Large-scale Security Analysis of the Web: Challenges and Findings

Tom van Goethem, Ping Chen, Nick Nikiforakis,
Lieven Desmet, and Wouter Joosen

iMinds-DistriNet, KU Leuven
3001 Leuven, Belgium
{firstname.lastname}@cs.kuleuven.be

Abstract As the web expands in size and adoption, so does the interest of attackers who seek to exploit web applications and exfiltrate user data. While there is a steady stream of news regarding major breaches and millions of user credentials compromised, it is logical to assume that, over time, the applications of the bigger players of the web are becoming more secure. However, as these applications become resistant to most prevalent attacks, adversaries may be tempted to move to easier, unprotected targets which still hold sensitive user data.
In this paper, we report on the state of security for more than 22,000 websites that originate in 28 EU countries. We first explore the adoption of countermeasures that can be used to defend against common attacks and serve as indicators of "security consciousness". Moreover, we search for the presence of common vulnerabilities and weaknesses and, together with the adoption of defense mechanisms, use our findings to estimate the overall security of these websites. Among other results, we show how a website's popularity relates to the adoption of security defenses and we report on the discovery of three, previously unreported, attack variations that attackers could have used to attack millions of users.

1 Introduction

Over the last decade, the web has become extremely popular. Businesses heavily depend on the web for their day-to-day operations, and billions of users interact on social networking websites on a daily basis. As a consequence of this enormous growth in popularity, the web has also drawn increased attention from attackers. A whole range of web attacks exists in the wild, ranging from Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF), and SQL injection, to the exploitation of broken authorization and session management. Moreover, as the technologies that support the web increase in numbers and complexity, new opportunities for exploitable vulnerabilities increase with them.

To assess a website's security, website owners typically choose security consulting firms for internal penetration testing, and code reviewing. It is difficult, however, for outsiders like government and supervisory organizations to assess a website's security externally, especially when the assessment needs to be done

at a larger scale, e.g., involving a large number of websites belonging to a country, or a specific industry sector. Such an assessment may be desirable since the citizens of each country depend more and more on certain web applications for their daily lives. An example of a real-world equivalent is the mandatory assessment of the structural safety of buildings in order to protect people from future disasters that could have been straightforwardly avoided.

In this paper, we investigate the feasibility of external security evaluations through a large-scale security analysis of the web. In particular, we evaluate the security stance of popular websites in the European Union (EU), and investigate the differences among countries.

To evaluate a website's security, existing approaches typically focus on the discovery of vulnerabilities in websites. For example, WhiteHat publishes yearly reports on website security statistics [31], highlighting the ten most common vulnerabilities, and discussing new attack vectors. Contrastingly, our approach not only accounts for common vulnerabilities and weaknesses, but also measures the presence of security mechanisms deployed on the investigated websites. These mechanisms have been developed by the security community as a response to web application attacks, making their adoption a crucial step towards a more secure web. The presence or absence of each of these mechanisms can be passively detected and can be used as an indicator of the "security consciousness" of each individual site.

In addition, in order to be able to compare websites by their security posture, we also propose a security scoring system for assessing a website's security level, and based on the scoring system, we present a comparative security analysis of European websites. Finally, because of the breadth of our analysis, we report on the discovery of novel variations of existing web application attacks. In one of the discovered cases, an attacker can register an expired Google Code project and serve malicious JavaScript to millions of users of sites that once trusted that specific project for remote code.

Our findings allow the community to assess the adoption of security mechanisms by websites at a large scale, and also prioritize corrective action, based on the severity of the discovered issues. Moreover, we list the challenges that we faced in our experiment, and provide possible directions towards future research in the area.

2 Data Collection

2.1 Dataset

For our experiment, we selected popular websites from the EU as the targets, to evaluate website security, and investigate the presence of potential differences between countries. The 28 member states in the EU represent a diverse set of communities, each with their own demographic characteristics. For each EU country, we selected the top 1,000 websites ending with a country code top-level domain (ccTLD) from Alexa's list of the top 1 million sites. For example, 1,000

websites ending with the Belgian ccTLD '`.be`' are extracted to represent the Belgian web. Note that several small EU countries (such as Luxembourg and Malta) do not have 1,000 websites in Alexa's top 1 million list, so we end up with a few countries having less than 1,000 websites in our dataset. We then obtained up to 200 webpage URLs for each website by querying the Bing search engine [1] for the popular webpages of each website. In total, we analyzed more than 3 million webpages for 22,851 EU websites with an average of 141 webpages per website.

2.2 Crawler setup

After the webpage URLs are obtained, PhantomJS [5], a headless browser, is used to visit the URLs and retrieve data from webpages. By loading every webpage within PhantomJS, we mimicked the behavior of a regular visitor using a Chrome browser. In order to crawl a large number of webpages in reasonable time, we run the experiment in a distributed fashion using 60 networked machines. As a result, our crawling experiment took approximately five days.

3 Security Scoring System

In order to compare the security level among different websites, and among different EU countries (represented by the websites of each country), we developed a security scoring system that gives quantitative security scores for each website. The security scores for a website consist of two parts: a positive score to represent the defense mechanisms adopted by the website (such as the `X-Frame-Options` and `Content-Security-Policy` headers), and a negative score for vulnerabilities or weaknesses (such as vulnerable remote JavaScript inclusions and insecure SSL implementations) found on it. For each defense mechanism and vulnerability/weakness, the security scoring system assigns a weighted positive and negative score. The overall positive and negative score for a website, is obtained by summing up each weighted positive and negative score respectively.

