RES MILITARIS
REVUE EUROPEENNE D ETUDES EUROPEAN JOURNAL OF MILITARY STUDIES
Social Science Journal

Using Python to Detect Web application vulnerability

By

Ann Zeki Ablahd
Computer Engineering Department, Kirkuk Technical College, Northern Technical University, Iraq
Email: drann@ntu.edu.iq

Abstract

The number of web application grows sharply because of a web application is a common way of delivering all services via the Internet. The developing such application with a fewer experience and without testing caused a huge vulnerability in it. The web application vulnerability is a weak point resulted through web application designing. There are many attackers exploit this vulnerability for gaining access to all unauthorized internal objects to compromise the application, modify data and steal the most important information. The aim of this proposed system is to detect the web application vulnerabilities before exploited by an attacker. A special scanner was built using python 3.7 built-in tools like AST, CFG, Flask, and Django to detect these vulnerabilities. There are different risks infect a web application caused by this vulnerability two types of them were solved in this proposed system. The proposed scanner detects the injection flaws command execution and Cross-Site Scripting (XSS) injection. The fixed-point algorithm is used for finding web application vulnerabilities after analysis and extracts its features. The proposed scanner called SCANSCX. SCANSCX has been created with flexible tools. In order to test and evaluate the ability of SCANSCX, a number of vulnerable applications were designed. All designed examples are identified as being vulnerable. The SCANSCX is a realistic application because it runs on windows and linx operating systems. SCANSCX is a big project that spends very long time on analysis, designed an application and was therefore terminated.

Keywords: web app, vulnerabilities, scanner, Flask, AST.

Introduction

The growth of the World Wide Web, internet and web application has an important impact on all professional and social human being life. The web application everyday was growing rapidly and more complex because of considering that it is the first face of all businesses in the world. The existing vulnerabilities in a web application due to developing such application with fewer programmer's experience and without testing the designed web application from loopholes. The code injection which is executed by a vulnerable application is a general term for threat, classified as number one in web application risk. This risk occurs when the web application accepts and executed input commands with a parameter (Salman, Khalaf, & Abdulsahib, 2019). The injection flaws infect the web application due to the vulnerability of it. This vulnerability came from different factors like an insecure layer network, the complexity of web technology and dynamic web applications that controlled most of the time by end-user. The attacker can exploit any web application weakness, 96% of them have critical vulnerabilities (Cenzic, 2014) and can be infected by different attacks due to the less experience of web application designers (Ann Zeki, 2020; Khalaf & Abdulsahib, 2019; Khalaf, Abdulsahib, & Sabbar, 2020).

The web application vulnerabilities are very dangerous and can be exploited by an attacker. Cross-Site Scripting injection (XSS), command execution attacks are examples of web

application attacks (Ogudo, Muwawa Jean Nestor, Ibrahim Khalaf, & Daei Kasmaei, 2019; Zheng, Zhang, & Ganesh, 2013). The XSS and command injections are the most important dangerous attack founded in all network classes. The Untrusted web application enables the attack to damages and steals sensitive information by unauthorized users. The results of these attacks are stealing secure data, breaking the integrity of data, and affect the availability of web app. It is an important step is detecting the vulnerabilities of a web app to avoid exploiting these vulnerabilities by attackers. The related works of command injection attacks are limited while focusing on CSS vulnerabilities Cross-Site Scripting. In 2004 Hunan used precondition in examining input data by "Tainted Data Tracking ". There is a drawback in this paper by assuming a technique accurately expressed for sensitive functions. In 2004 Boyd introduced an "SQL Randomization" this based on instruction set randomization, by appending random number after SQL keyword. The SQL parser in the runtime finds the injected attack but does not stop the attack. In 2005 Vale introduced the research "Intrusion Detection Based on Static Analysis". This research built a monitor app with a query model for distinguishing not match query at runtime (Khairnar, 2017; Khalaf, Abdulsahib, Kasmaei, & Ogudo, 2020). This research produced a guarantee of a huge number of false positives. In 2005 Tadeusz and Chris used Context-Sensitive String Evaluation for defending against the Injection attack; this paper does not have a dedicated and specialized tool for detecting a command injection attack. In 2007 Jin-Cherng proposed a gateway security for preventing code injection attack. It used numerical results for preventing code injections attacks (Lin & Chen, 2007). But the deployment practice needs modification in existing infrastructure. In 2011 Papagiannis used PHP Aspis in l taint tracking in protecting from injection attack (Papagiannis, Migliavacca, & Pietzuch, 2011). But PHP is not suitable for designing a web app and it is a poor coding in finding the vulnerability of web app. In 2013 presents RT-WASP tool but does not detect Injections (like XSS and SQL), thus the author plans for extending the RTWASP tool to encompass both the above attacks (Pietraszek & Berghe, 2005). In 2016 A. Alazab presents a new model for detecting and preventing injection at runtime, this model has successful against any type of vulnerability. The drawback is imposed very low overhead for the system that will be determined by the network speed and the access of database server (Medhane, 2013).

