Automated Detection of Vulnerabilities on Websites*

Anonymous
Removed

ABSTRACT

The creation of the concept of networking has open the door to many possibilities such as information sharing across nodes, creation of the Internet and web applications. Unfortunately, this also have a side effect. No longer attackers need to attack just a company software application but they are now capable of exploit and attack a company through their websites and web applications. It is estimated that there are at least 1 billion websites on the Internet. Those websites are power by various languages such as PHP, ASP.NET(C#) as well as open source projects such as Wordpress, Druapl, Symfony and ultimately inheritance the risks that comes with those tools and languages.The automated tool will take a given URL and provides vulnerabilities checklist and their scores which can assist company to patch the vulnerabilities in the development stages prior to release to the public. The goal is to help company maintain the security principles of computing security such as confidential, available, integrity and non-repudiation.

KEYWORDS

Computer Security, Vulnerabilities, SQL Injections, Malware, Website Vulnerabilities

ACM Reference format:
Anonymous. 2016. Automated Detection of Vulnerabilities on Websites. In *Proceedings of ACM Conference, Washington, DC, USA, July 2017 (Conference'17),* 5 pages.
DOI: 10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

According to the security threat report conducted by Sophos in 2012, approximately 30,000 websites are attacked on daily basis [12]. That is roughly 10800000 million websites hacked a year. Breaches of companies website doesn't merely ruin the reputation or several billions of dollars in damages, depending on the nature of the website, it also destroy the lives of the users. For example, in 2015 and 2016, 35.4% of the data breaches occurred in the medical and health center sector [3]. We are talking about the leak of sensitive patient's information including medical histories, social security number and among other sensitive information. In additional, 8.1% of the breach also occurred in the government or military sector [3], those often contain top secret information or even dangerous tools such as the hacking tools that NSA use to spy on other foreign countries [11] or even the top encryption cracking tools. In 2014, 76 millions Americans were affected when JPMorgan Chase was hacked [6]. Given that hacking is not going anywhere and the increase in level of sophisticated hacked, it is important that developers and cybersecurity have the necessary tools to help them migrate risk against their application. For this project, our focus will be on building a tool that will help web developers to migrate security risks on their web application according to OWASP security guidelines [9] such as configuration management, Authentication, Session Management, Data validation and among other.

*Produces the permission block, and copyright information

Conference'17, Washington, DC, USA

DOI: 10.1145/nnnnnnn.nnnnnnn

2 VULNERABILITIES

Below are the most common web security vulnerabilities, a brief explanation, and some risks posed by them.

2.1 SQL Injections

One of the more well known examples, where statements can be modified by the request to execute arbitrary queries. There's a number of ways it can be utilized, such as using escape characters to break the query or cracking weak type-checks to substitute in other data.

The impact of this largely depends on what data is accessible by the arbitrary queries. It's also possible that the attacker might not be trying to retrieve information, but to disrupt the target by deleting or corrupting data. One common example would be DROP TABLE to completely delete targeted tables.

2.2 Cross Site Scripting (XSS)

Malicious entities are capable of loading scripts into the client-side of a website. This will typically be HTML or Javascript, but can affect other client-side languages. These scripts rely on the website running the data without any safeguards, much like SQL Injections, but is harder to fully protect. At the very least, sanitizing input text serves as a good start, but isn't good enough on it's own.
This is considered one of the most common vulnerability for all industries except banking [5], but it still accounts for a large number of attacks to banking. The risk posed by it varies depending on what else is present on the website, such as login/passwords and other sensitive user data.

There are two main variants that are being tested: Stored and Reflected. Stored is kept on the website permanently, typically in the database by some comment system, while Reflected is typically a modified link with a malicious script within it.

2.3 Broken Authentication

Broken Authentication or more common known as broken access control post a major security risk to web application when not properly configured.Such issue might be an unauthorized user accessing resources that should not have been available to them. In

additional, improper management of session is also consider a broken authentication. For example, it is expected that a session id be rotated after successful login instead of using the same session id prior to successful login. Exposed of sensitive information in the url can also be consider a broken authentication. Very often the "GET" request are not private, therefore an attacker monitoring the network traffic on tools like wireshark would have been able to capture the sensitive information that is submitted in the GET request or POST request if not proper handled or encrypted.

2.4 Cross-Site Request Forgery (CSRF)

Cross-Site Request Forgery (CSRF) is exploiting a trust that site provides a URL to users [1]. There is a number of ways to exploit CSRF. This vulnerability appears in emails, ads, or website that links to another website. It is one of the most common attacks on users. The hacker will send an email with trustworthy pictures, text, etcetera and at the same time, create a malware hyper-link in one of those components.