Due to our ethically-guided choice of conducting passive analysis for the majority of our tests, our search was limited to eight defense mechanisms, and ten vulnerabilities/weaknesses for each website. In principle, however, the security scoring system is scalable to more measurements. In the following sections, we briefly describe these defense mechanisms and vulnerabilities/weaknesses and elaborate on the scoring system we adopted.

3.1 Defense mechanisms

In our security assessment for defense mechanisms, we searched whether each website had adopted one or more of the following eight mechanisms:

– **HTTP Strict-Transport-Security (HSTS):** HSTS is a web security policy mechanism where a web server can force complying browsers to interact

with it using only HTTPS connections [15]. By sending out the HSTS policy via an HTTP response header named `Strict-Transport-Security`, a web server specifies a period of time during which the user's browser is instructed that all requests to that website need to be sent over HTTPS, regardless of what a user requests. As a result, HSTS can effectively thwart SSL-striping attacks and other Man-in-the-Middle (MitM) attacks [19,22].

- **Secure Cookies:** Website operators can make use of the `Secure` flag when sending out `Set-Cookie` headers. By doing so, the scope of a cookie is limited to only secure channels [10], which makes the cookie less likely to be stolen via eavesdropping.
- **Content Security Policy (CSP):** To mitigate a wide range of injection vulnerabilities, such as Cross-Site Scripting (XSS), a website operator can make use of the CSP mechanism. CSP provides a standard HTTP header that allows website owners to declare approved sources of content that browsers should be allowed to load on any given webpage [27]. Whenever a requested resource originates from a source that is not defined in the policy, it will not be loaded [28]. Hence, if the policy does not allow in-line JavaScript, then even if an attacker is able to inject malicious JavaScript in the webpage, the code will not be executed.
- **HttpOnly Cookies:** By default, cookies are accessible to JavaScript code, which allows attackers to steal a user's cookies in an XSS attack. To protect against the theft of cookies, a website operator can use the `HttpOnly` flag on cookies. An `HttpOnly` cookie will be used only when transmitting HTTP/HTTPS requests, making them unavailable to client-side JavaScript.
- **X-Frame-Options (XFO):** When an attacker is able to load a website, or part of a website in a `frame` or `iframe` element, the website might be vulnerable to ClickJacking attacks. More precisely, by redressing the user interface, an attacker can trick the user into clicking on the framed page while the click is intended for the bottom-level page [18]. To avoid ClickJacking attacks, the XFO HTTP response header [24] can be used to instruct a user's browser whether a certain page is allowed to be embedded in a frame.
- **Iframe sandboxing:** The `sandbox` attribute for the `iframe` element, introduced in HTML5, enables a set of extra restrictions on any content loaded in a frame. By specifying the `sandbox` value, a website operator can instruct the browser to load a specific frame's content in a low-privilege environment, allowing only a limited subset of capabilities to be made available to that frame [30].
- **CSRF Tokens:** The most popular defense for Cross-Site Request Forgery (CSRF) attacks is the inclusion of a secret token with each request and validation of that token at the server side [11]. This secret token, often referred to as a "nonce", should be pseudo-random and of a certain length so it cannot be guessed or brute-forced by an attacker. To check for nonces, we searched for forms that contained a hidden form element that was most likely used as a nonce. More specifically, form elements were marked as nonces when their name contained the keywords "token", "nonce", or "csrf", and when

their value was a long alpha-numerical string. These form elements were then manually verified in order to filter out any false positives.
- **X-Content-Type-Options:** Internet Explorer has a MIME-sniffing feature that will attempt to determine the content type for each downloaded resource. This feature, however, can lead to security problems for servers hosting untrusted content. To prevent Internet Explorer from MIME-sniffing, thus reducing exposure to attacks, a web server can send the `X-Content-Type-Options` response header with the `nosniff` value .

Apart from the sandboxing of frames and CSRF tokens, all the above defense mechanisms are communicated to the browser via HTTP response headers, and hence can be discovered straightforwardly by parsing a server's response headers. For the sandboxing of frames and the presence of CSRF tokens, we searched for `iframe` and `form` elements in the response body of each crawled webpage.

3.2 Vulnerabilities and Weaknesses

For the assessment of vulnerabilities and weaknesses, we focus on the following ten measurements:

