Smudged Fingerprints: Characterizing and Improving the Performance of Web Application Fingerprinting

Brian Kondracki
Stony Brook University
bkondracki@cs.stonybrook.edu

Nick Nikiforakis
Stony Brook University
nick@cs.stonybrook.edu

Abstract

Open-source web applications have given everyone the ability to deploy complex web applications on their site(s), ranging from blogs and personal clouds, to server administration tools and webmail clients. Given that there exists millions of deployments of this software in the wild, the ability to fingerprint a particular release of a web application residing at a web endpoint is of interest to both attackers and defenders alike.

In this work, we study modern web application fingerprinting techniques and identify their inherent strengths and weaknesses. We design WASABO, a web application testing framework and use it to measure the performance of six web application fingerprinting tools against 1,360 releases of popular web applications. While 94.8% of all web application releases were correctly labeled by at least one fingerprinting tool in ideal conditions, many tools are unable to produce a single version prediction for a particular release. This leads to instances where a release is labeled as multiple disparate versions, resulting in administrator confusion on the security posture of an unknown web application.

We also measure the accuracy of each tool against real-world deployments of the studied web applications, observing up to an 80% drop-off in performance compared to our offline results. To identify causes for this performance degradation, as well as to improve the robustness of these tools in the wild, we design a web-application-agnostic middleware which applies a series of transformations to the traffic of each fingerprinting tool. Overall, we are able to improve the performance of popular web application fingerprinting tools by up to 22.9%, without any modification to the evaluated tools.

1 Introduction

From its static origins, the web has evolved into the ubiquitous position it occupies in our lives in large part due to the proliferation of *web applications*. Web applications provide the dynamic functionality required for the numerous online services we have come to depend upon. Additionally, open-source web application software, such as WordPress [35] and Drupal [20], has provided anyone the ability to create immersive experiences on their own websites, with minimal effort. The popularity of open-source web applications is at an all time high, with 43% of *all* websites on the Internet running WordPress [32].

This immense popularity, along with the sensitive nature of transactions conducted by online services, has increased the attention paid to web applications by attackers. Moreover, the widespread use of web applications has further widened the asymmetric gap enjoyed by attackers, as a single vulnerability in a version of a popular web application allows attackers to exploit all websites utilizing it. This has led to instances where millions of websites each running the same vulnerable version of a particular web application have been attacked at the same time [9]. Moreover, the failure of many website administrators to keep web application deployments up-to-date, in conjunction with public cataloging of known attack payloads, can make it trivial for even non-sophisticated attackers to execute attack campaigns [24]. Due to the impracticality and blind spots of manually cataloging and updating services at scale, organizations with large numbers of servers are highly susceptible to these attacks. In the past, vulnerable web application deployments on such networks have resulted in large-scale compromise, leading to billions in damages [16, 17]. The most infamous of these was the 2017 Equifax data breach in which attackers gained access to the private financial data of over 140 million consumers due to the failure of administrators to update a vulnerable deployment of Apache Struts on their network [5].

Therefore, it has become increasingly apparent that the ability to identify the type and version of web application software running at a particular web address is paramount not only for attackers, but also defenders to find and remedy vulnerable endpoints before they are exploited. This form of intelligence gathering, known as *web application fingerprinting*, assesses the content and behavior of a web endpoint to garner information regarding the application that produced it. Web application fingerprinting tools such as BlindElephant [4], WhatWeb [10], and Wappalyzer [34] have grown in popularity,

providing a quick and easy way for anyone to identify the type and version of an active web application.

In this paper, we analyze the state of web application fingerprinting, filling a knowledge gap on the techniques utilized, and their effectiveness in practice. We perform a literature review of all web application fingerprinting techniques, ultimately distilling this information into a list of six web application fingerprinting tools that fully summarizes the scope of actively-utilized academic and commercial work in this area. We then develop WASABO, a container-based web application testbed system allowing us to stage any version of a web application for laboratory testing of each fingerprinting tool, as well automate large-scale online fingerprinting campaigns.

Using WASABO, we audit the performance of each fingerprinting tool against 1,360 versions of popular web application software in their default installation states. Through this, we learn the distinct behaviors of each web application fingerprinting technique, as well as their upper-bound performance in the most ideal fingerprinting settings. By cross-referencing the results produced by each fingerprinting tool with the most severe vulnerabilities recorded for each of the web applications we study (average severity score of 7.7 out of 10), we find 82 instances where a vulnerable version of a web application would be incorrectly labeled as a non-vulnerable version.

Finally, we gauge the performance of the studied fingerprinting tools against real-world web application deployments, determining the effect of administrator-applied customizations. We find that the performance of each tool suffers greatly in this setting, with accuracy scores decreasing 20%-80% compared to laboratory performance. However, using the network middleware module of WASABO, we bridge this performance gap by applying a series of *scanner-agnostic* transformations to the traffic produced by the fingerprinting tools, including the use of a real browser to communicate with each site, reducing instances of bot-detection. Overall, we find WASABO is able to improve the performance of web application fingerprinting tools by up to 22.9% against real web applications in the wild.

Our main contributions are as follows:

- We summarize the current state-of-the-art in actively-utilized web application fingerprinting techniques, including the most popular tools in the space.
- We design and develop a web application sandbox framework, called WASABO, to automate the processes of deploying a wide-range of web applications, and the auditing of web application fingerprinting tools.
- We audit the performance of each fingerprinting tool in ideal and real-world deployments of web applications, allowing us to identify common website customizations that stifle their effectiveness.
- As part of WASABO, we build a network middleware module that dynamically alters the traffic transmitted by each fingerprinting tool, allowing us to improve their performance in a scanner-agnostic manner

```
<link href='https://example.com/wp-content/...'
     />
    ...
<meta name="generator" content="WordPress 5.8" />
    ...
<p>
    Powered by
    <a href="https://wordpress.org">WordPress</a>
</p>
```

Listing 1: *Examples of potential fingerprinting sources in dynamic content generated by WordPress.*

To enable future research in understanding and evaluating web-application fingerprinting, we are making our WASABO system available to other researchers [1].

2 Background on Web Application Fingerprinting

The ability to accurately identify the web application responsible for the content present on a particular web endpoint is a valuable tool for both attackers and defenders. Due to the public cataloging of exploits in CVE databases, attackers who can pinpoint the version of a web application can easily produce a list of exploits to compromise it. Conversely, administrators of large, undocumented/partially-documented networks must identify and patch vulnerable web applications before they are exploited. Current fingerprinting solutions rely heavily on the content generated by each particular web application–with decision boundaries drawn based on the dynamic and static content produced.

Dynamic Content

One of the primary uses of web applications is to generate dynamic content, based upon the current user and context. This means that visits to the same endpoint by two distinct users, or subsequent visits by the same user, will likely result in unique content returned. However, certain elements within this content are likely to remain static, leading to patterns that could be used to fingerprint the application. This includes HTML elements (e.g., `meta` tags), filesystem artifacts present in hyperlinks (e.g., the WordPress content directory, `wp-content`), and static strings within the content body (e.g., page headings/section titles). Moreover, some web applications include the entire version string within these elements by default, with the option to disable it either through configuration settings or the installation of third-party plugins. An example of this is demonstrated in Code Listing 1, which shows lines taken from a real WordPress deployment that contains these sources of information leakage.

Leakage of this information in dynamic content results in trivial identification of the underlying web application. The consistent format of software version strings allows

for the usage of regular expression searches of web content, reducing the effort required to create fingerprinting scripts. Moreover, this content is typically included in multiple locations throughout a website, including locations commonly frequented by visitors. This complicates the process of distinguishing fingerprinting bots from real users.

Static Content

In support of dynamically-generated HTML content, web applications also serve static content consisting of supplementary material (e.g., CSS and JavaScript files), documentation (e.g., `README` and Changelog files), and multimedia (e.g., image files). As this content is not state-dependent, it remains consistent for each user, and each subsequent visit to a site. Changes to these files will only occur when a new version of the particular web application is released, coinciding with changes to application functionality (e.g., the addition of new lines of code in a JavaScript file to support a new feature).

As this content will always remain the same for any deployment of a particular web application, its modification history can be used as a proxy for the version history of the web application as a whole. For instance, if one knows that only the newest version of a particular web application contains a certain function in a JavaScript file, they could accurately identify deployments of that version in the wild by requesting that JavaScript file, and checking for the existence of that function. To automate and extend this process, one could pre-compute the hashes of all static files in all versions of a web application, allowing for the identification of any version of a web application by determining which release has the greatest number of matching hashes to that of an arbitrary site. An example of this is demonstrated in Listing 2 in which the file `wp-includes/js/media-models.js` had a single change between WordPress versions 5.8 and 5.8.1, resulting in an entirely different hash value.

This method of fingerprinting does not rely on the inclusion of a version string within content, as is the case of dynamic content fingerprinting. However, it is prone to version collisions, where two or more web application versions are equally likely, due to limited changes occurring in static content between those versions. Additionally, web administrators conscious of site fingerprintability can, in theory, limit the effectiveness of this approach by denying access to certain static files, or modifying them so as to change the expected hashes.