For example, users (John) would like to access the bank account page and transfer $1,000 to another user (Jane). The real URL transfer will display as:

```
http://bank.com/account/?from=john&?to=jane?transfer=1000
```

Let's say, hacker creates an email and send it to John. Within the email, the hacker includes a hyper-link stating "Confirm a transfer" or "Verify the transfer". However, in the hyper-link, hacker included a malware URL display as:

```
http://bank.com/account/?from=john&?to=hacker1?transfer=1000
```

As soon the user click that link, it will automatically make a transfer $1,000 to a hacker without letting user (John) know.

2.5 Brute Force

This is one of the most common website vulnerabilities. There are many websites that provides all possible combination of user names and password. This allows an attacker to brute forcing every possible combination of username and password until it is successfully cracked it. They can used cracked username and password and take an advantage of the system or the user. Many websites does not implement a detection if login form is being attacked. The best way to implement a fix is to delay for every failed attempts or lock an account out after certain number of tries.

2.6 Failure to restrict files, folders, and URL access

This allows a user to access unauthorized folders or files on the website with a given unlisted URL. This could create a potential vulnerability depending on what kind of directories or files it contains. For example, if the file is sensitive data or directory contains personally identifiable information (PII) files which are not suppose to show in public.

How would the attacker gain access to folder or files URL? There are few different type of attacks: force browsing and brute-force. Force browsing is an attack which is used to gain access to unauthorized folders or files in a web server by entering a URL directly[2]. Brute force is a search of unlisted URL address on the web page that is not shown in public. In addition to that, attacker can guess by using most common folder or file names are "/Admin", "Log", "/Images", "/Backups", "/Resources", "/Data", "log.txt", "backup.sql", etcetera.

3 DESIGN CONSIDERATIONS

We developed an application that takes a given Universal Resource Locator (URL) and vulnerability types, then attempts to perform vulnerabilities checking. The application automatically check for different security principles that have been violated as well as adding a score to each violation and classify which category each violation breached such as confidentiality, availability, integrity, ease of access, awareness, access control, and among other.

We ended up not using public websites without consent as it would be considered illegal hacking and unethical. See more information on Ethical and Legal Issues. Instead, we used using Damn Vulnerable Web Application (DVWA) [4] running localhost that is exposed to one or more and not limited to the vulnerabilities listed below:

- SQL Injections
- Cross Site Scripting (XSS)
- Broken Authentication
- Cross-Site Request Forgery (CSRF)
- Brute Force
- Failure to restrict files, folders, URL access, and disclosure of sensitive files such as web configuration information

In order to classify the rating for each vulnerabilities. We used an OWASP rating methodology as the basis and customized it to match our needs. For example, according to OWASP rating methodology [8], it relies on two likelihood (threat agent factors and vulnerability factors) and two impacts (technical impact factors and business impact factors) to determine the risks of a vulnerabilities. Instead, we are focusing on one likelihood (vulnerability factors) and one impact (technical impact factors). The vulnerability factors are ease of discovery, ease of exploit, awareness, and intrusion detection. The technical impact factors are confidentiality, integrity, and availability. These are rating from 0 to 10 being from low risk to critical risk. We have decided to eliminate "Loss of Accountability", business impact factors (ie: financial, rep damage), and threat agent factors as we cannot determine their motive and their current financial state. See below is an example of a rating system:

```
Brute Force Rating and Impact:
==============================
Likelihood: 8 - HIGH
Impact: 3 - MEDIUM
Overall Risk: HIGH
```

Figure 1: Example of a rating system

However, there is one issue. We are not planning using neural networks or machine learning to determine if the info or leaked

Table 1: Modified Rating System

Vulnerability	CIA (#/10)	Vulnerability Impact (#/10)
Brute Force	3, 1, 5	9, 9, 9, 5
CSRF	3, 1, 5	9, 5, 4, 5
SQL Injection	9, 9, 9	9, 9, 9, 8
XSS	3, 1, 5	9, 5, 4, 5
Sensitive	1, 1, 3	3, 1, 1, 5

data are private or sensitive. It will be difficult to determine. Instead, we will give an estimate score for each vulnerability and provide links on how to fix a vulnerability. See below, Modified Rating System.

