Our (in)Secure Web: Understanding Update Behavior of Websites and Its Impact on Security

Nurullah Demir[1], Tobias Urban[1], Kevin Wittek[1,2], and Norbert Pohlmann[1]

[1] Institute for Internet Security—if(is)
Westphalian University of Applied Sciences Gelsenkirchen
{*lastname*}@internet-sicherheit.de
[2] RWTH Aachen University

Abstract. Software updates take an essential role in keeping IT environments secure. If service providers delay or do not install updates, it can cause unwanted security implications for their environments. This paper conducts a large-scale measurement study of the update behavior of websites and their utilized software stacks. Across 18 months, we analyze over 5.6M websites and 246 distinct client- and server-side software distributions. We found that almost all analyzed sites use outdated software. To understand the possible security implications of outdated software, we analyze the potential vulnerabilities that affect the utilized software. We show that software components are getting older and more vulnerable because they are not updated. We find that 95 % of the analyzed websites use at least one product for which a vulnerability existed.

Keywords: updates · vulnerabilities · security · web measurement.

1 Introduction

Nowadays, we use the Web for various tasks and services (e.g., talking to our friends, sharing ideas, to be entertained, or to work). Naturally, these services process a lot of personal and valuable data, which needs to be protected. Therefore, web services need to be hardened against adversaries, for example, due to imperfections of software. An essential role in every application's security concept is the updating process of the used components [9]. Not updating software might have severe security implications. For example, the infamous *Equifax* data breach that affected 143 million people was possible because the company used software with a known vulnerability that has already been fixed in a newer version [26].

However, keeping software up to date is not always easy and, from the security perspective, not always necessary (i.e., not every update fixes a security issue). Modern applications require a variety of different technologies (e.g., libraries, web servers, databases, etc.) to operate. Updating one of these technologies might have unforeseeable effects and, therefore, updates might create potentially high overhead (e.g., if an update removes support of a used feature). More specifically, service providers might object to install an update because they do not directly

profit from the new features (e.g., changes in an unused module). Hence, it is reasonable not always to install every available update (e.g., to ensure stability).

In this work, we show that this challenge can have grave implications. To understand how up to date the utilized software on the Web is and to understand its possible security implications, we conduct a large-scale measurement. Previous work also analyzed update behavior on the Web (e.g., [19,23]) but – to the best of our knowledge – our measurement is more comprehensive than the previous studies. While we analyze over 5.6M sites and nearly 250 software (SW) products, other work in this field often only analyzed one specific type of software or a small subset. Therefore, our results are more generalizable and provide a better overview of the scale of the problem.

To summarize, we make the following contributions:

1. We conduct a large-scale measurement that evaluates 246 software products used on 5.6M websites over a period of 18 months, to determine update behavior and security impact of not updating.
2. We show that 96 % of the analyzed websites run outdated software, which is often more than four years old and is getting even older since no update is applied.
3. We show that a vast majority of the analyzed websites (95 %) use software for which vulnerabilities have been reported, and the number of vulnerable websites is increasing over time.

2 Background

In this section, we discuss the principles of how web applications work and how known vulnerabilities are publicly managed, both necessary to appreciate our work.

2.1 Preliminaries

We start by introducing key terminology. In this work, we use the term *site* (or *website*) to describe a registerable domain, sometimes referred to as *eTLD+1* ("extended Top Level Domain plus one"). Examples for sites are `foo.com` and `bar.co.nz`. Each site may have several *subdomains* (e.g., `news.foo.com` and `sport.foo.com`). Following the definition of RFC 6454 [1], we call the tuple of protocol (e.g., HTTPS), subdomain (or hostname), and port *origin*. This distinction is important since the well-known security concept *Same-Origin Policy* (SOP) guarantees that pages of different origins cannot access each other. We use the term *page* (or *webpage*) to describe a single HTML file (e.g., a webpage hosted at a specific URL).

2.2 Web Technologies & Updating

To implement modern web applications, service providers rely on a diverse set of server-side (e.g., PHP or MySQL) and client-side technologies (e.g., HTML

or JavaScript). This combination of different technologies often results in a very complex and dynamic architecture, not always under full control of the service provider (e.g., usage of third parties [8]). Furthermore, the update frequency of web technologies is higher compared to desktop software [20]. Web applications are commonly composed of different modules that rely on each other to perform a given task. Hence, one vulnerability in any of these modules might undermine the security of the entire web app, depending on the severity of the vulnerability. Once a vulnerability of an application is publicly known or privately reported to the developers (see also Section 2.3), the provider of that application (hopefully) provides an update to fix it. Therefore, service providers need to check the availability of updates of the used components and their dependencies and transitive dependencies regularly. However, it should be noted that not all updates fix security issues, and, therefore, it is not necessary or desired (e.g., for stability reasons) to install all updates right away.

