulnerabilities.

Fingerprinting web technology installed on the server is routinely used by network security administrators for security assessment (Abdelnur et al., 2008), security analysts for penetration testing, and by researchers for research purposes (Li et al., 2009; Shamsi and Loguinov, 2017).

Web server fingerprinting is also a common approach that enables the adversaries to explore the existing configurations, collect information, and prepare for a more sophisticated compromise.

The arsenal of server fingerprinting techniques falls into two groups: passive and active techniques. Passive fingerprinting techniques rely on sniffing mechanisms to infer the web server applications. Although stealthy, they are known to be less accurate. On the other hand, active fingerprinting can achieve higher accuracy, but requires active and often aggressive probing of a remote server.

The classical approach to an active web server fingerprinting relies on *banner grabbing* through HTTP protocol that involves sending crafted requests to server to illicit a response. The information available in a returned banner often contains specific markers that can be associated with the web application technology. Unless modified, these markers remain stable and can be easily matched with application fingerprints. Hence, a common mitigation approach followed by practitioners is to manipulate or obfuscate the identifiable information preventing fingerprinting. The existing fingerprinting countermeasures typically aim to modify banner information (Apache, 2022; Microsoft, 2009) or introduce variations in the server response to render automatic fingerprinting ineffective (Yang et al., 2010). In spite of the long history of fingerprinting countermeasures, their effectiveness against fingerprinting techniques has not been studied.

In this work, we present a first study that investigates this. Specifically, we explore the capabilities of prevalent web server fingerprinting tools in a presence web server identity masking. We explore eight available server fingerprinting mechanisms, primarily open-source and widely used in practice, on the example of four popular web servers: Apache, Microsoft IIS, Nginx and Lighttpd. We discover that most of the fingerprinting mechanisms fail even in the presence of not sophisticated mitigation measures.

[a] https://orcid.org/0000-0003-1931-2234
[b] https://orcid.org/0000-0003-2764-0991
[c] https://orcid.org/0000-0001-6316-7789
[d] https://orcid.org/0000-0003-1923-319X

Kar, A., Natadze, A., Branca, E. and Stakhanova, N.
HTTPFuzz: Web Server Fingerprinting with HTTP Request Fuzzing.
DOI: 10.5220/0011328900003283
In *Proceedings of the 19th International Conference on Security and Cryptography (SECRYPT 2022)*, pages 261-271
ISBN: 978-989-758-590-6; ISSN: 2184-7711

To this extend, we propose a new method for fingerprinting web server applications irrespective of applied fingerprinting mitigation techniques. We design *HTTPFuzz* approach that leverages random mutation fuzzing.

Coming from software testing, fuzzing is a well known technique that allows to generate massive amounts of erroneous, unexpected, or random test cases to observe their effects on the target program aiming to identify cases that can trigger software problems or bugs. In our context, we leverage fuzzing approach to generate unexpected HTTP requests that may consequently illicit unexpected response revealing the true identity of a web server. Note that while in a traditional setting fuzzing often aims to crash a target application or a system, in our context, this is an undesirable outcome. Our goal is to determine a set of HTTP requests that produce server responses that can most accurately expose web server technology.

As opposed to fuzzing that generates massive amounts of cases, most modern servers are equipped to rate-limit the amount of incoming traffic, hence restricting the number of requests that can be potentially sent to a server for fingerprinting. We thus further design fuzzing-guided heuristics to select HTTP requests agnostic to the applied fingerprinting measures.

Our contributions in this work are as follows:

- We present an effectiveness analysis and discuss limitations of fingerprinting countermeasures against prevalent in industry web server fingerprinting techniques.
- We propose a new method called HTTPFuzz for fingerprinting web server applications agnostic to the applied fingerprinting mitigation techniques.
- We explore the proposed HTTPFuzz approach in practice by performing a fingerprinting of over 100K unknown servers.

We offer a prototype of HTTPFuzz to the security community in an effort to facilitate research in this area[1].

2 BACKGROUND

A web server is a combination of hardware and software that uses HTTP/HTTPS protocols as a convention to respond to client requests. An example of HTTP exchange of request and response message from IIS configured Virtual Machine is given in Figure 1.

[1]https://cyberlab.usask.ca/datasets/httpfuzz-main.zip

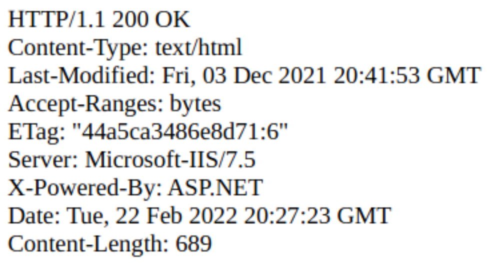
```
HTTP/1.1 200 OK
Content-Type: text/html
Last-Modified: Fri, 03 Dec 2021 20:41:53 GMT
Accept-Ranges: bytes
ETag: "44a5ca3486e8d71:6"
Server: Microsoft-IIS/7.5
X-Powered-By: ASP.NET
Date: Tue, 22 Feb 2022 20:27:23 GMT
Content-Length: 689
```

Figure 1: An example of HTTP response from IIS web server to '*GET / HTTP/1.1*' request.