- **Vulnerable Remote JavaScript Inclusion:** A website that chooses to include JavaScript from untrustworthy third-party sources opens itself up to a range of security issues. Recent research by Nikiforakis et al. [21], identified four different types of vulnerabilities that are related to the practice of unsafe remote JavaScript inclusions. In our assessment, we searched for the most dangerous of these vulnerabilities, called "Stale Domain-name-based Inclusions", where remote JavaScript is requested from a domain that has expired and is available for registration, which means the attacker can buy the domain and use it to serve malicious JavaScript.
- **Mixed-content Inclusion:** When migrating to HTTPS, many websites fail to fully update their applications, resulting in mixed-content inclusions where the main webpage is sent over a secure HTTPS channel, while some additional content included on that page, such as images and scripts, are delivered over non-secured HTTP connections. As a result, an active network attacker can attack the TLS-enabled website by intercepting and modifying any of the mixed content that is loaded over HTTP [13].
- **SSL-stripping Vulnerable Form:** For performance reasons, some websites only implement HTTPS for certain webpages that contain sensitive information (such as a log-in page), which may result in forms vulnerable to SSL stripping [19]. In this scenario, the form is displayed on an HTTP page, however the form action points to an HTTPS link. As a result, a MitM attacker can replace all HTTPS form links on the HTTP page to HTTP links, which will allow the attacker to intercept the form data sent from the user's browser.
- **Insecure SSL Implementation:** SSL is important for website owners since it provides end-to-end security. At the same time, however, it turns out that

it is not easy to deploy SSL correctly. According to Qualys' latest SSL survey of the most popular websites in December 2013, about half of the HTTPS websites have security issues associated with their SSL implementations [6]. In our assessment, we use a fast SSL scanner called `sslyze` [7] to search for SSL implementation issues including the support of SSL v2.0, use of weak ciphers, and the vulnerability to the recently discovered BEAST [14] and CRIME [23] attacks.

- **Weak Browser XSS Protection:** Most modern browsers include security mechanisms to protect a user against reflected Cross-Site Scripting attacks [20], and these features are, in general, enabled by default. While web servers can instruct a user's browser to disable this protection by means of the `X-XSS-Protection` response header, we consider such behavior as a weakness of the website, because disabling it might allow an attacker to successfully exploit an, otherwise unexploitable, XSS vulnerability.
- **HTTP Parameter Pollution (HPP):** When a website fails to properly sanitize user input, they might be vulnerable to HPP attacks. These attacks consist of injecting encoded query string delimiters into other existing parameters. By doing so, an attacker is able to compromise the application logic to perform client-side and server-side attacks. In our assessment, we searched for HPP vulnerabilities in a manner similar to the methodology of Balduzzi et al. [9].
- **Outdated Server Software:** It is important to keep web servers up-to-date, since an outdated server usually contains vulnerabilities that may lead to attacks. In our assessment, we searched for outdated server software for popular web servers including Apache, Microsoft-IIS, and Nginx.
- **Outdated Content Manage Systems (CMSs):** Many popular websites nowadays are built using a CMS, since CMSs allow non-technical users to build dynamic websites, and are usually free of charge. Similar to web servers, it is also recommended to keep a CMS up-to-date, as outdated CMSs often contain vulnerabilities. In our assessment, we looked for outdated CMSs for websites using WordPress, Joomla, vBulletin, and MediaWiki.
- **Information Leakage:** Many websites generate error messages and display them to users, which may reveal implementation details or information that is useful to an attacker. In our assessment, we searched for various categories of information leakage including SQL error messages, website directory listings, IIS error messages, PHP/ASP/JSP source code and error messages.
- **Sensitive Files:** A website may accidentally expose sensitive files such as configuration files and source code to the public, when moving files from the development server to the production server. The degree of vulnerability depends on the sensitive file that is exposed, ranging from information disclosure, to disclosure of source code containing credentials. In our assessment, we searched for the following files that were most likely to contain sensitive information: `phpinfo.php` or `test.php`, containing system information from the `phpinfo()` function, website configuration files, such as `Web.config`, and two version control system folders, namely `.svn/` and `.git/`.

Most of the aforementioned vulnerabilities and weaknesses can be discovered through passive analysis with PhantomJS visiting webpages, except for the finding of HPP vulnerabilities and sensitive files, where we actively scanned a limited number of webpages from each website, taking the necessary precautions not to stress or harm websites.

3.3 Scoring System Details

The scoring system used to estimate the state of security of websites is based on the Common Weakness Scoring System (CWSS) [3]. The CWSS provides a quantitative measurement of security weaknesses in software applications, and is mainly used to prioritize the remediation of reported weaknesses. In essence, the score appointed to a weakness by the CWSS aims to reflect the impact and likelihood of exploitation by adversaries. For instance, not "escaping" user-controlled data in an HTML document could lead to Cross-Site Scripting attacks, and may allow an attacker to extract sensitive user information. This weakness obviously has a high impact and is remotely exploitable; thus it will receive a higher score than, say, an insecure SSL implementation weakness.

The reason for using the CWSS over other scoring systems as a base for our scoring system is twofold. First, the CWSS is a well-established and commonly used mechanism to give a quantitative score to weaknesses. It has been extensively reviewed, which gives, to a certain extent, a guarantee that the score appointed to a weakness reflects the magnitude of the induced threat. Second, the CWSS gives scores to weaknesses, rather than to actual vulnerabilities as is done in the Common Vulnerability Scoring System (CVSS) [2]. This is important because most features we analyzed are security indicators rather than actual vulnerabilities.

