MACHINE LEARNING FOR WEB VULNERABILITY DETECTION

DR. T. S. GHOUSE BASHA[1], P SANDHYA[2], P. SUPRIYA[3], P. NAVYA[4]

[1]Professor and Department of ECE, Malla Reddy Engineering College for Women (UGC-Autonomous) Maisammaguda,Hyderabad, TS, India.

[2,3,4] UG students Department of ECE, Malla Reddy Engineering College for Women (UGC-Autonomous) Maisammaguda,Hyderabad, TS, India.

ABSTRACT

In this project, we propose a methodology to leverage Machine Learning (ML) for the detection of web application vulnerabilities. Web applications are particularly challenging to analyses, due to their diversity and the widespread adoption of custom programming practices. ML is thus very helpful for web application security: it can take advantage of manually labeled data to bring the human understanding of the web application semantics into automated analysis tools. We use our methodology in the design of Mitch, the first ML solution for the black-box detection of Cross-Site Request Forgery (CSRF) vulnerabilities. Mitch allowed us to identify 35 new CSRFs on 20 major websites and 3 new CSRFs on production software.

Keywords: ML, CSRF, widespread, web application.

INTRODUCTION

Web applications are the most common interface to security sensitive data and functionality available nowadays. They are routinely used to file tax incomes, access the results of medical screenings, perform financial transactions, and share opinions with our circle of friends, just to mention a few popular use cases. On the downside, this means that web applications are appealing targets to malicious users (attackers) who are determined to force economic losses, unduly access confidential data or create embarrassment to their victims. Securing web applications is well known to be hard.

There are several reasons for this, ranging from the heterogeneity and complexity of the web platform to the adoption of undisciplined scripting languages offering dubious security

guarantees and not amenable for static analysis. In such a setting, black-box vulnerability detection methods are particularly popular. As opposed to white-box techniques which require access to the web application source code, black-box methods operate at the level of HTTP traffic, i.e., HTTP requests and responses. Though this limited perspective might miss important insights, it has the key advantage of offering a language-agnostic vulnerability detection approach, which abstracts from the complexity of scripting languages and offers a uniform interface to the widest possible range of web applications. This sounds appealing, yet previous work showed that such an analysis is far from trivial. One of the main challenges there is how to expose to automated tools a critical ingredient of effective vulnerability detection, i.e., an understanding of the web application semantics. Example: Cross-Site Request Forgery (CSRF) Cross-Site Request Forgery (CSRF) is a well-known web attack that forces a user into submitting unwanted, attacker controlled HTTP requests towards a vulnerable web application in which she is currently authenticated. The key concept of CSRF is that the malicious requests are routed to the web application through the user's browser, hence they might be indistinguishable from intended benign requests which were actually authorized by the user.

A typical CSRF attack works as follows:

1) Alice logs into an honest yet vulnerable web application, e.g., her preferred social network. Session authentication is implemented through a session cookie that is automatically attached by the browser to any subsequent request towards the web application;

2) Alice opens another tab and visits an unrelated website, e.g., a newspaper website, which returns a web page including malicious advertisement;

3) The malicious advertisement sends a cross-site request to the social network using HTML or JavaScript, e.g., asking to "like" a given political party.

Since the request includes Alice's cookies, it is processed in her authentication context at the social network. This way, the malicious advertisement can force Alice into putting a "like" to the desired political

party, which might skew the result of online surveys.

Notice that CSRF does not require the attacker to intercept or modify user's requests and responses: it suffices that the Preventing CSRF

To prevent CSRF, web developers have to implement explicit protection mechanisms. If adding extra user interaction does not affect usability too much, it is possible to force re-authentication or use one-time passwords / CAPTCHAs to prevent cross-site requests going through unnoticed. In many cases, however, automated prevention is preferred: the recently introduced SameSite cookie attribute can be used to prevent cookie attachment on cross-site requests, which solves the root cause of CSRF and is highly recommended for new web applications. Unfortunately, this defense is not yet widespread and existing web applications typically filter out cross-site request by using any of the following techniques:

1) checking the value of standard HTTP request headers such as Referrer and Origin, indicating the page originating the request;

2) checking the presence of custom HTTP request headers like X-Requested-With, which cannot be set from a cross-site position;

3) checking the presence of unpredictable anti-CSRF tokens,set by the server into sensitive forms.

A recent paper discusses the pros and cons of these different solutions. However, all three options suffer from the same limitation: they require a careful and fine-grained placement of security checks. For example, tokens should be attached to all and only the security-sensitive HTTP requests, so as to ensure complete protection without harming the user experience.