The structure of the typical HTTP request includes several elements:

- *Request-Line*: the first line of an HTTP request that typically contains a `<method>` token, that tells the server what to do with the resource (e.g., GET, HEAD, POST), followed by the `<URI>`, that specifies the resource on the server, `<protocol>` and its `<version>`. The tokens in the request line are separated by '/'. Nine methods have been standardized for use in HTTP requests. Among them, web servers are required to support the GET and HEAD methods, while other methods are optional (Fielding and Reschke, 2014).
- *Request Headers*: may be present to provide additional context for a request. For example, by including conditional fields for the resource state, indicating accepted formats for the response or media types (e.g., Content-Length, Accept-Encoding), or including information about the user, user agent, and resource (e.g., User-Agent, Referer, From).
- *Request Body*: an optional part that may provide additional information to correctly retrieve requested data.

When a request is received, the server constructs a HTTP response that includes several elements:

- *Status-line*: contains the protocol and its version followed by a numeric status code with its short textual description of the status code. The status code is a 3-digit number that indicates a result of HTTP request execution.
- *Response Headers*: similar to a request header, response header aims to provide additional information to complement what is already given in status-line. Note that the header is optional, and the response may contain 0 or more headers.
- *Response Body*: typically provides the resource requested by a client or an error message in case of failure.

3 RELATED WORK

Web server fingerprinting is a widely-studied topic. Early studies leveraged differences in TCP/IP stack implementation for fingerprinting servers. For example, host operating system identification based on analysis of encrypted communication was introduced by Beverly (Beverly, 2004). Shamsi et al. proposed to automatically generate server signatures based on TCP/IP packets for large-scale fingerprinting (Shamsi and Loguinov, 2017).

Differences in network system implementation were also leveraged by Yang et al. (Yang et al., 2019) for fingerprinting of IoT devices. The approach relied on Neural Network classification model build with features extracted from the network layer, transport layer, and application layer. Another concept of fingerprinting for the IoT platform traffic was introduced by designing a set of IoT platform fingerprinting workflows via traffic analysis (He et al., 2022). The authors manually analyzed the deciphered traffic and found that some traffic in IoT platforms using private protocols had obviously distinguishable characteristics.

There has been a significant research done in the area of browser fingerprinting. Browser fingerprinting is the process of collecting data from a client's web browser in order to create a device's fingerprint (Laperdrix et al., 2020). Browser fingerprinting usually gathers a massive amount of data about a user's device, ranging from hardware to operating system to browser configuration (e.g., user's device model, operating system, screen resolution, user timezone, preferred language setting, browser version, tech specification of user's CPU, graphics card, and etc.).

As opposed to browser fingerprinting, web server fingerprinting aims to determine the software characteristics of the server. Lee was one of the first researchers to point out that different web servers implement the HTTP response differently despite RFC specification outlining the proper HTTP response (Lee et al., 2002). Hence, Lee developed HMAP, an automated tool that leveraged a method that uses the characteristics of HTTP messages to determine the identity of an HTTP server with high reliability. For fingerprinting web servers, three types of characteristics from HTTP responses were taken into consideration: syntactic, semantic, and lexical. HMAP works with variations of GET, HEAD request lines using the wrong capitalization of protocol name, version, and long URIs and compares each of the responses with a list of known server characteristics. The tool does not take into consideration of other available HTTP methods (e.g., DELETE, TRACE). The approach is based on the explicit assumption that server header is present and provides trustworthy information.

The study performed by Saumil et al. applied the tool HTTPrint to analyze web server fingerprinting (Shah, 2003b). The primary focus of this work was the analysis of server banners from common web servers. Only a few HTTP requests were considered including DELETE, improper HTTP version, junk request.

Shrivastava (Shrivastava, 2011) provides examples of fingerprinting mechanisms such as HTML data inspection, presence of the files based on HTTP response codes, checksum-based identification. The author focused on the application fingerprinting on the application level.

Auger outlined fingerprinting techniques based on web architecture, server, application software, backend database version. Banner grabbing technique of the HTTP responses were highlighted as server headers are likely to reveal identifying information, e.g., intermediate agents, via header, server version, and error pages (Auger, 2009). The study analyzed the lexical, syntactic, and semantic information provided in HTTP response produced by abnormal requests.

Lavrenovs et al.(Lavrenovs and Melón, 2018) carried out analysis of website extracted from Alexa's top one million list and presented a research on the security of the most known websites. Although the study was not focused on server fingerprinting, it provided an insight on how much information can be revealed through server-side headers. The analysis reached two conclusions: a) the more popular domains leak less information and b) HTTP sites are less restrictive than HTTPS served sites in terms of the information that they provide, mostly for server related headers.

The study conducted by Book et al. (Book et al., 2013) applied machine learning techniques for generating server fingerprinting automatically. The authors used Bayesian inference without building initial server features. They used a set of 10 specialized HTTP requests on 110,000 live servers. The analysis was performed on the response codes and MIME types returned by the server. The authors calculated unique fingerprint for each type of web server and then matched the responses of unknown web servers against the developed fingerprint set.