The CWSS uses 18 different factors across three metric groups to calculate the total score for a weakness. The first group, named the "Base Finding" group, reflects the risk of the weakness, the finding confidence and the presence of built-in defense mechanisms. The second group, called the "Attack Surface" group, reflects the exploitability of a weakness. A vulnerability which is easy to exploit, such as a stale JavaScript inclusion, will consequently receive a higher score for this group. The last group, named the "Environmental" group, indicates, among others, the impact on the business in case the weakness is exploited, as well as the likelihood of discovery and exploitation. Each group is appointed a subscore which constitutes of a weighted score of its factors. The total score appointed to a weakness is calculated by multiplying the score for the "Base Finding" group (value between 0 and 100) by the two other groups (values between 0 and 1).

In order to give a metric to security features on a similar scale as weaknesses, the CWSS was also used to appoint scores to these defense mechanisms. As the CWSS only works for weaknesses, we calculated the score for security measures by determining the metric for the vulnerability or weaknesses they attempt to prevent. Additionally, we took the effectiveness of the countermeasure into account, as security features that completely block certain attacks should receive

Defense Mechanism	Score
Content Security Policy	58.93
`X-Frame-Options`	45.21
HTTP Strict-Transport-Security	33.52
CSRF tokens	32.73
Secure cookies	31.84
HttpOnly cookies	28.21
Iframe sandboxing	25.32
`X-Content-Type-Options`	8.02

Vulnerabilities and Weaknesses	Score
Vulnerable remote JavaScript inclusion	67.50
Sensitive files	41.81
SSL-stripping Vulnerable Form	30.16
`X-XSS-Protection`	28.33
Outdated CMS	18.30
Insecure SSL implementation	18.10
HTTP Parameter Pollution	18.06
Mixed-content inclusions	13.42
Information leakage	9.44
Outdated Server Software	8.71

Table 1: Calculated scores for defense mechanisms and vulnerabilities

a better score. For instance, the `HttpOnly` flag on cookies may prevent sensitive cookies to be stolen in Cross-Site Scripting attacks, but it will not mitigate all consequences of these attacks, something that a properly written Content Security Policy may do.

Table 1 shows the score appointed to each defense mechanism and weakness. Due to reasons of brevity, we limit the discussion of the rationale for the calculated scores to one example. As can be seen in the table, the vulnerable inclusion of remote JavaScript received the highest score (67.50). The high impact, i.e., the execution of arbitrary JavaScript code on multiple web pages, and the ease of exploitability, i.e., the registration of a stale domain name, are the main factors that contribute to this high score. Additionally, no control mechanisms (e.g. Content Security Policy) were found on the vulnerable websites that attempt to mitigate this vulnerability. Consequently, a score of 90 was calculated for the "Base Finding" group subscore. As victims will be exploited upon visiting the vulnerable web site, a score of 1 was appointed to the "Attack Surface" group. The score for the "Environmental" group is 0.75. The main factor that contributed to this score is the business impact, which is mostly case-specific. While the execution of arbitrary JavaScript code may have a very high impact on security-sensitive websites (e.g. a banking website), the potential impact on a purely informational website that stores no sensitive data is considerably less. The total score, 67.50, is then calculated by multiplying the three subscores (90 * 1 * 0.75).

Security mechanism	# of websites	% of websites	Estimated year of adoption
`HttpOnly` cookie	7,658	33.51	2007
CSRF token	3,815	16.70	NA
Secure cookies	1,217	5.33	2007
`X-Frame-Options`	1,029	4.50	2008
`X-Content-Type-Options`	467	2.04	2008
`Strict-Transport-Security`	116	0.50	2010
Content Security Policy	13	0.06	2011
Iframe sandboxing	10	0.04	2010

Table 2: Results from the analyzed websites that enable security features

4 Findings

4.1 General findings

Out of the 22,851 analyzed websites, we found that 10,539 (46.12%) enabled at least one security feature. As can be seen in Table 2, the most popular defense mechanism is the `HttpOnly` attribute on cookies, which was present in 33.51% of the evaluated websites. This defense mechanism is followed in popularity by the presence of a CSRF token in forms, which was found in 16.70% of the websites. Interestingly, these two most popular security features are mitigations for the most critical web application flaws according to the OWASP Top 10 project [4]. This table also shows that, in general, the popularity of a defense mechanism is related to the time it was adopted by popular browsers, i.e., the older a security feature, the more widely it is used.

In our evaluation, we found that 12,885 (56.39%) websites contained at least one vulnerability or weakness. Table 3 shows the distribution of the number of websites found to be vulnerable. While only 5,113 websites provided at least one page over HTTPS, we found that the majority (80.32%) had content originating from an insecure channel on their website, or had SSL implementation issues. Likewise, although we only evaluated 17,910 websites for the presence of HTTP Parameter Pollution (HPP) vulnerabilities, we found 15.24% of these websites to be vulnerable. As HPP is very closely related to XSS in the sense that they are both caused by improper encoding of certain characters, we manually analyzed a subset of the webpages vulnerable to HPP for XSS vulnerabilities. This showed us that approximately 75% of the websites vulnerable to HPP are also vulnerable Cross-Site Scripting attacks.

