Automated Detection of Vulnerabilities on Websites (Tentative Title)*

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 (Tentative Title). In *Proceedings of ACM Conference, Washington, DC, USA, July 2017 (Conference'17),* 4 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 [10].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 [9] 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[7] such as configuration management,Authentication,Session Management,Data validation and among other.

2 APPLICATION DESIGN

We will be developing an application that takes a given Universal Resource Locator (URL) and attempts to perform vulnerabilities checking. The application will 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, least privilege, access control, and among other.

We will not be using public websites without consent as it would be considered illegal hacking and unethical. Instead, we will be 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
- Failure to restrict files, folders, and URL access
- Cross-Site Request Forgery (CSRF)
- Disclosure of sensitive files such as web configuration information

2.1 To-do List

- Develop python scripts that will perform vulnerabilities
  - SQL Injections
  - Cross Site Scripting (XSS)
  - Broken Authentication
  - Failure to restrict files, folders, and URL access
  - Cross-Site Request Forgery (CSRF)
- Display output(s)
  - Display score for each violations
  - Classify which violated
  - Display a fix for it (prevent it)
- Document the codes and how to use it
- Follow Object Oriented Programming (OOP) standard

We have implemented several vulnerabilities on a test environment using DVWA to simulate an actual website. In addition to

*Produces the permission block, and copyright information

Conference'17, Washington, DC, USA

DOI: 10.1145/nnnnnnn.nnnnnnn

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. 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.

Below is a preliminary design (tree diagrams) to see what type of scripts and classes are being used. We will be adding more scripts and classes as we progress.

```
Automated-Detection-Vulnerabilities/
    Attacks/
    |   ├── ActiveSQLInjection.py
    |   ├── BruteForce.py
    |   ├── DirectoriesFilesTraversal.py
    |   ├── Fuzz.py
    |   ├── PassiveSQLInjection.py
    |   ├── Sensitive.py
    |   └── XSS.py
    |       ├── XSS (Stored)
    |       └── XSS (Reflected)
    ├── Utilities/
    |   ├── Files.py
    |   ├── Input.py
    |   ├── Link.py
    |   ├── Classification.py
    |   └── Requests.py
    ├── Files/
    |   ├── activeSQL.txt
    |   ├── passiveSQL.txt
    |   ├── sensitive.txt
    |   ├── userspass.txt
    |   ├── directories.txt
    |   └── filenames.txt
    ├── README.MD
    ├── Test.py
    └── main.py
```

2.2 Completed

Developed a command line style program, it takes two user's arguments. Based on the user's inputs, it will execute a single vulnerability or all vulnerabilities.

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

We have implemented few vulnerabilities and helper class:

- **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.
- **Fuzzer(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.
- **File.py**: Class file (OOP) that returns the file path.
- **Requests.py**: Class file (OOP) that returns the session of the page. To prevent logging out when performing vulnerabilities on the web page.
- **ActiveSQLInjection.py**: Class file (OOP) This is partial completed. It takes the Link(Link.py) and go through the list of links and their input and attempt multiple active SQL injections vectors.
- **PassiveSQLInjection.py**: Class file (OOP) This is partial completed. It takes the Link(Link.py) and go through the list of links and their input and attempt multiple passive SQL injections vectors.
- **DirectoriesFilesTraversal.py**: Class file (OOP) 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.

2.3 What's next

For the next few weeks, we will be implementing more vulnerabilities and document the codes/process that are listed in To-do List section. In addition to that, we will be performing a code review to eliminate any duplicate codes, leftover codes, and ensure the code is meeting the coding standard design. At the same time, we will be doing some testing to ensure the program does not break when executing specific tasks.

If we have time, we will be focusing on the performances. The way our program works is that it executes tasks one at a time. We are planning on implementing multi-threading that executes multiple tasks at the same time. Thus, it will speed up the process.

3 VULNERABILITIES

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

3.1 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.

3.2 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.
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.

3.3 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.

3.4 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.

3.5 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.

4 ETHICAL AND LEGAL ISSUES

We will not be testing public websites with this tool, 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[8]:

- 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 machine, with the vulnerabilities deliberately exposed. This will keep possible impacts on others to a minimum during this project.

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.

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] OWASP. 2017. Web Application Security Testing Cheat Sheet. (2017). https://www.owasp.org/index.php/Web_Application_Security_Testing_Cheat_Sheet.

[8] 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.

[9] 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.

[10] Sopho. 2012. Security Threat Report 2012. (2012). https://www.sophos.com/medialibrary/pdfs/other/sophossecuritythreatreport2012.pdf.