Grenze International Journal of Engineering and Technology, June Issue

Web Entity Protection System

Saurabh Yadav[1], Tushar Rangroo[2], Nishanth M[3], Suhan B Revankar[4] and Dr. Annapurna D[5]
[1-4]Computer Science & Engineering, PES University, Bangalore, India
Email: saurabhlmy@gmail.com, tusharrangroo@gmail.com, nishanth.murali2017@gmail.com, suhanbrevenkar29@gmail.com
[5]Professor, Computer Science & Engineering, PES University, Bangalore, India
Email: annapurnad@pes.edu

***Abstract*—Cyber-attacks can be extremely dangerous for not only organizations, but the nation too. These attacks if not controlled at early stages will have a severe impact on the economy. The increase in the number of attacks has made security measures mandatory. They include personal information, property, government information theft, Denial of Service, Infrastructure damage etc. This paper aims at Identifying potential vulnerabilities in a web facing entity and suggesting recommendations or patches to fix it. The main reason and goal of the project is to improvise web apps security before it is deployed and made available for public use. The user / developer can use and perform automated scans to detect possible vulnerabilities, store detailed information obtained from them for further analysis and also suggest possible countermeasures if within scope. This aims to help a non-security person or a very small startup with minimal knowledge on security practices to detect presence of vulnerabilities in their product and mitigate them before a hacker takes advantage of vulnerability by exploiting the same. This serves more like a pre-deployment tool.**

***Index Terms*— IDS, SQL injection, Cross-Site Scripting, Denial of service.**

I. INTRODUCTION

Security is one of the most important aspects that comes to picture when speaking of the internet. Today, a large number of people spend the majority of their time online, whether it be to make purchases, communication, education, transactions, online bookings, leisure, shopping etc. Therefore, rendering protection on the web and making websites safer against cyber-crimes and attacks is very important in order to avoid any negative events like threats. Cybercrime is a harmful attack that a business, person, or website may face. There are numerous instances where a cyber-attack caused a firm or an individual to suffer a significant loss as a result of a data breach. Any website or web application deployed, will mostly have interaction with the users. This user can be a good actor or a malicious attacker with bad intentions to cause harm to web applications. Therefore proper testing of the web app in terms of security is of utmost priority before making a web-app public facing.
Major exploited vulnerabilities stats are shown in Fig.2
So our prime focus stays on detecting 3 of OWASP 2021 top 10 vulnerabilities i.e SQL injection, Insecure design - XSS, Vulnerable and outdated components – DOS.
Therefore detection and prevention of these vulnerabilities before web-apps face the internet is really important else it can lead to serious damages. This project is being taken up to overcome the same problem i.e to detect mentioned 3 vulnerabilities and suggest effective preventive measures to avoid its exploitation by malicious 3rd party.

Grenze ID: 01.GIJET.9.2.8_1

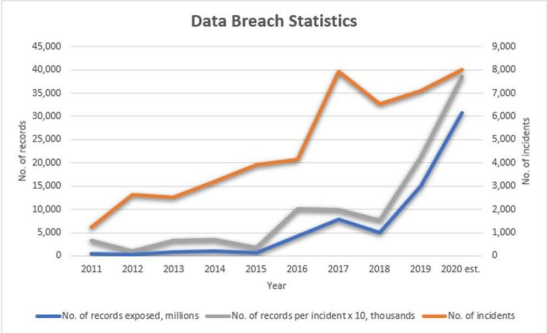

Fig.1 : Data Breach Statistics 2011-2022

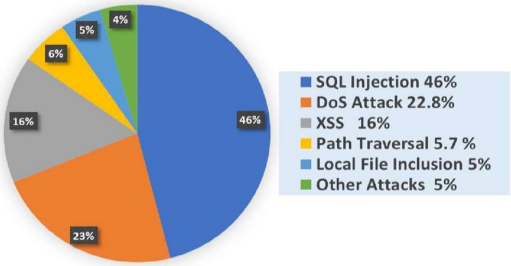

Fig.2 : Top five attacks on web app of healthcare institute 2017

II. Related Works

A. Boyer Moore Algorithm

Balasundaram I, Ramaraj E.[7] discusses a solution in their analysis about an automated scanning technique using Boyer Moore algorithm in order to scan potential SQLI vulnerabilities in a web application. In this technique a software which takes user inputs is used, which is then sent for static pattern matching. The Boyer Moore algorithm is then used for scanning the user input string by string and from left to right. This helps in matching the string to a static pattern list that maintains all the anomaly patterns, which helps to determine whether to affiliate the user input to a SQLI query or not. After the scanning an anomaly value is generated of the user generated query for each pattern in the static pattern list in case the string does not match any pattern in the list . If the calculated score tends to be greater than the threshold then an alarm is generated. The query is then transferred to the admin which manually checks whether the query affects SQLIA or not. In case it affects then the static list is appended. The security and accuracy here fully depends upon the contents of the static list.