4.2 Incorrect security-header usage

By making use of headers, website administrators are capable of instructing a user's browser to enable a certain security feature. Browsers, however, require the value of the security-header to be correct. Values that are incorrect, for example headers containing a typing error or headers with incorrect syntax,

Vulnerability	# of websites	% of websites
Outdated server Software	6,412	28.06
Mixed-content inclusion	3,442	15.06
SSL-stripping Vulnerable Form	2,884	12.62
HTTP Parameter Pollution	2,731	11.95
Outdated CMS	2,041	8.93
Insecure SSL implementation	1,945	8.51
Information leakage	1,231	5.39
Sensitive files	1,068	4.67
Vulnerable remote JS inclusion	91	0.40
`X-XSS-Protection`	91	0.40

Table 3: Results from the analyzed websites that contain vulnerabilities

will be ignored by the browser. The presence of such headers in websites is a strong indication that the website administrator is under the impression that he successfully secured his website. Nonetheless, if the security-header contains a syntactical or typographical error, adversaries might be able to successfully exploit a vulnerability on a website.

In our analysis, we found several instances where the website operator tried to protect his website against ClickJacking attacks by using the `X-Frame-Options` header, but failed to do so by using an incorrect directive, for instance specifying `SAME-ORIGIN`, instead of the correct `SAMEORIGIN` directive.

Additionally, we found that 15 out of 116 (12.93%) analyzed websites that make use of the `Strict-Transport-Security` header to prevent SSL-stripping attacks, used the header in an improper fashion. The majority of these websites sent the `Strict-Transport-Security` header over an HTTP connection, without referring the user to an SSL-connection. Since browsers will ignore HSTS headers that are sent over an unencrypted channel, users of these websites can still fall victim to SSL-stripping attacks. The remainder of websites that implemented HSTS incorrectly, either forgot the `max-age` directive, or set this directive to the value `0`, which signals the user's browser to delete the HSTS policy associated with the website.

4.3 Security by Alexa rank

As the set of evaluated websites is distributed over the Alexa's list of the top 1 million websites, we evaluated how the rank of a website relates to the score we appoint it. We found that on average, the rank of a website is positively correlated with the positive score we appoint it, i.e., a high-ranked website is more likely to have a relatively high positive score. Contrastingly, we found that the negative score of a website is unrelated to its popularity according to Alexa. This indicates that popular websites try to improve their security by the adoption of defense mechanisms, rather than by tackling vulnerabilities. Figure 1 depicts the relation between the security score and the Alexa rank. Each entry coincides with the average positive or negative metric of the evaluated websites that fall within a range of 10,000 Alexa ranks.

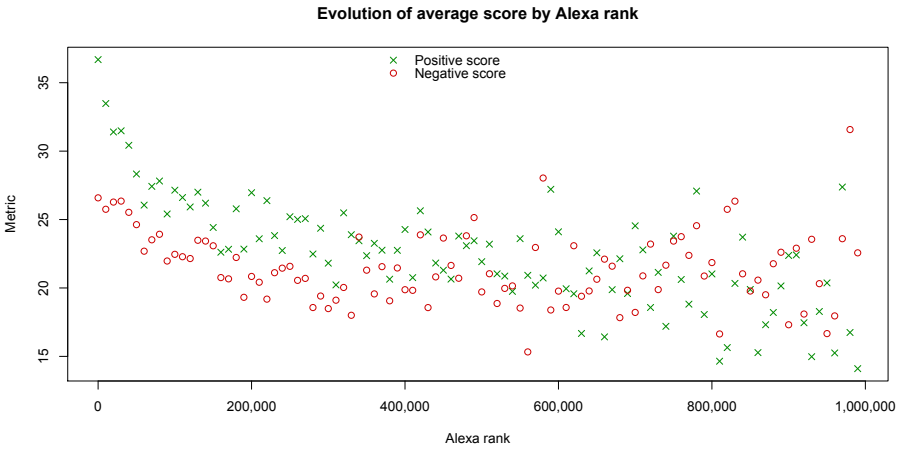

Figure 1. Average metric by analyzed websites grouped by 10,000 Alexa entries

Additionally, we found that, in general, there is no correlation between the positive metric and negative score of a website, which strengthens the indication that websites try to improve their security by adding security mechanisms in an ad-hoc fashion. Moreover, we found that a large number of websites apply a certain defense mechanism to a limited fraction of the URLs we visited, e.g. the majority of websites that make use of the `X-Content-Type-Options` header to prevent XSS attacks in Internet Explorer due to MIME sniffing, only add the header to a small fraction of their pages.

4.4 Security by country

We found that the scores for websites located in different countries were similar. Figure 2 shows the cumulative distribution function of both the positive as well as the negative score for websites of a set of four randomly selected countries. From this set, Germany has more websites with a higher positive score than the other countries. However, the same country scores worse than the rest on the negative score. This again shows that there is no relation between the number of enabled security features and the number of weaknesses or vulnerabilities we were able to find on a website.

The variance of scores between different countries are most likely due to the unequal distribution of the countries' websites over the Alexa rank. The distribution of Alexa rank for the subset of four countries is shown in Figure 3. Compared to the distribution of the positive score, it is clear that the countries with the most high-ranked websites have a better positive score. This indicates that, in general, the security of a website is unrelated to its geographical location or the policies its hosting country may have.