2.3 Common Vulnerabilities and Exposures

Once vulnerabilities in software systems are discovered, reported to a vendor, or shared with the internet community publicly, they are published in vulnerability database platforms (e.g., in the *National Vulnerability Database* (NVD)). The NVD utilizes the standardized *Common Vulnerabilities and Exposures* (CVE) data format and enriches this data. Each CVE entry is provided in a machine-readable format and contains details regarding the vulnerability (e.g., vulnerability type, vulnerability severity, affected software, and version(s)). The primary purpose of each CVE entry is to determine which software is affected by a vulnerability and helps to estimate its consequences. Each entry in the NVD database is composed of several data fields, of which we now describe the one most important for our work. In the NVD database, the field `ID` of a CVE entry uniquely identifies the entry and also states the year when the vulnerability was made public, followed by a sequence number (e.g., `CVE-2020-2883`), the field `CVE_data_timestamp` indicates when the CVE entry was created. Furthermore, each CVE entry also includes a list of known software configurations that are affected by the vulnerability (field `configurations`), formally known as *Common Platform Enumeration* (CPE). CPE defines a naming scheme to identify products by combining, amongst other values, the vendor, product name, and version. For example the CPE (in version 2.3) `cpe:2.3:a:nodejs:node.js:4.0.0:[...]` identifies the product `node.js` provided by the vendor *nodejs* in version 4.0.0. Furthermore, the `configurations` field lists all conditions under which the given vulnerability can be exploited (e.g., combination of used products). Finally, the field `impact` describes the practical implications of the vulnerability (e.g., a description of the attack vector) and holds a score, the *Common Vulnerability Scoring System* (CVSS), ranging from 0 to 10, which indicates the severity of the CVE (with ten being the most severe). Again, it is worth noting that it not definite that if one uses a software product – for which a vulnerability exists – that it is exploitable by an attacker. For example, if an SQL-Injection is possible via the comment function of a blog,

it can only be exploited if the comment function is enabled. Thus, our results can be seen as an upper bound.

3 Method

In this work, we want to assess the update behavior of web applications, measure if they use outdated software, and test the security implications of using the vulnerable software. To accomplish that, we collect the used modules (software and version) of the websites present in the *HTTPArchive* [4] over a period of 18 month, extract known vulnerabilities from the *National Vulnerability Database* database, and map them against the used software versions of the analyzed sites.

Identifying Used Software To assess the update behavior of websites, we need to identify the software versions of the software in use. To do so, we utilized data provided by *HTTPArchive* [6], which includes all identified technologies used by a website. HTTPArchive crawls the landing page of millions of popular origins (mobile and desktop) based on the *Chrome User Experience Report* (CrUX) [3] every month, since January 2019. In CrUX, Google provides publicly metrics like load, interaction, layout stability of the websites that are visited by the Chrome web browser users on a monthly basis. This real-world dataset includes popular and unpopular websites [5]. In our study, we analyze all websites provided in HTTPArchive. Hence, we can use 18 data points in our measurement (M#1 – M#18). The data provided by HTTPArchive includes, among other data: (1) the date of the crawl, (2) the visited origins, and (3) identified technologies (software including its version). HTTPArchive uses *Wappalyzer* [24] to identify the used software, which uses different information provided by a site to infer the user version and technology stack. In order to make version changes comparable, we converted the provided data to the *semantic versioning* (SemVer) standard (i.e., `MAJOR.MINOR.PATCH`) [14] and validate also the version information from HTTPArchive as well as from NVD and check if provided versions are in a valid SemVer format. This unification allows us to map the observed versions of the known vulnerabilities. If we find an incomplete SemVer string, we extended it with ".0" until it fits the format.

Identifying Vulnerable Software To better understand the security impact of updates, we map the software used by an origin to publicly known vulnerabilities. We collect the vulnerabilities from the *National Vulnerability Database* (NVD) [3]. Each entry in the NVD holds various information, but only three are essential to our study: (1) the date on which it was published, (2) a list of systems that are affected by it, and (3) the impact metrics how it can be exploited and its severity. In this work, we only focus on vectors that can be exploited by a remote network adversary.

[3] We used the database published on 04/07/20.

3.1 Dataset Preparation and Enrichment

Here, we describe the steps taken to enrich our dataset to make it more reliable.

Release History To get a firm understanding of the update behaviour of websites, it is inevitable to know the dates on which different versions of a software were released ("release history"). To construct the list of release dates of each software product, we used *GitHub-API* for the official repositories, on *GitHub*, of the products and extracted the date on which a new version was pushed to it and store the corresponding SemVer. If a product did not provide an open repository on *GitHub*, we manually collected the official release dates from the product's official project webpages, if it's published.

Dataset Preparation Since the Web is constantly evolving and Web measurements tend to be (strongly) impacted by noise, we only analyzed software products on a site for which we found version numbers in at least four consecutive measurements. Furthermore, we dropped all records with polluted data (e.g., blank, invalid versions, duplicates, dummy data) from our dataset. Finally, in order to make a valid match between CVE entries and software in our dataset, we manually assigned each software in our dataset their CPE (naming scheme) using the *CPE Dictionary* [12] provided by NVD.

3.2 Analyzing Updating Behavior & Security Implications

In this section, we describe how we measure update behavior and identify vulnerable websites.

Updating Behavior To understand update behavior in our dataset, one needs to measure the deployed software's version changes over time. Utilizing the release dates of each software product, we know, at each measurement point in our dataset, whether a site/origin deploys the latest software version of a product or if it should be updated. If we found that an outdated product is used, we check if it was updated in the subsequent measurements (i.e., if the SemVer increases). This approach allows us to test if a product is updated after all and to check how long this process took. In our analysis, we call an increasing SemVer an *update* and decreasing version number a *downgrade*. In this analysis, we compare the `MINOR` and `PATCH` part of a product's SemVer, utilizing the release dates of each version, and not the `MAJOR` section because service providers might not use the latest major release due to significant migration overhead. For example, we would consider that an origin is "up to date" if it runs version 1.1.0 of a product even if version 2.1.0 (major release change) is available. However, if version 1.1.1 would be available, we consider it "out of date".

Identifying Vulnerable Websites One way to measure the impact of an update on the security of a site is to test if more or less vulnerabilities exists for the new version, in contrast to the old version. To identify vulnerable software on

a website, we retrieve the relevant CVEs for the identified software and then check if it is defined in these CVE entries – with consideration of *versionStart[Excluding/Including]* and *versionEnd[Excluding/Including]* settings. We map a vulnerability to a crawled origin if and only if (1) it uses a software for which a vulnerability exists and (2) if it was published before the crawl was conducted. Utilizing the *Common Vulnerability Scoring System* (CVSS) of each vulnerability, we can also assess the theoretical gain in security.

