2022 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW) | 978-1-6654-7679-9/22/$31.00 ©2022 IEEE | DOI: 10.1109/ISSREW55968.2022.00043

Detecting and Defending CSRF at API-Level

Shun Wang[†§], Chao Ni[*], Jianbo Wang[‡], and Changhai Nie[†]
[†]State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China
[*]School of Software Technology, Zhejiang University, Ningbo, China
[‡]School of Petroleum Engineering, Northeast Petroleum University, Heilongjiang, China
roy.wang123@gmail.com, chaoni@zju.edu.cn, w1186855763@163.com, changhainie@nju.edu.cn

***Abstract*—Cross-Site Request Forgery (CSRF) vulnerabilities are severe web vulnerabilities since their characteristics of extreme concealment and heavy harmfulness. However, they have received marginal attention from both the academic and the industry and the detection and protection of CSRF vulnerabilities are still performed predominantly manually.**

This paper proposes CSRFSolver for API-level CSRF detection and protection with two components: CSRF detector and CSRF defender. The former helps to identify and locate CSRF points where they need CSRF protection, and the latter provides CSRF protection by generating and verifying CSRFToken. We evaluate the effectiveness and efficiency of CSRFSolver on Cisco Webex public URL APIs with the state-of-the-art method. The results indicate that CSRFSolver can effectively and efficiently protect the system from CSRF attacks and have no side effects on systems' functionality. Meanwhile, the practical usefulness of CSRFSolver has also been verified through four years of deployment in Cisco Webex.

***Index Terms*—Cross-Site Request Forgery, Security Defense, CSRFToken**

I. Introduction

Cross-Site Request Forgery (CSRF) [1] vulnerabilities have been continuously ranked as one of the most dangerous web security attacks, which also include SQL injection [2] attacks, Cross-Site Scripting (XSS [3]) attacks, Access Control [4] attacks and so on. In the untrusted/vulnerable web application, even a brief visit may induce the victim's browser to perform authenticated, security-sensitive operations without the victim's awareness or consent. Therefore, successful CSRF attacks may result in serious loss to users including illicit money transfers [5], user account takeover [6], or remote server-side command execution [7]. Meanwhile, existing studies have found that CSRF attacks are ubiquitous on the internet, including ING Direct [8], Netflix [9], Mexican Bank [10] as well as Gmail [11].

Despite its popularity, CSRF has not received much attention from the academic to the industry since its characteristic of extreme concealment. Most of the previous works aim at proposing active [12] or passive defense mechanisms [13]. Besides, as for practical usages, we find that the API layer's CSRF vulnerabilities widely exist in several APIs provided by Cisco's online conferencing, and Cisco's APIs are widely used in many IT companies. Cisco's APIs contain both hundreds of public URL APIs (e.g., server XML APIs as well as other functionally specific APIs) and several internal APIs. All of these APIs may potentially expose users to danger. Unfortunately, little approaches has been proposed to help practitioners to effectively detect this class of vulnerabilities, and none of the existing work are easily applicable to CSRF vulnerability especially for API-level CSRF ones.

Our Approach – We take a step forward by proposing CSRFSolver to enable the detection and protection of CSRF vulnerabilities. To the best of our knowledge, this is the first work that targets the API-level CSRF detection and protection, which has two components: CSRF detector and CSRF Defender. As for the former one, it helps to identify and locate CSRF points where the URL may need CSRF protection by analyzing the intrinsic mechanism of the CSRF attack. As for the latter one, it provides the CSRF protection by adding the generation and verification of CSRFToken on the original API service.