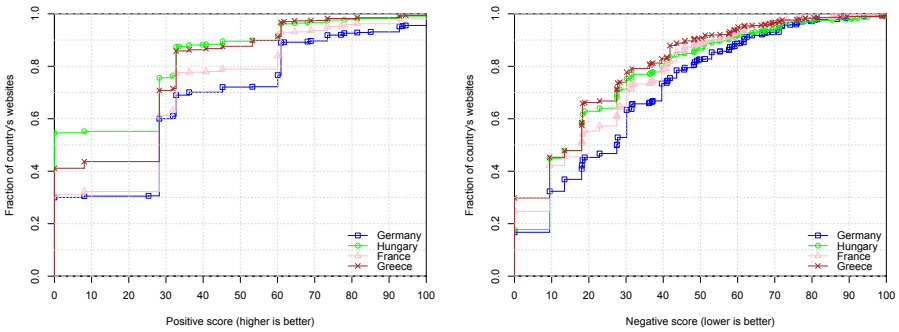

Figure 2. Distribution of positive and negative score for several countries' websites

4.5 Novel attack techniques

In the course of our analysis, we encountered a new attack technique in the Cross-Origin Resource Sharing mechanism as well as two variations on the insecure inclusion of remote JavaScript code on a webpage. Both are related to remote trust relations in the sense that the website operator trusts a certain domain or URL to be benign, which may become malicious in the future.

By sending out the `Access-Control-Allow-Origin` header, a website operator can instruct a browser to allow a third-party website to make XHR-requests towards his website and read out the result. When the `Access-Control-Allow-Credentials` header is included as well, these requests can be authenticated. It is in the best interest of a website administrator to only allow trusted websites to extract the response of an XHR-request targeting his website. Interestingly, we found a case where a website sent out the `Access-Control-Allow-Origin` containing a `.local` domain. This allows an attacker to trick a user on the local network in visiting his webpage located at the `.local` domain. The attacker is then able to make the victim's browser send XHR-requests to the vulnerable website while being able to read out CSRF-tokens from forms.

In the aforementioned work by Nikiforakis et al. [21], the authors analyzed the inclusion of JavaScript files from expired domains. In the course of our analysis, we encountered two variations on this type of attack. More specifically, we found that several websites remotely include JavaScript files from domains that were marked as "for sale" by their owner on `sedo.com`, a large domain marketplace. Similar to the attack described by the authors, an attacker is able to buy such a domain, and serve malicious JavaScript to unsuspecting users. The second variation on this type of attack occurs when websites include JavaScript files directly from project hosting websites, such as GitHub or Google Code. The files hosted on these services are linked to a project or a user. However, upon deletion, that project or account, becomes again available for registration. This way, an adversary is able to host malicious JavaScript, which may be included by a large

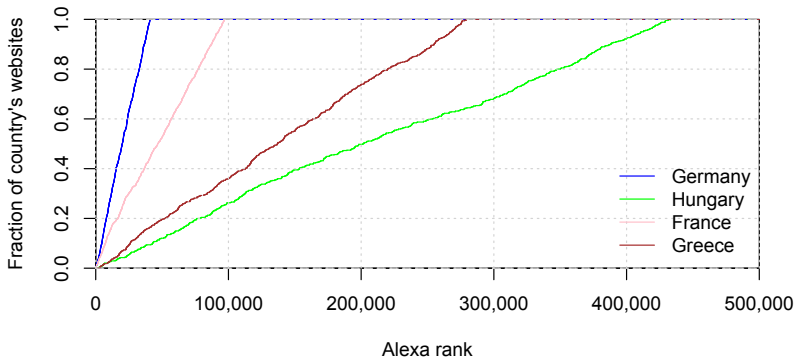

Figure 3. Distribution of Alexa rank for several countries

set of websites. To show the importance of this type of attack, we registered a stale project on Google Code, and made available the last available version of project's JavaScript files, with a minor addition which allowed us to analyze the number of including websites and affected users. During a month's time, we registered a total of 3,879,701 requests, originating from 1,104,497 unique IP addresses. In total, there were 3,400 websites, including a prominent Chinese news website, which directly included JavaScript files from this Google Code project. In every single one of these requests, an attacker could have have served malicious JavaScript that steals a user's cookies, exfiltrates private user information and even attempts to launch a drive-by download.

4.6 Miscellaneous

In our analysis, we found that the presence of certain security features, such as the `HttpOnly` attribute in the `Set-Cookie` header, is more common in websites that are powered by frameworks which facilitate the system-wide usage of these security features [25]. More precisely, through the `X-Powered-By` header we found that although the majority (49.53%) of the analyzed websites are powered by PHP, only 16.36% of these websites enable the `HttpOnly` attribute. The second most popular framework is ASP.NET, used by 22.80% of the crawled websites. Interestingly, we found that 54.74% of these ASP.NET websites enable `HttpOnly` (three times as much as PHP sites).