B. Model Checking

An approach of finding security flaws that are caused due to vulnerable and outdated components is model checking.Hao Chen, Drew Dean, and David Wagner[12] suggested this approach in which specific classes of vulnerabilities are formalized and the program model checks for violations of the formalized properties. The advantage of this tool over other formal methods is that in case a failure gets detected , the model checker comes up with a concrete counter example that can be used as a regression test case. This tool is also counted as a useful tool for serving our purpose but it requires the vulnerabilities to be formalized which might not be the case every time.

C. Stack Guard

Another approach that was suggested by Crispin Cowan, Calton Pu, Dave Maier, Jonathan Walpole, Peat Bakke, Steve Beattie, Aaron Grier, Perry Wagle, Qian Zhang, and Heather Hinton[11] was using a stack guard. In this approach stack guard which is an automatic detection and prevention tool was suggested in order to mitigate a program's ability to do damage. Using this approach is especially useful when a program is already deployed.

D. Testing the binary

In this approach the binary of the program to be scanned is susceptible to various forms of testing and analysis. After testing the binary of the program the security leaks are reported to the vendor. The types of techniques

used for this analysis are fuzzing testing and fault techniques which were suggested by Jeffrey Voas and Gary McGraw[13]. This approach is argued by many authors who find this approach not contributing to the improvement of software quality.

E. Vulture tool

The Vulture Tools was first inspired by the pilot study of Schroter et al. [14], who first observed that imports correlate with failures.While Schroter et al. examined general defects, the present work focuses specifically on vulnerabilities. To our knowledge, this is the first work that specifically mines and leverages vulnerability databases to make predictions.

III. METHODOLOGY

A modular design approach so that any further extensions, additional features to be added etc. so that all these can be easily accommodated within the existing project in the future time. Each module here is assigned with their respective responsibility and will handle the same.

A. Architecture and Features

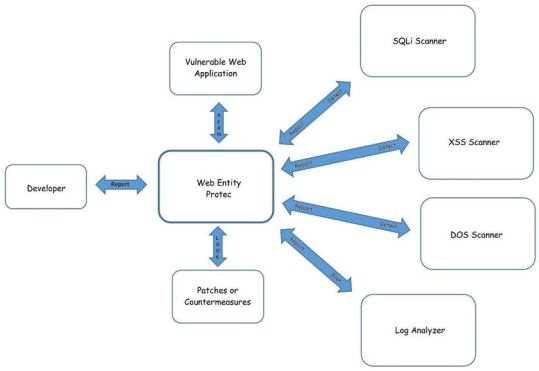

Fig.3 : System at a black-box point of view

The architecture of the system that is being proposed is given in Fig.3. Each module here is assigned with respective responsibilities and will handle the same, like SQLi scanner looks up on possibility of SQLi vulnerability on the web-app, XSS instance will look up the presence of XSS vulnerability on the web-app, DOS detector will check the target web-app for the presence of vulnerability to conduct a specific type of DOS attack for now and the Log analyser module will convert raw log files to human readable format.

All these sub-modules will interact with the main module i.e. Web Entity Protection module. The main module will analyze the response from each sub module and then take appropriate action regarding patching or at least inform the user / developer to look out for these possible vulnerabilities present in the web-app and are exploitable and inform the developer to take appropriate actions to secure the web-app. The user or developer has access to the main module

i.e 'Web Entity Protec' module only to initialize the tool and apart from that he can only access the report generated by the main module.

B. Module details

Initially the tool will need to accept the URL from the user/developer as input to scan for presence of possible vulnerabilities. Based on this input using requests.get and BeautifulSoup object, all the paths that the code found by crawling the given URL and writing the same to crawl_paths.csv under the directory belonging to that particular scan number.

1. xss_detector : This module is responsible for

detecting XSS vulnerability in a given website. Since user inputs and forms are typically used to exploit this type of vulnerability, one needs to add some malicious javascript code to every input field and forms we find in the online application. The payload we used is

"<img src/onerror=prompt(8)>" .

