Cross-Site Challenge-Response Attacks

Nethanel Gelernter
Dept. of Computer Science
College of Management Academic Studies
nethanel.gelernter@gmail.com

Itamar Peretz
Dept. of Computer Science
Ben-Gurion University of the Negev
itamarpe@post.bgu.ac.il

***Abstract*—The challenge-response mechanism is a computer security method used to prevent access by unauthorized parties. Similar to passwords, it involves a group of protocols in which one party is asked a question and must provide a correct answer in order to be authenticated. This work shows that using challenge-response as a cross-site request forgery (CSRF) countermeasure puts the challenge-response itself at risk.**

The *cross-site (XS) challenge-response* attack uses brute-force on the challenge-response mechanism in a cross-site manner, relying on advanced techniques to bypass the same-origin policy.

We present and analyze two variants of the XS challenge-response attack: (1) an unauthenticated variant in which visitors to the attacking page are abused as bots that attempt to break the challenge-response, and (2) an authenticated variant that directly targets visitors to the malicious page.

Our work surveys the use of challenge-response by popular websites, content management systems, IoT devices, and routers to show that XS challenge-response vulnerabilities are common. We created proofs of concept for the vulnerabilities discovered, and ethically evaluated attacks under real-world conditions to demonstrate the tangibility of the threat. Several vendors have already confirmed these vulnerabilities, which affect millions of users, and are working on fixes.

I. Introduction

The challenge-response method of authentication uses a group of protocols in which one party presents a challenge, and the other party must provide a correct response to be authenticated.

Our work focuses on the *feasibly guessable* challenge-response. These challenges are hard for a hacker to guess, yet possible given many tries. Two examples of challenge-response that are known to almost every web user are the following:

1) Password. Although these can be chosen from extremely large sets of options, it is known that passwords can be guessed [13], [37], [32].
2) Secret code that is sent during the password activation or reset process. Although they might be long, it is still feasible to guess them. For example, Facebook uses a six-digit password reset code, which can be guessed with a success probability of one to a million per guess.

Workshop on Measurements, Attacks, and Defenses for the Web (MADWeb) 2019
24 February 2019, San Diego, CA, USA
ISBN 1-891562-60-6
https://dx.doi.org/10.14722/madweb.2019.23xxx
www.ndss-symposium.org

Feasibly guessable challenge-response (from now on referred to as simply *challenge-response*) are widely used for authentication. Passwords, the most common challenge-response mechanism, are used as the primary account protection mechanism on the web. But passwords and other sensitive pieces of information (e.g., confirmation number, ID number, or order number) are also used as challenge-response for other purposes. For example, users might be asked to provide a confirmation/order ID to access the details of a purchase. Another example is old passwords, which might be requested when the user is changing to a new one. The most common purpose of challenge-response on the web is to prevent remote access to an account by unauthorized parties; for example, to prevent attackers from signing in. Challenge-response is also used to prevent local access. For example, it can be used to prevent a change in account settings. Although passwords are usually required when modifying sensitive account information, websites may request different sensitive user information such as email address, phone number, credit card details, and so forth. In this way, even if an attacker physically accesses a computer to which the user is authenticated, he cannot carry out his malicious plan without overcoming the challenge.

Physical or remote access without any interaction with the victim is not the only way to manipulate accounts. In the absence of appropriate countermeasures, attackers can launch a *cross-site request forgery* (*CSRF*) [31], which essentially forces innocent web users to perform specific operations. For CSRF, attackers often abuse the browsers of web-users who surf their page, and trick them into sending requests to different services. There exist several techniques that can be used to prevent CSRF. Some of them rely on indications that are sent via the browser [9], [6]. However, the most reliable countermeasures that work for every browser, rely on a value that is not known to the cross-site attacker and is attached to the request. The server then verifies the existence and the correctness of that value before it services the request. Section II-A further explains the cross-site attacker model. Sensitive operations such as logging or changing credentials should be protected against all of the above-mentioned attacks. In many cases, a single challenge-response is used to protect against all of these attacks. For example, requiring the user to send the current password when setting a new one, is expected to block both local and cross-site attackers from gaining access. We argue that relying on feasibly-guessable challenge-response, such as passwords, to protect against the cross-site

attacker exposes the challenge-response itself to a potentially distributed cross-site brute-force attack. In brute-force attacks, attackers try to break a challenge-response by guessing solutions until the correct one is found.

In the *cross-site (XS) challenge-response* attacks presented in this work, the attacker exploits the lack of dedicated CSRF countermeasures to guess the challenge-response in a cross-site manner. Basically, a cross-site attacker conducts a brute-force attack by sending those requests in a cross-site manner, each time using a different guess for the challenge-response. Throughout the attack, the attacker uses response differentiation techniques to detect whether a guess was successful or not, bypassing the Same-Origin Policy [28] (see Section II-A) when necessary. These techniques are used to identify a specific characteristic of the cross-site HTTP response, for example response size or status code. The fact that the attack can be launched in a cross-site manner, means it costs nothing for the attacker. Moreover, the ability to launch the attack in a distributed manner, allows the attacker to bypass rate-limit restrictions that are based on the source IP. We present two variants of the attack:

Unauthenticated XS challenge-response. This variant is characterized by sending cross-origin challenges to interfaces that do not require authentication. The most common example is a login interface. Obviously, users should not be authenticated to web services in order to send sign-in requests. In this example, the browser of the user visiting the attacking pages is used to send many cross-site login requests, each with a different password guess. The attacker uses cross-site response differentiation techniques to detect whether the requests were successful [19]. When a successful login is detected, this indicates that the password guess was correct. See Figure 1.

Authenticated XS challenge-response. This variant focuses on sending cross-origin requests that require authentication to the account. For example, email modification can usually be done only when the user is authenticated; but here, a password is also needed to complete the change. In this example, the attacker sends cross-origin email modification requests, each with a different value for the current password. The attacker exploits advanced techniques to distinguish between HTTP responses and check each request to see whether or not it succeeded. Similar to the unauthenticated variant, detection of a successful request indicates that a particular password was guessed correctly. See Figure 2.

To the best of our knowledge, this work provides the first extensive study on performing cross-site attacks to steal sensitive information. We evaluate the efficiency of authenticated and unauthenticated XS challenge-response attacks. We further introduce a survey we conducted of the *Alexa top 100 websites in the USA*, showing that many of them are vulnerable to the attack. We also tested the three most common content management systems (CMS) and found that two of them (30% of sites in the world [35]) are vulnerable to the unauthenticated XS challenge-response attack via their login interfaces. Moreover, we surveyed routers and IoT devices that use web interfaces and found that most of them are vulnerable.