In (Alazab & Khresiat, 2016) the Decaf represents a new instruction for taint tracking engine, which controlled by tiny code generator used to accomplish optimization while ensuring taint propagation with high precision. In (Henderson et al., 2016) the Feng C introduce a model based on taint analysis, this model detects stack over web app vulnerabilities. Most of the previous tools mentioned in this paper took a long time for scanning any web application, that why it needs a perfect scanner for parsing any web application with a short time and precise results and test two typesof injection. The proposed scanner SCANSCX was built, based on Python 3.7 programming language in analyzing, detecting and evaluating is able to detect Cross-Site Scripting injection (XSS) and command execution injections in the web app in high success rate. The Web app is like any computer system app but it works with the Internet (Abdulsahib & Khalaf, 2018; Khalaf, Abdulsahib, & Sadik, 2018).

Web app architecture is a client with a server component, including user interfaces, databases, and middleware. Most attacks (approximately 75%) today come with a web app. The client-side consists of static web pages and embedded scripting languages like JavaScript that be executed by browser (Feng & Zhang, 2017). The client sent a URL (Uniform Request Locator) a request to a web server using the internet. But, the server-side processed a client request using dynamic web pages such as HTML, PHP, ASP, CSS, or JAVA Servlets to supply suitable respond as shown in Figure 1.

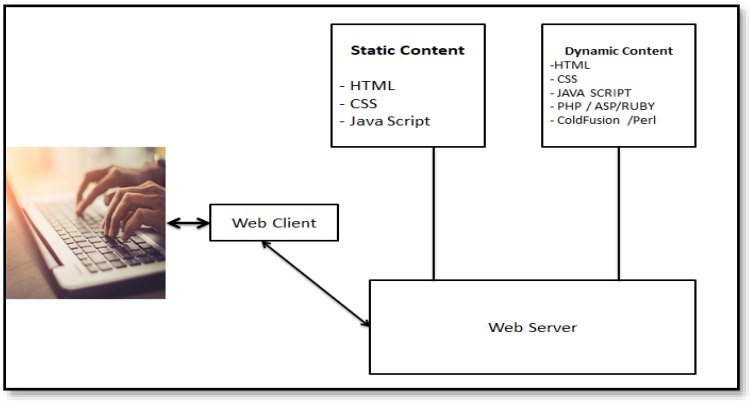

Figure.1 *Architecture of Web App*

The web vulnerability is a weakness in designating web application. This vulnerability creates a hole to be exploited by the attacker in harming a web application by accessing and stealing sensitive data(Razaghpanah et al., 2017). The input validation is the most common problem of the web app vulnerability because of lacking validation and sufficient mechanism on user input, in addition to that failing through handling errors and closing connection not properly (Wassermann & Su, 2007). Most of the web app is poor in programming and has a highly vulnerable and exposed. Python is a dynamic programming language. Produced in the 1990s by Guido van Rossum (f3.7) is the last version was used in building SCANSCX. There are many programming paradigms supported by python including object-oriented programming and functional.

The Python web framework is divided into two sets: first is non-full stack and second: is a full-stack framework. The first one is a framework that does not have all packages for developing a complete web app (Backes, Bugiel, & Derr, 2016; Stock, Pellegrino, Rossow, Johns, & Backes, 2016). But the second (full-stack framework) is a framework having all needing parts for developing a web app. The most common full-stack web framework is Django was chosen. A non-full stack micro web framework is Flask was chosen. It is a web framework tool that contains what you need to design a complete web application. Django used architecture (MVC) Model View Controller. The web app of Django consists of a collection of apps. The Django power is the easiest reuse and linked together using URLs (Hunt, 2019; Natarajan & Moh, 2016). The app of Django has the following modules:

- The main module of the app to start code executing.
- Test module used for app testing.
- View module for app visualization.
- URLs module used for URLs mapping.
- Models module for modeling databases.
- Apps module for apps nested.

There are many well-known web sites or frameworks used in Django like Mozilla, Flask, and Instagram.

It is a micro web framework in python 3. The importance of flask appears in a small number of codes you can design a web page. It is very flexible in the developer of databases. Easy and powerful in designing server development and packages. In python is used input () command for inputting data, while the command eval (value) used for handling these data without filtration. But with using the flask tool it input data with sanitization (Nithya, 2013).