Techniques for detecting web servers from the banner information, HTTP response characteristics (order of server and date headers), and special HTTP requests were introduced by Huang et al. (Huang et al., 2015). Through special HTTP requests which included correct and undefined request methods, the authors analyzed the web servers based on the servers' different processing procedures.

In this work, we leverage fuzzing to create ambiguous HTTP requests that may potentially provoke servers to disclose identifiable information. This is not the first use of fuzzing for security purposes. Barreaud et al (Barreaud et al., 2011) examined implementations of the HTTP protocol embedded in smart cards for the presence of vulnerabilities. The authors created mutators to represent the various mutation types that were then used to automatically evaluate the application's behavior with the goal of exploiting vulnerabilities on the servers. In a similar vein, Jabiyev et al. (Jabiyev et al., 2021) looked at HTTP protocol exploitation through HTTP Smuggling. Using a grammar-based fuzzer, the approach aimed to automatically exploit the HTTP communication.

4 MITIGATING FINGERPRINTING

The majority of the web server applications are shipped with numerous configuration options. These settings are easily identifiable and in essence form a server fingerprint that can be later matched during a fingerprinting process. Based on the complexity of the fingerprint and the process, the fingerprinting techniques can be broadly divided into several groups:

- A direct identification of a technology-based on server response fields, e.g., *X-POWERED-BY* or *SERVER* values in Figure 1.
- Inference-based identification that leverages information leading to server technology identification (e.g., presence of files, libraries identification by tools like Aquatone, WhatWeb, Wappalyzer, Nikto).
- Heuristics-based fingerprinting that infers the corresponding web technology by combining and analyzing various patterns of HTTP response elements to those contained in the database of fingerprints (e.g., httprint, httprecon).

4.1 Fingerprinting Tools

Nmap. is arguably one of the most dominant and versatile tools for network analysis and fingerprinting (Lyon, 2009). By default, for an open port, nmap produces a series of TCP packets that constitute a generic 'null' probe followed by a 5 second pause. The probe packets typically include a probe string, i.e., an arbitrary ASCII string. The server response, if any is produced, is compared to a list of signature regular expressions within nmap database. If a full match is not found, nmap proceeds with a more specific (usually probably service-oriented) probes that may also be strengthened by increased probe intensity. To avoid contamination of results, Nmap typically starts a new connection for each probe which adds a significant overhead to a fingerprinting process. While the tool is favored by practitioners for isolated scans, the process is clearly unfeasible for large-scale fingerprint analysis. Nmap generally does not anticipate significant variations from the expected server response, as a result, various system modifications might produce an illusion of a completely different service.

HTTPrint. (Shah, 2003a) aims to overcome the challenges of pure signature-based approaches that can be easily deceived with web server banner configuration. The server customization might produce various deviations in HTTP response. To account for these variations, HTTPrint leverages fuzzy logic and assigns confidence ratings to choose the most probable signature and consequently to determine the type of HTTP server.

Httprecon. (Ruef, 2017) was designed as the successor to the **HTTPrint**. Similarly, Httprecon aims to leverage the fact that most servers may exhibit dominant behavior which allows for their quick identification. The tool sends 9 HTTP requests (including malicious requests), each of which might be repeated 22 times resulting in 198 requests per server. The obtained responses are analyzed for the presence of known dominant characteristics which are then summarized and matched to a database containing known Key Analysis Indexes (KAI).

Wappalyzer. (Alias, 2017) is an open-source community-driven tool. Among other things, Wappalyzer can recognize CMS (content management systems), web server software, web frameworks, analytical tools, and commonly used web front-end libraries. The approach is based on the premise that every technology leaves distinct traces, hence, as opposed to many other techniques, Wappalyzer uses HTML code and web page content to determine the presence of web technology.

Aquatone. (Henriksen, 2019). Designed for reconnaissance, Aquatone is a versatile tool capable of discovering subdomains, fingerprinting servers, and identifying visually similar web pages. Based on

Wappalyzer's fingerprinting engine, Aquatone sends one HTTP GET request per host with a randomly selected set of HTTP headers. If a server response is produced, it is analyzed using regular expression pattern matching with respect to the fingerprints contained in the corresponding fingerprint database.

WhatWeb. (Horton, 2017) is a fingerprinting tool for identification of web technology installed on the web server. Similar to other fingerprinting techniques, WhatWeb analyzes the HTTP responses using regular expression pattern matching and fuzzy logic to illuminate variations in server response. In addition to this, WhatWeb considers meta-data of the webpage (e.g., email addresses, web framework modules) to recognize potential web technology.

FavFreak. (Batham, 2020) uses a direct approach to web server fingerprinting based on the hash values of favicon icon file present on the website.

Nikto. (Andress, 2011) is an open-source Web server analysis tool that tests for vulnerabilities on the server-side. Nikto indexes all the *files and directories*, that it encounters on the target web server to locate the technical information. The fingerprinting approach is based on the presence of identifiable and traceable web components (e.g., favicon.ico files).