Using a token to protect a "like" button is useful to prevent the attack discussed above, yet having a token on the social network homepage is undesirable, because it might lead to rejecting legitimate cross-site requests, e.g., from clicks on the results of a search engine indexing the social network. In the end, finding the "optimal" placement of anti-CSRF defenses is typically a daunting task for web developers. Modern web application development frameworks provide

Automated support for this, yet CSRF vulnerabilities are still routinely found even in top-ranked websites. This motivates the need for effective CSRF detection tools. But how can we provide automated tool support for CSRF detection if we have no mechanized way to detect which HTTP requests are actually security-sensitive.are passed - No splits.

EXISTING SYSTEM

In the existing system Securing web applications is well known to be hard. There are several reasons for this, ranging from the heterogeneity and complexity of the web platform to the adoption of undisciplined scripting languages offering dubious security guarantees and not amenable for static analysis. Though this limited perspective might miss important insights, it has the key advantage of offering a language-agnostic vulnerability detection approach, which abstracts from the complexity of scripting languages and offers a uniform interface to the widest possible range of web applications.

PROPOSED SYSTEM

Cross-Site Request Forgery (CSRF) is a well-known web attack that forces a user into submitting unwanted, attacker controlled HTTP requests towards a vulnerable web application in which she is currently authenticated. The key concept of CSRF is that the malicious requests are routed to the web application through the user's browser, hence they might be indistinguishable from intended benign requests which were actually authorized by the user. The CSRF does not require the attacker to intercept or modify user's requests and responses: it suffices that the victim visits the attacker's website, from which the attack is launched. Thus, CSRF vulnerabilities are exploitable by any malicious website on the Web.

MODULES DESCRIPTION

User:

The User can register the first. While registering he required a valid user email and mobile for further communications. Once the user register then admin can activate the customer. Once admin activated the customer then user can login into our system. User can do the data preprocess. First required running website name. By using that website the user can test the csrfs. By help of bolt tool the user can fetch related all csrfs and generated algorithm names. The result will be stored in json files. Later the user can get the results of Mitch

dataset. The mitch dataset tested for POST method as well GET method to. The result will be displayed on the browser.

Admin:

Admin can login with his credentials. Once he login he can activate the users. The activated user only login in our applications. The admin can set the training and testing data for the project of the Mitch Dataset. The user search all urls related csrf token admin can view in his page. The admin can also check the POST method performed data from the dataset and GET method related data also.

False Positives and False Negatives:

Mitch produces a false positive when it returns a candidate CSRF that cannot be actually exploited. This is something relatively easy to detect by manual testing, though this process is tedious and time-consuming. In general, it is not possible to reliably identify when Mitch produces a false negative, because this would require to know all the CSRF vulnerabilities on the tested websites. To estimate this important aspect, we keep track of all the sensitive requests returned by the ML classifier embedded into Mitch and we focus our manual testing on those cases. This is a reasonable choice to make the analysis tractable, because we first showed that the classifier performs well using standard validity measures.

Machine Learning Classifier:

The ML classifier used by Mitch was trained from a dataset of around 6000 HTTP requests from existing websites, collected and labeled by two human experts. The feature space X of the classifier has 49 dimensions, each one capturing a specific property of HTTP requests. Those can be organized into following categories.

following set of numerical features:

numOfParams: the total number of parameters;

numOfBools: the number of request parameters bound to a boolean value;

numOfIds: the number of request parameters bound to an identifier, i.e., a hexadecimal string, whose usage was empirically observed to be common in our dataset;

numOfBlobs: the number of request parameters bound to a blob, i.e., any string which is not an identifier;

reqLen: the total number of characters in the request, including parameter names and values.

Home page:

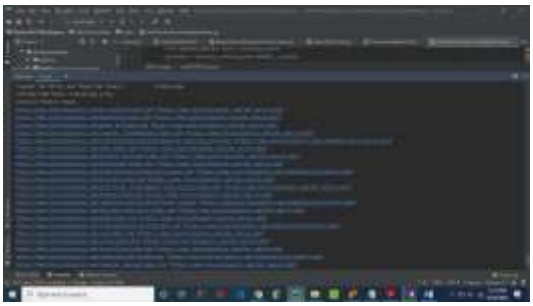
User Registration Form

User Login Form:

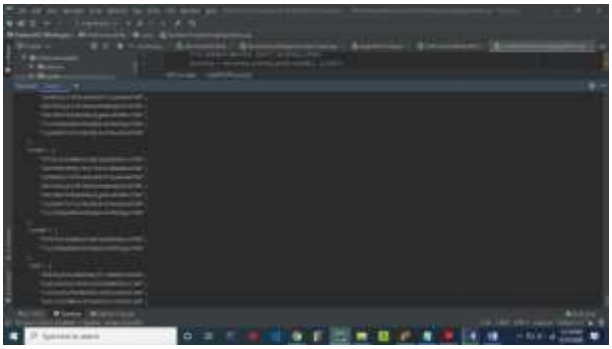
User Home:

Getting website csrfs:

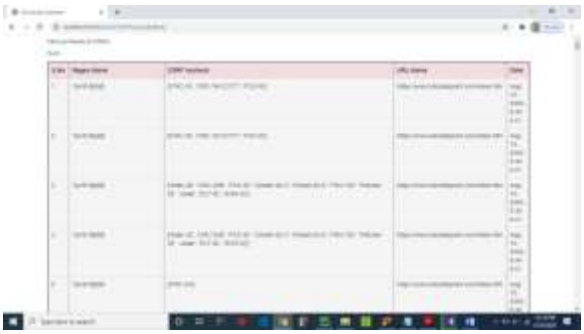
Scanning urls:



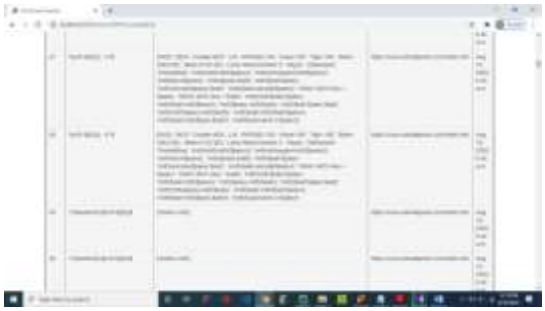
CSRF token:



Given website csrf results



MD5 Token



Mitch Detected sites:

Machine Learning Results:

Admin Login:

CONCLUSION

Web applications are particularly challenging to analyse, due to their diversity and the widespread adoption of custom programming practices. ML is thus very helpful in the web setting, because it can take advantage of manually labeled data to expose the human understanding of the web application semantics to automated analysis tools. We validated this claim by designing Mitch, the first ML solution for the blackbox detection of CSRF vulnerabilities, and by experimentally assessing its effectiveness. We hope other researchers might take advantage of our methodology for the detection of other classes of web application vulnerabilities.

REFERANCES

[1] Stefano Calzavara, Riccardo Focardi, Marco Squarcina, and Mauro Tempesta. Surviving the web: A journey into web session security. ACM Comput. Surv., 50(1):13:1–13:34, 2017.

[2] Avinash Sudhodanan, Roberto Carbone, Luca Compagna, Nicolas Dolgin, Alessandro Armando, and Umberto Morelli. Large-scale analysis & detection of authentication cross-site request forgeries. In 2017 IEEE European Symposium on Security and Privacy, EuroS&P 2017, Paris, France, April 26-28, 2017, pages 350–365, 2017.

[3] Stefano Calzavara, Alvise Rabitti, Alessio Ragazzo, and Michele Bugliesi. Testing for integrity flaws in web sessions. In Computer Security - 24rd European Symposium on Research in Computer Security, ESORICS 2019, Luxembourg, Luxembourg, September 23-27, 2019, pages 606–624, 2019.

[4] OWASP. OWASP Testing Guide. https://www.owasp.org/index.php/ OWASP Testing Guide v4 Table of Contents, 2016.

[5] Jason Bau, Elie Bursztein, Divij Gupta, and John C. Mitchell. State of the art: Automated black-box web application vulnerability testing. In 31st IEEE Symposium on Security and Privacy, S&P 2010, 16-19 May 2010, Berkeley/Oakland, California, USA, pages 332–345, 2010.

[6] Adam Doup´e, Marco Cova, and Giovanni Vigna. Why johnny can't pentest: An analysis of black-box web vulnerability scanners. In Detection of Intrusions and Malware, and Vulnerability Assessment, 7th International Conference, DIMVA 2010, Bonn, Germany, July 8-9, 2010. Proceedings, pages 111–131, 2010.

[7] Adam Barth, Collin Jackson, and John C. Mitchell. Robust defenses for cross-site request forgery. In Proceedings of the 2008 ACM Conference on Computer and Communications Security, CCS 2008, Alexandria, Virginia, USA, October 27-31, 2008, pages 75–88, 2008.

[8] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. Foundations of Machine Learning. The MIT Press, 2012.

[9] Michael W. Kattan, Dennis A. Adams, and Michael S. Parks. A comparison of machine learning with human judgment. Journal of Management Information Systems, 9(4):37–57, March 1993.

[10] D. A. Ferrucci. Introduction to "This is Watson". IBM Journal of Research and Development, 56(3):235–249, May 2012.