To evaluate the effectiveness and efficiency of CSRFSolver, we compare it with the state-of-the-art baseline (e.g., Deemon [14]) on Cisco Webex public URL APIs. The results indicated that CSRFSolver can effectively and efficiently achieve good performance (i.e., 98.7%) of accuracy, which improves Deemon by 13.3% and uses only about 5% execution time of Deemon does. Meanwhile, CSRFSolver can protect CSRF attacks on both database-related ones and file system-related ones, while Deemon can only protect against CSRF attacks on the database. Moreover, we conduct stress testing on a system equipped with CSRFSolver and find that CSRFSolver has no side effects on original APIs' functionality and has little effect on response times on average. In a special scenario (i.e., "Missing or Invalid CSRFToken"), CSRFSolver can quickly prevent CSRF attacks and decrease response times by 80.2% on average. Lastly, the practical usefulness of CSRFSolver has also been verified through four years of deployment in Cisco Webex, which also verifies the portability of configuration and fills the gap of protection from CSRF attack at API-level.

To summarize, we make the following contributions:

- To the best of our knowledge, we are the first one to propose a novel approach named CSRFSolver for protecting CSRF attacks at the API level. CSRFSolver contains two components: CSRF detector and CSRF defender. The CSRF detector helps to identify and locate CSRF points that need CSRF protection, while the CSRF defender helps to prevent the vulnerable point at the API level.
- We comprehensively investigate the value of CSRFSolver in terms of effectiveness and efficiency. The results indi-

[§]Work done while at Cisco.
Chao Ni and Changhai Nie are the corresponding authors.

978-1-6654-7679-9/22/$31.00 ©2022 IEEE
DOI 10.1109/ISSREW55968.2022.00043

cate that CSRFSolver effectively and efficiently outperforms the state of the art (e.g., a higher accuracy by 11.6 percentage points and a lower response time with 5% execution time of baseline's).

II. Background

This section presents the basic knowledge about concepts of cross-site request forgery (CSRF), the working mechanism of CSRF as well as the attacking mode of CSRF.

A. Cross-Site Request Forgery

Cross-site request forgery [15], abbreviated as CSRF or XSRF, is a malicious usage of web applications. It is used to deceive the server through the third party to forge the user request by impersonating users' identification and users' rights. This kind of attack usually takes effect only if a user clicks it, therefore it also has the name "one-click attack" [16]. Meanwhile, some people also treat it as "the session riding" [15] since it needs the user authenticated identity session to work. Once the CSRF attack happens successfully, it will cause the results of "confused deputy" [17].

B. CSRF Working Mechanism

The main reason that CSRF widely exists is the defects of the current web identity authentication, including the following mechanisms: (1) implicit authentication [18], (2) same origin policy [19], (3) cross-domain resource sharing, (4) cookie security policy [19] and (5) flash security policy [20].

Implicit Authentication. Most web applications use Cookie/Session to identify the users and maintain their session state. when the user completes the authentication, the user's successor operations of accessing the site do not need to relogin certification and the browser can identify each request automatically [20, 21] since the site has been certified such a cookie/session successfully. This type of authentication is named implicit authentication [22, 23]. The problem with implicit authentication [18] is that once a user login into a website and clicks on a link to enter any page, his authenticated identification on this site could be illegally exploited.

Same Origin Policy. Same Origin Policy (SOP) [24] means that the browser accesses an address requiring a network security protocol with the same protocol, same domain name, and same port [20]. However, the SOP does not prevent the script which requests to access other sites. Therefore, it is one of the flaws in SOP.

Cross-Origin Resource Sharing. The SOP [25] is used to ensure that non-same resource is not allowed. However, in practice, users usually need to request other domain resources. Cross-Origin Resource Sharing (CORS) defines how browsers and servers should communicate when cross-origin resource access is requested. CORS, by default, does not exchange cookies except *AccessControl-Allow-Credentials* is allowed, which makes it possible for CSRF attacks and increases the risk of CSRF attacks.

Cookie Security Policy. The cookie [26] includes two types: persistent one and temporary one. The persistent cookie will facilitate the CSRF attack therefore it is not recommended to set the identity of the cookie to be persistent. As for the temporary cookie, it is mainly stored in session [27] and is destroyed if the user logins out or closes the browser. Therefore, CSRF may use the logged-in users' authentication to complete risky operations.