4.2 Fingerprinting Countermeasures

A fingerprinting mitigation approaches tend to manipulate or obfuscate the identifiable information preventing fingerprinting. The existing fingerprinting countermeasures typically fall into the following categories:

- *Hiding Identifiable Information*: this includes removing response headers containing identifiable information (e.g., `<Server>`), or completely disabling banners to limit the information disclosure.
- *Deceiving the Fingerprinting Process*: the deception techniques range from modifying or misrepresenting banner information to replacing external libraries or files to mislead the fingerprinting and cause incorrect identification. These methods often target inference-based fingerprinting tools. Other techniques include modification of HTTP responses to introduce variations and render heuristic-based fingerprinting ineffective (Yang et al., 2010).

Table 1: Configurations of servers in a controlled environment.

Web server	OS	(Mis)Configurations
No configuration		
Microsoft-IIS v.7.5	Windows Server 2008	Plain
Nginx v.1.14.1	Centos 8.5.2111	Plain
Nginx v.1.18	Fedora 32	Plain
Lighttpd v.1.4.55	Centos 8.5.2111	Plain
Lighttpd v.1.4.59	Debian 11.0	Plain
Lighttpd v.1.4.45	Ubuntu 18.04.6	Plain
Apache v.2.4.37	Centos 8.3.2011	Plain
Configured		
Microsoft-IIS/10.0	Windows Server 2016	Disabled Server Banner
Microsoft-IIS v.8.5	Windows Server 2012	Disabled Server Banner
Microsoft-IIS/10.0	Windows Server 2016	Disabled X-Powered-By
Nginx v.1.16	Debian 11.0	Disabled Server Banner
Nginx v.1.18	Ubuntu 20.04.3	Set `<Server>` value to Apache/2.4.52
Lighttpd v.1.4.55	Fedora 30	Set `<Server>` value to Microsoft-IIS/7.5
Apache v.2.4.29	Ubuntu 18.04.3	Disabled X-Powered-By
Apache v.2.4.51	Fedora 34	Set `<Server>` value to lighttpd/1.4.55
Apache v.2.4.46	Ubuntu 21.04	Disabled Server header and X-Powered-By

4.3 Analysis of Fingerprinting Tools

To evaluate the accuracy of fingerprinting in the presence of various countermeasures, we have setup and configured four different types of web servers: Microsoft-IIS, Nginx, Apache, and Lighttpd. The servers were installed on different operating systems to explore the behaviour of the mentioned fingerprinting techniques. Since the objective of this work is to explore the fingerprinting capabilities in a presence of mitigation, we further apply various mitigation measures resulting in the nine configurations presented in Table 1.

Hiding Identifiable Information. As a first approach, we configured servers to remove response headers that directly state the installed web server software (such as `<Server>`, `<X-Powered-By>`, `<X-AspNet-Version><X-AspNetMvc-Version>`). To disable this information, we installed and configured libapache2-mod-security2 module on Apache servers (Apache, 2022), *nginx-extras* package on

Nginx servers and configured system settings on IIS servers. It was not possible to completely remove response headers and disable banner information for Lighttpd servers.

Deceiving the Fingerprinting Process. To deceive the fingerprinting, we have modified the existing HTTP headers to supply incorrect information. `<Server>` header.

- for *Apache*, using libapache2-mod-security2, we modified the server value in *security.conf* file.
- for *Nginx*, we added *more_set_headers* variable in *nginx.conf* file to change the `Server` header value.
- for *Lighttpd*, we modified the value of *lighttpd.conf* file's *server.tag* variable.
- We were not able to modify the `Server` header value for IIS server.

Since at least two of the tools are known to rely on the presence of favicon.ico file in their fingerprinting process, we have created random icon files to replace favicon.ico in two of the servers.

Our goal was to mitigate fingerprinting by configured the servers and application settings without crashing the servers or making the applications fail to run or behave unexpectedly/abruptly.

4.4 Results

To create a baseline for our analysis, we launch all fingerprinting tools against plain installations of seven servers. All tools were able to correctly identify web server technology for all server installations. The fingerprinting results were drastically different when fingerprinting mitigation measures were applied as the results in Table 2 show.

After modifying and hiding the headers along with technology information, the majority of the tools failed to detect the accurate environment on the web servers. For servers, the identification was mostly blank or incorrect detection.

None of the inference-based tools could give any proper information regarding technology and server information against the 9 configured servers. It is quite obvious that they largely rely on the banner grabbing information in spite of their difference in fingerprinting approaches. For example, with modified `Server` header, the majority of tools (5 out of 8) simply extracted the value without any further verification. In cases when the identifiable headers were present, the majority of tools provided no identification returning blank response.

Hence, even in a presence of simple mitigation these techniques failed to properly recognize the technology. Among the heuristic-based tools, Nmap and HTTPrint tool were able to detect only one IIS server despite of information hiding. Httprecon tool performed the best among the tools only misclassifying Nginx server as IIS. However, the versions of the servers were not detected properly.

5 HTTPFuzz DESIGN

Our analysis of fingerprinting techniques revealed their inadequacy to provide accurate recognition of web server technology in a presence of even small deviations from the expected results. Our goal is to design an automated approach for fingerprinting web server applications insensitive to variations that might be introduced by various mitigation techniques. The flow of the proposed HTTPFuzz approach is introduced in Figure 2.