3 Experimental Setup and Methodology

In this section, we first present our process for selecting web application fingerprinting tools. We then describe WASABO, our system for evaluating the performance of each technique in a laboratory setting. Lastly, we detail the methodology and motivation behind each experiment.

WordPress 5.8: a380109124ae1089684a00c8ac9c5c68

```
   1  /******/ (function(modules) { // webpackBootstrap
        ...
1412            posts_per_page: 40
        ...
1708  /******/ });
```

WordPress 5.8.1: 3086430ffe51587d0ac5ae51b4404df7

```
   1  /******/ (function(modules) { // webpackBootstrap
        ...
1412            posts_per_page: 80
        ...
1708  /******/ });
```

Listing 2: *Change in file wp-includes/js/media-models.js between WordPress versions 5.8 and 5.8.1. A minor change in this file between these subsequent versions results in a completely different MD5 hash of the file contents, leading to a potential fingerprinting source.*

3.1 Collection of Web Applications and Fingerprinting Tools

Due to their widespread use and history of serious vulnerabilities, open-source PHP web applications are likely targets for attackers. Furthermore, the techniques used to fingerprint these web applications (i.e. the static and dynamic fingerprinting methods reviewed in Section 2), and the countermeasures utilized to prevent such fingerprinting, apply to any web application, regardless of the underlying web framework. Therefore, we narrow the scope of our work to open-source PHP web applications, without any loss of generality.

Of all the possible web applications, we select the following for our study: WordPress [35], Drupal [20], Joomla [22], MediaWiki [25], and phpMyAdmin [30]. Together, these five web applications power over 45% of all websites [33], providing services such as: content management, wiki development, and database management. With a wide range of functionalities and use-cases, these web applications provide a diverse set of test cases to gauge the performance of each fingerprinting technique.

For each of the five web applications, we download all available mainline releases from the previous decade (e.g., 2013–2023). We do not test any development versions of the web applications to avoid the bugs introduced in these versions from influencing the results of the subsequent experiments. Moreover, as most public websites are unlikely to deploy development versions of their chosen web applications, the fingerprinting results against these versions would not accurately describe expected real-world performance. Statistics on all web applications studied in this work are located in Table 5 in the Appendix.

To systematically search for all current web application fingerprinting tools, we conducted a literature review in the field of web application fingerprinting. Additionally, we utilized search engine queries for terms such as “Web Application Fingerprinting Tool” and “Web Application Identification Tool”. This search resulted in a large collection of web application fingerprinting tools, ranging from academic tools

Table 1: *List of all web application fingerprinting tools we audit in this work, including the type of fingerprinting techniques utilized as well as the upper bound of requests sent to target sites. Cells with two request quantities indicate the tool has varying behavior when scanning a known/unknown web application.*

Tool	Fingerprinting Type		# Requests	
	Dynamic	Static	Default	Aggressive
BlindElephant [4]	○	●	$10^0/10^1$	$10^0/10^4$
WhatWeb [10]	●	●	10^0	10^2
VersionInferer [8]	●	●	10^0	10^1
Wappalyzer [34]	●	○	10^0	10^0
Metasploit-Joomla [6]	●	○	10^0	-
Metasploit-WordPress [7]	●	○	10^2	-

to commercial services. However, we limit the fingerprinting tools we audit in this study to only those that provide support for the chosen PHP web applications, and are open-source. Inspecting the source code of each fingerprinting tool allows us to fully understand the utilized techniques and their limitations.

Table 1 lists the web application fingerprinting tools we audit in this study. For each tool, we examine the source code to determine the techniques utilized to fingerprint web applications. We note that the six fingerprinting tools chosen for this study do not represent an exhaustive list of all available fingerprinting tools, but represent a comprehensive summary of the techniques used by available fingerprinting tools. For instance, there exists a number of WordPress-specific fingerprinting tools such as WPScan [37] and Plecost [31]. However, as these tools are specific to only one web application and utilize the same fingerprinting techniques as the more general tools we identified, we consider them out-of-scope for this study. We chose to include two Metasploit modules for WordPress and Joomla since the Metasploit framework provides scaffolding for the authors of each module, allowing us to treat the overall framework as a single generic tool. Moreover, we note prior work has presented other methods to fingerprint web applications using analysis of HTML XPaths [48], as well as JavaScript and CSS attributes [43], but since these works have not materialized into available tools that can be utilized from the client-side, we also consider them to be out-of-scope. We qualitatively compare these papers against the ones that are in-scope for this paper in Section 7.

The selected fingerprinting tools are diverse in behavior, with five utilizing dynamic content to identify each web application and three utilizing static content–including WhatWeb and VersionInferrer that utilize both to varying degrees. We observe that the majority of these tools provide the user with the option to tune the number of probes sent to each site, sending a greater number of probes to potentially increase performance while decreasing stealthiness. For this study, we measure the performance of each tool in both the default "out-of-the-box" settings as well as the maximum settings, referred to as *Default* and *Aggressive*, respectively. We note that the Aggressive setting varies for each tool,

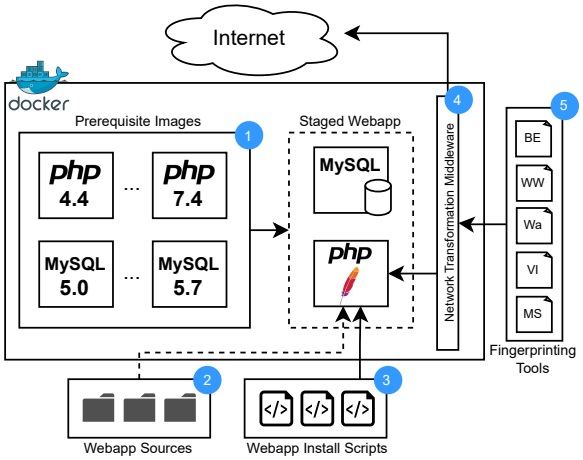

Figure 1: *Architecture of* WASABO.

with some providing a simple command line flag for a more "aggressive" scanning approach, with others simply allowing the user to specify a maximum number of probes. In the latter case, we set this number to total number of probes available to each tool, determined by examining the source code.

3.2 WASABO: Web Application Sandbox

To comprehensively audit the performance of each fingerprinting tool against the chosen web applications, we design and develop WASABO (Web Application SAndBOx), a web application sandbox framework. This Docker-based [19] system, illustrated in Figure 1, allows for the automated testing of web applications in both an offline and online setting. Additionally, traffic between any web client and the end web application can be inspected and modified using our network middleware module. Below, we describe the technical details of WASABO, and explain its benefits. Upon publication of this paper, we will be releasing WASABO to inspire the community to perform additional work in the space of web-application fingerprinting.

Offline Web Application Testing

To facilitate the large-scale offline testing of web applications, WASABO provides a method to automate the deployment of any version of a web application from its source code. The offline functionality of WASABO is divided into five main categories: (1) Prerequisite Docker images, (2) Web Application Source Code, (3) Web Application Installation Scripts, (4) Network Transformation Middleware, and (5) Test-case Scripts. Web-application-specific options for each of these five categories are specified in configuration files for that web application version.

Each web application chosen for this study, and all relevant versions, require a unique set of prerequisite technologies to function properly. Specifically, they each require an installation of PHP as well as access to a MySQL database. However, the version combinations of each of these technologies can vary drastically between each web application, and over time (releases between 2013-2023). It is for this reason that we

chose to utilize Docker as the foundation of WASABO. The ability to isolate installations of PHP and MySQL into distinct images, and then join arbitrary versions of each using Docker's software-defined networking features, allow us to efficiently create the proper execution environments of any version of the five chosen web applications using only a handful of Docker images. We note that the PHP Docker image also contains an Apache web server [14] to host all web application content.

When a particular version of a web application is chosen for deployment, its configuration file is read by WASABO, specifying the particular versions of PHP and MySQL it requires. Docker containers for each of these images are then created and configured, including mounting the source code for the chosen web application version into the Apache web content directory of the PHP container.

Once a web application is deployed in its required execution environment, it must still be configured by completing its default installation steps. For each of the web applications chosen, this involves completing a series of HTML forms received when visiting the IP address of the site. The user provides information such as the title of the newly created site, and the address of the MySQL server. Typically, this process is completed through a GUI interface in a browser window. However, since we aim to completely automate the process of deploying web applications, we capture the HTTP POST requests transmitted by a browser during the installation process of each web application version, and encode them into installation scripts to be replayed at a later date. When the HTML installation forms do not change between versions, one captured installation session can be reused by WASABO to install multiple web-application versions. The encoding process is manual and exhaustive in order to ensure each version of each web application can be installed consistently for each execution of WASABO. Once this encoding process is completed, no additional manual effort is required to (re)deploy all versions of that particular web application.

Finally, upon verification of successful installation (by identifying strings within the content implying success), a user-provided test-case script is executed on the staged web application. For our purposes, we create simple scripts to execute fingerprinting attempts from each of the fingerprinting tools, and log the results of each run. However, any arbitrary code can be executed, meaning WASABO could – in the future – be used for cases unrelated to fingerprinting or web application security.