5 Limitation and Challenges

5.1 Accuracy of passive analysis

Due to legal and ethical considerations, our analysis of vulnerabilities in websites was limited to a passive analysis, with a few exceptions. Consequently, the results described in the previous section only show an estimation on the state of security of European websites. In order to assess the accuracy of these estimates, we compared the scores of websites likely to be insecure, to websites expected to be secure. The set of most-likely vulnerable websites consisted of websites with a Cross-Site Scripting vulnerability which was publicly known and had not been patched in over one year.[1] The set of websites probable to be secure, was made up from a set of 20 respectable banking websites. This comparison showed that the average positive score for the known-vulnerable set (35.21) was lower than that of the set of banking websites (41.62). Also the average negative score, which was 27.22 on the insecure set and 12.80 on the probably secure websites, indicates that, despite the fact that only a fraction of a website's state of security could be assessed, we were still able to differentiate between vulnerable and secure websites. At the same time, we are aware of the coarse-granularity of our analysis and we highlight the antithesis between the invasiveness of an external security assessment, and the coverage obtained by it. It would be worthwhile to investigate whether website administrators would be willing to consent to a more invasive security assessment, in return for obtaining the results free of charge.

5.2 Scoring system

In order to evaluate the general state of security of a website, we developed a scoring system based on CWSS, as was described in Section 3.3. However, this scoring system is subject to two types of limitations. Firstly, the total positive score assigned to a website originates from an individually assigned score of eight security features, while the total negative score is derived from a score attributed to ten potential weaknesses and vulnerabilities. As a result, the positive score for a website is on a different scale from the negative score. This prevents us from being able to compare the positive score, to the negative score. Moreover, as the total score appointed to a website originates from a limited set of factors, the total score may not always reflect the actual state of security of a website. However, as we evaluate diverse aspects which are highly related to a website's security, we believe that our scoring system provides a good estimate on the general state of security of a website.

The metric appointed to each weakness and security measure is derived from a list of 18 factors, some of which are subject to the opinion of the authors or are often case-specific. For instance, the impact of exploiting a certain vulnerability may differ based on the type of website. In order to account for these differences, each metric was calculated for a general website. Consequently, the appointed

[1] `http://www.xssed.com`

metrics are not website-specific and the score of one feature is relative to the other scores. This allows us to appoint a comparable score which gives an estimation of the state of security for each tested website.

6 Related Work

To the best of our knowledge, there exists no large-scale analysis which evaluates security features as well as weaknesses in a broad range of websites. Nonetheless, several evaluations on the presence of specific vulnerabilities in web applications have been carried out. For instance, WhiteHat Security evaluates, on a yearly basis, the data on several types of vulnerabilities they collect from their customers [31]. Contrastingly to our research, their security analysis has the permission of their clients and is thus more aggressive, which enables them to find additional types of vulnerabilities, such as SQL Injections and XSS errors.

Another large-scale evaluation of websites in a specific demographic area is presented in research by Alarifi et al. [8], that evaluates the security of popular Arabic websites. Their analysis explores the presence of phishing and malware pages in 7,000 domains. To detect malicious scripts hosted on webpages, they make use of APIs offered by known website scanners. Kals et al. developed the SecuBat tool, which was used for an automated detection of XSS and SQL Injection vulnerabilities in a selection of 100 security-sensitive websites [16]. Similarly, Zeller et al. performed an analysis on the presence of CSRF vulnerabilities in popular websites [32], finding vulnerabilities in four major websites. Nikiforakis et al. presented a large-scale analysis of remote JavaScript inclusions [21]. Additionally, in their paper, they also proposed a metric called Quality of Maintenance (QoM) to characterize a website's security consciousness. Their QoM adopts several features such as `HttpOnly` cookies, `X-Frame-Options`, that are also included in our assessment. As earlier discussed, the presence of these defensive mechanisms give an indication for a website's security.

Vasek and Moore found that some website features, such as server software and CMSs, can serve as positive risk factors for webserver compromise [29]. Their study shows that some server types and CMS types are more risky than others (e.g., servers running Apache and Ngnix are more likely to be compromised than those running Microsoft IIS).

Lekies et al. performed a large-scale detection of DOM-based XSS vulnerabilities in the top 5,000 Alexa websites [17]. In their evaluation, they found a total of 6,167 unique vulnerabilities distributed over 480 domains, demonstrating that 9.6% of the evaluated websites are vulnerable to this type of attack. Son et al. analyzed the implementation of the HTML5 `postMessage` mechanism in the Alexa top 10,000 [26]. They found that 84 popular websites were exploitable to several attacks, including XSS and content injection, due to the lack of proper checks in the cross-origin communication mechanism.

A feature that we did not include in our study was the security of a site's hosting provider. Sites situated on shared hosting environments are expected to be at a greater risk of compromise, since a vulnerability of another co-located

tenant can be used to attack the entire server. Canali et al. recently investigated the ability of shared hosting providers to detect compromised sites hosted on their servers [12], finding that the vast majority of providers cannot detect even the most straightforward attacks.

7 Conclusion

Websites have become the main target for numerous attacks originating from adversaries who attempt to monetize a user's sensitive data and resources. In order to protect themselves from this threat, website operators are provided with several security mechanisms to defend against a wide range of vulnerabilities. In this paper, we evaluated the usage of security features, as well as the presence of vulnerabilities and weaknesses, in 22,851 EU websites. We found that a large part of the evaluated websites showed weaknesses, and some even contained severe vulnerabilities. Moreover, we discovered that the state of security of a website is unrelated to its demographic characteristics. In spite the fact that popular websites are more likely to prevent attacks by implementing security features, we found that the presence of weaknesses and vulnerabilities is unrelated to a site's popularity. We hope that our study can inspire similar systems at a country- or sector-level, and help the owners of sites to discover and prioritize the adoption of security mechanisms, and the correction of existing vulnerabilities.