4 Results

After describing our approach to analyze the update behavior of websites and its possible security impact on websites, this section introduces the large-scale measurement results. Overall, we observed 8.315.260 origins on 5.655.939 distinct domains using 342 distinct software products. After filtering, we were left with 8.205.923 origins (99 %) on 5.604.657 domains (99 %) using 246 (72 %) software products. We collected 31.909 releases for 246 software products. Furthermore, we collected 147.312 vulnerabilities of which 2.793 (2 %) match to at least one identified product. Overall, we found an exploitable vulnerability for 148 (60 %) of the analyzed software products. Note that products with no public release history are excluded from analyzing update behavior and security analysis if they don't have a known vulnerability. Note also we have full access to all the segments of the `MAJOR.MINOR.PATCH` for 98.5% of our data. In total, we identified 12.062.618 software updates across all measurements. Table 1 provides an overview of all evaluated records of each measurement run.

4.1 Update Behavior on the Web

In the following, we analyze the impact of adoption of releases on the Web on website level and from software perspective.

Update Behavior of Websites The first step to understand the update behavior of websites is to analyze the fraction of used software products that are fully patched, according to our definitions. Remember that we assume that a software product should be updated if a newer minor version or patch is available (i.e., we exclude the major version (see Section 3.2)). In our dataset, we identified a median of 3 (min: 1, max: 17, avg: 3.37) evaluable software products for each website. Overall, we identified that across all measurement points, on average, 94 % of all observed websites were *not* fully updated (i.e., at least for one software product exists a newer version). Only 6 % of the observed sites used only up to date software while 47 % entirely relied on outdated software types. The mean fraction of out of date software products is 74 % for each observed website across our measurement points. These numbers show that websites often utilize outdated software. While at domain granularity, almost all analyzed sites use outdated software, it is interesting to analyze if subdomains show different update behavior. Figure 1 compares the fraction of up to date software utilized

M.	Date	#Sites	#Origins	#Products	#dist. Ver.	#Updates	#Vuln.
M#1	01/19	2.5M	3.4M	208	15,436	—	2,201
M#2	02/19	2.3M	3.1M	204	15,178	0.4M	2,224
M#3	03/19	2.3M	3.1M	205	15,390	0.5M	2,235
M#4	04/19	2.7M	3.5M	205	16,145	0.6M	2,291
M#5	05/19	2.8M	3.6M	216	16,741	0.4M	2,298
M#6	06/19	2.8M	3.6M	217	17,013	0.7M	2,310
M#7	07/19	3.0M	3.9M	215	17,438	0.6M	2,286
M#8	08/19	3.0M	3.9M	215	17,474	0.5M	2,316
M#9	09/19	3.0M	3.9M	215	17,682	1.0M	2,390
M#10	10/19	3.0M	3.8M	217	17,873	0.8M	2,424
M#11	11/19	3.0M	3.8M	217	17,958	1.0M	2,468
M#12	12/19	3.0M	3.8M	216	18,122	1.0M	2,478
M#13	01/20	2.9M	3.8M	217	18,173	0.8M	2,502
M#14	02/20	2.7M	3.4M	211	17,558	0.4M	2,526
M#15	03/20	3.1M	3.9M	217	18,558	0.4M	2,412
M#16	04/20	3.3M	4.2M	217	19,321	0.6M	2,467
M#17	05/20	3.1M	4.0M	220	19,353	0.8M	2,460
M#18	06/20	3.4M	4.4M	218	20,118	0.6M	2,475

Table 1. Overview of all measurement points.

on subdomains (e.g., *bar.foo.com*) against the root domains (e.g., *foo.com*), along our measurement points. In the figure, zero means that all software is up to date and one means that all software is outdated. Our data shows that most software products are not updated to the newest release, but it is still interesting to analyze the update cycles websites use in the field. On average, we observed 0.7M version changes between two measurement runs. 97 % of them were upgrades (i.e., the SemVer increased) and consequently 3 % were downgrades.

Update Behavior from a Software Perspective Previously, we have shown that websites tend to use outdated software. In the following, we take a closer look at the used software to get a better understanding if the type of used software has an impact on its update frequency. Across all measurements, the software used on the live systems is 44 months old (M#1: 40, M#18: 48), and the trend during the measurement is that it gets even older (18 days each month on average). To determine how the average age changes by software types, we measured the average age of the top ten used software types for all measurement points. These top ten account for 65 % of all analyzed software types. In Figure 2 we show the corresponding results. Our finding clarifies that client-side software (e.g., JavaScript Libraries) is older than server-side software (e.g., Web Servers). A closer observation of the releases SW shows that the server-side software has shorter release cycles than client-side software in the measured period (e.g., *nginx* has 18 and *jQuery* only has 6 releases). While the age of the software itself is not necessarily a problem per se, it is notable that the average number of months a utilized software is behind the latest patch is 48. The ANOVA test ($\alpha = 0.05$)