Flash Security Policy. Improper configurations on *cross-domain.xml* [20] will risk the websites since it stores sensitive information and could result in loss of sensitive information.

To sum up, the mainstream security strategy of websites is not good enough to prevent CSRF attacks, to a certain extent, cannot guarantee the user's network security.

C. CSRF Attacking Mode

CSRF attacks are covert and lead to the users unconsciously visiting the attacker's carefully constructed web page or clicking on an unknown website link. Usually, a CSRF attack involves three phases: (1) looking for the occasion, (2) setting traps, and (3) launching an attack.

(1) Looking for the Occasion. Considering the secrecy of CSRF attacks, users cannot easily identify such an attack. Besides, CSRF can not only attack across domains but also attack in the same domain [28]. Risky sites, messages, pictures, or links are the trigger of attacks.

(2) Setting Traps. A variety of beautiful pictures, coupons, or rewards are carefully designed in web pages or emails, which are used to lure a user into a trap. If a CSRF attack is stored on a website database, it is called storage CSRF [29], otherwise, it is referred to as reflective CSRF (e.g., emails).

(3) Launching an attack. Firstly, the user logins in the websites with correct authentication. Then, the cookies are exchanged for the following operation without login again. Following that, the user may click on a link with risk, which causes the user drop into a trap in the authenticated site.

In conclusion, the CSRF attack mainly requires the users to click the particular links to lure them into a trap.

III. Methodology

In this section, we propose a novel method named CSRFSolver for CSRF attacks involving two main components: (1) CSRF detector and (2) CSRF defender. The CSRF detector helps to identify and locate CSRF points while the CSRF defender helps to prevent the vulnerable point at the API-level according to the detection results from the CSRF detector. Figure. 1 illustrates the overall framework of CSRFSolver and the details of each component are introduced as follows.

A. CSRF Detector

CSRF attacks are highly covert, destructive, and easy to be overlooked. Therefore, effectively identifying and locating CSRF attack points is extremely important, which can further help to design software architecture to prevent these attacks.

However, at present, there is no tool/approach working well on the identification and location of CSRF vulnerability since most of the CSRF attacks are deliberately designed traps. Existing tools, including static scan tools (e.g., Coverity and Jtest) and security dynamic penetration tools (e.g., AppScan

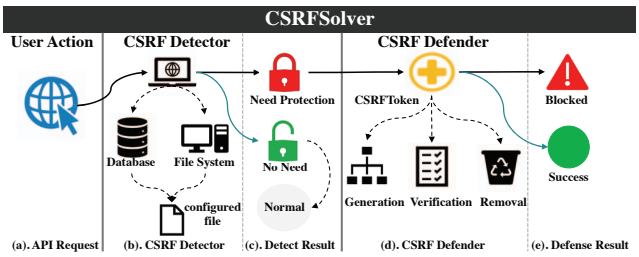

Fig. 1: The framework of CSRFSolver.

and Zap), can only scan a few potential CSRF attacks and the resutls need to be manually confirmed.

Considering the importance of CSRF attacks, the Cisco Webex security team has paid much attention on researching the core function of the web application for CSRF protection based on the priority of the possibility of attacks. However, API-level CSRF protection is still not treated seriously for the following reasons:

- The public API interface has been provided to users. Changing the interface may cause a huge impact on customers.
- The quantity of API as well as its type is huge, which brings the difficulty to solve these threats at one time.
- The API's security development and security testing are often overlooked since its complex protocols and data structures (e.g., Rest, JSON, and XML).

Through many years of experience in tracking and practicing in various fields of web security, we figure out the intrinsic attacking mechanism of CSRF and design a CSRF detector named QIL-CSRF for quickly identifying and locating CSRF.

The workflow of the CSRF Detector can be described as follows. When a request to the server requires authentication, the user must pass through the permission verification successfully. These operations will cause the addition/deletion/modification to the database or file system, which will bring the opportunity for the attacker to forge the request as a CSRF attack and consequently lead to the threats to users' data, privacy, permission, and so on.