Online Web Application Testing

Next to experimenting with web applications in a laboratory setting, WASABO also makes it easy to perform tests against web application deployments in the wild. Similarly to the offline testing module, arbitrary code can be run against one or more live URLs on the Internet. This reduces the time and effort required for one to perform large-scale web application testing.

Network Middleware

When performing experiments with web applications, either online or offline, it may be useful to inspect or modify network traffic in transit. This can assist in understanding the functionality of a web application, or debugging any particular issues that may arise in its operation. The network middleware module of WASABO enables this functionality by intercepting all network traffic to and from each web application deployment using mitmproxy [28]. This includes both cleartext traffic as well as encrypted traffic, assuming the client is configured to accept the mitmproxy TLS certificate. Similar to the PHP and MySQL deployments used in the offline module of WASABO, mitmproxy is deployed using a Docker image, greatly decreasing the time required to configure the testing environment.

Users may provide a mitmproxy addon script [29] which can contain arbitrary logging or transformation instructions for network traffic. It is important to note that since all web content transformations occur at the network level, they are *completely agnostic* to both the client as well as web application. This means that, for the fingerprinting use case described in this paper, web content modifications to improve the performance of fingerprinting tools can be made on-the-fly without modifying the source code of any fingerprinting tool. In Section 5, we describe the network middleware addon scripts we utilized to improve the real-world performance of the web application fingerprinting tools studied in this work.

4 Laboratory Fingerprinting Performance

To gauge the upper-bound performance of the identified web application fingerprinting tools, we measure their identification accuracy when encountering fresh installations of web applications, with no user-customizations applied. This means that steps commonly taken to "harden" the deployment of each web application are not included, such as deleting or removing access to unnecessary files, or installing anti-fingerprinting plugins. Additionally, we note that we also do not modify any web application-specific configuration options that are not necessary to complete installation. This means all "out-of-the-box" settings are configured, and any default behaviors are present. By testing against fresh installations of each web application, we ensure that each fingerprinting tool is provided the greatest amount of information regarding the identity of the web application it is interacting with.

As mentioned in Section 3.1, we record the performance of each tool in both Default and Aggressive scanning modes. In doing so, we can determine the expected performance of each tool without any fine-grained tuning of arguments. Conversely, we can observe the change in performance at the most aggressive state of each tool, and compare the number of probes required to achieve that performance. Intuitively, a fingerprinting tool that is able to achieve high identification accuracy while keeping the number of probes low is superior to one which requires a large number of probes (since the former increases the stealthiness of the attack).

Additionally, we note that one of the evaluated fingerprinting tools, BlindElephant, permits the user to specify the type of web application running on the target, allowing for more focused probes for determining the exact version of that particular web application. For this tool, we test the two options in which the user provides the name of the web application, versus when they do not. In doing this, we can observe the effect of this additional input on its performance.

Using WASABO, we record the outputs of each fingerprinting tool in the states previously described against all web application versions. Prior to conducting any experiments, we ensure each tool is updated to the most current release and, if relevant, trained on the source code of all web application versions. This "training" process is required for VersionInferrer and BlindElephant, allowing each tool to build maps of static file hashes to web application versions, as described in Section 2.

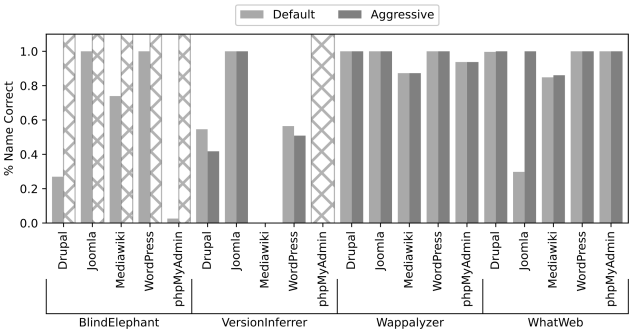

Figure 2: *Performance of fingerprinting tools when attempting to guess web application name with both default and aggressive scanning modes. Areas shaded with "X" indicate that particular result is not applicable due to fingerprinting tool capability.*

4.1 Experimental Evaluation and Results

We leverage the automated web application testing functionality of WASABO to audit each tool against 1,360 versions of popular web applications dating back as far as 2013. We note that this scale of laboratory testing would not have been possible without WASABO due to the manual effort required to install and configure each web application release.

Analysis of the performance of web application fingerprinting tools must occur on a sliding-scale: there is not always one correct answer. Rather, the amount of information garnered by a fingerprinting scan can range from no result, to only the type of the web application produced, to finally the entire version string. Therefore, we separate the analysis of these tools into two distinct categories: the ability to produce the correct web application type (e.g., WordPress vs. Drupal), and the ability to produce the correct web application version (e.g., WordPress 5.8 vs. WordPress 5.9). While producing the entire version string is optimal for the user, there is still value in producing only the web application type. Moreover, we note that the former group is a subset of the latter, as a fingerprinting tool that is able to produce the full version string is also able to produce the web application type.

We also explore the tendency of each fingerprinting tool to output more than one version for each fingerprinting scan. While a particular fingerprinting tool may be able to produce a correct result (i.e., a correct web application type and version), that prediction will not hold much value if it is simply one of a large set of possible versions. An optimal result for a fingerprinting tool would be one that is able to produce one, correct, prediction for each host it encounters.

4.2 Web Application Type Prediction

Figure 2 shows the performance of each fingerprinting tool in producing the correct web application type, as a percentage of all versions of each web application, in both default and aggressive scanning modes. We note that the probes contained within the Metasploit framework are not listed in this figure as they are specific to particular web applications and, thus, their use presumes that the user has knowledge of the web application type. Additionally, as BlindElephant does not support an aggressive scanning mode when guessing the web application type, only the results for the default scanning mode are present. Likewise, as VersionInferrer does not currently support fingerprinting phpMyAdmin releases, such results are not applicable. All such non-applicable cases are signified by areas shaded with Xs.

In ideal conditions, we observe varying accuracy among each fingerprinting tool. The two tools that only utilize static content fingerprinting, BlindElephant and VersionInferrer, perform consistently worse than Wappalyzer and WhatWeb, which utilize regular expression searches of dynamic content. This indicates that, in most cases, identification of a web application is performed best when utilizing strings present in dynamic content. This makes sense as many web applications will, by default, include information about their identities within webpage content in either HTML `meta` tags, or page titles and headers. Meanwhile, using only the hashes of static content can lead to misidentification, if content that is utilized by a wide-range of websites (e.g., common JavaScript libraries) is attributed to the wrong web application.

Generally, utilizing a more aggressive scanning approach does not garner greater performance. The only case in which this is true is WhatWeb's fingerprinting of Joomla deployments. In its Default scanning mode, WhatWeb attempts to reduce the number of requests sent to the target site by only downloading the URL provided by the user, and searching the returned content with regular expressions for each web application plugin supported by it. By studying WhatWeb's performance, we discovered errors in the regular expressions specified in its Joomla plugin file [12]. This prevents WhatWeb from trivially identifying over 50% of all Joomla installations in its Default scanning mode. It is only when additional probes are

Table 2: *Performance of web application fingerprinting tools in guessing web application version when provided with no prior knowledge of web application type. D indicates that the default settings were used, whereas A indicates most aggressive scanning.*

Tool	Webapp	Type	Version Matched (%) Major	Minor	Full
VersionInferrer	Drupal	D	53.6	53.6	53.6
		A	40.8	40.8	40.8
	Joomla	D,A	100.0	100.0	100.0
	Mediawiki	D,A	0.0	0.0	0.0
	WordPress	D	56.5	56.5	55.9
		A	50.9	50.9	50.2
	phpMyAdmin	D,A	0.0	0.0	0.0
Wappalyzer	Drupal	D,A	96.4	8.2	0.0
	Joomla	D,A	0.0	0.0	0.0
	Mediawiki	D,A	87.3	87.3	87.3
	WordPress	D,A	100.0	100.0	100.0
	phpMyAdmin	D,A	0.0	0.0	0.0
WhatWeb	Drupal	D	0.0	0.0	0.0
		A	97.4	72.4	65.5
	Joomla	D	0.0	0.0	0.0
		A	100.0	100.0	100.0
	Mediawiki	D	46.1	46.1	46.1
		A	46.7	46.7	46.7
	WordPress	D,A	100.0	100.0	100.0
	phpMyAdmin	D	31.2	31.2	31.2
		A	34.9	34.9	34.9

transmitted in Aggressive mode that a correct identification is made for these releases. In all other cases, Aggressive scanning mode has no discernible positive effect: it either provides no additional performance benefits, or causes confusion by transmitting unnecessary probes, reducing performance.