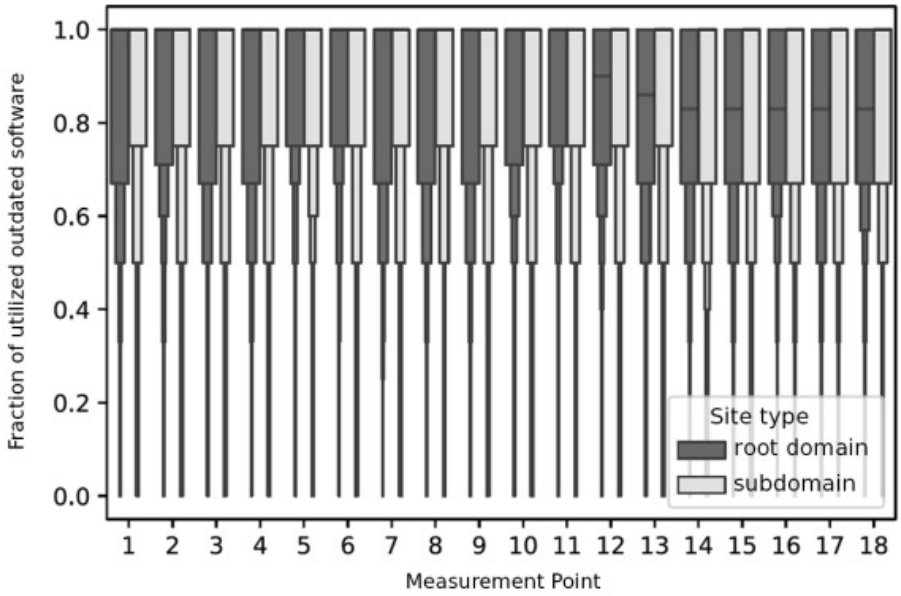

Fig. 1. Fractions of utilized outdated software products on the analyzed domains in comparison to their subdomains (1 = no product is up-to-date).

showed no statistical evidence that the popularity of a website, according to the *Tranco* list [11], has an impact on the age of the used software (i.e., popular and less popular websites use outdated software alike). Using software that is four years old might be troubling, given that on average 41 newer version exists, because the software might have severe security issues. We have shown that overall mostly outdated software is used. However, it is interesting to understand if this applies to all types of software alike or if specific products are updated more frequently.

Adoption of Software Releases To get a better understanding of the update behavior of websites, we observe the adoption of releases. We find that every month, on average, 67 % of the software used has a new release. However, our observations show that only a few service providers install the release promptly. We record that on average, only 7 % of available updates are processed (min: 4 %, max: 11 %). The mean time between two updates for any of the used software on one website is 3.5 months (SD: 5.4). To get a more in-depth understanding of the adoption of software releases, we measure it in a time span of 30 days after the release. Figure 3 shows the fraction of processed updates by websites in that time span for the top eight software types. The top eight types account for 60 % of all used software. In general, we see that PATCH level releases are processed most frequently. Furthermore, we observe that the adoption of release types differ based on the software types. E-Commerce software process PATCH

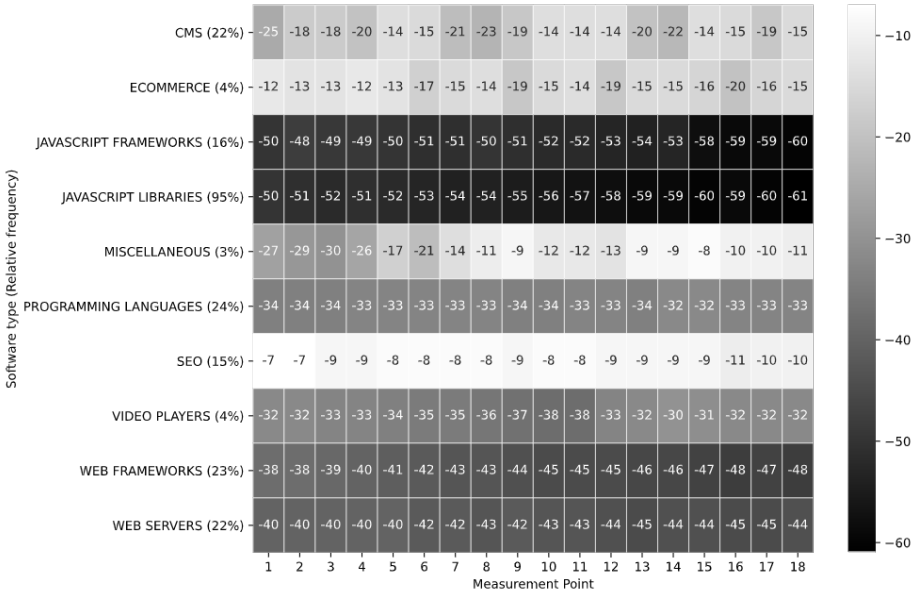

Fig. 2. Average age (in month) of top utilized 10 software types for all measurement points.

releases most frequently and search engine optimization software (SEO) MAJOR releases respectively. We assume that integrated automatic background updates play an important role why specific software types are updated. For example *WordPress* and *Shopware*, two popular content management systems, provide an auto update functionality [25,17].

Summary Based on our dataset, we have shown that the used software on the Web is often very old and not updated frequently. While differences in the update behavior between different types of software exist, the majority of all times is still not updated. However, the impact of this not-updating is not clear and needs more investigation.

4.2 Security Impact of *Not* Updating

Experts agree that updating is one of the most critical tasks one should do to harden a system or to avoid data leaks [15]. Therefore, we are interested in the security impact of the identified tend to use outdated software.