Therefore, by monitoring the working flow, CSRF Detector can quickly identify and locate these URLs which are needed to be protected from CSRF attacks. Meanwhile, these URLs will be also stored in "csrftoken-verify-list.xml" file, which is a special configuration file recording all information of URLs for protection. In addition, CSRF Detector also adds these URLs to another special file named "csrftoken-issue-list.xml" to facilitate subsequent resolution of these CSRF attacks.

B. CSRF Defender

CSRF Detector aims to quickly and accurately identify and locate CSRF vulnerabilities, but it cannot protect such a vulnerability. Though many works have been proposed for defending against CSRF attacks, most of them are neither effective nor suitable for all scenarios. Therefore, we propose a light but effective CSRF Defender at the API level based on our practice experience in industry as well as the in-depth understanding of the intrinsic mechanism of CSRF attacks.

The core of CSRF Defender on protecting CSRF attacks is the addition of CSRFToken parameters. CSRFToken is tightly bound to the users logged in to a system. A prerequisite for the success of a CSRF attack is that an attacker uses a victim to log on to a site for legal status. Therefore, for resources without the need to log in to a system, they do not need to utilize CSRFToken protection and consequently reduce the servers' pressure for operations in CSRF protections. CSRF Defender has at least the following advantages: (1) Light-weighted. Compared with existing approaches (e.g., OWASP[1]), CSRF Defender does not need database service to store and validate CSRFToken values since our CSRF Defender is designed as a dynamical variable; (2) Flexibility and Configuration-Ability. CSRF Defender is managed with the configurable file (i.e., "csrftoken-verify-list") to easily add new APIs or delete old APIs which may need or remove the CSRF protection. CSRF Defender can also specify the particular URLs to provide CSRFToken protections.

Overall, CSRF Defender involves three main steps for CSRF protection: CSRFToken Generation, CSRFToken Verification and CSRFToken Removal. We introduce them as follows.

CSRFToken Generation. When a user calls an API, the system will generate a CSRFToken and return it to the API caller. Afterward, all the subsequent operations executed by the caller on this API must be verified with the CSRFToken generated by the system. In our implementation, we adopt the instantaneous and unique value generated by UUID functionality, which is a machine-generated identifier involving MAC address, timestamp, namespace, random or pseudo random number, and time series. Besides, CSRF Defender stores the generated CSRFToken in the cookie and to ensure the security, it sets both "setSecure" and "setHttpOnly" as true to avoid stealing cookies.

CSRFToken Verification. Though CSRF Detector can quickly and accurately identify and locate CSRF vulnerabilities, not all APIs need CSRF protection. In our implementation, the APIs that need to be protected are configured in the special file (i.e., "csrftoken-verify-list.xml"), which indicates the risk of CSRF attack in these URLs and all requests from these URLs are needed to be verified on its CSRFToken.

The CSRFToken verification is to calculate the CSRFToken checksum. In the process, CSRF Defender first determines whether CSRFToken parameter exists or not. Then, CSRF Defender will return error information directly if the CSRFToken parameter is empty or the value of CSRFToken is *NULL*, which will cause the abort of the program and consequently block the potential CSRF attacks.

For the scenario that CSRFToken is not empty, CSRF Defender extracts the CSRFToken value in the parameters and obtains the CSRFToken value from the user's local cookie. Then, CSRF Defender makes a comparison to check the consistency between the value from the user's cookie and the value stored in the server-client. For inconsistent scenarios, for example, if the value of the current CSRFToken does not

[1]https://owasp.org/www-project-csrfguard/

belong to the user or the value is false or expired, CSRF Defender will abort the program and prevent the potential CSRF attacks. For a consistent scenario, all requests to API functions will be accepted and executed normally.