In this paper we make the following contributions:

- Introduce two variants of the XS challenge-response attack: authenticated and unauthenticated. We show that using a single challenge-response to protect against cross-site attacks puts the challenge-response itself at risk.
- Describe a new cross-site response differentiation technique based on the Cache API mechanism.
- Analyze browser features and describe how they can be used by an attacker to launch a large-scale attack.
- Survey the Alexa top 100 websites in the USA, most popular content management systems (CMS), and IoT devices and routers to understand the prevalence of XS challenge-response vulnerabilities. We contacted the many vulnerable vendors, some of whom already noted they will block the vulnerabilities, which can affect millions of users.
- Implement and evaluate the attack under real-world conditions, proving that its risks are tangible.
- Propose solutions and defenses, with an emphasis on the ease and practicality of quick deployment.

A. Ethics and Vulnerability Disclosure

The research in this paper exposes risks that exist on many websites, including popular ones, alongside risks in IoT devices and routers. We contacted all the vulnerable vendors surveyed in this paper, most of them, close to the submission deadline. We also created a website, in which we detail their responses, and update it when the vulnerabilities are patched or when vendors send their permission to be listed [11]. In our survey of the top 100 websites, there were several websites that we either failed to contact or for which we cannot publish details. In consultation with experts, we decided not to publish any of the top-100 websites vulnerable to the attack. When all the vulnerabilities are patched, we will publish a comprehensive list. For similar reasons, we do not provide details about the vulnerable models and versions of the tested IoT devices and routers.

Upon request from the general chair, we will deliver the full list to the reviewers. We designed and conducted all the experiments with ethics as our main consideration. For details, see Section VI-A.

II. Preliminaries

This section explains the cross-site attacker model and the attacker's limitations in Section II-A. It lists the browser features that are used to launch the attack efficiently in Section II-B.

A. Cross-site Attacker Model

Modern browsers enforce security restrictions that allow one website to send requests to another website of a different origin, but prevent the sending website from reading the response. A request that is sent from one website to another is called a *cross-site* or *cross-origin* request. The *Origin* of a website is defined by the combination of protocol, host, and

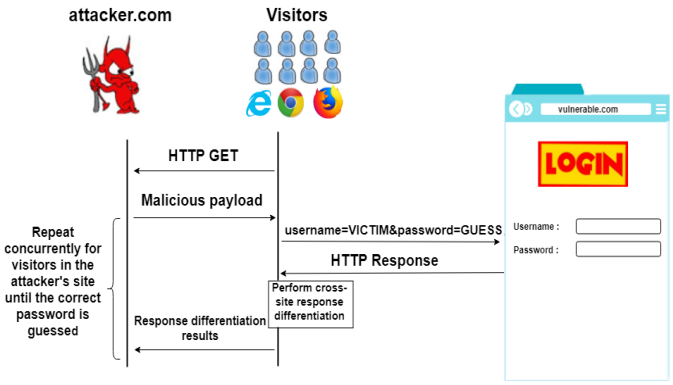

Fig. 1: Example of unauthenticated XS challenge-response attack. The adversary exploits visitors to send cross-origin login requests that try to crack the password of a victim. Then, he uses response differentiation techniques to detect whether password attempt was correct.

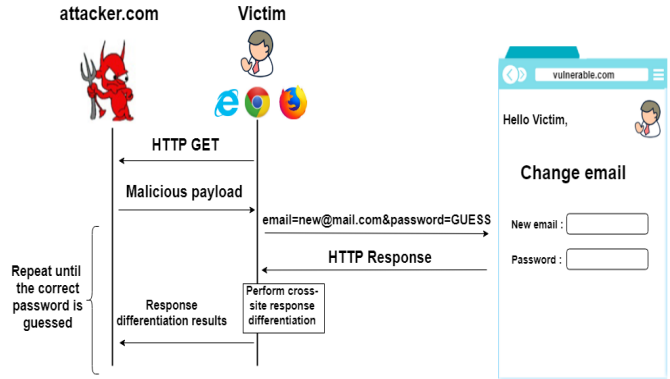

Fig. 2: Example of authenticated XS challenge-response attack. The adversary exploits a victim that visits his website to send authenticated cross-origin email modification requests that try to crack the victim's password. Response differentiation techniques are then used to detect a successful email modification.

port. This restriction is referred to as the *same-origin policy* (*SOP*) [28].

Since cross-site requests are necessary for web connectivity, and can be sent by any visited web page, websites must validate the origin of requests for sensitive operations. Without this validation, cross-site attackers can manipulate users' accounts (CSRF attack [31]) and even steal information [15], [24].

Many approaches have been suggested for blocking CSRF attacks [14], [12], [27]. However, the only approach that is effective without being dependent on client settings (e.g., browser support) involves attaching to each request a secret that cannot be guessed by the attacker. Examples of such mechanisms are the use of CAPTCHA [7], re-authentication, and the use of anti-CSRF tokens. Among the aforementioned solutions, anti-CSRF token is the only one that aims to protect *solely* against cross-site attackers. Anti-CSRF tokens are unpredictable strings that are tied to the current session. The tokens are generated by the website's server, and are embedded in pages the server returns. The server will serve an incoming request only if it contains a valid anti-CSRF token. When tokens are long enough and there are no implementation errors, it is considered unfeasible to guess these tokens. Because the cross-site attackers do not have access to the victim's session, and they cannot generate anti-CSRF tokens that match the victim's session, the server will not serve their cross-site requests.

B. Advanced Browser Features

One of the main challenges of XS challenge-response attacks is the need to send a massive number of requests within a short time interval (i.e., the time in which users stay on the attacking page). The more requests sent, the greater the probability that the brute-force attack will succeed. The attacker can use advanced browser techniques to create a large number of requests for a targeted website. For example, service worker [10] scripts are separate from web pages; they run in the background and are independent of the web page in which they are created.

To maximize the effectiveness of the attack, the attacker can use another advanced browser feature known as the Fetch API. This API, which is available in all modern browsers, can be used to send a batch of HTTP requests in short time intervals.

III. Cross-Site Response Differentiation

The XS challenge-response attack uses advanced side-channel techniques to extract information that cannot be achieved in a cross-site manner. This section surveys techniques that can be used to bypass those limitations noted in Section III-A. In Section III-B, we describe the cross-site response differentiation technique we discovered; this technique is possible due to the behavior of the Cache API mechanism. We use these techniques as a black-box in the description of the attack in the following section.

A. Known Response Differentiation Techniques

The same-origin policy [28] prevents attackers from accessing the content of cross-site HTTP responses. Previous research also suggests several techniques to extract some information. We briefly survey the relevant techniques and pieces of information that can be extracted.

HTTP response size indications. Recent work shows that a cross-site attacker can use advanced timing side-channel techniques to extract the size of cross-site HTTP responses, and obtain personal information regarding a user's state [15], [33].

HTTP response headers difference. Websites may conditionally return responses with different headers, depending on the user's status. For example, Grossman [19] describes how to detect whether a response message was returned with an *X-FRAME-OPTIONS* header.

HTTP response content. Depending on the website implementation, it may even be possible to fully or partially extract the content of HTTP responses. For example, one can exploit a

vulnerable JSONP endpoint to read the contents of a response. JSONP is a method for sharing data in a cross-domain manner. If a JSONP receiving endpoint is vulnerable, an attacker might exploit this to bypass the SOP by injecting malicious Javascript code into the response. An attacker could use the Javascript as defined in Listing 1 to perform cross-site response differentiation based on a vulnerable JSONP callback. It should be noted that dynamically changing Javascript is an action that can be detected, as described by Lekies et al. [22].