Interestingly, we find that VersionInferrer is unable to identify any release of Mediawiki. Analysis of the behavior of both VersionInferrer and Mediawiki reveals the web application unintentionally bypasses the fingerprinting tool's static content analysis with one of its default behaviors. Specifically, Mediawiki by default redirects visitors requesting the domain root path to a subdirectory containing the name of the website's homepage [26]. When requesting static files to analyze, VersionInferrer simply appends the relative path of each file to the end of the redirected URL, rather than the domain root, resulting in requests for non-existing files. While this behavior prevents correct fingerprinting in this scenario, it is not necessarily incorrect. For instance, if a web application is in fact hosted in a sub directory of a domain's root, this would ensure static files could still be reached. Conversely, BlindElephant, which ignores such redirects, would not be able to retrieve any static files in this scenario. Therefore, it follows that in order to ensure static files could be reached even in the scenario in which a web application is hosted in a subdirectory of a domain's root, future fingerprinting techniques should include checks for such behavior and dynamically generate request URLs to match the current circumstances.

4.3 Web Application Version Prediction

While identifying the type of web application behind a particular website is useful in narrowing the scope of reconnaissance scans, the vast number of releases available for any popular web application makes it difficult to draw any meaningful conclusion about a site's security posture. Therefore, a web application fingerprinting tool that can not only identify the type of web application, but also the version currently present allows attackers to craft exploits for known vulnerabilities, and defenders to discover and patch vulnerable hosts.

Table 2 presents the accuracy of each fingerprinting tool in determining the version of a web application with both Default and Aggressive scanning settings, when provided no prior information on the type of web application present. We note that while BlindElephant is able to operate without prior information to produce the type of the current web application, it does not provide a version prediction in this setting. Therefore, it is excluded from this table. Table 3 shows the performance of the fingerprinting tools when provided with prior information on the type of web application present.

The three result columns in these tables (Major, Minor, and Full) indicate the percentage of web application releases in which each tool was able to identify: only the major version, only the major and minor version, and the entire version string, respectively. The former of these groups is a subset of the latter (i.e., a tool that can identify the full version string can also identify just the major version). Similar to the distinction made between the ability of a fingerprinting tool to determine the type and version of a web application, there exists a performance gradient based on the granularity of the version string produced. For instance, a fingerprinting tool that can determine the full version string (including the major, minor, and incremental version numbers) is more useful than a tool that can only identify the major version.

We find that, generally, these tools do not decrease the granularity of their prediction in the face of uncertainty. That is, the accuracy of each tool does not increase as we focus on major and minor versions of each web application compared to the entire version string. The nature of the fingerprinting techniques utilized makes it such that a tool is unable to determine a "rough-estimate" of the version present. In the case of static content fingerprinting, a prediction is generated by matching the hashes of enough static files to determine one or more potential versions. Similarly, dynamic content fingerprinting will either find a version string within the webpage, or it will not. In the majority of cases where the major version of a web application is predicted correctly but not the minor version or full version string, we find that these correspond to web application versions that are bordering two major versions (i.e., the last or first release of a major version).

The only case in which this is not true is Wappalyzer's fingerprinting of Drupal releases. We find that this tool utilizes a number of sources within Drupal's content that only reveals

Table 3: *Performance of web application fingerprinting tools in guessing web application version when provided with prior knowledge of web application type. D indicates the default settings were used, whereas A indicates most aggressive scanning.*

Tool	Webapp	Type	Version Matched (%) Major	Minor	Full
BlindElephant	Drupal	D	18.8	18.8	18.8
		A	71.4	71.4	70.4
	Joomla	D,A	100.0	100.0	100.0
	Mediawiki	D,A	98.2	97.6	95.2
	WordPress	D,A	100.0	100.0	93.9
	phpMyAdmin	D,A	54.2	54.2	54.2
Metasploit	Joomla	D	100.0	100.0	100.0
	WordPress	D	100.0	100.0	100.0

the major version of the current release. For example, Drupal includes an `X-Generator` header to all responses with the current release's major version (e.g., Drupal 7, Drupal 8, etc.). Wappalyzer uses this header to identify all versions of Drupal, meaning the most information this tool will ever provide the user is the major version of any Drupal deployment. While the intended purpose of the Drupal `X-Generator` HTTP header is to announce the major version of the current deployment, we note that this can be interpreted as a full version string if the user assumes this is the first release of a major revision (i.e., 7.0, 8.0, etc.). Therefore, we count the cases in which Wappalyzer identifies such a release as a correct prediction.

We observe varying performance advantages between the two studied fingerprinting techniques, likely due to the content produced by each web application. For instance, VersionInferrer, which leans heavily on static content fingerprinting, is able to produce a correct version string for less than half of WordPress releases; while Wappalyzer, Metasploit, and WhatWeb, which make use of dynamic content fingerprinting, are able to produce full version strings for all WordPress releases. Conversely, Wappalyzer and WhatWeb struggle to identify Joomla releases in their default scanning modes, while VersionInferrer and Metasploit exhibit near-perfect accuracy.

This dichotomy is best understood by analyzing the WhatWeb's plugin files for both WordPress [13] and Drupal [11]. These plugins separate behavior into "passive" and "aggressive" modes (equivalent to our Default and Aggressive labels). Typically, in WhatWeb's Default scanning mode, dynamic content fingerprinting is utilized to identify a web application using a single request to the target site's homepage. In Aggressive scanning mode however, WhatWeb greatly increases the number of requests transmitted and includes static content fingerprinting to expand the breadth of information sources. Therefore, in cases such as WordPress, which commonly divulges the current version in a number of locations, dynamic fingerprinting is sufficient. This leads to WhatWeb's consistent performance across both Default and Aggressive scanning modes: the static fingerprinting probes sent in Aggressive scanning mode are not necessary as the version determination is made using the, less aggressive, dynamic fingerprinting probes.

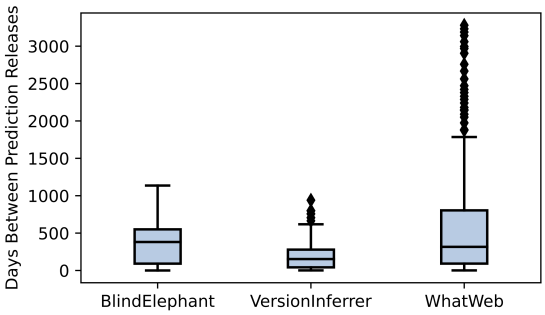

Figure 3: *Distribution of the number of days between the minimum and maximum releases in each prediction.*

4.4 Version Collisions

Even though the accuracy of a fingerprinting tool is a vital measure of its overall performance, the number of predictions required to produce a correct result is also an important factor in determining the practicality of that tool. It is common for web application fingerprinting tools to output a set of predictions in cases where it is not certain which of the group the current target is using. However, even if a fingerprinting tool is able to produce a correct prediction, it will not be of much use to the user if that is one of many predictions.

Figure 4 shows the CDFs of the number of predictions produced per version of each web application in both default and aggressive scanning modes. Here, we see the benefits of dynamic content fingerprinting when compared to static content content fingerprinting. Wappalyzer, Metasploit, and WhatWeb (in Default scanning mode) each utilize only dynamic fingerprinting, meaning a prediction is produced if and only if a version string if found within the content. Therefore, there will not be a case in which there is an equal probability between two or more different releases: there will either be a correct prediction, or there will be no prediction produced.

On the other hand, tools such as VersionInferrer, BlindElephant, and WhatWeb (in Aggressive scanning mode), utilize static content fingerprinting to varying degrees. As described in Section 2, this means that an identification is made by matching one or more hash values of static files from a target site against a pre-computed database. Unlike dynamic content fingerprinting, this technique could result in collisions between versions if there are a number of sequential releases in which these static files do not change. This severely degrades the practicality of using such tools as the operator must decipher which prediction of the group is correct. For instance, while WhatWeb in its Aggressive scanning mode is able to correctly identify all WordPress releases, over 75% of those result in collisions between more than one possible version. This shows how using accuracy as the only measure of a fingerprinting tool's perfor-

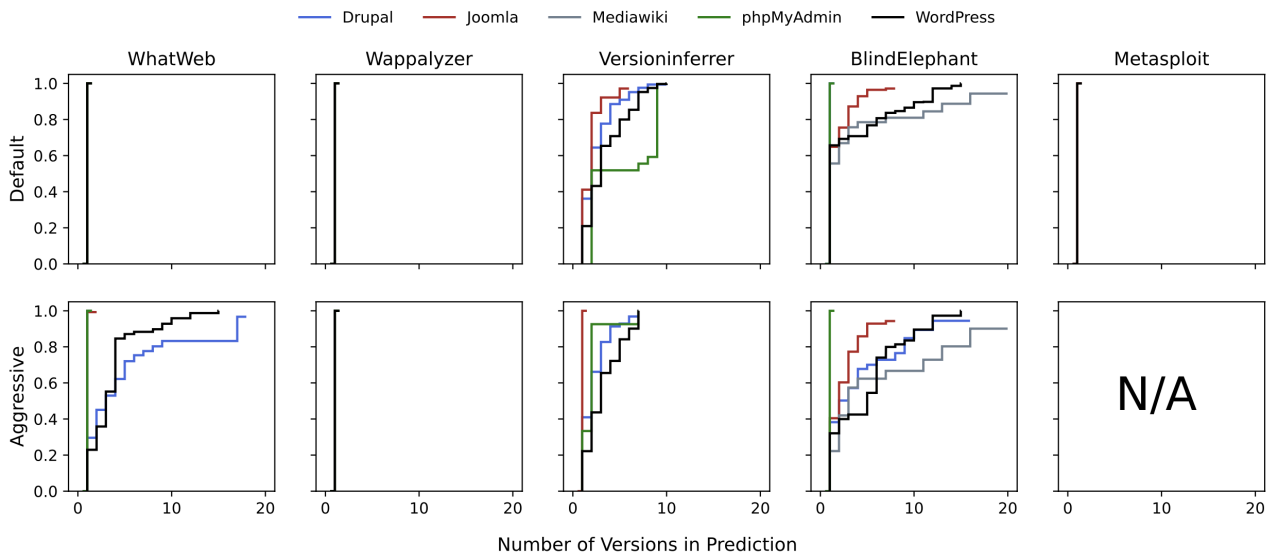

Figure 4: *Distribution of number of versions returned for each fingerprinting attempt.*

mance can be misleading, as a correct identification does not hold much value if it is one of over 10 possible releases.