So, to satisfy the first requirement, a function that can scrape and then fetch all the forms from the HTML content of any web page once this is done, another function that can submit any given form to the intended destination. The POST request has hard coded malicious XSS payload and takes form details as an input and submits this form using requests.get() or requests.post() methods. Results are written to a csv file named xss_results.csv in a column named 'status'.
3 possible values of the column are: 'YES' - path IS vulnerable to XSS. 'NO' - path IS NOT vulnerable to XSS. 'NoSc' - the detector was unable to scan the given particular path, maybe that respective path is responding etc.
2. Dos_detector : This module is responsible to check whether a web-app is vulnerable to DOS attack or not. At this point of time, focus is on one particular DOS attack type i.e Slowloris attack.
To detect Slowloris vulnerability Nmap is used in the backend. A script called "dos-detector" written using Nmap Script Engine establishes two connections to the
server, each of which lacks the CRLF [Carriage Return, Line Feed] end. The second connection transmits an additional header after ten seconds. Both connections are currently idle while they wait for the server to time out. If the second connection timeout 10 seconds or more after the first one, we can infer that the server is vulnerable to a slowloris attack and that adding additional headers lengthened the second connection's timeout duration. The server is vulnerable to timeout period extension-related assaults if the result is "LIKELY VULNERABLE" but again, the http server's architecture and resource constraints must be taken into account; a total DOS is not always attainable. A proper complete testing requires conducting an actual DoS condition, attack and measuring server responsiveness. DoS results are stored as Dos_results.csv
3. SQLi Detector: making use of Sqlmap A bash script written to initiate dynamic scanning and based on the results given by the exploited forms, the URL path is matched with the main URL returned by the main module.
A detailed report of the scan is also stored in .txt format, as per user needs, if the user wants to perform detailed analysis of what's exploited and how it was exploited in the later time. All these .csv results are read by the main model and makes a consolidated report of the same by appending the results of each scan to the same file named dataset.csv, so that all the results are available in the same file for future analysis and reporting.
4. There is also a 'log extractor' feature, which is independent of the above. This accepts large log files which have all the details of the packets that passed through any proxy. And the analysis of logs is the first and foremost step taken in case of any security breach. But log files are usually in .xml format and not human-friendly to read, therefore the 'log extractor' that makes use of xml.etree, urllib.parse, base64 python libraries to convert the log files to a human readable .csv format. Based on the output csv the developer/SOC analysts can proceed further with the information of the attack in detail.
After detection is completed, those files are categorized based on scan number and fed to another module, This module analyzes the CSV files such that if a particular vulnerability exists despite many scans, then it throws a warning to look into that particular vulnerability

C. Technologies and Software

Overall coding necessary for the project was done using Jupyter notebook's Python (version 3.9.13) environment and minimal usage of bash scripts is also done, which helps in initializing sqlmap for testing sql injection vulnerability. The entire system is in integration between different modules for detecting their respective vulnerabilities.
At this stage working is automated with few prompts required in-between to proceed further after each phase of detection. As all results are presented to the developer in .csv files or displayed on the terminal there is no real use of GUI as of now and left for future scope.
Pandas library used to write results to csv files.
bs4 Module used to scrape information from web pages in the XSS detector module. OS Module used to set the results path and also to fetch source file location.
pathlib Module used to combine different file-system paths to remove file location dependency. Requests Module used to send HTTP requests using python in XSS detector. xml.etree.ElementTree module used to parse XML files
in log analyzer.
Nmap script engine used in backend to write script to detect slowloris DoS.

IV. Result and Discussion

In order to mitigate the three stated vulnerabilities (i.e SQLI, XSS and DOS) we use our tool on a website which is scanned when it has vulnerability mitigation set to low, low-moderate, moderate-high. The tool scans the web

application under these 3 states and creates a csv file which is used to keep track of the reoccurring vulnerabilities that a particular web application has and after a certain continuous repetition of these vulnerabilities suggests a patch which can be used to fix those vulnerabilities.If the user wants the tool displays a set of steps which can be used to mitigate those attacks and can also generate a csv log which tells the user about the number of urls in which the respective vulnerability has occurred. These scans are used to check the accuracy and effectiveness of the tool in scanning vulnerabilities on a certain web application.