Listing 1: Response differentiation based on a vulnerable JSONP callback

```
function abuse_jsonp(){
  s = document.createElement("script");
  s.src = "https://www.vulnerable.com?jsonp=
  malicious_callback";
  document.body.appendChild(s);
}

function malicious_callback(response){
  if(response.length > THRESHOLD){
    handle_valid_challenge();
  }else{
    handle_invalid_challenge();
  }
}
```

B. Response Differentiation Using the Cache API

The methods described in Section III-A are up and running, yet they do not provide a complete set of abilities to differentiate between HTTP responses. Modifications that have been implemented in the cache mechanism limit the efficiency of previous works that deal with cross-origin response differentiation by abusing the behavior of this mechanism. For example, the Chrome browser adds virtual padding to responses, and prevents HTTP response size indications attacks [16]. In addition, works that rely on AppCache are only efficient under special conditions [21]. We discovered a new response differentiation technique to improve the XS challenge-response attacks' ability when it comes to detecting whether a correct challenge was used.

The Cache API provides a storage and retrieval mechanism to resources. Further, this API allows the caching of both same and cross origin resources. To determine the storage usage of cache, one can use the *estimate* method. However, this method does not provide the correct usage of storage, as its main purpose is to prevent the size of cross origin resources from being leaked [18].

We found that caching a resource that redirects to a previously cached resource, results in a slight difference in the used storage. The difference amounts to a few bytes, and it is only affected by the length of the URL for the originally fetched resource. In practice, the pseudo-code listed in Algorithm 1 on the following page allows us to use this aspect to perform cross-site response differentiation. This technique can be employed to support both of the XS challenge-response attacks.

For example, websites often return different responses to HTTP requests, based on the login status of the user. Specifically, an unauthenticated access to authenticated interfaces is usually redirected to the login interface. Examples for such URLs are shown in Listing 2. Given that, cross-origin login detection is feasible. First, an attacker finds a resource that only performs a redirect for a single type of access: authenticated or unauthenticated. Next, two resources will be downloaded and stored in cache: the redirect destination, and the original page itself. After downloading and saving each of the resources, the attacker will measure the storage used by the cache. The difference between the storage usage measurements can reflect the login status of a victim, and indicate whether an unauthenticated XS challenge-response attempt used a correct challenge.

Similarly, some websites return different responses based on the validity of an HTTP request. Listing 3 shows URLs of password modification interfaces that only redirect to another location after a valid password modification attempt is made.

Consequently, we can use our response differentiation technique to confirm whether an authenticated XS challenge-response attack used the correct challenge.

Listing 2: Examples of web pages that redirect unauthenticated access attempts to login interface

```
'https://www.facebook.com/settings' ->
'https://www.facebook.com/login.php?next=...'

'https://vimeo.com/manage/videos' ->
'https://vimeo.com/log_in'

'https://www.reddit.com/prefs/' ->
'https://www.reddit.com/login?dest=...'
```

Listing 3: Examples of change password interfaces that redirect after successful password modification

```
'https://twitter.com/settings/password' ->
'https://twitter.com/settings/passwords/
password_reset_...'

'https://www.tumblr.com/settings/account' ->
'https://www.tumblr.com/login?redirect_to=...'

'https://www.imdb.com/registration/
changepassword' ->
'https://www.imdb.com/registration/
accountsetting...'
```

IV. XS Challenge-Response Attack

This section introduces the XS challenge-response attack. Sections IV-A and IV-B describe the *unauthenticated* and *authenticated* variants of the attack, respectively. Section IV-C summarizes and compares the attack variants. During the

Algorithm 1: Cross-site response differentiation based on the behaviour of the Cache API mechanism

```
Function isRedirect(firstUrl, secondUrl):
    cache.put(firstUrl)
    firstResourceSize = storage.estimate()
    cache.put(secondUrl)
    secondResourceSize = storage.estimate()
    diff = abs(secondResourceSize - firstResourceSize)
    if(diff < secondUrl.length()):
        return True
    else:
        return False
```

description of the attack, techniques that were surveyed in Section II are used as a black-box.

A. Unauthenticated XS Challenge-Response

Unauthenticated requests are requests that do not require an authenticated session attached to them in order to be handled by the server. When unauthenticated requests include a challenge-response, such as a password for login requests, but do not include a dedicated anti-CSRF token, it is possible to launch an unauthenticated XS challenge-response attack. This attack sends the request in a cross-site manner with a guess of the challenge-response and then detects whether or not the guess is correct. Because authentication of the browser is not required, the attacker can send the requests from the browser of every visitor on his malicious web-pages. In short, the attacker can easily launch a low-cost, distributed, brute-force attack.

An HTTP request $\mathcal{R}$ can be abused for an unauthenticated XS challenge-response attack if the following conditions hold:

1) $\mathcal{R}$ will be served by the website even when sent by an unauthenticated user.
2) $\mathcal{R}$ includes a feasibly-guessable challenge-response.
3) $\mathcal{R}$ does not include a dedicated anti-CSRF countermeasure.
4) The responses for $\mathcal{R}$ with valid and invalid challenge-response differ by status code or size, or can be differentiated by any other technique.

The most common countermeasure against brute-force attacks is a rate limit. Here, the server limits the amount of requests that can be sent from a single source. Usually, this is done based on the source IP address or the data that is targeted in the requests.

IP-based rate-limit constrains the rate of requests that are sent from a single IP address. This limit can be easily circumvented by the unauthenticated XS challenge-response variant. The attacker sends the requests in a distributed manner from the browsers of different web-users who surf to the attacking pages.

Data-based rate-limit restricts the number of requests that are related to some piece of information. For example, it is common to limit the number of login attempts to a specific account. Given such a limit, XS challenge-response attacks cannot efficiently break a challenge-response that is related to a particular account. However, the attacker can still abuse the visitors of his website, by trying the limited number of guesses (e.g., most common passwords) for many accounts.

An attacker can also abuse the data-based rate-limit to conduct a distributed denial of service (DoS) attack. This can be done either for specific data (e.g., specific account) or on a large scale for many accounts.

1) Examples: Beyond the login password brute-force attack that was already brought as an example in Section I, there are other cases in which an unauthenticated XS challenge-response attack is possible. Two examples are as follows:

Credit-card brute-force. In a large ticketing website, we found that it is possible to retrieve order data by giving the full credit card number. Although the classical brute-force attack was blocked by an IP-based rate-limit, we easily bypassed this limitation using a distributed XS challenge-response attack.

Identification number and ID. We found a local government website in which login can be done by giving the personal ID and the date the identity document was created. Using the XS challenge-response attack, we were able to simulate the attack on ourselves. We bypassed the IP-based rate-limit, and were able to successfully access private data.