Vulnerability of Websites Towards understanding the threats that result from the usage of outdated software, we first analyze the scope of affected websites. On average, 94 % of the analyzed websites contain at least one potential vulnerable software, which was slightly increasing over the course of our measurements

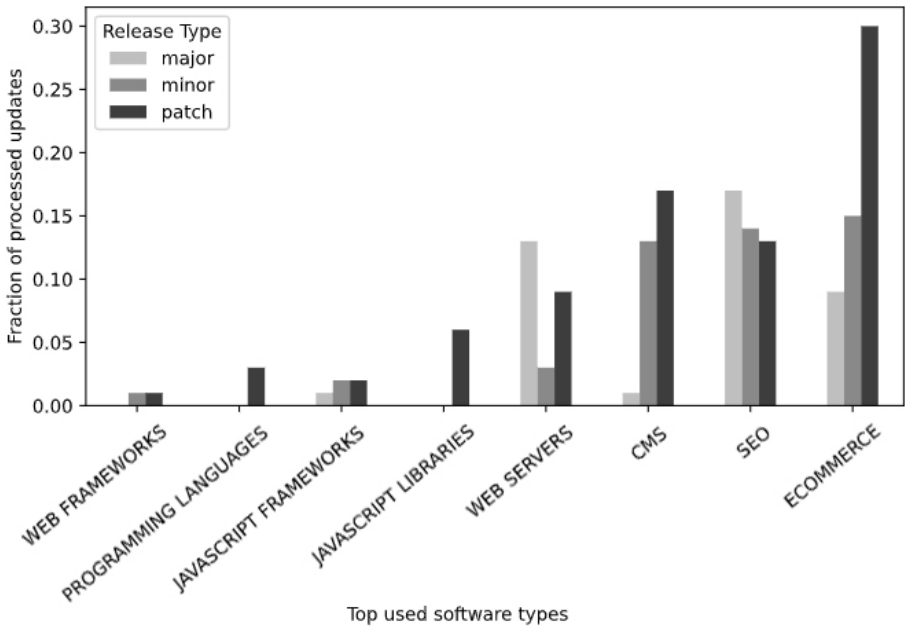

Fig. 3. Fraction of processing a new release for top used eight software types.

(M#1: 92 % to M#18: 95 %). We also record that each analyzed software has on average 8 vulnerabilities and that websites are affected, on average, by 29 (min: 0, max: 963). Our data shows that the number of exploitable vulnerabilities is decreasing over time for both per software (0.4 per month) and per websites (0.14 per month). Hence, overall the number of websites that have at least one vulnerability increases but the amount of vulnerabilities per site decreases.

Each vulnerability has a different security impact on a website, and, therefore, the number of identified vulnerabilities does not directly imply the severeness of them. The NVD assigns a score to each vulnerability to highlight its severeness (i.e., the CVSS score). Figure 4 shows the mean CVSS scores for the analyzes websites their rank. By inspecting the figure, one can see that less popular sites (the rank is higher) are affected by more severe vulnerabilities. The Spearman test ($\alpha = 0.05$) showed a statistical significant correlation between the rank and the mean CVSS score of the identified vulnerabilities (p-value < 0.007). Table 2, in Appendix A lists the most common vulnerabilities in our last measurement point (M#18). A stunning majority of websites (92 %) is theoretically vulnerable to *Cross-site Scripting* (XSS) attacks. In our dataset, *jQuery* is the software that is most often affected by a CVE (92 %). A list of the most prominent CVEs is given in Appendix C. Given the wide occurrence of vulnerabilities in our dataset, the question arises which threats websites and users actually face.

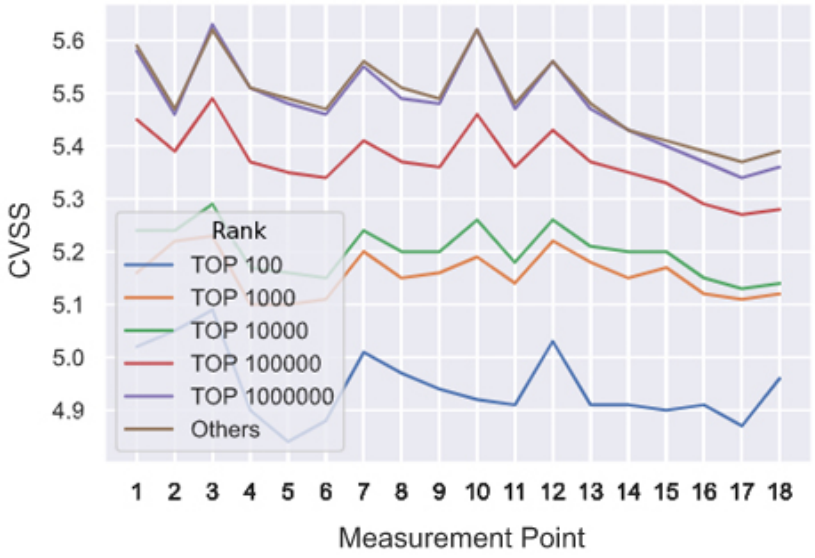

Fig. 4. AVG CVSS by popularity of websites. Vulnerability severity is significantly lower for high-ranked websites.

Analysis of Available Vulnerabilities Figure 5, shows the distribution of severity of identified vulnerabilities on websites based on the *Common Vulnerability Scoring System* (CVSS). Our results show that the number of websites with the most severe vulnerability (CVSS: 10) steadily decreases. The average number of vulnerable websites with a severity "HIGH" (CVSS: 7–10) is decreasing (M#1: 43 %, M#18: 39 %), while the number of vulnerable websites with "MEDIUM" (CVSS: 4–7) remains almost constant (M#1: 47 %, M#18: 49 %). For this analysis, we only used the most severe vulnerability for each website.

Given the result that the average age of used software depends on its type (see 2), we find that older software has more dangerous vulnerabilities. For example, the average CVSS/age of JavaScript-Frameworks was 4/50 in M#1 and 6/62 in M#18, while the score and age for programming languages go from 9/34 to 8/33. This confirms that older software *does* have more vulnerabilities and highlights the need for better update processes of websites. Furthermore, our analysis shows that performing updates has a significant impact on the security of software. The average value of CVSS for software for which an update is available is 6.4 ("MEDIUM"). However, after applying the update(s), the CVSS is lowered to 2.4 ("LOW").