A consequence of version collisions is the potential of a large temporal range of the predicted versions. A prediction from a web application fingerprinting tool can include web application releases that are multiple months or years apart from each other, with drastically different behaviors and potential vulnerabilities. Figure 3 shows the distributions in temporal range of the three fingerprinting tools that produce multiple predictions for at least one web application. We observe that each of these tools produce a large number of predictions that have release date ranges spanning over the course of multiple years. The most drastic of these cases occurs with WhatWeb's fingerprinting of WordPress 3.7.36 in which the versions 3.7, 3.7.1 and 3.7.36 are produced, with 2,758 days (i.e., 7.5 years) between the earliest and latest versions.

CVE Coverage

Administrators require web application fingerprinting tools to discover outdated and vulnerable hosts on large, undocumented, networks. One critical example of such a need would be when a serious vulnerability is discovered in a particular web application, and documented in a Common Vulnerability Enumeration (CVE) database. CVE entries list the severity of a vulnerability, the versions of the application that are affected by the vulnerability, and details regarding how the vulnerability is triggered. In such a scenario, network administrators would utilize a web application fingerprinting tool to determine if deployments of the affected web application on their networks are vulnerable to the particular CVE, or not. Therefore, collisions between vulnerable and non-vulnerable versions could lead to confusion where an administrator believes a particular deployment is patched, when in reality it is not.

To determine the extent in which this scenario is possible, we explored the tendency for each fingerprinting tool to label a vulnerable version of a web application as a non-vulnerable version. To do this, we recorded the top-10 most severe CVEs (by CVE severity score) on cvedetails.com [18], affecting versions of the web applications we study (i.e., we do not include vulnerabilities that affect versions earlier than 2013). On average, the CVEs we include in this experiment have a CVE severity score of 7.7 out of 10, indicating they correspond to attacks that result in high damage. We map these CVEs to the results of our experiments, and search for cases in which a web application version affected by one of these vulnerabilities is labeled as a non-vulnerable version. All recorded CVEs are listed in Table 6 in the Appendix.

In total, we find 82 cases in which a vulnerable web application was predicted to be a non-vulnerable version. Each of these cases occur with the three tools that utilize static content fingerprinting: BlindElephant, VersionInferrer, and WhatWeb. This is alarming as we only study a small portion of all vulnerabilities affecting each web application version. Furthermore, since the studied vulnerabilities allow attackers to affect serious harm on a website, such as exfiltrating sensitive information or compromising administrator accounts, any such case where this mislabeling occurs can lead to disastrous consequences. We emphasize that defenders are more negatively affected by misclassifications such as these, as attackers can still perform exploits to determine vulnerability while defenders are completely reliant on the output of monitoring/telemetry tools.

For instance, CVE-2020-13664 is a severe vulnerability in recent versions of Drupal which could allow for an attacker to remotely execute code on the victim web server. This vulnerability affects Drupal version 9.0.0, and was patched in version 9.0.1. When attempting to fingerprint this vulnerable version of Drupal using VersionInferrer in its aggressive scanning mode, the user is presented with the results: Drupal 9.0.0 and Drupal 9.0.1. Common intuition would suggest to trust

the most recent version when presented with more than one prediction. Therefore, an administrator would likely assume this particular Drupal deployment is not vulnerable, and take no action, allowing for potential exploitation by attackers.

5 Real-world Fingerprinting Performance

In Section 4, we investigated the expected "best-case" performance of popular web application fingerprinting tools. Generally, we observed high accuracy scores from tools utilizing both static and dynamic fingerprinting against unaltered ("out-of-the-box") web application deployments. However, performance in a laboratory setting does not necessarily translate to equally-strong real-world performance, due to the possible changes made by website administrators to fit their needs. To learn how these techniques perform in the wild, we direct each fingerprinting tool to live websites online utilizing the target web applications. We aim to determine how successful each tool is in identifying a web application behind a completely unknown website, as well as the modifications made that hinder fingerprintability, either inadvertently or on purpose.

To do this, we must first compile a list of real websites powered by the web applications we study in this work. We note, however, that curating a groundtruth list of websites utilizing specific releases of web applications is not a straightforward process. Website administrators are unlikely to advertise the type or version of web application they are using for obvious security reasons. Moreover, third-party lists that provide this information, such as BuiltWith [15], can not be trusted as a groundtruth source as they themselves utilize fingerprinting techniques to identify web technologies.

Therefore, we utilize the "showcase" pages on each web-application vendor website, which advertise examples of popular websites utilizing their software. We find that four of the five web applications we included in this study provide such lists [21, 23, 27, 36]; with phpMyAdmin, a web application primarily used as an internal service, not providing such a list. We therefore exclude it from this section. Additionally, we supplement our list using a small dataset of known real-world web application deployments which we curate by contacting third parties with first-hand knowledge of the type of software running on their endpoints. This list contained additional sites for: WordPress, Drupal, and Joomla.

In total, we curated a list of 726 websites powered by the web applications studied in this work. The breakdown of our set is as follows: 231 WordPress web applications, 231 Joomla, 169 Drupal, 95 Mediawiki. We note that our only source of ground truth is the presence of a particular type of web application at any web address, not the exact version utilized. We therefore measure the performance of each tool by determining how many sites each tool can correctly predict the type of web application powering it. Similar to Section 4.2, we exclude Metasploit from this analysis as its probes operate under the assumption that the user is already aware of the web application type utilized by a site, whereas the remaining tools require no such prior knowledge.

Ethical considerations: As these are real websites, we limit our scans to only the Default scanning modes for each tool. In this scanning mode, each of the studied fingerprinting tools produces a minimal number of GET requests (fetching the main page and, optionally, a small number of static resources), resulting in behavior similar to that of a regular user or benign automated crawler. Therefore, we are confident this experiment does not negatively affect any of the tested sites. We describe our ethical considerations in more detail in the A.1 appendix.

5.1 Network Middleware

Next to determining the performance of each fingerprinting tool against real-world web-application deployments, we also seek to determine the cause of any performance decreases compared to laboratory scanning, and attempt to remedy these sources of degradation. To do this, we develop a series of network transformation scripts that transparently modify the traffic generated by each fingerprinting tool.

To identify which transformations WASABO should support, we used both our domain expertise in web security as well as experience when using the evaluated web-application fingerprinting tools and the ways that they unexpectedly failed when scanning real-world websites. A guiding principle for all our transformations was that they can be performed at the network level (no modification of scanning tools required) and that they are not tied to any specific tool (i.e. all current and future request-issuing clients can benefit from WASABO). Below, we describe the functionality and intended purpose of each WASABO transformation:

Cache-breakers. It is common for websites that experience large amounts of user traffic to place static resources behind caching proxy servers. One common feature of caching services is to minify static web resources [3]. While this can help to improve the performance of sites, it has the added side-effect of thwarting fingerprinting by introducing a discrepancy between the content expected by a fingerprinting tool and what is actually returned. That is, if a fingerprinting tool requests a non-minified version of a resource, but the caching server returns a minified version, the produced file hash will differ from what is expected. To prevent this from occurring, our WASABO middleware appends a query parameter containing a random string to all requests produced by each fingerprinting tool (following the format *msgID=a23k...*). This prevents any potential proxy server from returning a cached version of a requested resource, and instead retrieving the actual file from the origin web server.