B. Authenticated XS Challenge-Response

Authenticated requests are requests that are only handled by a server if they are attached to an authenticated session. Similar to the unauthenticated XS challenge-response attack, when authenticated requests that include a challenge-response are not protected from CSRF, an authenticated XS challenge-response attack can be performed.

An HTTP request $\mathcal{R}$ can be abused for the authenticated attack under the conditions described in Section IV-A, but with a change of the first condition: the request must be sent as part of an authenticated session. Unlike the unauthenticated variant, the attack can be launched only against users who are authenticated to the target website. Additionally, a visit to the attacking page of browser with an active session of the target site for some user, can be abused only against that particular user. Hence, authenticated XS challenge-response attacks cannot be launched in a distributed manner.

1) Examples: In Section I, we briefly described how an authenticated XS challenge-response attack puts passwords in danger by abusing credential-change requests. Although this seems to be the most common example, we bring two more scenarios in which the authenticated attack is feasible.

Pin code brute-force. We tested a website in which a pin code was required for password modification. We found that no protection from brute-force was implemented in this interface. Because the pin code length was limited, we were able to perform the attack and extract the pin code.

Email address modification. Changing an account's email address is considered a highly sensitive operation, since often the email address can be used to reset the password. In our

	Unauthenticated	Authenticated
Accounts in risk	All accounts	Visitors of the attacking page
Data-based rate-limit effect	Can be abused for DoS attack	Failed
IP-based rate-limit effect	Distributed attack works	Irrelevant

TABLE I: Differences between authenticated and unauthenticated XS challenge-response attacks.

survey (see Section V), we found seven websites that require a password for email modification, but do not use dedicated anti-CSRF countermeasures. A XS challenge-response attacker can abuse the email-modification request to guess the password, change the account's email address, and take over the account. On websites that send notification/verification email to the new address, the attacker can detect the success of the attack without the techniques mentioned in Section III-A; this is because a successful guess of the password triggers the website to send an email to the new email address, which is controlled by the attacker.

C. Authenticated versus Unauthenticated XS Challenge-Response

Table I summarizes the differences between the variants. As shown in the table, the authenticated variant cannot effectively handle a strict rate-limit. However, it is important to note that rate-limit is less common and less strict when applied for authenticated requests. Authenticated requests are harder to forge and remote attackers should not be able to send them at the beginning of the attack. Hence, they are less prone to brute-force attacks. For example, login requests (unauthenticated requests) can be easily abused for brute-force attack, hence, it is more common to apply rate-limit countermeasures on them. On the other hand, usually only logged-in users can change their password (authenticated request). Our survey in the section that follows shows that among the vulnerable websites, rate-limit was more commonly used in the unauthenticated requests we tested. To demonstrate the difference, we conducted a small study and compared the rate-limit policies applied on login requests (unauthenticated) and credential change (authenticated). This was done on the 10 most popular websites in the US [5] and on the 3 most popular CMSs. We dropped duplicates (e.g., Google and YouTube) and websites that do not require passwords for either login or credential changes. The results can be found in Appendix A. Notice, not all the websites in this brief study are vulnerable to the XS challenge-response attack.

V. Real-World Survey

The previous section introduced authenticated and unauthenticated XS challenge-response attacks. To understand whether these attacks present a widespread threat, we conducted a survey of the most popular websites in the US [5] and examined the three most widely used CMS systems [35]. We also surveyed the web interfaces of different IoT devices. Our survey results indicate that XS challenge-response vulnerabilities are quite common. Since we could not test every kind of request that includes sensitive information as challenge-response, we chose to focus on the most common cases for each variant. We focused on *login requests* as unauthenticated requests, and on *credential modification* as authenticated requests. For websites in which email alone can be used to reset the password, we also examined email modification requests. Almost every website with users supports all of the requests noted here. In the survey, we consider a request to be vulnerable to the XS challenge-response attack if it contains a password but does not contain a dedicated CSRF countermeasure. Websites that included only anti-CSRF mechanisms, or did not include any protection mechanisms, were not counted. We provide a more detailed explanation about how we reported the vulnerabilities and the subsequent responses in Appendix B.

A. Top Alexa Websites

Testing XS challenge-response attacks on a large scale is difficult. Moreover, accurately detecting anti-CSRF mechanisms is challenging, given the many different technologies used in websites. This is even more difficult to test using automatic registration for websites and attempting to correctly detect the credential modification request.

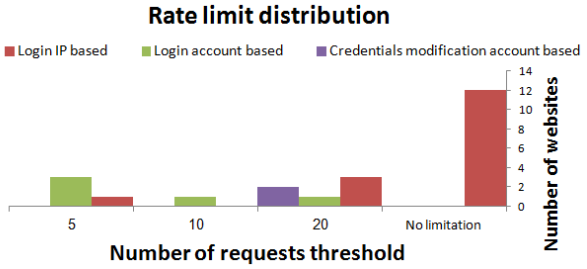

Fig. 3: Number of websites that impose brute-force limitations on authenticated and unauthenticated requests, considering IP-based and account-based rate limits. The IP-based rate-limit is irrelevant for authenticated requests.

Hence, to evaluate the existence of XS challenge-response vulnerabilities, we manually audited the top 100 websites in the US according to Alexa [5] [1]. We examined the login process for each website and classified them based on the security countermeasures used. Then, we created an account for each website and examined the security mechanisms used in their credential modification process. Our survey discovered 21 and 13 XS challenge-response vulnerabilities for the unauthenticated and authenticated variants, respectively. Among the 100 tested websites, 9 were vulnerable to both variants, 12 only to the unauthenticated variant, and 4 only to the authenticated variant.

[1] We used the first 100 websites to which we could register without paying. For example, we did not survey banks.

Among the 13 websites that were vulnerable to the authenticated variant, 12 were vulnerable through password modification, 3 through email modification, and 2 through both of them. Our survey also evaluated the brute-force countermeasures in the vulnerable websites. Among the websites that are vulnerable to the authenticated variant, only 2 applied brute-force protection in authenticated interfaces. Among the 21 websites that are vulnerable to the unauthenticated variant, 8 applied brute-force protection in the login interface. Figure 3 shows the brute-force protection evaluation in unauthenticated interfaces, with consideration for IP-based and account-based limitations. For each vulnerability, we created an exploit and simulated the attack successfully on our own accounts in the vulnerable website. Namely, we succeeded in distinguishing between the requests that included the correct guess and other requests that did not, in a cross-site manner (see Section III-A). We are keeping the identity of the vulnerable websites confidential due to ethical limitations [2]. Our findings indicate that the sensitive data belonging to millions of users registered on those websites is exposed to attacks. Moreover, the results reveal the high volume of vulnerable websites that can be targeted by attackers.