4 IMPLEMENTATION

We have implemented several vulnerabilities on a test environment using DVWA to simulate an actual website. In addition to that, we have used several python libraries that help us to develop the scripts for each vulnerability. The python libraries are BeautifulSoup, Requests, and ArgParse, and url-normalize. The BeautifulSoup parse the data from Hypertext Markup Language (HTML) web pages. The Requests allows the developer to send data (POST) to the website and get the output data (GET). The Argparse creates a python script usage and displays the message if a user inputs the wrong arguments. The url-normalize eliminates unnecessary characters in URL, for example, removing extra dashes at the end of the url (http://example.com/// to http://example.com/).

Below is a final design (tree diagrams) to see what type of scripts and classes are being used.

```
Automated-Detection-Vulnerabilities/
   Attacks/
   |   ActiveSQLInjection.py
   |   BruteForce.py
   |   DirectoriesFilesTraversal.py
   |   Fuzz.py
   |   PassiveSQLInjection.py
   |   Sensitive.py
   |   CSRF.py
   |   XSS.py
   |      XSS (Stored)
   |      XSS (Reflected)
   Utilities/
   |   File.py
   |   Link.py
   |   Classification.py
   |   Requests.py
   Files/
   |   activeSQL.txt,  passiveSQL.txt,  XSSAttacks.txt,
   |   sensitive.txt,  userspass.txt,  directories.txt,
   |   filenames.txt
   README.txt
   Main.py
```

5 ARCHITECTURE

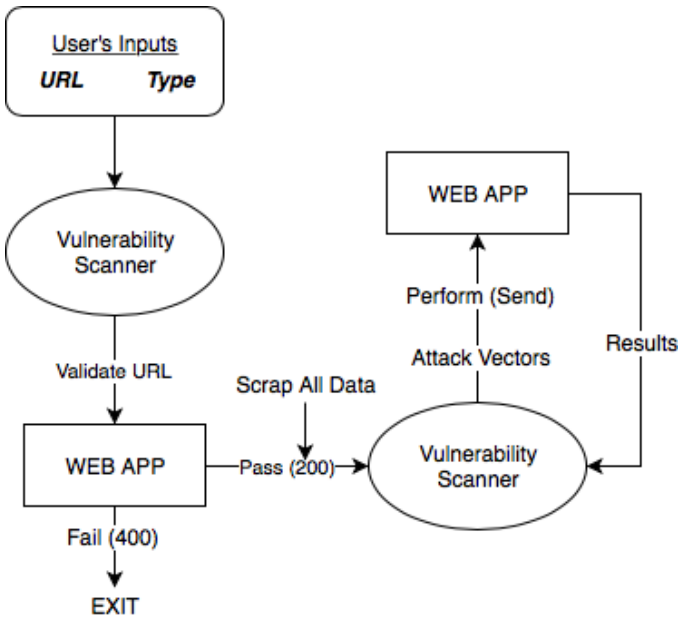

Figure 2: Automated Detection of Vulnerabilities Pipeline

It takes two user's arguments, two being required. Based on the user's inputs, it will execute a single vulnerability, multiple, or all vulnerabilities. In order to print program usage, type "python3 main.py -h"

- -v {BRUTE, CSRF, P-SQL, A-SQL, XSS, DIR-TRA, SENSITIVE}
- -u Website URL

Under "Attacks" folder, we have implemented seven vulnerabilities and follow the standard OOP design:

- **ActiveSQLInjection.py**: It takes the Link (Link.py) and go through the list of links and their input and attempt multiple active SQL injections vectors. Active SQL Injection performs read and write.
- **Brute Force (BruteForce.py)**: Brute force the login form with a list of possible user name and passwords. If it login with a correct user name and password, it will print out the information.
- **DirectoriesFilesTraversal.py**: This use the File object to return a list of possible directories and attempt to find if we are able to traverse and access them without proper permission.
- **Fuzz.py** - It scraps the website, collects a link of URLs, inputs, and relevant data and the page contents. This is done by returning the Link class object which will have the following attributes: url, list of inputs for the specific url, and page content.
- **PassiveSQLInjection.py**: It takes the Link (Link.py) and go through the list of links and their input and attempt multiple passive SQL injections vectors. Passive SQL Injection perform read, not write.