HTTPFuzz is a multi-stage approach that leverages fuzzing to generate mutated HTTP requests. These requests are directed towards configured HTTP servers set up in our controlled environment for testing. The responses are collected and analyzed to reduce the massive set of mutated requests and to select mutations that are likely to produce correct identification. Finally, the requests generated for the selected mutation types are deployed to fingerprint technology of unknown web servers in the wild.

Fuzzing Module. One of the insights our analysis of fingerprinting techniques revealed is that different technologies respond to ambiguities in HTTP request's fields differently, hence our approach aims to systematically explore possible discrepancies.

Our fuzzing module uses a grey box paradigm, i.e., it generates mutated requests given a valid HTTP request syntax and elements. Although HTTP specifications allow an HTTP request to contain several elements, our fuzzer restricts its mutations to `<request-line>` fields, i.e., optional elements (`<headers>` and `<body>`) are not generated. All `<request-line>` fields are considered mutable and undergo character-level manipulations: insertion, deletion, swapping. The character pool includes ASCII character set, i.e., capital and small letters are considered to be different characters. For insertion, up to 256 random characters can be appended. In addition to that, protocol `<version>` included the use of float and integer values.The examples of mutations are shown in Table 3.

Table 2: Fingerprinting servers with the existing techniques.

Tools	**Default Configuration**	**Hiding Headers**					**Deceptions**			
		Server			X-Powered-By		Server Value Changed			Modified favicon.ico
		Nginx	IIS	Apache	IIS	Apache	Nginx	Lighttpd	Apache	All Servers
Whatweb	✓	×	×	×	×	×	Apache	IIS	Lighttpd	×
Aquatone	✓	×	×	×	×	×	Apache	IIS	Lighttpd	×
Nikto	✓	×	×	×	×	×	Apache	IIS	lighttpd	×
Wappalyzer	✓	×	×	×	×	×	Apache	IIS	Lighttpd	×
Nmap	✓	×	✓	×	×	×	Apache	IIS	Lighttpd	×
FavFreak	✓	×	×	×	×	×	×	×	×	×
Httprint	✓	×	✓	×	×	×	×	Apache	✓	×
HttpRecon	✓	IIS/6.5	✓	✓	×	×	IIS/7.5	✓	✓	×

✓ correct identification
× blank/no identification
value identified information (correct/incorrect)

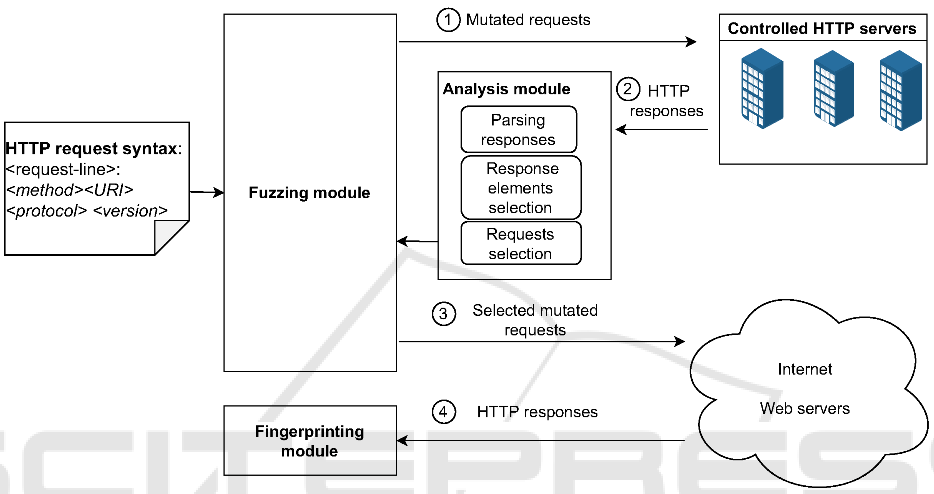

Figure 2: The flow of HTTPFuzz approach.

Table 3: The examples of a HTTP GET request mutations.

Mutation	**HTTP request line**
Deletion	GT / HTTP/1.1
Insertion	GETA / HTTP/1.1
Swapping	GxT / HTTP/1.1

Analysis Module. The HTTP responses from the mutated requests are collected and parsed to extract features for further fingerprinting. The analysis stage aim is two-fold: reducing the number of request and selecting the elements of response that are indicative of the employed web technology.

The fuzzing approach generates a significant number of requests. Using all mutated requests is infeasible due to traffic rate-limiting policies commonly set by servers. Flooding servers with the requests is similarly not efficient for large-scale analysis. Hence, we need to select requests that trigger discrepancies and are likely to produce behaviour indicative of the employed web applications. Similarly, since the mutated requests produce often unexpected HTTP response, we need to select elements of the response that we can use for accurate fingerprinting.

For these purposes, we leverage machine learning classification. In this work, we explore three classification algorithms: Neural Networks (NN), Decision Trees (DT), and Random Forest (RF). The classification parameters of the algorithms employed in this study are shown in Table 4.

For classification, we derive features that characterize any HTTP response, i.e., all elements that might be potentially present in the response, including status line, headers, and body, and characteristics of the applied request mutations. For each response header, we create three additional features: indicating the presence of a header, its value, name, and capitalization pattern. Since this initial step is performed on the known servers, the corresponding server's technology is used as a ground truth label in this classification.