B. Wordpress, Joomla, and Drupal

Since we were not able to manually audit millions of websites, we decided to audit the most popular content management systems (CMS), used by millions of websites. CMSs are applications that facilitate content and user management in websites. *Wordpress, Joomla and Drupal* are the most popular CMSs used today [35]. Wordpress is used by 28.7% of all the websites; Joomla and Drupal are used by 3.2% and 2.3%, respectively. We found that the login interfaces of Wordpress and Drupal are vulnerable to unauthenticated XS challenge-response attack. This is due to their lack of anti-CSRF mechanisms, other than the password itself. Unlike Wordpress, Drupal has an IP-based rate-limit. Yet, this limit cannot prevent effective distributed attacks. None of the systems we audited are vulnerable to the authenticated variant of the attack via password or email modification. In Wordpress, we also considered the use of security plugins. We audited *Wordfence* and *All In One WP Security*, the most popular Wordpress security plugins, with more than $2,600,000$ active installations [38], [8]. The plugins add a layer of protection to Wordpress login by implementing an IP-based rate-limit as a countermeasure to brute-force attacks. However, since they do not implement any CSRF countermeasures, the many websites that use Wordpress with these security plugins are still vulnerable to unauthenticated XS challenge-response attacks.

C. IoT Devices and Routers

At the time of writing, there are approximately 20 billion connected IoT devices [29] in the world. Considering this growing number of IoT devices, we surveyed popular products that use web interfaces in order to detect whether they are vulnerable to unauthenticated XS challenge-response attacks. We did not survey the feasibility of the authenticated variant since users are less likely to be authenticated to IoT web interfaces through their browsers.

IoT devices are considered insecure due to their widespread use of default usernames and passwords. Using automatic tools, attackers can scan for public IoT web interfaces that use default credentials and use the results to take over the devices [30], [25]. The obvious countermeasure is for users to change the password. However, it is possible to launch unauthenticated XS challenge-response attacks on IoT devices if their login interface does not have added anti-CSRF protection. We researched the login interfaces of several IoT devices available to us, including popular models of closed-circuit televisions (CCTV) manufactured by: Sony, Hikvision, Defeway, and Foscam. Out of 14 different devices we surveyed, 8 were found to be vulnerable. We also surveyed the login interfaces of routers available to us. In contrast to IoT devices, most routers are only accessible from the local network of the victim. Hence, attackers cannot launch unauthenticated XS challenge-response attacks in a distributed manner. Many routers use default passwords, and when anti-CSRF mechanisms are not deployed on sensitive functions like changing the DNS server, the routers are exposed to attacks from the local network [26], [1].

Our survey examined the login interfaces of 5 routers available to us from 4 popular manufacturers: TP-Link, D-Link, Huawei, and Asus. Of these routers, 3 out of 5 (from 2 out of 4 manufacturers) were found vulnerable to unauthenticated XS challenge-response attack. Our results indicate that changing the default password of devices does not provide sufficient protection. As long as web interfaces of IoT devices and routers use password as challenge-response without proper CSRF countermeasures, the password can be bruteforced in a cross-site manner.

VI. Evaluation

Section V showed the widespread existence of XS challenge-response vulnerabilities. We created a proof of concept of the attack for each vulnerable website, CMS, and IoT device. Although we measured their rate-limit, we could not test the attack on each of them by sending millions of requests, since this goes beyond what is ethical. This section describes a more in-depth simulation of the attack under real world conditions, where millions of requests might be necessary to complete the attack. Based on the simulations, we evaluated the effectiveness of the attack. Specifically, we conducted two experiments that simulated the variants of the attacks on websites surveyed in Section V-A. Section VI-A outlines the design and the execution process of both experiments. Sections VI-B and VI-C describe the experiments for the unauthenticated and the authenticated XS challenge-response attacks, respectively. Section **??** evaluates the effectiveness of using advanced browser features, as surveyed in Section II-B.

[2]We have not yet succeeded in contacting all of them. Moreover, for some websites we reported the vulnerabilities through bug bounty programs that prohibit publication.

A. Experimental Outline

Design. We chose two vulnerable websites from the survey, one for each XS challenge-response variant. We built a website for each of them, and made the requests and the response in the relevant interfaces completely identical. The only difference in the requests was the hostname. The responses for the requests for both correct and incorrect password guesses were identical to the original websites. To distinguish between correct and incorrect guesses, we used the same methods created for the original websites.

Participants. To perform all the experiments, we recruited 105 students from our institutes; all of them are studying security courses. The students did not know the goal of the experiment, but were told that their participation might lead to slower performance of some of their digital devices for a limited time period. Although we did not expect this to happen, the performance degradation could be influenced by many cross-site requests. The students gave their permission at the beginning of the semester and were told that the experiment would take place sometime during the semester. We encouraged them to detect any manipulation we tried to apply on them. Retrospectively, it turned out that many students expected us to launch phishing attacks on them.

Execution. We added a reference to an external resource for two exercises during the semester. This external resource was a webpage from which we simulated the attack. We chose to lure the victims to the attacking page that way, because in reality, attackers would make similar attempts. For example, attackers can publish a resource that is relevant for students in a forum of students.

We did not embed the malicious script in the attacking page, but included it as an external file. In the title of this script file we wrote a message for the students and asked them to contact us if they read it. This way, we could learn if the attack was detected.

Ethics. To receive IRB approval, we designed the experiments with two purposes in our mind: (1) avoid harming websites and (2) avoid harming users. Below we detail the steps taken.

We easily achieved the first goal by launching the attack on our own servers. It was more challenging to avoid any potential damage to the participants. Although we informed the users that they might feel some performance degradation, we made efforts to avoid it. We first tested the attack on many computers and browsers to make sure the attack did not cause any damage or degrade the user experience. Additionally, we did not launch the attack on mobile devices that surfed to the experiment page, as their bandwidth is sometimes limited. In total, the traffic that was generated by the attacking page was less than an average YouTube page's traffic; the CPU change measured was negligible. The cookies we planted in the users' browsers were not linked to specific users, and did not contain any information other than statistics about the visits to the attacking page. Except for the known fact that the students are participants of security courses, we avoided collecting any additional data about them.

B. Unauthenticated XS Challenge-Response Attack Evaluation

We conducted the experiment on a website that simulated the login interface of a vulnerable website from Alexa Top 100 in USA. The experiment confirmed the feasibility of unauthenticated XS challenge-response attacks that try to crack the password of a predefined user in the targeted website.

On the client side, similar to the original website exploited, we used a SOP bypass, as described in Section III-A, to detect whether a login attempt succeeded. Specifically, we exploited a vulnerable JSONP file that conditionally had different content for unauthenticated and authenticated users. On the server, we stored a large list with half a million different password guesses. When a user surfed to the attacking page, the page retrieved 50 password guesses that had not been tested. The attacking page checked if any of the passwords guessed were correct, and then continued to test another batch of 50 guesses. For efficiency, the attacking page retrieved the next guesses while the previous batch was being tested. Once the test of a batch ended, the attacking page sent a summary of the test to the server. Guesses were retrieved from the list in their order.