CSRFToken Removal. When the user wants to log out of a system, the generated CSRFToken assigned to the user will be destroyed by setting its value as an empty one in the cookie. For the next logging, another new completely different CSRFToken will be generated for the same user, which prevents the API-level CSRF attacks from attackers since they cannot easily predict the CSRFToken.

IV. Experiments and Results

In this section, we introduce the experimental settings, research questions as well as results.

A. RQ1: How well does CSRFSolver perform on API-level CSRF attack protections?

Motivation. CSRF attacks are highly covert and are easy to be overlooked, therefore, they will cause huge risk to a system and consequently damage the users' data and privacy. However, most of the existing approaches can only work in specific scenarios, which is unsuitable for other usage cases. In this paper, we propose a novel method named CSRFSolver to address CSRF attacks at the API-level, which has good adaptability. Therefore, we want to investigate the effectiveness of CSRF attacks.

Method. We conduct experiments on Cisco Webex public URL API, which totally contains 78 public APIs with the need for CSRF token protection. Besides, to the best of our knowledge, there is no study on API-level protection for CSRF attacks. Therefore, to better indicate the performance of our approach, we adopt the state-of-the-art approach named Deemon [14] as our baseline. Deemon is the first automated security testing framework to discover CSRF attacks and is based on a new modeling paradigm that captures multiple aspects of web applications, including execution traces, data flows, and architecture tiers in a unified and comprehensive property graph.

We evaluate the performance of the two methods by considering two aspects: effectiveness and efficiency. For evaluation of effectiveness, we adopt the widely used performance metrics accuracy. For evaluation of efficiency, we report the running time of the two methods.

TABLE I: Comparison between CSRFSolver and Deemon on protection performance and protection efficiency

Method		# DB_A	# DB_M	# DB_D	# FS_A	# FS_M	# FS_D	# All
Perf.	CSRFSolver	22	36	10	5	3	1	77
	Deemon	22	36	10	–	–	–	68
Time	CSRFSolver	6.7	16.5	3.1	7.9	4.9	1.4	40.5
	Deemon	131	272	76	–	–	–	479

∗**DB**: Database, **FS**: File System, **A**: Add, **D**: Delete, **M**: Modify. **Perf.**: Performance. **Times**: Second.

Results. Table. I shows the experimental results of CSRFSolver and Deemon. Each column shows the functionality of the corresponding protection type for CSRF attacks. Firstly, our approach CSRFSolver can support the attacks on the database as well as the attacks on the file system. However, Deemon can only protect the attacks on the database. Therefore, CSRFSolver is more powerful than Deemon. As for attacks on the database, we find that CSRFSolver achieves the same results with Deemon, which indicates the effectiveness of the two methods. However, CSRFSolver can perform extremely quickly than Deemon using about 5% times of Deemon. Overall, CSRFSolver can successfully prevent 77 attacks achieving 98.7% accuracy, while Deemon successfully prevents 68 attacks achieving 87.1% accuracy. For the only one (i.e., p.php?AT=LO[2]) escaping from the protection of CSRFSolver, we find that this API is used to log out of a system. However, it neither causes the change to the database nor makes changes to the file system. Therefore, CSRFSolver cannot successfully identify that it needs to be protected against CSRF attacks.

B. RQ2: How does CSRFSolver affect the performance of the original system?

Motivation. CSRFSolver can protect the system from CSRF attacks by adding CSRFToken generation and verification. Meanwhile, it should not have an impact on the functionality of the original system and it should affect the service performance (i.e., response time) as least as possible. Therefore, we want to check whether it has side effects on systems' functionality and performance.

Approach. As for the functionality verification, we check whether the original system and the new one (i.e., CSRFSolver) have the same results. Therefore, if CSRFSolver is executed successfully by calling the same API, the running result should be the same one. In addition, we also use the stress testing with JMeter to simulate hundreds of concurrent calls to some APIs. Then, we can compare the response time difference between the one without CSRF protection and the one with CSRF protection.