Web Path Prediction. Another potential source of fingerprinting tool performance degradation is the serving of web resources from unexpected subdirectories. For instance, if an administrator moves all WordPress installation files from the

Algorithm 1 Algorithm used to predict alternate path for requested resource

```
function REQUEST(url)
    resp ← http_get(url)
    if resp.status_code != 404 then
        return resp
    tree ← get_resource_tree(url.hostname)
    branch ← get_url_branch(url)
    candidate_paths ← branch_search(tree, branch, 2)
    for path in candidate_paths do
        resp ← http_get(path)
        if resp.status_code == 200 then
            return resp

function BRANCH_SEARCH(tree, branch, level)
    paths ← []
    nodes ← tree.get_level_n_nodes(level)
    for node in nodes do
        if branch.head == node then
            paths.append(node.parent.path + branch)
    return paths
```

web server root to a subdirectory called *wp*, all resource requests would need to include the new parent directory at the start of all queries (e.g., */readme.html* → */wp/readme.html*). To address this possibility, our WASABO middleware analyzes the homepages of each targeted website to generate a file tree of all known resources. Using this tree, we can identify common subdirectories between those on each site's homepage and those requested by the fingerprinting tools. This process is demonstrated in Algorithm 1. By utilizing a site's homepage as groundtruth on the web server's filesystem layout, we can identify common parent directories of static resources, and transparently modify request paths to accommodate for these changes.

Real-browser Web Requests. To prevent resource strain and potential abuse, it is common for sites to restrict access to perceived web bots. This is accomplished by analyzing attributes of all requests received to identify signs of automated browsing. One of the most common methods utilized is browser fingerprinting, specifically to determine if a client is utilizing a fully-fledged browser or not [38]. The fingerprinting tools we study in this work utilize the light-weight network requests libraries of popular scripting languages. This greatly increases the efficiency of website scanning, at the cost of potential bot detection. To resolve this possibility, we utilize the network middleware of WASABO to intercept all HTTP GET requests originating from each fingerprinting tool and instead make the same requests with a real web browser, using the popular undetected-chromedriver library [2]. This library allows for automated browsing with a patched version of the chromedriver engine to remove sources of information that anti-bot services utilize, greatly decreasing the chances of detection [39].

5.2 Experimental Results

Table 4 shows the scan results from each fingerprinting tool against our curated list of real web application deployments. Generally, we find that the performance of each fingerprinting tool decreases by 20%-80% when compared to our offline lab results, with BlindElephant seeing the greatest decrease between offline and online tests. For instance, while it was able to correctly identify nearly all WordPress releases out-of-the-box, it was only able to identify 46% of real WordPress sites encountered. Conversely, Wappalyzer had the least performance degradation, though it still saw double-digit accuracy decreases for nearly all web applications.

This performance discrepancy can be explained by the inevitable noise added to web application deployments when they are customized for each site owner's needs. By analyzing the varying performance of each tool when under the influence of each middleware module, we can begin to understand the peculiarities of not only the websites scanned, but also the fingerprinting tools themselves. In the cases of BlindElephant and VersionInferrer, tools entirely dependent on static content fingerprinting, the greatest instances of performance improvement can be seen using the Cache Break middleware module. As caching servers may modify the static resources they serve for performance reasons, retrieving this content directly from origin web servers allows these tools to successfully identify the underlying web applications. Most strikingly, VersionInferrer is able to fingerprint twice as many real-world Mediawiki sites through our WASABO middleware than by default.

For the tools that utilize regular expression searches of dynamic content (Wappalyzer and WhatWeb), bypassing bot detection services such as Cloudflare is of the utmost importance, as being redirected to a CAPTCHA page can completely neutralize any fingerprinting possibility. While the use of a fully-fledged browser to initiate requests does generally help improve performance, it is not a perfect solution. We find that both Wappalyzer and WhatWeb are still flagged by Cloudflare bot detection on a handful of sites, with this affecting Wappalyzer more often due to the larger number of requests it transmits to each site. Surprisingly, these same tools can also suffer from adding cache breaking query parameters to requests, as is the case of WhatWeb when fingerprinting Drupal sites. In its default scanning mode, WhatWeb places high importance on the `X-Drupal-Dynamic-Cache` HTTP header, which is removed from responses of a handful of sites when including a cache break, preventing fingerprinting even with an HTML Generator tag specifying the use of Drupal.

Overall, utilizing each of the three middleware modules in unison provides the greatest fingerprinting performance boost for nearly all web application-fingerprinting tool combinations. We see an average fingerprinting accuracy increase of 5.8%, with a maximum increase of 22.9%. However, this improvement is heavily weighted on static content fingerprinting tools. This is due to the fact that this method of fingerprinting

Table 4: *Performance of each web application fingerprinting tool in guessing the web application type of a set of real world websites, both with and without the intervention of our network middleware module. Lab accuracy of each tool (Section 4.1) included for contrast. The "Combined" column indicates the performance when combining all transformations into one, with the Δ being the difference between the default and Combined performances and the Improvement Factor being the percentage change of fingerprintable sites.*

			Real-world Fingerprinting Accuracy (%)						
Tool	Webapp	Lab Accuracy (%)	Default	Cache Break	Path Predictor	ChromeDriver	Combined	Δ	Imp. Factor (%)
BlindElephant	Drupal	26.97	10.00	12.94	12.94	12.94	**13.02**	3.02	30.18
	Joomla	100.00	29.74	30.17	32.76	32.76	**33.77**	4.02	13.53
	Mediawiki	73.94	15.79	16.84	15.79	15.79	**17.89**	2.11	13.33
	WordPress	100.00	46.52	56.09	56.52	57.39	**59.74**	13.22	28.41
VersionInferrer	Drupal	54.60	30.00	31.76	30.00	36.47	**42.01**	12.01	40.04
	Joomla	100.00	12.07	12.50	12.93	18.10	**23.81**	11.74	97.28
	Mediawiki	00.00	3.16	**6.32**	3.16	4.21	**6.32**	3.16	100.00
	WordPress	56.45	39.39	40.00	40.87	53.04	**62.34**	22.94	58.24
Wappalyzer	Drupal	100.00	**73.96**	**73.96**	70.41	71.60	71.60	-2.37	-3.20
	Joomla	100.00	60.17	58.44	61.90	**66.23**	**66.23**	6.06	10.07
	Mediawiki	87.27	90.53	90.53	90.53	90.53	**91.58**	1.05	1.16
	WordPress	100.00	67.53	66.67	68.83	68.83	**70.13**	2.60	3.85
WhatWeb	Drupal	99.67	58.58	57.40	58.58	**59.76**	59.17	0.59	1.01
	Joomla	29.78	18.18	19.91	18.18	19.48	**21.21**	3.03	16.67
	Mediawiki	84.84	85.26	85.26	85.26	89.47	**90.53**	5.26	6.17
	WordPress	100.00	65.37	64.94	65.37	69.26	**69.70**	4.33	6.62

is more sensitive to content changes than dynamic content fingerprinting. By applying the middleware improvements to these tools, we are able to bring the content closer to the expected out-of-the-box state. Meanwhile, dynamic content fingerprinting techniques can either find the particular strings they are searching for, or they cannot. This makes it more difficult for application-agnostic network middleware like WASABO to tease out greater performance from these tools.

6 Discussion and Future Work

6.1 Key Takeaways

• **Benefits of Web Application Containerization:** Our comprehensive analysis of web application fingerprinting techniques would not have been possible had it not been for the development of our web application sandbox framework, WASABO. This Docker-based framework allows for the automated deployment of any version of a supported web application using only a handful of Docker containers, and the improvement of *any* web application fingerprinting tool through the use of its network middleware module. While we utilized WASABO for evaluating the performance of web application fingerprinting tools, we emphasize that it is not limited to only collecting such data. Rather, the modularity of WASABO allows for test scripts to be created to serve any web application testing purpose, such as website response time analysis or web application extension compatibility tests. We plan to release WASABO to the community, upon publication of this paper.

• **Improved Web Application Fingerprinting:** Our results have demonstrated the limitations of current web application fingerprinting techniques. The over-reliance on content remaining unchanged between the development and training of tools to their deployment on the web, reduces their robustness. Therefore, we emphasize the need of creating improved web application fingerprinting techniques which utilize the functional aspects of application content to conduct fingerprinting, rather than all content in general. For instance, rather than utilizing the hashes of JavaScript files as a whole, one could normalize these files by removing all whitespace and comments prior to computing their hash values. Similarly, utilizing the structure and style of dynamic HTML web pages could prove more robust [45]. As mentioned previously, the works of Marquardt et al. [48] and Dresen et al. [43] provide a great starting-point for the development of better performing tools. Future work could explore the ideal amount of normalization that increases the robustness of these tools without increasing the rate of version collisions. Until then, we have shown that utilization of a network middleware such as ours can immediately improve the performance of current web application fingerprinting tools without making any changes to each individual tool.

• **Fingerprinting Defenses:** Current fingerprinting techniques are highly reliant on the assumption that website administrators will not modify or remove access to web application content provided "out-of-the-box." As such, minor changes to the state of this content is enough to severely degrade the ability for these tools to accurately identify a web application. Therefore, it is trivial for administrators to prevent fingerprinting of their web application deployments by modeling the changes we demonstrated in our experiments. Similar to how we improved web application fingerprinting performance, reverse proxy server middleware can be deployed at the egress of networks to apply web-application-agnostic measures to all outgoing traffic, preventing fingerprinting by attackers on the

Internet, while also allowing for the continual internal use of fingerprinting tools to discover and patch outdated systems.