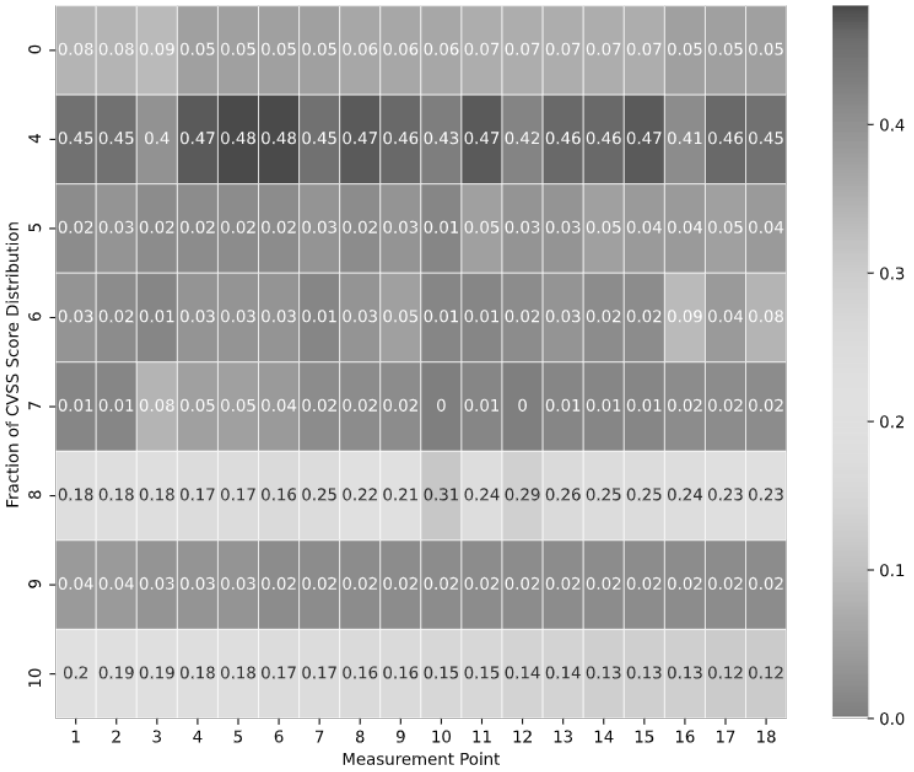

Fig. 5. Fraction of CVSS score distribution on websites for all measurement points (10 = Critical, 0=No Vulnerability).

5 Limitations

Although we have put a great effort while preparing our dataset, our study is impacted by certain limitations. Our approach comes with the limitation that, on the one hand, *HTTPArchive* only crawls landing pages and does not interact with the website, which might hide the complexity of an origin [21], and, on the other hand, *Wappalyzer* might not detect all used software for the website. Although NVD is one of the most popular vulnerability databases, there are some discussions around the accuracy of the data provided by NVD e.g., [13,16]. In our study, we assume that software utilized by a website is vulnerable if the NVD provides a CVE entry for it. For ethical reasons, we did not validate if successful exploitation of the CVE requires any interaction or enabled functions. We also don't examine any mechanism for the validity of CVE entries.

6 Related Work

To the best of our knowledge, our study is the first one that measures update behavior and security implications by evaluating all utilized server and client-side software on a website and by conducting multiple measurements. In the following, we discuss studies related to our research.

Update Behavior Update behavior of software has been previously studied. Tajalizadehkhoob et al. [19] measure the security state of software provided by hosting providers to understand the role of hosting providers for securing websites. Vaniea et al. [23] conduct a survey to understand the update behaviour of software. They ask 307 survey respondents to provide software update stories and analyze these stories to determine the possible motivations for software updates. Stock et al. [18] examine the top 500 websites per year between 1997 and 2016 utilizing archive.org dataset. In their measurement, they mainly evaluate security headers and analyse usage of outdated *jQuery* libraries.

Security Implications Prior literature has proposed various techniques to measure websites' security in terms of different metrics. Lauinger et al. [10] study widely used 72 client-side JavaScript libraries usage and measure security across Alexa Top 75k. Van Goethem et al. [2] report the state of security for 22,000 websites that originate in 28 European countries. Their analysis is based on different metrics (e.g., security headers, information leakage, outdated software). However, they use only a few popular software products for their measurement. Huang et al. [7] measure the security mechanisms of 57,112 chinese websites based on vulnerabilities published on Chinese bug bounty platforms between 2012 and 2015. Van Acker et al. [22] scrutinize the security state of login webpages by attacking login pages of websites in the Alexa top 100k.

7 Discussion and Conclusion

In this work, we measured the update behavior and possible security implications of software products utilized on more than 5.6M websites. Our measurement highlights the current state of the Web and shows the update behavior of websites over the course of 18 month. We show that most of the Web's utilized software is outdated, often by more than four years. Running outdated software is not a security problem per se because the old software might not be vulnerable. However, we found several sites that use software products for which vulnerabilities have been reported. Furthermore, we show that the number of vulnerable websites increases over time while the average severity of identified vulnerabilities decreases. For instance, we record that 95 % of websites *potentially* contain at least one vulnerable software. It has to be noted that the identified vulnerabilities in our work must be seen as an upper bound because utilizing a product for which vulnerabilities exist does not automatically mean that it can be exploited (e.g., the vulnerable module of the product is deactivated or not used). Our results still

highlight that website providers need to take more care about their update processes, even if this comes with a potential overhead, to protect their users and services.

Acknowledgment

This work was partially supported by the Ministry of Culture and Science of North Rhine-Westphalia (MKW grant 005-1703-0021 "MEwM" and "connect.emscherlippe") and by the Federal Ministry for Economic Affairs and Energy (grant 01MK20008E "Service-Meister").