In essence, classification analysis allows for *intelligent* and *adaptive* selection of requests that are able to provoke servers to disclose their service-level information. Note that this process is not reliant on the stability of server behaviour or the knowledge of applied fingerprinting mitigation mechanisms.

The corresponding mutation information for responses that are successfully classified are forwarded

Table 4: The parameters of the classification algorithms.

Alg.	Hyperparameters
Neural Networks	max_iter=10000, learning_rate='adaptive', solver='adam', alpha=0.001, random_state=42
Decision Tree	max_depth=6, criterion='entropy', splitter='best'
Random Forest	n_estimators=50, criterion='entropy', bootstrap=True, min_samples_split=2, min_samples_leaf = 1, oob_score = True, max_features="auto", max_leaf_nodes=None, min_impurity_decrease=0.0, max_depth=6

to the fuzzing module to further guide fuzzing for fingerprinting in the wild.

Fingerprinting Module. The classification analysis yields a small set of mutated requests are then used for fingerprinting unknown servers. The mutated requests are sent to target servers and the received responses are forwarded to the fingerprinting module.

6 EXPERIMENTS

Experimental Setup. Our approach was implemented using the Python language (v 3.9) with the scikit-learn library (v 0.23.2). The request to web servers were sent using CURL utility. A summary of the classification algorithms' parameters used in the prediction module is given in Table 4. A 5-fold cross-validation was employed to measure the accuracy of all machine learning models.

6.1 Experiments with Servers in a Controlled Environment

As the first step of the experiments, we focused on analysis of fingerprinting accuracy on a diverse set of web servers set up in the controlled environment (steps 1-3 of the HTTPFuzz approach). For the four different servers set up in the controlled environment, the fuzzing module generated 7,411 mutated requests that were sent to 16 web servers which included both plain and configured ones to mitigate fingerprinting.

The obtained servers' responses were collected and parsed to remove features with a low variance ($var = 0$) as they are unlikely to contribute to the classification model. We also standardized features by removing the mean and scaling them to unit variance. As servers unless configured to hide information, commonly respond with (optional) headers that indicate the server's web technology (e.g., `Server` and `X-Powered-by`), we excluded these headers from the analysis. The resulting set contained 68 features.

To shed light on the most statistically relevant features, we have decided to utilize Information Gain (IG) to identify the importance of each feature. Tables 6 show the top selected features with $IG > 0.01$. All further experiments were conducted with this selected set of 11 features.The highly ranked feature is 'filtered headers', a string indicating an order in which headers appear in the server response. Note that this feature does not contain header values but rather indicates that depending in the server technology headers appear in a distinct order which allows the fingerprinting of the server technology.

We have further classified the parsed requests from our controlled four web servers with the selected features using three machine learning algorithm. Table 7 shows the accuracy of these classifiers for fingerprinting.

Our results show that we can fairly accurately (93.67% accuracy with RF and DT) identify the type of web technology even in the presence of fingerprinting countermeasures. Compared to the results of commonly employed fingerprinting utilities (Table 2), this is a significantly improved performance.

A close manual analysis of the results revealed that only a few mutation categories were able to provoke the servers to disclose their information. Among them are *insertion* by appending a random number of characters on several request line methods (GET, HEAD, DELETE, TRACE, OPTIONS), protocol and its version, and *swapping* of characters on request line method and protocol. The total of 32 categories listed in Table 5.

6.2 Fingerprinting Web Servers in the Wild

The selected mutation categories are forwarded to the fuzzing engine for fingerprinting servers in the wild. Based on these categories, HTTPFuzz generates HTTP requests (one per category) and sends them to web servers to be identified.

Collected Data. For this step, we selected domain names from a list of the top 1 million domains called Majestic Million list [2]. From the list of ranked domains, a set of 350,000 domains has been selected at random for our experiments. These domains were scanned using nmap utility for possible web server fingerprinting. Out of these servers, 127,169 responded. The vast majority of the servers (89%) were not identified by nmap, i.e., nmap produced no-match results (Table 8). 12,450 (10%) of domains were identified by nmap. Furthermore, 10,091 (82%) of these domains sent HTTP responses that contained

[2]https://majestic.com/reports/majestic-million

Table 5: The categories of mutated requests.