A. Scan number 1

The first scan taken into consideration is when the vulnerability mitigation is taken as low. The output (Table1) shows the result on the web application "Demo.testfire. net" which is a vulnerable banking web application which will help us in checking whether the tool is working accurately or not. Also we are using a Python script to scan the number of rows(Urls) in which these three attacks are displayed(Table2) and also keeping track of the Scan number in order to check which type of vulnerabilities are being repeated in the particular web application.

TABLE I.

Scanned Urls	**Scan No**	***Denial of Service***	***Cross Site Scripting***	***SQL Injection***
http://Demo.testfire.net//Index.jsp	1	YES	YES	NO
http://demo.testfire.net//login.jsp	1	YES	YES	YES
http://demo.testfire.net//index.jsp?content=inside_contact.htm	1	YES	YES	NO
http://demo.testfire.net//feedback.jsp	1	YES	YES	NO
http://demo.testfire.net//status_check.jsp	1	YES	NoSc	NO
http://demo.testfire.net//swagger/index.html	1	YES	NO	NO
https://github.com/AppSecDev/AltoroJ/	1	YES	NO	NO
http://www-142.ibm.com/software/products/us/en/subcategory/SWI10	1	YES	NoSc	NO

TABLE II.

No of Urls in which XSS is Found	No of Urls in which DOS is Found	No of Urls in which SQL Injection is found
4	8	1

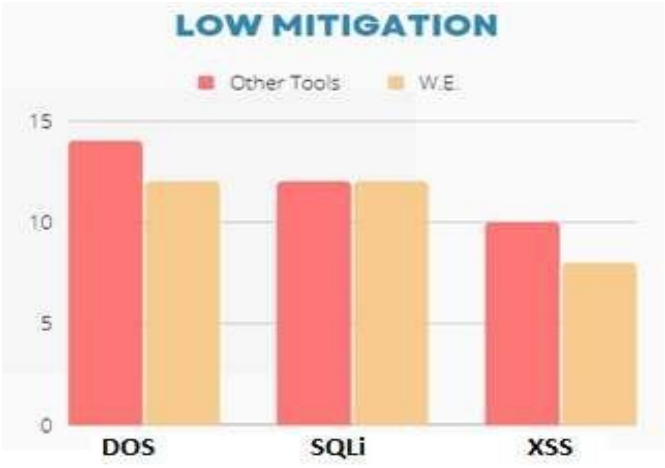

Fig.4 : Accuracy Comparison of tool set as benchmark

We have compared (Fig4) W.E. (Web Entity protection System) with a fairly accurate tool which we will use to find the accuracy of our tool in three different scans (low, moderate, high). In the first scan where the mitigation is set to low, the number of Dos vulnerabilities found by other tool are 14 whereas the no of Dos vulnerabilities found by W.E. are 12. In SQL the vulnerabilities found by other tool are 12 and by W.E. is 12. For XSS the number of vulnerabilities found by other tools are 10 whereas for W.E. are 9. Hence accuracy of W.E. when the mitigation is set to low is 91.6%.

B. Scan number 2

The second scan taken into consideration is when the vulnerability mitigation is set as low- moderate. The output (Table3) shows the result of the scan on the Web application. This scan also helps the developer to get a glance of how the tool will perform in the real environment as the maximum Web applications in the real world will have the vulnerability mitigation set to moderate or high. Also the counter that was set in the previous scan is also used in order to detect the trend of vulnerabilities that is occurring in a targeted Web application. This will help the user of the tool to keep a track about the mistakes that have been repeated while developing the Web application. Also the tool after every scan will suggest the user about the techniques that can be used while developing (Eg a New functionality in the site) the Web application so that the focused vulnerability does not occur in the Web application again. Also mitigation techniques that could be used after the development of the Web Application(Eg a Functionality) will also be suggested. The total no of Urls consisting of the respective vulnerabilities will also be updated as the no of scan increases and the user can access the result of the latest scan(Table 4).

TABLE III

Scanned Urls	Scan No	*Denial of Service*	*Cross Site Scripting*	*SQL Injection*
http://Demo.testfire.net//Index.jsp	2	YES	YES	NO
http://demo.testfire.net//login.jsp	2	YES	YES	YES
http://demo.testfire.net//index.jsp?content=inside_contact.htm	2	YES	NO	NO
http://demo.testfire.net//feedback.jsp	2	YES	YES	NO
http://demo.testfire.net//status_check.jsp	2	NO	NoSc	NO
http://demo.testfire.net//swagger/index.html	2	YES	NO	NO
https://github.com/AppSecDev/AltoroJ/	2	NO	NO	NO
http://www-142.ibm.com/software/products/us/en/subcategory/SWI10	2	NO	NoSc	NO

Table IV

No of Urls in which XSS is Found	No of Urls in which DOS is Found	No of Urls in which SQL Injection is found
3	5	1

In scan number 2(Fig 5) the mitigation is set low-moderate. The other tool used for the accuracy comparison has detected 13 DOS vulnerabilities and W.E. has detected 11 vulnerabilities. For SQL the numbers are 10 and 10. For XSS they are 9 and 8 respectively. The total accuracy of
W.E. during low-moderate mitigation is 90.62%.