References

1. Barth, A.: The Web Origin Concept. RFC 6465, Internet Engineering Task Force (2011), https://tools.ietf.org/html/rfc6454
2. van Goethem, T., Chen, P., Nikiforakis, N., Desmet, L., Joosen, W.: Large-Scale Security Analysis of the Web: Challenges and Findings. In: International Conference on Trust and Trustworthy Computing. TRUST (2014). https://doi.org/10.1007/978-3-319-08593-7_8
3. Google Inc.: Chrome User Experience Report | Tools for Web Developers. https://developers.google.com/web/tools/chrome-user-experience-report?hl=de (2020), online; Accessed: 2020-06-08
4. HTTP Archive: About HTTP Archive. https://httparchive.org/about (2020), [Online; accessed 20. Oct. 2020]
5. HTTP Archive: Methodology — The Web Almanac by HTTP Archive. https://httparchive.org (2020), [Online; Accessed: 18. Jan. 2021]
6. HTTP Archive: The HTTP Archive Tracks How the Web is Built. https://httparchive.org (2020), [Online; Accessed: 20. Oct. 2020]
7. Huang, C., Liu, J., Fang, Y., Zuo, Z.: A study on Web security incidents in China by analyzing vulnerability disclosure platforms. Computers & Security **58** (2016). https://doi.org/10.1016/j.cose.2015.11.006
8. Ikram, M., Masood, R., Tyson, G., Kaafar, M.A., Loizon, N., Ensafi, R.: The Chain of Implicit Trust: An Analysis of the Web Third-Party Resources Loading. In: International Conference on World Wide Web. WWW, International World Wide Web Conferences Steering Committee (2019). https://doi.org/10.1145/3308558.3313521
9. Kula, R.G., German, D.M., Ouni, A., Ishio, T., Inoue, K.: Do Developers Update their Library Dependencies? Empirical Software Engineering **23**(1) (2018). https://doi.org/10.1007/s10664-017-9521-5
10. Lauinger, T., Chaabane, A., Arshad, S., Robertson, W., Wilson, C., Kirda, E.: Thou Shalt Not Depend on Me: Analysing the Use of Outdated JavaScript Libraries on the Web. In: Symposium on Network and Distributed System Security. NDSS (2017). https://doi.org/10.14722/ndss.2017.23414
11. Le Pochat, V., Van Goethem, T., Tajalizadehkhoob, S., Korczyński, M., Joosen, W.: Tranco: A Research-Oriented Top Sites Ranking Hardened Against Manipulation. In: Symposium on Network and Distributed System Security. NDSS (2019). https://doi.org/10.14722/ndss.2019.23386
12. National Institute of Standards and Technology: Official Common Platform Enumeration (CPE) Dictionary. https://nvd.nist.gov/products/cpe (2020), [Online; accessed 19. Oct. 2020]

13. Nguyen, V.H., Massacci, F.: The (Un)Reliability of NVD Vulnerable Versions Data: An Empirical Experiment on Google Chrome Vulnerabilities. In: ACM Symposium on Information, Computer and Communications Security. AsiaCCS (2013). https://doi.org/10.1145/2484313.2484377
14. Preston-Werner, T.: Semantic Versioning 2.0.0. https://semver.org/ (2020), online; Accessed: 20. Oct. 2020
15. Redmiles, E.M., Kross, S., Mazurek, M.L.: How I Learned to Be Secure: A Census-Representative Survey of Security Advice Sources and Behavior. In: ACM Conference on Computer and Communications Security. CCS (2016). https://doi.org/10.1145/2976749.2978307
16. Shahzad, M., Shafiq, M.Z., Liu, A.X.: A Large Scale Exploratory Analysis of Software Vulnerability Life Cycles. In: International Conference on Software Engineering. ICSE (2012). https://doi.org/10.5555/2337223.2337314
17. shopware AG: Updating Shopware. https://docs.shopware.com/en/shopware-5-en/update-guides/updating-shopware (2020), [Online; accessed 20. Oct. 2020]
18. Stock, B., Johns, M., Steffens, M., Backes, M.: How the Web Tangled Itself: Uncovering the History of Client-Side Web (In)Security. In: USENIX Security Symposium. SEC (2017). https://doi.org/10.5555/3241189.3241265
19. Tajalizadehkhoob, S., Van Goethem, T., Korczyński, M., Noroozian, A., Böhme, R., Moore, T., Joosen, W., van Eeten, M.: Herding Vulnerable Cats: A Statistical Approach to Disentangle Joint Responsibility for Web Security in Shared Hosting. In: ACM Conference on Computer and Communications Security. CCS (2017). https://doi.org/10.1145/3133956.3133971
20. Torchiano, M., Ricca, F., Marchetto, A.: Are Web Applications More Defect-prone than Desktop Applications? International Journal on software tools for technology transfer **13**(2) (2011)
21. Urban, T., Degeling, M., Holz, T., Pohlmann, N.: Beyond the Front Page: Measuring Third Party Dynamics in the Field. In: International Conference on World Wide Web. WWW (2020). https://doi.org/10.1145/3366423.3380203
22. Van Acker, S., Hausknecht, D., Sabelfeld, A.: Measuring Login Webpage Security. In: Symposium on Applied Computing. pp. 1753–1760. SAC (2020). https://doi.org/10.1145/3019612.3019798
23. Vaniea, K., Rashidi, Y.: Tales of Software Updates: The Process of Updating Software. In: Conference on Human Factors in Computing Systems. CHI (2016). https://doi.org/10.1145/2858036.2858303
24. Wappalyzer: Identify technology on websites—Wappalyzer. https://www.wappalyzer.com (2020), online; Accessed: 2020-06-08
25. WordPress: Configuring Automatic Background Updates (2019), https://wordpress.org/support/article/configuring-automatic-background-updates, [Online; accessed 20. Oct. 2020]
26. Zou, Y., Mhaidli, A.H., McCall, A., Schaub, F.: "I've Got Nothing to Lose": Consumers' Risk Perceptions and Protective Actions after the Equifax Data Breach. In: Symposium on Usable Privacy and Security. SOUPS (2018). https://doi.org/10.5555/3291228.3291245