TABLE II: The response time of the system with (without) CSRF protection. (Unit: millisecond)

# Requests	# Avg.(ms)	# Min.(ms)	# Max.(ms)	# No-CSRFToken
250	61 (63)	20 (20)	237 (244)	21
500	97 (91)	20 (20)	230 (225)	20
1,000	140 (159)	19 (20)	238 (300)	20
2,000	146 (146)	20 (19)	254 (265)	20
4,000	131 (140)	19 (18)	245 (324)	21

Results. The comparison results between the system equipped with CSRFSolver and the system without CSRFSolver are shown in Table. II. In this table, we report the system response time for different numbers of concurrent requests. The last column reports the response time in the scenario that there is no CSRFToken or the CSRFToken is incorrect. That is, CSRFSolver can quickly return "Missing or Invalid CSRFToken" error and interrupt the execution of such an attack.

First, we compare the functionality of the system equipped with/without CSRFSolver and find that there is no obvious difference in functionality. That is, CSRFSolver executes correctly and the calls to API are not changed functionally.

[2]https://community.cisco.com/t5/cloud-collaboration/webex-api-rs-missingorinvalidcsrftoken/m-p/3551198

Second, according to the response time, we also find that on average the system equipped with CSRFSolver performs similarly to the system, which indicates that CSRFSolver will not affect the original system's performance. In addition, compared with one without or with invalid CSRFToken, the systems' response time will be increased by 80.2% on average.

V. Related Work

Considering the heavy impact of CSRF attacks, several approaches have been proposed to address CSRF attacks in the literature and can be classified into the following types.

Changing GET mode to POST mode [8]. Developers originally believed that the main reason why CSRF is successfully attacked is the simulation of a running URL. That is, the attacker simulates a running URL and the parameters in the URL displayed in the browser's address bar, which makes it easy for attackers to forge the same URL. Afterward, developers discovered that this protection only slightly increases the threshold of the attackers. It is easy to change the service request mode from GET to POST. However, this kind of protection is relatively not truly a CSRF attack but adds the threshold from an attack.

Protecting HTTP Referer header [30]. Developers thought the main reason for CSRF attach is the cross-domain forged URL requests. Then, developers believed that checking the HTTP Referer header is an effective way to avoid CSRF attacks. It is not allowed to access if the link to the domain and the site are inconsistent with the original one. This does bring a certain protective effect. However, the current browser Referer header can tamper and the Referer header in the pop-up windows is usually empty, which leads to the defects in Referer header solution. Meanwhile, Referer headers can only prevent cross-domain attacks. For CSRF attacks on the same domain, it can do nothing.

Installing a plug-in/extension in the browser [31]. This plug-in is used to restrict whether a link can jump to another one or not. Many security engineers and programmers think it is a good way since it does not need to make changes to the code of each application. However, such a solution is not user-friendly from two aspects: (1) users may be unaware of such types of plug-ins for safely accessing the website; (2) users may not be good at configuring the plug-in. Meanwhile, browser plug-in protection can only prevent cross-domain CSRF attacks and it cannot protect the attacks from the same domain. Lastly, the client plug-in can be easily disabled by other programs. Therefore, once the plugin is disabled, this protection is invalid.

Confirmation on key operation [15, 32] or two-times authentication (e.g., Graphics verification code, SMS authentication code, payment password, etc.). For example, you may need to relog in when you need to log out (e.g., Amazon). Besides, some applications require the final checkout operation with entering the payment password (e.g., Alipay). Similar to online payment, the system will send dynamic code through a message to the user to confirm the payment behavior, which is widely used by many companies. As for key operations, it often requires the user to enter a graphics verification code or to select the appropriate graphics (e.g., 12306 ticketing system). If the user does not confirm this operation with appropriate actions, such an operation will be blocked and consequently, it avoids unnecessary losses.

CSRFToken protection [33]. Considering the dynamic of CSRFToken, the attackers cannot predict in advance. Therefore, each key link cannot be constructed by the attacker and finally, it avoids the CSRF attack. This method is the most secure CSRF protection and is the most widely used one by software companies. It also does not change users' behavior and does not increase the burden of frequent user certification solutions.