Mutation	Mutation Category	Request
Insertion	Target: GET	GETA / HTTP 1.1
Insertion	Target: OPTIONS	OPTIONSA / HTTP 1.1
Insertion	Target: HEAD	HEADA / HTTP 1.1
Insertion	Target: DELETE	DELETEA / HTTP 1.1
Insertion	Target: TRACE	TRACEA / HTTP 1.1
Insertion	Target: URI	GET /A HTTP 1.1
Insertion	Target: URI	OPTIONS /A HTTP 1.1
Insertion	Target: URI	DELETE /A HTTP 1.1
Insertion	Target: Protocol	GET / HTTPA 1.1
Insertion	Target: Protocol	OPTIONS / HTTPA 1.1
Insertion	Target: Protocol	HEAD / HTTPA 1.1
Insertion	Target: Protocol	DELETE / HTTPA 1.1
Insertion	Target: Protocol	TRACE / HTTPA... (18 ”A” appended) 1.1
Insertion	Target: Protocol version	OPTIONS / HTTP 1.11
Insertion	Target: Protocol version	HEAD / HTTP 1.11111111111111111
Insertion	Target: Protocol version	DELETE / HTTP 1.111111111
Insertion	Target: Protocol version	TRACE / HTTP 1.111... (180 1 appended)
Insertion	Target: Protocol version	GET / HTTP 1.111111
Insertion	Target: URI	TRACE /A HTTP 1.1
Swapping	Target: HEAD	aEAD / HTTP 1.1
Swapping	Target: DELETE	DELERE / HTTP 1.1
Swapping	Target: GET	aET / HTTP 1.1
Swapping	Target: Protocol	DELETE / HpTP 1.1
Swapping	Target: Protocol	GET / HITP 1.1
Swapping	Target: Protocol	TRACE / HVTP 1.1
Swapping	Target: Protocol	OPTIONS / HSTP 1.1
Swapping	Target: Protocol	HEAD / HcTP 1.1
Swapping	Target: Protocol	HEAD / HTTd 1.1
Swapping	Target: Protocol version	GET / HTTP 5.1
Swapping	Target: OPTIONS	aPTIONS / HTTP 1.1
Swapping	Target: TRACE	dRACE / HTTP 1.1
Deletion	Target: Protocol version	GET / HTTP .1

`<Server>` header and the value of this header agreed with the server identification produced by nmap.

The servers were then sent the selected mutated requests. We have further performed two sets of experiments targeting web servers labeled by nmap and those that were not identified by nmap.

Evaluating HTTPFuzz on Wild Servers Identified by Nmap. As another validation step, and in an absence of large datasets with labeled web servers, we have turned to a list of domains that their HTTP responses provided web server identification in the `<Server>` header. We have selected a subset that was also correctly (according to the value of this header) fingerprinted by Nmap. Although the correct server recognition by Nmap cannot be guaranteed, we feel that this was a reasonable verification.

To evaluate our fingerprinting approach, we apply the HTTPFuzz to this subset of 10,091 servers. In this analysis, we retain the selected set of 15 features, i.e., `Server` and `X-Powered-by` related features are excluded from this analysis.

The results given in Table 7 confirm the effectiveness of the proposed approach. With fairly high accuracy (98.8% with RF), HTTPFuzz is able to determine the web server technology. Since this set provided a larger pool of samples (including other servers beyond the originally considered), we have further retrained the model on this set. The results were similar to the ones archived with an earlier model (97% with RF). Similarly, the top 10 features included the features selected with controlled servers. We therefore used this model in fingerprinting unknown servers.

Table 6: Feature Importance.

Feature	IG
filtered_headers	0.506496
content-type_value	0.166241
connection_value	0.040618
accept-ranges_case	0.039472
status_message	0.039399
accept-ranges_value	0.033453
status_code	0.033364
allow_value	0.029266
content-length_case	0.026900
allow_case	0.017372
content-type_case	0.011032

Table 7: Classification accuracy of fingerprinting web servers.

	NN	DT	RF
Controlled environment	93.9%	94.29%	94.29%
Web servers on the wild identified by nmap	87.66%	98.39%	98.45%

Fingerprinting Unknown Servers. We explore the practicality of our approach on the set of 102,752 servers not identified by Nmap. In this analysis, we have leveraged the model built by HTTPFuzz during the validation step allowing us to label unidentified servers. Similar to the previous experiments, we have relied on a small set of features that excluded server identifiable information if it was present.

Since these servers have no corresponding ground truth, the resulting responses were classified and the final result was selected based on the majority label of the classified responses (over 80% of responses contained the corresponding label). The prediction results are given in Table 9. The majority of the servers were classified as Apache and Nginx. This is a pre-

Table 8: Data collected in the wild.

Total Domains	127,169
Invalid Domains	11,967 (9.4%)
Valid Domains	115,202 (90.6%)
Domains not identified by NMAP	102,752 (89.2%)
Domains fingerprinted by NMAP	12,450 (10.8%)
Among them:	11,881 (95.4%)
Apache	5,995 (50.45%)
IIS	148 (1.24%)
Lighttpd	17 (0.1%)
Nginx	5,721 (48.1%)
Domains with no `<Server>`	20,727 (18%)
Domains with known `<Server>`	94,475 (82%)
Domains fingerprinted by NMAP with known `<Server>` header value:	
Results do not agree	2,161 (17.6%)
Results agree	10,091 (82.4%)
Among them:	
APACHE	5536 (55.21%)
IIS	51 (0.50%)
LIGHTTPD	9 (0.09%)
NGINX	4431 (44.19%)

Table 9: Fingerprinting unknown servers.

Server	Frequency
Tengine	41
Lighttpd	112
Varnish	170
Microsoft-IIS	579
Cowboy	709
Python	852
Caddy	1,703
Apache	37,648
Nginx	60,938
Total	102,752

dictable outcome as these two servers have the majority market share.