A Overview of the Top Identified CWEs

In this appendix, we show our findings related to identified CWEs. Table 2 lists the most common CWEs on websites that we identified in the last measurement

Vulnerability Type (CWE)	Relative Frequency
CWE-79 Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	0.92
CWE-20 Improper Input Validation	0.32
CWE-400 Uncontrolled Resource Consumption	0.27
CWE-200 Exposure of Sensitive Information to an Unauthorized Actor	0.24
CWE-476 NULL Pointer Dereference	0.24
CWE-601 URL Redirection to Untrusted Site ('Open Redirect')	0.22
CWE-125 Out-of-bounds Read	0.22
CWE-119 Improper Restriction of Operations within the Bounds of a Memory Buffer	0.20
CWE-787 Out-of-bounds Write	0.19
CWE-190 Integer Overflow or Wraparound	0.17
CWE-284 Improper Access Control	0.17

Table 2. Top 10 vulnerabilities in our last measurement point (M#18) by relative frequency on websites.

Software	CVE	CVE Publication	CWE	CVSS	Public exploit	Vuln. Websites	Total usage
jQuery	CVE-2020-11023	04.2020	XSS	4.3	✗	3.98M	4M
Apache	CVE-2017-7679	06.2017	Buffer Over-read	7.5	✓	0.26M	0.46M
PHP	CVE-2015-8880	05.2016	Double free	10	✓	0.45M	0.46M
PHP	CVE-2016-2554	03.2016	Buffer Over-read	10	✓	0.23M	0.46M
WordPress	CVE-2018-20148	12.2018	Deserialization of Untrusted Data	7.3	✓	0.18M	0.46M
WordPress	CVE-2019-20041	12.2019	Improper Input Validation	7.3	✗	0.31M	0.46M

Table 3. Some examples of vulnerabilities identified on analyzed websites that run outdated software.

run (June 2020). While the vulnerability *Cross-site Scripting* (XSS) occurs in almost all websites, a closer analysis of the same measurement point (M#18) shows that only 28% of software is vulnerable to this vulnerability.

B Average age of the 20 used software by website-ranking

Figure 6 shows the most popular software types, their average age (in month), and the rank of the websites on which they are used. We record that most of the widely used software on the web is often very old. We also found that the average age of utilized software on a website is unrelated to its popularity, according to the *Tranco* list [11].

C Case Studies

Table 3 illustrates the most common CVE entries identified in our study. *CVE-2020-11023* is the most common vulnerability with the severity "MEDIUM" – based on our last measurement. Some of the vulnerabilities require certain functions or enabled functions (e.g., CVE-2017-7679 for *Apache* requires *mod_mime* and CVE-2016-2554 for *PHP* requires file uploading functionality) In some cases, the running software requires interaction between more than one component to abuse an exploit. The listed vulnerabilities for *WordPress* and vulnerability CVE-2015-8880 for *PHP* do not require any interaction or enabled features and can be exploited directly.

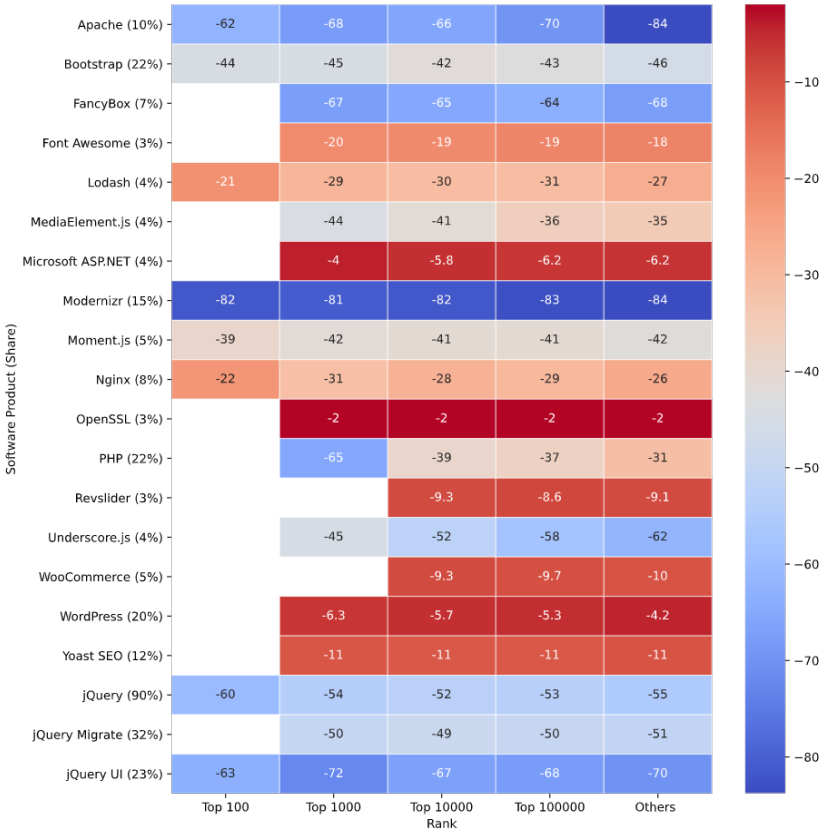

Fig. 6. Average age (in month) of the top 20 used software by website ranking. The share of software in our dataset is shown in brackets – Blank cells: no website identified in the corresponding ranking.