- **Sensitive.py**: Check to see if there are any leak of sensitive information such as configuration files; for example phpinfo, config.ini or backup logs etc using predefined sensitive keywords
- **XSS.py**: It takes the Link and goes through the list provided to attempt various XSS vectors. This tests all forms in a page to see if there's a match for any script and if it would execute based off the text format.
- **CSRF.py**: Takes link object and read the link object such as the content, inputs, and url to see if the CSRF token is in the link object. The user also have the ability to specified their own CSRF token

Under "Utilities" folder, we have implemented helper class and follow the standard OOP design:

- **File.py**: Returns the file path for our pre-written attack scripts.
- **Link.py**: Returns the link object consist of url, inputs, and page contents
- **Classification.py**: If the vulnerability exist in the website, it will give the rating of the vulnerabilities and classify what violation has breached.
- **Requests.py**: Returns the session of the page. To prevent logging out when performing vulnerabilities on the web page.

6 RELATED WORK

There are several commercial products that offer functionality similar to what we implemented. This is largely because there is value in making sure the website is secure the first time around instead of waiting for something to go wrong and trying to mend it after the fact. Most notably, there is a patent [7] on the general cycle of checking and testing, which is extremely similar to what was created for our research into the topic. Since we are not pursuing distribution, but simply researching to see what challenges there are, we do not have to worry too much about the patent, but will have to revisit it if something changes down the line.

7 CURRENT STATUS & FUTURE WORK

7.1 Current Status

We have successfully implemented and tested all vulnerabilities across three platforms (Windows 10, Mac OS X 10.13, and Linux). We have performed a code review to ensure there is no duplicate codes (or leftover codes) and ensure the code is meeting the coding standard design. We also provided our code for a peer-review to other groups, allowing us to get fresh opinions from outsiders to our project. Thanks to this review, we were able to identify small changes that needed to be done to improve our code. Finally, we have ensued that the code is documented properly for future reference, along with further cleanup to make it easier to review at a later date.

7.2 Future Work

Given the time we had for this project, we wished to implement few more vulnerabilities such as insecure direct object references, session management, missing function level access control, and few more. Additionally, we wanted to consider multi-threading to further speed up our code and allow for testing multiple errors at once. Additionally, we would've liked to expand the details of some functions, such as adding more exhaustive XSS tests or trying combinations of exploits. Although we have fixed the issue regarding website with HTTPS not able to use the tool, we would like to give developers ability to add certificate path if the HTTPS flag is enabled. One other thing that would've been nice is a GUI, which could make our program more user-friendly for peer testing. However since this program is not intended for distribution, it was given a very low priority.

8 LESSONS LEARNED

8.1 Response to Investigating Teams

The peer reviews have helped us identify three main issues.

The first was an issue with the CSRF, which was identified directly after distributing a copy for reviewing by ourselves. This has since been worked on, and should be working as intended, without any crashes.

The second is some issues with the inability to connect to HTTPS according to the group reviewing. We have added supports for the HTTPS protocol thus our scanner is now able to work correctly with both HTTP and HTTP protocol. However, for future work, we would like to give developer ability to specify the certificate path (file) for the HTTPS as well as ability on whether to detect HTTPS certificate is outdated.

The third issue identified was based on the DVWA tools we were using, as it was observed that the evaluation could change based on the directory name being changed, with no other changes. As a result, our program may have some rigidity when it comes to evaluating an application's structures, and needs to be reviewed. We were not able to replicate this issue on our end. As critical this type of feedback is, without the steps on how to replicate this behavior, we were not able to issue a fix for this in this release.

8.2 Other Lessons

We have learned a plethora of vulnerabilities, and the challenges of automating testing for these vulnerabilities. We had to go in depth on Python libraries designed to help read web pages, gathering all fields necessary, and automating our tests. We learned how to test vulnerabilities by hand with the DVWA tools, but ran into challenges trying to automate testing. One of the bigger ones was trying to handle page-changes, as that was challenging to implement. Building a crawler can be challenging, one of the issue we had with our crawler in the begin was how to kept it from crawling outside links and focus only on the internal links. Getting the crawler right can influence the result of the attack vectors.

9 ETHICAL AND LEGAL ISSUES

We have decided that testing public websites with this tool is not a good idea, as it could be unethical and/or illegal hacking. The definition of illegal hacking depends on what type of hacking we might be classified as. There are three types of hacking and along with details what each means[10]:

- White hat Hacking

  - White hat hacking as known as ethical hacking. They are referred to perform some penetration testing against the software/hardware to ensure that the data is safe and prevents others accessing unauthorized data. These people are typically hired to specifically perform the testing by whoever owns the website or service.
- Black hat Hacking
  - Black hat hacking as known as unethical hacking. They would hack a software or website to gain data, such as credentials, credit card, or PII for personal use or sell it to third party users. The general idea is the hacker is gaining personal boons in some way by taking advantage of weaknesses.
- Gray hat Hacking
  - Gray hat hacking is neither ethical or unethical. They would hack a software or website that would break the law, but they would not use it for personal gain or sell it to third party users. Often, these people will do it for the fun of cracking a system, and nothing else.