6.2 Limitations

While WASABO allows for automated testing of a large number of web application versions, providing compatibility for each web application requires manual effort to map out the installation steps required for each web application version. It is for this reason that we limit the number of web applications included in our study. However, we emphasize that the chosen web applications make up a large portion of the overall web application marketshare, representing a wide-range of functionalities and information sources that are common to all web applications. Likewise, we acknowledge that the web application fingerprinting tools chosen for this study do not constitute an exhaustive list of all fingerprinting tools. Rather, we focus our efforts on evaluating fingerprinting tools that allow for the identification of a wide-range of web applications, rather than tools specialized for a particular web application that could overly rely on specific artifacts of their targets, as opposed to generalizable fingerprinting techniques.

7 Related Work

To the best of our knowledge, this work is the first to systematically explore the performance of current web application fingerprinting techniques. In addition to works that have introduced the fingerprinting tools which we study in this paper [53], prior work has utilized static and dynamic fingerprinting techniques to create web application fingerprinting and vulnerability scanning tools [31, 37, 46, 47, 49].

Marquardt et al. propose a web application fingerprinting approach that builds upon those studied in this work by using only the HTML and associated assets of the root URL of each website, decreasing the number of required requests [48]. In addition to utilizing the hash values of CSS and JavaScript files included in the homepage of each site, the authors also utilize XPath profiling to parse out the HTML-tree structure of the website homepage Generally, the reported performance of this technique is equal to, or lower than the tools we study in this work.

Dresen et al. evaluate to what extent attackers outside of critical networks can identify services (e.g. the ones running on IoT devices) in these networks [43]. Because these devices cannot be directly probed, the authors have to use a victim user's browser to probe for resources on the user's local network, as much as the Same-Origin Policy allows them to. Because of that constraint, the tool that the authors develop (CORSICA) can either check for the presence of files (e.g. is a specific PNG of a specific IoT device present on an internal IP address) or, for limited cases, check for the effects of other files in the user's browser (e.g. try to load a JS library and then check for the existence of global variables and functions that should be present if that library was indeed loaded and executed). CORSICA's file-existence checks are a weaker form of static content fingerprinting (where the content is hashed by tools such as BlindElephant) compared to the tools we evaluated in WASABO. Contrastingly, their JavaScript checks are on-par with dynamic-content fingerprinting (such as the one used by Wappalyzer and WhatWeb).

In addition to utilizing the content produced by web applications to identify them [50], prior work has also presented methods to utilize side-channel data to fingerprint web application platforms [40, 41]. Schmitt et al. demonstrated that in addition to fingerprinting the web application name and version powering a website, attackers can also fingerprint web application firewall (WAF) rules [52].

Prior work has also explored the use of network middleware to improve the performance of security-focused scanning tools. Particularly, Drakonakis et al. recently introduce ReScan, a network middleware used to increase the effectiveness of popular web vulnerability scanners [42]. Unlike WASABO, ReScan takes a more active role in improving the performance of vulnerability scanners by seeking out relevant information regarding a web application's attributes and appending this information to responses so the underlying scanner can "discover" them. Conversely, the network transformation scripts we developed for this work passively normalize web application content to reduce the effects of real world customizations.

8 Conclusion

In this paper, we studied modern web application fingerprinting techniques, measuring the performance of six popular tools utilizing information from both static and dynamic web application content, against 1,360 releases of five of the most popular web applications in use today. We designed and developed WASABO, a web application sandbox framework to analyze the identification accuracy of each fingerprinting tool offline in ideal conditions, in which all possible web application content is available. While 94.8% of web application releases are identified by at least one tool, many are unable to produce a single version prediction for any release, instead returning many disparate versions, sometimes distributed over 7 years apart. Moreover, we find 82 instances in which a web application release which contains a severe vulnerability is labeled as a non-vulnerable version.

Next to evaluating web-application fingerprinting tools in ideal conditions, we measured their robustness to the organic noise added to web application deployments in the wild. We find that, by default, these tools struggle to determine the web application present on a host, with accuracy decreasing 20%-80% compared to our offline results. To understand the reasons for this performance degradation and to remedy it, we designed network middleware modules for WASABO capable of transparently applying a series of traffic modifications. In doing so, we were able to increase the real-world performance of the studied

fingerprinting tools by up to 22.9%. Crucially, WASABO provides these improvements in a *scanner-agnostic* manner, meaning any current or future web application fingerprinting tool immediately benefits from them without modification.

Availability. We built WASABO to automate the process of auditing web application fingerprinting tools against thousands of versions of popular web applications, each with distinct prerequisite technologies and installation procedures. To assist the research community in further understanding the fingerprintability of web applications and inspire additional research, we are making our system publicly available [1].

Acknowledgements

We thank the anonymous reviewers for their helpful feedback. This work was supported by the Office of Naval Research (ONR) under grant N00014-24-1-2193 as well as by the National Science Foundation (NSF) under grants CNS-2211575 and CNS-1941617.

References

[1] Smudged fingerprints project website. https://pragseclab.github.io/smudged-fingerprints/.

[2] undetected-chromedriver. https://github.com/ultrafunkamsterdam/undetected-chromedriver.

[3] Using cloudflare auto minify. https://developers.cloudflare.com/support/speed/optimization-file-size/using-cloudflare-auto-minify/.

[4] Blindelephant. https://github.com/lokifer/BlindElephant, 2016.

[5] Equifax had patch 2 months before hack and didn't install it, security group says. https://www.usatoday.com/story/money/2017/09/14/equifax-identity-theft-hackers-apache-struts/665100001/, 2017.

[6] Metasploit joomla version scanner. https://www.rapid7.com/db/modules/auxiliary/scanner/http/joomla_version/, 2017.

[7] Metasploit wordpress scanner. https://www.rapid7.com/db/modules/auxiliary/scanner/http/wordpress_scanner/, 2018.

[8] Versioninferrer. https://github.com/wichmannpas/VersionInferrer, 2020.

[9] 1.6 million wordpress sites hit with 13.7 million attacks in 36 hours from 16,000 ips. https://www.wordfence.com/blog/2021/12/massive-wordpress-attack-campaign/, 2021.

[10] Whatweb. https://github.com/urbanadventurer/WhatWeb, 2021.

[11] Whatweb drupal plugin. https://github.com/urbanadventurer/WhatWeb/blob/master/plugins/drupal.rb, 2021.

[12] Whatweb joomla plugin. https://github.com/urbanadventurer/WhatWeb/blob/master/plugins/joomla.rb, 2021.

[13] Whatweb wordpress plugin. https://github.com/urbanadventurer/WhatWeb/blob/master/plugins/wordpress.rb, 2021.

[14] Apache web server. https://httpd.apache.org/, 2022.

[15] builtwith.com. https://builtwith.com, 2022.

[16] China-linked group hacks cow monitoring app to spy on six states. https://www.forbes.com/sites/leemathews/2022/03/14/china-linked-group-hacks-cow-monitoring-app-to-spy-on-six-states, 2022.

[17] Chinese hackers target taiwanese financial institutions with a new stealthy backdoor. https://thehackernews.com/2022/02/chinese-hackers-target-taiwanese.html, 2022.

[18] cvedetails.com. https://cvedetails.com, 2022.

[19] Docker. https://docker.io, 2022.

[20] Drupal. https://drupal.org, 2022.

[21] Drupal website showcase. https://www.drupal.org/case-studies, 2022.

[22] Joomla. https://joomla.org, 2022.

[23] Joomla website showcase. https://showcase.joomla.org, 2022.

[24] List of wordpress security vulnerabilities. https://www.cvedetails.com/vulnerability-list/vendor_id-2337/product_id-4096/Wordpress-Wordpress.html, 2022.

[25] Mediawiki. https://mediawiki.net, 2022.

[26] Mediawiki manual:short url. https://www.mediawiki.org/wiki/Manual:Short_URL, 2022.

[27] Mediawiki website showcase. https://www.mediawiki.org/wiki/Sites_using_MediaWiki/en, 2022.

[28] mitmproxy. https://mitmproxy.org/, 2022.

[29] mitmproxy - addons. https://docs.mitmproxy.org/stable/addons-overview/, 2022.

[30] phpmyadmin. https://phpmyadmin.net, 2022.

[31] Plecost. https://github.com/iniqua/plecost, 2022.

[32] Usage statistics and market share of wordpress. https://w3techs.com/technologies/details/cm-wordpress, 2022.

[33] Usage statistics of content management systems. https://w3techs.com/technologies/overview/content_management, 2022.

[34] Wappalyzer. https://www.wappalyzer.com/, 2022.

[35] Wordpress. https://wordpress.org, 2022.

[36] Wordpress website showcase. https://wordpress.org/showcase/, 2022.

[37] Wpscan. https://wpscan.com/wordpress-security-scanner, 2022.

[38] Babak Amin Azad, Oleksii Starov, Pierre Laperdrix, and Nick Nikiforakis. Web runner 2049: Evaluating third-party anti-bot services. In *Detection of Intrusions and Malware, and Vulnerability Assessment: 17th International Conference, DIMVA 2020, Lisbon, Portugal, June 24–26, 2020, Proceedings 17*, pages 135–159. Springer, 2020.