C. Scan number 3

The third scan takes into consideration the level of vulnerability mitigation as moderate-high. The output (Table 5) shows the result of the scan on the Web application. This is the scan that shows how the system will perform with respect to most modern websites today. This scan also shows us how our tool will work with the websites that have taken safety as a consideration while developing the website and how most of the professional websites that are built today by experienced programmers will react to our scanning tool. Since this the third scan and the program counter that web had used before is still keeping track of the vulnerabilities that have been consistently occurring in the Web application, after a certain number of scans before running the scan the user will get a patch suggestion regarding the fixing of the vulnerability that is consistently occurring in the scans of the targeted Web application. The csv file(log) which is used to keep the track of the Urls in which the respective vulnerabilities are occurring will also be updated so that users using the tool can access the updated log in order to analyze the Urls that require patching/fixing.

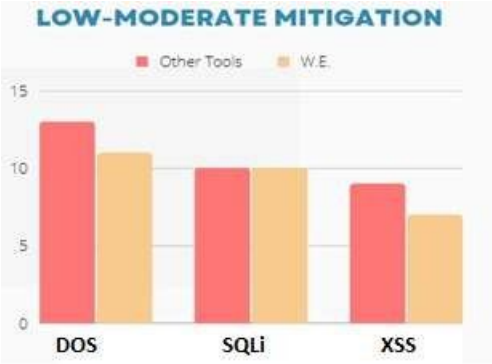

Fig.5 : Accuracy Comparison of tool set as benchmark

TABLE V

Scanned Urls	**Scan No**	***Denial of Service***	***Cross Site Scripting***	***SQL Injection***
http://Demo.testfire.net//Index.jsp	3	YES	NO	NO
http://demo.testfire.net//login.jsp	3	YES	YES	YES
http://demo.testfire.net//index.jsp?content=inside_contact.htm	3	NO	NO	NO
http://demo.testfire.net//feedback.jsp	3	YES	YES	NO
http://demo.testfire.net//status_check.jsp	3	NO	NoSc	NO
http://demo.testfire.net//swagger/index.html	3	NO	NO	NO
https://github.com/AppSecDev/AltoroJ/	3	NO	NO	NO
http://www- 142.ibm.com/software/products/us/en/subcategory/SWI10	3	NO	NoSc	NO

TABLE VI

No of Urls in which XSS is Found	No of Urls in which DOS is Found	No of Urls in which SQL Injection is found
2	3	1

In scan number 3(Fig 6) the migration is set moderate-high. The number of vulnerabilities for DOS are 11 and 9. For SQL they are 8 and 8. For XSS these are 7 and 6. Hence the total accuracy of W.E. when the mitigation was set moderate-high is 88.46%.

Fig.6 : Accuracy Comparison of tool set as benchmark

V. Future Scope

In the current project we were able to only suggest the user regarding the patches but we were not able to deploy the patches. The reason being that every web application is unique and hence its code is unique, making the general patches not fully compatible with the code of web application. Hence the future scope of this project is the execution and deployment of the patches for the web entity.

VI. Conclusion

At the end, we could understand different types of methodologies used by different people in their research to detect various vulnerabilities and how they managed to increase the accuracy of detecting those vulnerabilities and keep up the performance in an effective manner. This gives us an idea about all these methodologies, what to use in our project and what not to use. After doing the research for this project we were able to conclude that SQL injection is a direct vulnerability whereas insecure design, vulnerable and outdated components cause vulnerabilities. The mitigation for this is that an Automation script should be able to detect both direct vulnerabilities and consequent ones. This should help us efficiently implement the whole idea of Web entity protection. The vulnerabilities that we are dealing with are not new nor are they those vulnerabilities that do not have a mitigation, but with this the question arises then why are these vulnerabilities still prominent in today's date and still are listed on the OWASP vulnerabilities. The answer to this is that in the present scenario the developers are heavily dependent upon post deployment tools like web application firewall to do their job and protect the web application from the exploit of these vulnerabilities. Also Javascript being the prominent language for Web application development provides the developer with flexibility in coding but at the cost of security and human error will always remain a factor which we cannot deny during the development of any product whether it is a Web application or any other software. Although a Web application firewall provides substantial protection to the Web application but it also comes with a cost of speed and accuracy. This gives us the motive of using a tool which will help the developer to find vulnerabilities beforehand so that those vulnerabilities could be fixed immediately. As we can see from the execution of our web entity protection tool, it does a very decent job in finding those vulnerabilities even though the mitigation is moderate to high. In the end we could conclude that no site could be perfectly safe from vulnerabilities but detecting and mitigating those detected vulnerabilities is the right practice that a developer could perform.