Different from existing work, our CSRFSolver uses UUID to generate CSRFToken rather than the MD5 algorithm, which ensures efficiency for generation and verification. Besides, CSRFSolver can generate CSRFToken only when the user logins into a system and it only validates risky links confirmed by CSRF Detector. Lastly, CSRFSolver does not need a database for saving and validating CSRFToken.

VI. Conclusion and Future Work

In this paper, we propose a novel approach named CSRFSolver for addressing CSRF attacks which contains two components: CSRF defector and CSRF defender. The CSRF detector helps to identify and locate CSRF points while the CSRF defender helps to prevent the vulnerable point at the API-level according to the detection results from the CSRF detector. To verify the effectiveness and efficiency of our approach, we conduct several experiments and the results indicate that CSRFSolver can effectively and efficiently prevent CSRF attacks and have no side effects on system's performance.

Our future work involves extending our evaluation by considering more usage scenarios (e.g., CSRF attacks on the wireless router, mobile phones, cloud computing, etc.)

ACKNOWLEDGMENT

This work was supported in part by the National Key Research and Development Program of China (2018YFB1003800), National Natural Science Foundation of China (61902174,62072226), and Natural Science Foundation of Jiangsu Province (BK20190291).

References

[1] R. Feil and L. Nyffenegger, "Evolution of cross site request forgery attacks," *Journal in Computer Virology*, vol. 4, no. 1, pp. 61–71, 2008.

[2] H. Gu, J. Zhang, T. Liu, M. Hu, J. Zhou, T. Wei, and M. Chen, "Diava: A traffic-based framework for detection of sql injection attacks and vulnerability analysis of leaked data," *IEEE Transactions on Reliability*, vol. 69, no. 1, pp. 188–202, 2020.

[3] Y. Zhou and P. Wang, "An ensemble learning approach for xss attack detection with domain knowledge and threat intelligence," *Computers & Security*, vol. 82, no. MAY, pp. 261–269, 2019.

[4] Y. Chen, W. Sun, Z. Ning, Q. Zheng, and Y. T. Hou, "Towards efficient fine-grained access control and trustworthy data processing for remote monitoring services in iot," *IEEE Transactions on Information Forensics and Security*, vol. PP, no. 99, 2018.

[5] W. Zeller and E. W. Felten, "Cross-site request forgeries: Exploitation and prevention," *The New York Times*, pp. 1–13, 2008.

[6] A. Sudhodanan, R. Carbone, L. Compagna, N. Dolgin, A. Armando, and U. Morelli, "Large-scale analysis & detection of authentication cross-site request forgeries," in *2017 IEEE European symposium on security and privacy (EuroS&P)*. IEEE, 2017, pp. 350–365.

[7] J. Martin, "The three faces of csrf," 2007. [Online]. Available: https://deepsec.net/archive/2007.deepsec.net/speakers/index.html#martin-johns

[8] K. Sentamilselvan., P. S. Lakshmana, and N. Ramkumar., "Cross site request forgery: Preventive measures," *International Journal of Computer Applications*, vol. 106, no. 11, pp. 20–25, 2014.

[9] M. S. Siddiqui and D. Verma, "Cross site request forgery: A common web application weakness," in *IEEE International Conference on Communication Software & Networks*, 2011, pp. 538–543.

[10] Shalu and N. Kumar, "A survey on cross site request forgery web application attack."

[11] P. D., "Google gmail e-mail hijack technique," 2007. [Online]. Available: http://www.gnucitizen.org/blog/google-gmail-e-mail-hijack-technique

[12] M. Johns and J. Winter, "Requestrodeo: Client side protection against session riding," in *Proceedings of the OWASP Europe 2006 Conference*, 2006.