We wanted to measure how long it takes to break a password and how much time the user must remain on the site. Because password strength varies between passwords, we had to simulate cases in which the correct password is at the beginning of the list (easy to guess), where a few guesses are enough, and when the correct password appears at higher indexes (harder to guess). In the list of password guesses, we planted the correct password multiple times, in a set of indexes $I_u = \{1K, 10K, 25K, 50K, 100K, 250K, 500K\}$. Figure 5 shows how much time it took to discover the password that appears in index $i \in I_u$, and how much on-site user time was required. All the passwords were guessed correctly without false-positives. The passwords in the lower indexes that represent easy-to-guess passwords, were guessed quickly. The hardest passwords ($250K$ and $500K$ guesses) were completed within a few days. The total on-site user time needed to complete half a million guesses was 6 hours; it was

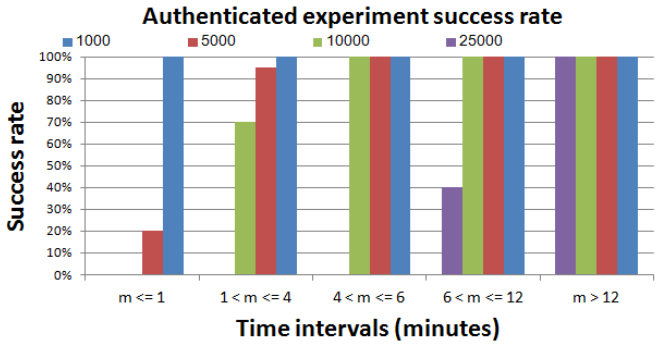

Fig. 4: Success rate in authenticated XS challenge-response experiment for passwords that were guessed correctly in the ith guess ($i \in I_a$), distributed based on the total on-site time of the users.

achieved within 5 days with at most 105 different web users who could visit the attacking page. The more visitors to the attacking page, the faster the attack could be completed.

C. Authenticated XS Challenge-Response Attack Evaluation

In order to validate the authenticated variant of XS challenge-response attacks, we created a website that simulates the credential modification interface of a vulnerable website from the Alexa Top 100 in the US. The attack exploits a form that requires the old password but does not have any dedicated anti-CSRF countermeasures. To detect if a password guess was correct, we relied on the original website's exploit, in which the responses were distinguished by their status code and size (see Section III-A). Unlike the unauthenticated XS challenge-response attack experiment (Section VI-B, the authenticated variant cannot be distributed. The attack is launched against users of the vulnerable site when they are signed into it. The visit of each user is exploited to break his own password. None of the participants were users of the target website that we created. Therefore, upon the first visit of the user to the attacking page, we created an account for the user in the target website. The attack phase was similar to the experiment of the unauthenticated variant (Section VI-B), but the attack was launched separately against each user. On the server side, there was a list of $25K$ guesses. The guesses were used against each account in the target website, once the corresponding participant surfed to the attacking page.

For each account, we put a correct guess in indexes $I_a = \{1K, 5K, 10K, 25K\}$. We measured the number of users for whom we detected a correct guess for every index, as a function of their visit duration on the attacking page. This experiment included 241 different accounts that were created, because some participants used different machines/browsers or cleared their cookies (e.g., incognito mode). Figure 4 shows the success rate of the attack for the different offsets of the correct password (I_a), as a function of the users total on-site time.

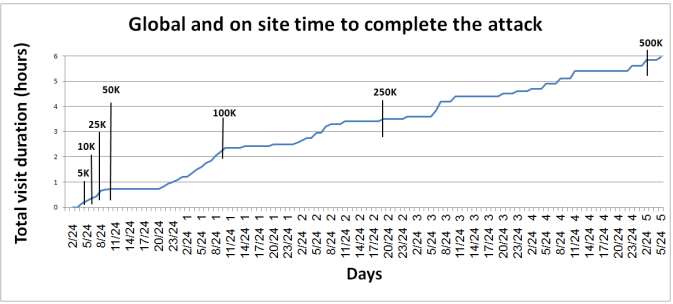

Fig. 5: Total visit duration in the attacking website during the unauthenticated XS challenge-response attack experiment. The black markers indicate successful guesses of the correct password and its offset.

VII. Defenses

The previous sections describe variants of the XS challenge-response attacks. We showed that popular websites, content management systems, and IoT web interfaces are vulnerable to the attacks. This leaves millions of websites and many more users in danger. In this section, we briefly survey possible defenses. Similar to other cross-site attacks [31], [15], the mitigation can be done by blocking cross-site requests. Specifically, every request that includes a challenge-response must be protected with a dedicated anti-CSRF countermeasure.

Servers can detect cross-site requests based on the *Origin* and *Referer* headers in HTTP requests [9]. Because in some cases attackers can omit those headers, the server must block every request that does not include them. Another protection mechanism is validation of the *anti-CSRF token*, which should be sent as a parameter of the request. The token should be randomly generated such that attackers cannot predict its value. A newer CSRF solution uses the *SameSite* attribute for website cookies. A *SameSite* cookie will not be sent along with cross-site requests. Using this attribute ensures that authenticated requests cannot be sent in a cross-origin manner, hence preventing the authenticated XS challenge-response attack. However, at the time of writing, this attribute is only supported by *Chrome and Opera* [23]. A rate limit based on the source IP cannot prevent unauthenticated XS challenge-response attacks, as seen in previous sections. Rate limits based on data that constrain the requests related to a particular account, can be used to limit the effect of the attack. However, without dedicated CSRF countermeasures, such a rate limit could lead to a DoS attack. The attacker can send many requests to reach the rate limit and to prevent legitimate users from doing so. Combining a strict IP-based rate limit and data-based rate-limit can mitigate the DoS threat, since the requests in the unauthenticated XS challenge-response attack do not arrive from the IP address of the attacked account. The use of strong (i.e., long and hard to guess) CAPTCHA [7] is an alternative to rate-limit and to other anti-CSRF mechanisms. Users must solve a challenge to send a request. However, CAPTCHA comes at the cost of degrading the user experience.

It could be argued that XS challenge-response attacks would not be successful without the ability to distinguish between two cross-site HTTP responses (Section III-A), and hence, mitigation should be done there. However, because information leaks occur in all browsers today via side-channels [15], [33], we believe this is not the recommended way to deal with the XS challenge-response threat. It appears that a practical, comprehensive solution for this browser-level problem does not exist, and will not appear in the near future. Hence, solutions should be applied at the website-level.

VIII. Related Work

Cross Site Attacks. Cross-Site Request Forgery (CSRF) is a web application attack that aim to perform an action on behalf on a victim. The most common related attack to our work is *login cross-site request forgery (CSRF)* [9]. In this attack, the attacker sends an unauthenticated request to log visitors of his webpage into his own account on third-party websites. When the victim is signed into the attacker-controlled account, the attacker can extract information about the victim's operations,

e.g., from history features that are available in many websites. A comprehensive survey on CSRF vulnerabilities in popular websites was performed by Zeller et al. [39] to gather information about awaresness of site administrators of the risks and existence of these vulnerabilities. They also offered server and client side tools to protect users from CSRF attacks.