Acknowledgment

This work was supported by Dr. Annapurna D, Professor, Department of Computer Science and Engineering, PES University. We would like to express our gratitude for her continuous guidance, assistance, and encouragement throughout the development of this project. We are grateful to the Capstone Project Coordinators, Prof. Animesh Giri, Assistant Professor and Dr.

Gokula Kannan, Assistant Professor, for organizing, managing, and helping with the entire process. We take this opportunity to thank Dr. Sandesh B J, Chairperson, Professor, Department of Computer Science and Engineering, PES University, for all the knowledge and support we have received from the department. We would like to thank Dr. B.K. Keshavan, Dean of Faculty, PES University for his help. We are deeply grateful to Dr. M. R. Doreswamy, Chancellor, PES University, Prof. Jawahar Doreswamy, Pro Chancellor - PES University, Dr. Suryaprasad J. Vice-Chancellor, PES University for providing us various opportunities and enlightenment every step of the way. Finally, this project could not have been completed without the continual support and encouragement we have received from our family and friends.

REFERENCES

[1] https://nmap.org/
[2] https://sqlmap.org/
[3] Balzarotti, Davide & Monga, Mattia & Sicari, Sabrina. (2006). Assessing the risk of using vulnerable components. 23. 10.1007/978-0-387-
[4] C. Cetin, D. Goldgof and J. Ligatti, "SQL-Identifier Injection Attacks," 2019 IEEE Conference on Communications and Network Security (CNS), 2019, pp. 151-159, doi: 10.1109/CNS.2019.8802743.
[5] Neuhaus, Stephan & Zimmermann, Thomas & Holler, Christian &Zeller, Andreas. (2007). Predicting Vulnerable Software Components. Proceedings of the ACM Conference on Computer and Communications Security. 529- 540. 10.1145/1315245.1315311.
[6] Fink, Laura & Prakash, Atul & Borders, Kevin. (2008). Analyzing websites for user-visible security design flaws. SOUPS 2008 - Proceedings of the 4th Symposium on Usable Privacy and Security. 117-126. 10.1145/1408664.1408680.
[7] https://www.cc.gatech.edu/home/orso/papers/halfond.orso.ICSEDEMO06.pdf
[8] Analysis of SQL Injection Detection and Prevention (indjst.org)
[9] T. A. Taha and M. Karabatak, "A proposed approach for preventing cross-site scripting," 2018 6th International Symposium on Digital Forensic and Security (ISDFS), 2018, pp. 1-4, doi: 10.1109/ISDFS.2018.8355356.
[10] https://www.whamtech.com/data-breaches-continued-to-rise-in-2019/
[11] Crispin Cowan, Calton Pu, Dave Maier, Jonathan Walpole, Peat Bakke, Steve Beattie, Aaron Grier, Perry Wagle, Qian Zhang, and Heather Hinton. StackGuard: Automatic adaptive detection and prevention of buffer-overflow attacks. In Proceedings of the 7th USENIX Security Conference, pages 63–78, San Antonio, Texas, January 1998.
[12] Hao Chen, Drew Dean, and David Wagner. Model checking one million lines of C code. In Proceedings of the 11th Annual Network and Distributed System Security Symposium (NDSS), pages 171–185, February 2004.
[13] Jeffrey Voas and Gary McGraw. Software Fault Injection: Innoculating Programs Against Errors. John Wiley & Sons, 1997.
[14] Adrian Schroter, Thomas Zimmermann, and Andreas Zeller. Predicting component failures at design ¨ time. In Proceedings of the 5th International Symposium on Empirical Software Engineering, pages 18–27, New York, NY, USA, September 2006. Association for Computing Machinery, ACM Press.