7 CONCLUSIONS

In order to fingerprint the technological aspects of a web server, gathering proper information plays a vital role. In the modern cyber world, the cyber attackers try to build intelligent attack methods and exploit remote web servers by gathering vendor name, vendor versions and the services running on the web servers. When specific software version of the server is revealed, the server becomes susceptible to attacks against software that is known to contain security vulnerabilities.

In this work, we proposed a novel web server fingerprinting approach that utilizes HTTP request fuzzing and machine learning classification algorithms to analyze the behavioural characteristics found in web server responses. We showed that technology detection does not require the server to disclose its exact identity and a small set of requests can be sufficient to achieve highly accurate fingerprinting without any advance knowledge of applied mitigation techniques.

REFERENCES

Abdelnur, H., State, R., and Festor, O. (2008). Advanced Network Fingerprinting. In Trachtenberg, A., editor, *Recent Advances in Intrusion Detection*, volume Volume 5230/2008 of *Computer Science*, pages 372–389, Boston, United States. MIT, Springer Berlin / Heidelberg.

Alias, E. (2017). Wappalyzer Project. https://github.com/AliasIO/wappalyzer.

Andress, J. (2011). Http fingerprinting and advanced assessment techniques.

Apache (2022). Apache Module mod_headers. https://httpd.apache.org/docs/current/mod/mod_headers.html.

Auger, R. (2009). Project: WASC Threat Classification. http://projects.webappsec.org/w/page/13246925/Fingerprinting.

Barreaud, M., Bouffard, G., Kamel, N., and Lanet, J.-L. (2011). Fuzzing on the http protocol implementation in mobile embedded web server. In *C&ESAR*.

Batham, D. (2020). FavFreak project. https://github.com/devanshbatham/FavFreak.

Beverly, R. (2004). A robust classifier for passive tcp/ip fingerprinting. In *International Workshop on Passive and Active Network Measurement*, pages 158–167. Springer.

Book, T., Witick, M., and Wallach, D. S. (2013). Automated generation of web server fingerprints. *arXiv preprint arXiv:1305.0245*.

Fielding, R. T. and Reschke, J. (2014). Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content. RFC 7231.

He, X., Yang, Y., Zhou, W., Wang, W., Liu, P., and Zhang, Y. (2022). Fingerprinting mainstream iot platforms using traffic analysis. *IEEE Internet of Things Journal*, 9(3):2083–2093.

Henriksen, M. (2019). AQUATONE project. https://github.com/michenriksen/aquatone.

Horton, A. (2017). WhatWeb project. https://github.com/urbanadventurer/WhatWeb.

Huang, Z., Xia, C., Sun, B., and Xue, H. (2015). Analyzing and summarizing the web server detection technology based on http. In *2015 6th IEEE International Conference on Software Engineering and Service Science (ICSESS)*, pages 1042–1045.

Jabiyev, B., Sprecher, S., Onarlioglu, K., and Kirda, E. (2021). T-Reqs: HTTP Request Smuggling with Differential Fuzzing. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 1805–1820.

Laperdrix, P., Bielova, N., Baudry, B., and Avoine, G. (2020). Browser fingerprinting: A survey. *ACM Transactions on the Web (TWEB)*, 14(2):1–33.

Lavrenovs, A. and Melón, F. J. R. (2018). Http security headers analysis of top one million websites. In *2018 10th International Conference on Cyber Conflict (CyCon)*, pages 345–370.

Lee, D., Rowe, J., Ko, C., and Levitt, K. (2002). Detecting and defending against web-server fingerprinting. In *18th Annual Computer Security Applications Conference, 2002. Proceedings.*, pages 321–330.

Li, Z., Goyal, A., Chen, Y., and Paxson, V. (2009). Automating analysis of large-scale botnet probing events. In *Proceedings of the 4th International Symposium on Information, Computer, and Communications Security*, ASIACCS '09, page 11–22, New York, NY, USA. Association for Computing Machinery.

Lyon, G. F. (2009). *Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning*. Insecure, Sunnyvale, CA, USA.

Microsoft (2009). IIS ServerMask. https://www.iis.net/downloads/community/2009/01/servermask.

Ruef, M. (2017). httprecon - Advanced Web Server Fingerprinting. https://www.computec.ch/projekte/httprecon/.

Shah, S. (2003a). Http fingerprinting and advanced assessment techniques.

Shah, S. (2003b). An Introduction to Http Fingerprinting.

Shamsi, Z. and Loguinov, D. (2017). Unsupervised clustering under temporal feature volatility in network stack fingerprinting. *IEEE/ACM Transactions on Networking*, 25(4):2430–2443.

Shrivastava, A. (2011). Web Application finger printing. https://anantshri.info/articles/web_app_finger_printing.html.

Yang, K., Li, Q., and Sun, L. (2019). Towards automatic fingerprinting of iot devices in the cyberspace. *Computer Networks*, 148:318–327.

Yang, K.-x., Hu, L., Zhang, N., Huo, Y.-m., and Zhao, K. (2010). Improving the defence against web server fingerprinting by eliminating compliance variation. In *2010 Fifth International Conference on Frontier of Computer Science and Technology*, pages 227–232. IEEE.