[13] A. Barth, C. Jackson, and J. C. Mitchell, "Robust defenses for cross-site request forgery," in *Proceedings of the 15th ACM conference on Computer and communications security*, 2008, pp. 75–88.

[14] G. Pellegrino, M. Johns, S. Koch, M. Backes, and C. Rossow, "Deemon: Detecting csrf with dynamic analysis and property graphs," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 1757–1771.

[15] W. A. Shaik and R. Pasupuleti, "Avoiding cross site request forgery (csrf) attack using twofish security approach," 2015.

[16] Z. Mao, N. Li, and I. Molloy, "Defeating cross-site request forgery attacks with browser-enforced authenticity protection," *DBLP*, 2009.

[17] E. Shernan, H. Carter, D. Tian, P. Traynor, and K. Butler, "More guidelines than rules: Csrf vulnerabilities from noncompliant oauth 2.0 implementations," *International Conference on Detection of Intrusions & Malware*, 2015.

[18] M. Johns and J. Winter, "Requestrodeo: client side protection against session riding."

[19] F. Kerschbaum, "Simple cross-site attack prevention," in *International Conference on Security & Privacy in Communications Networks & the Workshops*, 2013.

[20] C. Zhen, "Principle analysis and countermeasure research of csrf attack," *Fujian computer*, vol. PP, no. 6, 2009.

[21] R. Sankuru and M. Janjanam, "Web application security-cross-site request forgery attacks," 2014.

[22] W. Maes, T. Heyman, L. Desmet, and W. Joosen, "Browser protection against cross-site request forgery," in *ACM workshop on Secure execution of untrusted code*, 2009.

[23] P. D. Ryck, L. Desmet, T. Heyman, F. Piessens, and W. Joosen, "Csfire: Transparent client-side mitigation of malicious cross-domain requests," in *International Symposium on Engineering Secure Software & Systems*, 2010.

[24] B. Stritter, F. Freiling, H. Knig, R. Rietz, S. Ullrich, A. V. Gernler, F. Erlacher, and F. Dressler, "Cleaning up web 2.0's security mess-at least partly," *IEEE Security and Privacy Magazine*, vol. 14, no. 2, pp. 48–57, 2016.

[25] H. Shahriar and M. Zulkernine, "Client-side detection of cross-site request forgery attacks," in *IEEE International Symposium on Software Reliability Engineering*, 2010.

[26] E. Porat, S. Tikochinski, and A. Stulman, "Authorization enforcement detection," in *Acm on Symposium on Access Control Models & Technologies*, 2017, pp. 179–182.

[27] H. Selim, S. Tayeb, Y. Kim, J. Zhan, and M. Pirouz, "Vulnerability analysis of iframe attacks on websites," in *The 3rd ACM Multidisciplinary International Social Networks Conference*, 2016.

[28] Y. C. Sung, M. Cho, C. W. Wang, C. W. Hsu, and S. W. Shieh, "Light-weight csrf protection by labeling user-created contents," in *IEEE International Conference on Software Security & Reliability*, 2013.

[29] K. Sentamilselvan., "Survey on cross site request forgery (an overview of csrf)," in *IEEE - International Conference on Research and Development Prospects on Engineering and Technology (ICRDPET 2013)*, 2013.

[30] T. Alexenko, M. Jenne, S. D. Roy, and W. Zeng, "Cross-site request forgery: Attack and defense," in *Consumer Communications and Networking Conference (CCNC), 2010 7th IEEE*, 2010.

[31] A. Alameen, "Building a robust client-side protection against cross site request forgery," *International Journal of Advanced Computer ence and Applications*, vol. 6, no. 6, 2015.

[32] H. Moradi and H. K. Moghaddam, "Strategies and scenarios of csrf attacks against the captcha forms," *Journal of Advanced Computer Science & Technology*, vol. 4, no. 1, 2015.

[33] B. Chen, P. Zavarsky, R. Ruhl, and D. Lindskog, "A study of the effectiveness of csrf guard," in *IEEE Third International Conference on Privacy*, 2012.