[39] Ajay Sudhir Bale, Naveen Ghorpade, S Rohith, S Kamalesh, R Rohith, and BS Rohan. Web scraping approaches and their performance on modern websites. In *2022 3rd International Conference on Electronics and Sustainable Communication Systems (ICESC)*, pages 956–959. IEEE, 2022.

[40] Dominique Bongard. Fingerprinting web application platforms by variations in png implementations, 2014.

[41] Hyunseok Chang, Murali Kodialam, TV Lakshman, and Sarit Mukherjee. Microservice fingerprinting and classification using machine learning. In *2019 IEEE 27th International Conference on Network Protocols (ICNP)*, pages 1–11. IEEE, 2019.

[42] Kostas Drakonakis, Sotiris Ioannidis, and Jason Polakis. Rescan: A middleware framework for realistic and robust black-box web application scanning. In *2023 Network and Distributed Systems Symposium (NDSS)*, 2023.

[43] Christian Dresen, Fabian Ising, Damian Poddebniak, Tobias Kappert, Thorsten Holz, and Sebastian Schinzel. Corsica: Cross-origin web service identification. In *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security*, pages 409–419, 2020.

[44] Steven Englehardt and Arvind Narayanan. Online tracking: A 1-million-site measurement and analysis. In *Proceedings of ACM CCS 2016*, 2016.

[45] Thamme Gowda and Chris A Mattmann. Clustering web pages based on structure and style similarity (application paper). In *2016 IEEE 17th International conference on information reuse and integration (IRI)*, pages 175–180. IEEE, 2016.

[46] Hao He, Lulu Chen, and Wenpu Guo. Research on web application vulnerability scanning system based on fingerprint feature. 2017.

[47] Raghavendra Karthik, Sowmya Kamath, et al. W3-scrape-a windows based reconnaissance tool for web application fingerprinting. *arXiv preprint arXiv:1306.6839*, 2013.

[48] Fabian Marquardt and Lennart Buhl. Déjà vu? client-side fingerprinting and version detection of web application software. In *2021 IEEE 46th Conference on Local Computer Networks (LCN)*, pages 81–89. IEEE, 2021.

[49] Fabian Marquardt and Lennart Buhl. Large scale monitoring of web application software distribution to measure threat response behavior. *Electronic Communications of the EASST*, 80, 2021.

[50] Tony Nasr, Sadegh Torabi, Elias Bou-Harb, Claude Fachkha, and Chadi Assi. Chargeprint: A framework for internet-scale discovery and security analysis of ev charging management systems.

[51] Studies using OpenWPM. https://openwpm.readthedocs.io/en/latest/Papers.html, 2023.

[52] Isabell Schmitt and Sebastian Schinzel. Waffle: Fingerprinting filter rules of web application firewalls. In *WOOT*, pages 34–40, 2012.

[53] Pascal Wichmann. Automated inference of web software packages and their versions. 2018.

A Appendix

A.1 Ethical Considerations

In this paper, we evaluated web-application fingerprinting tools against off-the-shelf web applications in a laboratory setting as well as against real web applications deployed in the wild. Our decision to evaluate these tools against real-world web applications was based on our understanding that some of their fingerprinting vectors were brittle (such as the hashing of static files like JavaScript and CSS) and therefore highly susceptible to small changes at the server side. Our results

Table 5: *List of all web applications we audit in this work.*

Web App	Releases Studied		
	#	Earliest	Latest
WordPress [35]	558	3.5.1	6.1.1
Drupal [20]	304	6.28	10.0.2
Joomla [22]	141	2.5.10	4.2.7
MediaWiki [25]	165	1.19.10	1.39.1
phpMyAdmin [30]	192	3.5.6	5.2.1

confirmed these suspicions showing a drop-off in the tools' accuracy by as much as 80% compared to our lab experiments.

We are confident that our in-the-wild experiments were conducted ethically and caused no issues for the scanned web applications. First, all evaluated tools are not attempting to exploit any vulnerabilities on web applications. Instead, they request content that is already there and either parse that content searching for version information (such as in the case of dynamic-content fingerprinting) or hash that content at the client side and perform set-membership tests, associating hashes with known web-application versions. The tools perform a small number of HTTP GET requests (no POST or other state-changing requests) requesting commonly-accessed files by regular users, such as, the main page of a web application, JavaScript files, and CSS files. Moreover, whenever available, we always chose the tool configuration that emitted the fewest HTTP requests when scanning real-world web applications.

The transformations added by our WASABO system are also non-intrusive in nature. Namely, WASABO adds a random cache-breaking parameter at the end of the requests that the tools make, finds alternative hosting paths for server-side resources, and performs the requests over a real browser to avoid anti-bot mechanisms. None of these changes endanger the server side of the scanned web applications in any way. While we do have to try and evade any anti-bot mechanism that may be available at the server side (via the use of a real browser) that evasion is not harmful in our setting (i.e. where a handful of pages and static resources are requested from the web application). The need to avoid anti-bot technologies for ethically-conducted research is well established by prior work. For example, the OpenWPM web-privacy measurement framework by Engelhardt and Narayanan [44] has been used by 75 different studies (according to statistics tracked by the authors [51]) and uses specific anti-bot mechanisms (such as simulated mouse movements and page scrolls) to try and evade detection by the evaluated web applications.

Without these in-the-wild measurements, we would have greatly overestimated the accuracy of the evaluated fingerprinting tools thereby misleading penetration testers and giving a false sense of security to administrators and organizations who rely upon them for discovering software that needs to be updated.

Table 6: *CVEs studied to uncover cases in which fingerprinting tools label vulnerable versions of web applications as non-vulnerable versions. Version cells with a "*" indicate the vulnerability is believed to have affected all versions of the web application, until patched.*

Webapp	CVE ID	CVSS	Earliest Version	Patched Version
WordPress	CVE-2013-4338	7.5	*	3.6.1
	CVE-2013-4339	7.5	*	3.6.1
	CVE-2014-5203	7.5	3.9	3.9.2
	CVE-2015-2213	7.5	*	4.2.4
	CVE-2016-10033	7.5	*	5.2.18
	CVE-2016-10045	7.5	*	5.2.20
	CVE-2017-5611	7.5	*	4.7.2
	CVE-2017-14723	7.5	*	4.8.2
	CVE-2017-16510	7.5	*	4.8.3
	CVE-2018-20148	7.5	*	4.9.9
Drupal	CVE-2020-13664	9.3	8.8	8.8.8
	CVE-2020-13664	9.3	9	9.0.1
	CVE-2016-3168	8.5	6	6.38
	CVE-2014-1475	7.5	6	6.3
	CVE-2014-3704	7.5	7	7.32
	CVE-2015-6659	7.5	7	7.39
	CVE-2017-6920	7.5	8	8.3.4
	CVE-2017-6925	7.5	8	8.3.7
	CVE-2018-7600	7.5	*	7.58
	CVE-2019-6339	7.5	8.5.0	8.5.9
Joomla	CVE-2013-1453	7.5	3.0.0	3.0.3
	CVE-2014-6632	7.5	3.3.0	3.3.5
	CVE-2014-7228	7.5	2.5.4	2.5.26
	CVE-2014-7981	7.5	3.2.0	3.2.3
	CVE-2014-7984	7.5	2.5.0	2.5.19
	CVE-2015-7297	7.5	3.2.0	3.4.4
	CVE-2015-7857	7.5	3.2.0	3.4.5
	CVE-2016-9081	7.5	3.4.4	3.6.4
	CVE-2016-9836	7.5	*	3.6.5
	CVE-2016-10033	7.5	*	5.2.18
Mediawiki	CVE-2013-4304	7.5	1.21.0	1.21.2
	CVE-2013-4571	7.5	1.22.0	1.22.1
	CVE-2013-6453	7.5	1.20.0	1.21.4
	CVE-2014-9277	7.5	1.23.0	1.23.7
	CVE-2014-9487	7.5	1.24.0	1.24.1
	CVE-2015-6728	7.5	1.25.0	1.25.2
	CVE-2017-8809	7.5	1.29.0	1.29.2
	CVE-2019-12468	7.5	1.27.0	1.32.2
	CVE-2020-10534	7.5	*	1.34.1
	CVE-2021-31556	7.5	*	1.35.3
phpMyAdmin	CVE-2016-5703	7.5	4.4.0	4.4.15.7
	CVE-2016-5734	7.5	4.0.0	4.0.10.16
	CVE-2016-6620	7.5	4.6.0	4.6.4
	CVE-2016-9849	7.5	4.0.0	4.0.10.18
	CVE-2016-9865	7.5	4.6.0	4.6.5
	CVE-2019-6798	7.5	*	4.8.5
	CVE-2019-11768	7.5	*	4.9.0.1
	CVE-2019-18622	7.5	*	4.9.2
	CVE-2020-26935	7.5	5	5.0.3
	CVE-2020-26935	7.5	*	4.9.6