Acknowledgements We want to thank the anonymous reviewers for the valuable comments. This research was performed with the financial support of the Prevention against Crime Programme of the European Union (B-CCENTRE), the Research Fund KU Leuven, and the EU FP7 projects NESSoS, WebSand, and STREWS.

References

1. Bing Search API. `http://datamarket.azure.com/dataset/bing/search`.
2. Common Vulnerability Scoring System (CVSS). `http://www.first.org/cvss`.
3. Common Weakness Scoring System (CWSS). `https://cwe.mitre.org/cwss/`.
4. OWASP Top Ten Project. `https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project`.
5. Phantomjs: Headless webkit with javascript api. `https://www.phantomjs.org/`.
6. SSL Pulse. `https://www.trustworthyinternet.org/ssl-pulse/`.
7. sslyze. `https://github.com/iSECPartners/sslyze`.
8. Abdulrahman Alarifi, Mansour Alsaleh, and AbdulMalik Al-Salman. Security analysis of top visited arabic web sites. In *Advanced Communication Technology (ICACT), 2013 15th International Conference on*, pages 173–178. IEEE, 2013.
9. Marco Balduzzi, Carmen Torrano Gimenez, Davide Balzarotti, and Engin Kirda. Automated Discovery of Parameter Pollution Vulnerabilities in Web Applications. In *18th Annual Network and Distributed System Security Symposium*, San Diego, USA, 2011.

10. A. Barth. HTTP state management mechanism. *IETF RFC*, 2011.
11. Adam Barth, Collin Jackson, and John C. Mitchell. Robust defenses for cross-site request forgery. In *Proceedings of the 15th ACM conference on Computer and communications security*, CCS '08, pages 75–88, New York, NY, USA, 2008. ACM.
12. Davide Canali, Davide Balzarotti, and Aurélien Francillon. The role of web hosting providers in detecting compromised websites. In *Proceedings of the 22Nd International Conference on World Wide Web*, WWW '13, pages 177–188, 2013.
13. Ping Chen, Nick Nikiforakis, Christophe Huygens, and Lieven Desmet. A Dangerous Mix: Large-scale analysis of mixed-content websites. In *Proceedings of the 16th Information Security Conference*, ISC '13, Dallas, USA, 2013.
14. Thai Duong and Juliano Rizzo. *Here Come The ⊕ Ninjas*, 2011.
15. J. Hodges, C. Jackson, and A. Barth. HTTP strict transport security (HSTS). *IETF RFC*, 2012.
16. Stefan Kals, Engin Kirda, Christopher Kruegel, and Nenad Jovanovic. Secubat: a web vulnerability scanner. In *Proceedings of the 15th international conference on World Wide Web*, pages 247–256. ACM, 2006.
17. Sebastian Lekies, Ben Stock, and Martin Johns. 25 million flows later: large-scale detection of dom-based xss. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 1193–1204. ACM, 2013.
18. R. Lundeen, J. Ou, and T. Rhodes. New ways i'm going to hack your web app. 2011.
19. Moxie Marlinspike. New tricks for defeating ssl in practice. *Blackhat*, 2009.
20. Microsoft. IE8 Security Part IV: The XSS Filter. 2008.
21. Nick Nikiforakis, Luca Invernizzi, Alexandros Kapravelos, Steven Van Acker, Wouter Joosen, Christopher Kruegel, Frank Piessens, and Giovanni Vigna. You are what you include: large-scale evaluation of remote javascript inclusions. In *Proceedings of the 2012 ACM conference on Computer and communications security*, CCS '12, pages 736–747, New York, NY, USA, 2012. ACM.
22. Nick Nikiforakis, Yves Younan, and Wouter Joosen. HProxy: Client-side detection of SSL stripping attacks. In *Proceedings of the 7th Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA'10)*, 2010.
23. Juliano Rizzo and Thai Duong. Crime: Compression ratio info-leak made easy. In *ekoparty Security Conference*, 2012.
24. D. Ross and T. Gondrom. HTTP Header X-Frame-Options. *IETF RFC*, 2013.
25. D. Sellers. ASP.NET 2.0 and the new HTTP-only property. *MSDN Blogs*, March 2006.
26. Sooel Son and Vitaly Shmatikov. The postman always rings twice: Attacking and defending postmessage in html5 websites.
27. Sid Stamm, Brandon Sterne, and Gervase Markham. Reining in the web with content security policy. In *Proceedings of the 19th international conference on World wide web*, WWW '10, pages 921–930, New York, NY, USA, 2010. ACM.
28. B. Sterne and A. Barth. Content Security Policy 1.0. *W3C Candidate Recommendation*, 2012.
29. Marie Vasek and Tyler Moore. Identifying Risk Factors for Webserver Compromise. In *Proceedings of the Eighteenth International Conference on Financial Cryptography and Data Security*, FC' 14, 2014.
30. Mike West. Play safely in sandboxed iframes. 2013.
31. WhiteHat. Website Security Statistics Report. `https://www.whitehatsec.com/resource/stats.html`.
32. William Zeller and Edward W Felten. Cross-site request forgeries: Exploitation and prevention. *The New York Times*, pages 1–13, 2008.