Additionally, probing public websites without consent is likely to produce legal issues, so we shall avoid doing such things. As a result, we will be hosting Damn Vulnerable Web Application (DVWA) [4] on our local machines, with the vulnerabilities deliberately exposed. This will keep possible impacts on others to a minimum during this project. For others who are testing our program, we suggest using DVWA as well.

The programs and all scripts used to research this topic will be available for personal use after the project is completed. This is largely because we view this as a tool used to test one's own website for vulnerabilities. There are some concerns over some users using this maliciously to try and find weaknesses, but we feel that it would be better to publish this tool so people can use it properly to benefit from gaining increased security by knowing where they're currently vulnerable.

We also wish to warn that this tool does not test everything possible and that if the website passes all the checks performed, it is still not fully secure. As a result, we cannot recommend relying on only this tool to help secure one's website but to use a mixture of multiple tools and services to make sure the website is not at risk of being affected by potential attacks.

10 CONCLUSIONS

We were successful able to created a tool that can be use to perform security vulnerabilities testing on both websites and web applications following OWASP web security guidelines. Our security vulnerability scanner tool will help developers mitigate discovered security vulnerabilities before moving to a more advance security testing with other tools such as Uniscan, OWASP ZAP, Sqlmap and others. Not only developers will be shown the vulnerabilities that was discovered but as well as attack vectors, risk level as well as potential fixes and a link back to OWASP guideline on which contain an in depth information about the discovered vulnerability.

It is very important to fix the vulnerability that exists on a website. By taking precautions to test a website earlier, it could potentially prevent breaches and keep plethora of customers' data secure, not limited to credit card, address, or any sensitive information.

In addition, our project can assist with the 8 security principles laid out by Saltzer. Most notably, it assists with Open Design, as our tool finds problems, and points you to solutions that would meet the standards of open design. It also assists with Complete Mediation, as most of the problems are solved by having increased checks to verify everything remains secure.

REFERENCES

[1] R. Auger. 2010. The Cross-Site Request Forgery (CSRF/XSRF) FAQ. (2010).
[2] Barracuda. Forceful Browsing Attack. (????). https://campus.barracuda.com/product/webapplicationfirewall/doc/42049348/forceful-browsing-attack/
[3] John DiGiacomo. 2017. 2017 Security Breaches: Frequency and Severity on the Rise. (2017). https://revisionlegal.com/data-breach/2017-security-breaches/.
[4] DVWA. 2017. Damn Vulnerable Web Application (DVWA). (2017). http://www.dvwa.co.uk/.
[5] HackerOne. 2017. The Hacker-Powered Security Report 2017. (2017). https://www.hackerone.com/sites/default/files/2017-06/TheHacker-PoweredSecurityReport.pdf, Accessed February 8, 2018.
[6] Matthew Goldstein Jessica Silver-Greenberg and Nicole Perlroth. 2014. JPMorgan Chase Hacking Affects 76 Million Households. (2014). https://dealbook.nytimes.com/2014/10/02/jpmorgan-discovers-further-cyber-security-issues/.
[7] Thomas Mayberry George Friedman Larry Apfelbaum, Henry Houh. 2003. Automated security threat testing of web pages. (2003).
[8] OWASP. 2016. OWASP Risk Rating Methodology. (2016). https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology.
[9] OWASP. 2017. Web Application Security Testing Cheat Sheet. (2017). https://www.owasp.org/index.php/Web_Application_Security_Testing_Cheat_Sheet.
[10] M. Ruesink. 2017. Types of Hackers: White Hat vs. Black Hat & Every Shade in Between. (2 2 2017). http://www.rasmussen.edu/degrees/technology/blog/types-of-hackers/, Accessed February 8, 2018.
[11] Nicole Perloth Scott Shane and David E. Sanger. 2017. Security Breach and Spilled Secrets Have Shaken the N.S.A. to Its Core. (2017). https://www.nytimes.com/2017/11/12/us/nsa-shadow-brokers.html.
[12] Sopho. 2012. Security Threat Report 2012. (2012). https://www.sophos.com/medialibrary/pdfs/other/sophossecuritythreatreport2012.pdf.