In several websites that do not have history-based features, this attack is not considered a risk. Indeed, bug-bounty programs including Yahoo [4] and Dropbox [3] exclude this threat. Our work shows that even if a login-CSRF cannot be used to harm the victim by logging-in to an attacker-controlled account, it is possible to abuse cross-site login requests to crack passwords. In the unauthenticated XS challenge-response attack, the attacker launches a distributed attack from the browsers of visitors to his website. Antonatos et al. [20] first described this model, and called it *puppetnets*. The authors showed how a group of web clients who visit an attacker-controlled website can be abused to perform malicious activities in a distributed manner.

Cross-Site Response Differentiation. Cross-site response differentiation is a method to retrieve information on responses for requests sent in a cross-site manner. In XS challenge-response attacks, it is necessary to identify whether the correct challenge was used. To overcome this obstacle, we were required to find and implement techniques for distinguishing between different HTTP responses. Some of the methods we used are presented in previous studies.

Van Goethem et al. [16]) discovered several methods for exposing the size of a cross-origin resource. Their new techniques allow the discovery of resources size in short time intervals, using design flaws they found in storage mechanisms. Lee et al. [21] also proposed an attack that allows an adversary to detect the status code of cross-site requests. This attack exploits the cross-origin AppCache mechanism. Gelernter et al. [15], [24] presented timing side-channel attacks in order to extract private information by sending cross-origin requests. These attacks exploit the fact that various search interfaces are not protected by anti-CSRF countermeasures. As a result, an adversary can extract sensitive information about a victim by analyzing the responses of cross-site search requests.

Advanced Browser Features. Our work is not the first to use service workers for malicious purposes. Homakov [2] described how to build a botnet on service workers by exploiting a vulnerability that allows the infinite execution of Javascript code. Van Goethem et al. [33] showed how service workers are used in side-channel timing attacks.

Password Guessing. Work related to effective password guessing is orthogonal to our work, and can be applied to further improve the effectiveness of XS challenge-response attacks. Beyond relying on dictionaries of common passwords to break passwords, several works showed that information about the victim [36] or information about passwords used in other websites [13], [40] can be used to improve password guessing. Other works [37], [17] use training methods on existing password sets to create efficient password generators.

IX. Conclusions

In this paper, we introduced two vulnerabilities of the XS challenge-response class. Contrary to classic cross-site attacks which aim to perform an action on behalf of a victim, our introduced attacks use cross-site vulnerabilities to steal sensitive information. The main observation of our work is that the use of challenge-response without using CSRF countermeasures puts the challenge response at risk.

To understand the prevalence of this class of security vulnerabilities, we performed a real-world analysis of Alexa Top 100 websites in the USA, most popular CMS, IoT devices, and routers, to gain insight into the feasibility and efficiency of the attacks.

We reported our findings to the many vulnerable websites and vendors. Some of them, e.g., Wordfence [38], Reddit, Zillow, and Foscam, already confirmed the vulnerability and are working on fixing it. Further, we created a website [11] to keep an updated list of the vendors' responses. We believe that the publication of this paper will help increase the awareness of XS challenge-response attacks, and encourage websites and vendors to protect their users.

Finally, we proposed solutions and defenses against the attack, using well-known and efficient concepts, such as anti-CSRF token, headers validation and SameSite cookies.

Acknowledgements

We would like to thank Prof. Ehud Gudes and Tomer Brami for their help in promoting this research. This research was supported by a grant from the Ministry of Science and Technology, Israel.

References

[1] *Routers Default Passwords*, December 2012, https://www.itworld.com/article/2716804/security/if-your-router-is-still-using-the-default-password–change-it-now-.html .

[2] *Botnets on ServiceWorkers*, December 2016, https://sakurity.com/blog/2016/12/10/serviceworker_botnet.html.

[3] *Dropbox Bug Bounty Program*, May 2016, https://hackerone.com/dropbox.

[4] *Yahoo Bug Bounty Program*, November 2016, https://hackerone.com/yahoo.

[5] *Alexa Top USA Sites*, February 2017, http://www.alexa.com/topsites/countries/US.

[6] *OWASP SameSite Cookie*, April 2017, https://www.owasp.org/index.php/SameSite.

[7] L. V. Ahn, M. Blum, N. J. Hopper, and J. Langford, "CAPTCHA: Using Hard AI Problems for Security," in *EUROCRYPT*. Springer-Verlag, 2003, pp. 294–311. [Online]. Available: http://dl.acm.org/citation.cfm?id=1766171.1766196

[8] All In One WP Security, *All In One WP Security*, https://he.wordpress.org/plugins/all-in-one-wp-security-and-firewall/.

[9] A. Barth, C. Jackson, and J. C. Mitchell, "Robust Defenses for Cross-Site Request Forgery," in *ACM Conference on Computer and Communications Security*, P. Ning, P. F. Syverson, and S. Jha, Eds. ACM, 2008, pp. 75–88. [Online]. Available: http://doi.acm.org/10.1145/1455770.1455782

[10] Chrome Developers, *Chrome Service Workers*, https://developers.google.com/web/fundamentals/getting-started/primers/service-workers.

[11] Cross-Site Challenge-Response Researcher(s), *Cross-Site Challenge-Response Reports*, 2017, https://xsreports.weebly.com.

[12] A. Czeskis, A. Moshchuk, T. Kohno, and H. J. Wang, "Lightweight server support for browser-based CSRF protection," in *Proceedings of the 22nd international conference on World Wide Web*, 2013, pp. 273–284.
[13] A. Das, J. Bonneau, M. Caesar, N. Borisov, and X. Wang, "The tangled web of password reuse." in *NDSS*, vol. 14, 2014, pp. 23–26.
[14] P. De Ryck, L. Desmet, W. Joosen, and F. Piessens, "Automatic and precise client-side protection against CSRF attacks," in *Computer Security–ESORICS 2011*. Springer, 2011, pp. 100–116.
[15] N. Gelernter and A. Herzberg, "Cross-site search attacks," in *Proceedings of the 22nd ACM Conference on Computer and Communications Security*, ser. CCS '15, 2015, pp. 1394–1405.
[16] T. V. Goethem, M. Vanhoef, F. Piessens, and W. Joosen, "Request and conquer: Exposing cross-origin resource size," in *25th USENIX Security Symposium (USENIX Security 16)*. Austin, TX: USENIX Association, Aug. 2016, pp. 447–462. [Online]. Available: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/goethem
[17] B. Hitaj, P. Gasti, G. Ateniese, and F. Perez-Cruz, "Passgan: A deep learning approach for password guessing," *arXiv preprint arXiv:1709.00440*, 2017.
[18] Jeff Posnick, Google, *Estimating Available Storage Spacety*, https://developers.google.com/web/updates/2017/08/estimating-available-storage-space/.
[19] Jeremiah Grossman, "I Know What Websites You Are Logged-In To (Login-Detection via CSRF)," 2009. [Online]. Available: http://blog.whitehatsec.com/i-know-what-websites-you-are-logged-in-to-login-detection-via-csrf/
[20] V. Lam, S. Antonatos, P. Akritidis, and K. G. Anagnostakis, "Puppetnets: misusing web browsers as a distributed attack infrastructure," in *Proceedings of the 13th ACM conference on Computer and communications security*. ACM, 2006, pp. 221–234.
[21] S. Lee, H. Kim, and J. Kim, "Identifying cross-origin resource status using application cache," in *22nd Annual Network and Distributed System Security Symposium, NDSS 2015*, 2015. [Online]. Available: https://www.cc.gatech.edu/ slee3036/papers/lee:appcache.pdf
[22] S. Lekies, B. Stock, M. Wentzel, and M. Johns, "The unexpected dangers of dynamic javascript." in *USENIX Security Symposium*, 2015, pp. 723–735.
[23] Mozilla Developer Network, *Set-Cookie*, https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Set-Cookie.
[24] Nethanel Gelernter, "Timing Attacks Have Never Been So Practical: Advanced Cross-Site Search Attacks," in *Black Hat USA*, 2016.
[25] New York Times, *NYT Mirai*. [Online]. Available: lhttps://www.nytimes.com/2016/10/22/business/internet-problems-attack.html
[26] M. Niemietz and J. Schwenk, "Owning your home network: Router security revisited," *arXiv preprint arXiv:1506.04112*, 2015.
[27] Paul Petefish, Eric Sheridan, and Dave Wichers, *Cross-Site Request Forgery (CSRF) Prevention Cheat Sheet*, 2015, https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet.
[28] J. Ruderman, *Same Origin Policy for JavaScript*, 2001, https://developer.mozilla.org/En/Same_origin_policy_for_JavaScript.
[29] Statista, *IoT Usage*, https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/.
[30] TechRepublic, *IoT Attacks*. [Online]. Available: http://www.techrepublic.com/article/report-iot-attacks-exploded-by-280-in-the-first-half-of-2017/
[31] The Open Web Application Security Project, *Cross-Site Request Forgery*, 2010, https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF).
[32] B. Ur, S. M. Segreti, L. Bauer, N. Christin, L. F. Cranor, S. Komanduri, D. Kurilova, M. L. Mazurek, W. Melicher, and R. Shay, "Measuring real-world accuracies and biases in modeling password guessability." in *USENIX Security*, 2015, pp. 463–481.
[33] T. Van Goethem, W. Joosen, and N. Nikiforakis, "The clock is still ticking: Timing attacks in the modern web," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2015, pp. 1382–1393.
[34] W3Counter, *Browsers Usage*, https://www.w3counter.com/globalstats.php.
[35] W3Techs, *CMS Usage*. [Online]. Available: https://w3techs.com/technologies/overview/content_management/all

	Unauthenticated requests limit	Authenticated requests limit
Gmail	20	50
Facebook	20	150
Twitter	15	100
Reddit	10	No rate limit
Linkedin	5	200
Amazon	5	600
Netflix	6	150
Espn	5	No rate limit
Imgur	3	1300
Cragislist	3	No rate limit
Wordpress	No rate limit	No rate limit
Joomla	No rate limit	No rate limit
Drupal	5	No rate limit

TABLE II: Rate limit thresholds, surveyed against Alexa Top 10 Websites in the US, and below, for most popular CMS.

[36] D. Wang, Z. Zhang, P. Wang, J. Yan, and X. Huang, "Targeted online password guessing: An underestimated threat," in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*. ACM, 2016, pp. 1242–1254.
[37] M. Weir, S. Aggarwal, B. De Medeiros, and B. Glodek, "Password cracking using probabilistic context-free grammars," in *Security and Privacy, 2009 30th IEEE Symposium on*. IEEE, 2009, pp. 391–405.
[38] Wordfence, *Wordfence*, https://wordpress.org/plugins/wordfence/.
[39] W. Zeller and E. W. Felten, "Cross-site request forgeries: Exploitation and prevention," *The New York Times*, pp. 1–13, 2008.
[40] Y. Zhang, F. Monrose, and M. K. Reiter, "The security of modern password expiration: An algorithmic framework and empirical analysis," in *Proceedings of the 17th ACM conference on Computer and communications security*. ACM, 2010, pp. 176–186.

APPENDIX A
RATE LIMIT SURVEY

This appendix demonstrates the rate-limit difference between authenticated and unauthenticated requests, beyond the results presented in Section V-A (see also Figure 3). We conducted a survey on the Alexa Top 10 websites in the USA that are relevant to the claim [3]. We also surveyed the CMS sites mentioned in Subsection V-B: Wordpress, Joomla, and Drupal. For each, we tested how many requests can be sent until a rate limit is applied in authenticated (credentials modification) and unauthenticated (login) interfaces that use challenge-response. If no rate limit was applied after 1500 requests, we concluded that there is no rate limit.

Similar to the results from Section V-A, the results that appear in Table II, reflect the difference. Rate-limit was less common in the examined authenticated interfaces. Even when applied, the rate-limit in authenticated interfaces was consistently less strict as compared to unauthenticated interfaces. Joomla and Wordpress do not require a password for credential modification or for any other authenticated request that we examined.

[3]We excluded Yahoo.com, Wikipedia.com and Ebay.com, which do not require re-authentication with password to perform sensitive operations. Youtube.com is excluded since it uses the same authentication mechanism as Google.com, which is already surveyed.

APPENDIX B
VULNERABILITIES REPORTS AND RESPONSES

We reported the vulnerabilities described in our paper to websites and system developers. The contact was made through public / private bug bounty programs, email, and contact forms. In our paper, due to ethical considerations, we do not mention the specific names of vulnerable websites and systems that did not manage to fix the vulnerabilities we reported, or did not send a response.

A. Top Alexa Websites

Out of 25 vulnerable websites to which we sent a report, we can currently bring only the responses of Reddit and Zillow. **Reddit's** security team confirmed the vulnerability and fixed it by adding an anti-CSRF token to the login interface. **Zillow's** security team plans to implement a solution that will help mitigate the security risks of the attack.

B. CMSs

We contacted Wordpress, Wordfence, All In One WP Security, and Drupal. All confirmed our findings that make their systems vulnerable to XS Challenge-Response attacks. **Wordpress** said that individual websites should decide how to implement security solutions by using plugins, firewall rules, or monitoring systems. **Wordfence**, the largest Wordpress security plugin, confirmed that they will block the vulnerabilities, and are discussing possible solutions such as CAPTCHA or account lockout to fix the vulnerability. **All In One WP Security**, the second largest Wordpress security plugin is also considering fixing the issue. **Drupal** confirmed the vulnerability, and told us that future, possibly public discussions should be held to find a reasonable mitigation (that will not downgradge the user experience and performance).

C. IoT Devices and Routers

Currently, we can bring two responses from IoT or router vendors. The CCTV manufacturer **Hikvision** is taking this threat seriously and confirmed the vulnerability. However, they mentioned that their new devices are not vulnerable to XS Challenge-Response attacks, as they changed their design to counter general CSRF attacks. That said, old models cannot be remotely patched and updated, and are still vulnerable to the attacks. The CCTV manufacturer **Foscam** is examining the protection mechanisms described in our paper in order to fix the login interfaces of CCTV devices.