This sanitization used in the protection of web app vulnerability. A markup class used string. replace () command for input sanitizing. This class contains an escaping function that is responsible for escaping HTML markup for input sanitizing; see Figure 2 display flask codes for sanitizing user input.

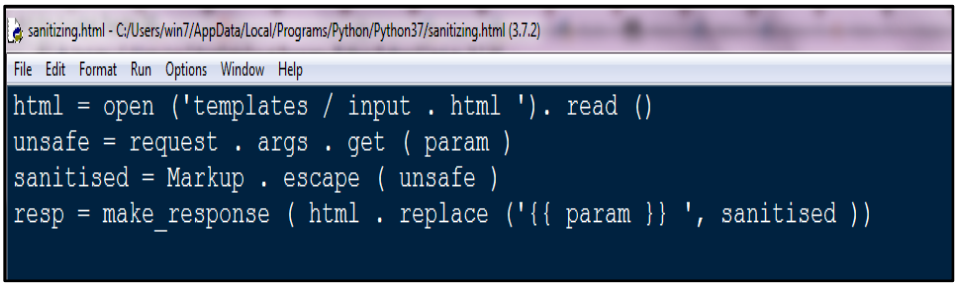

Figure. 2 *display flask code for sanitizing user input*

Flask has many functions that used in this paper like (Templates, Responses, Routing, Requests) functions. The "app.route" decorator is a routing function used in flask for binding functions to the URL path. Get and POST requests are other parameters of "app.route" (OWASP, 2018). The Get request used in fetching information from the server. But POST request used in sending data to a server used. Figure (3) explains an example of these requests in the login form.

```
from flask import Flask,request
@app . route ('/ login ', methods =[ 'GET ', 'POST '])
 def login ():
 if request . method == 'POST ':
 do_the_login ()
 else :
      show_the_login_form ()
```

Figure. 3 *login form with "GET and POST" requests*

The Flask tool is used in SCANSCX because it is simple, easy on testing and development a web app. The python Template is used for representing data in different way with attractive forms (Tian, Zhao, Zhang, & LI, 2014; Valeur, Mutz, & Vigna, 2005). Jinja2 is a flask template engine was used to keep application security. This Template used to escape all dangerous input. Django used a Template for generating dynamic HTML pages. A simple web app page was built using python tools like (flask, sqlite3, Django and virtualenvwrapper). The ("pip install tool") command used for setting every tool. Figure (4) shows a code for a designed web app using a flask tool (Rajeh & Abed, 2017).

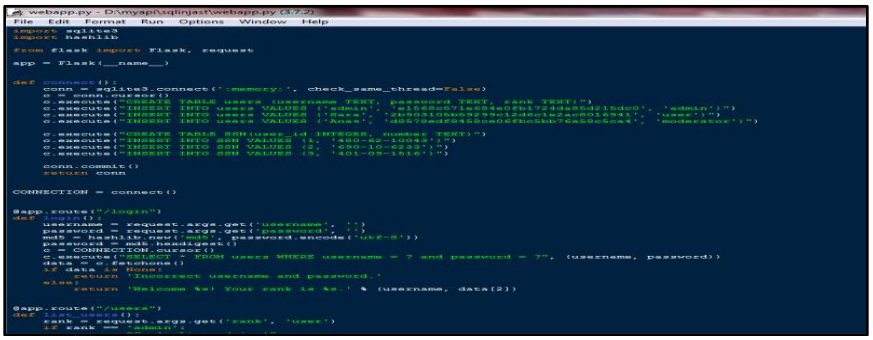

Figure. 4 *displays code of designed web application*

Abstract Syntax Tree (AST) is a standard library module in python. This module helps the python syntax grammar application in processing trees. The command ("import ast") used for calling the AST library. The source code of the tested web page converted to AST form as a step of parsing in finding web page vulnerability. The data structure for AST lets the python user to analyze, testing the grammer of python abstract grammar. By using this module, the application source code transferred to abstract tree (WU, CHENG, & HU, 2015). A standard tree was built from a different statement of the source code using a built-in class "ast.NodeVisitor". The "ast.NodeVisitor" is a class used for followed all the tree nodes. For each founded node call the function visit () to return value. When the returned value="none" that mean remove the node from tree.

Otherwise, value of the node value will be replaced by the return value. The prepared tool SCANSCX used for detecting vulnerabilities in a web application. This tool used the AST library. The standard AST library downloaded using a python 3.7 command ("pip install ast") (Chen, Jin, Yu, & Chen, 2018; Li, Ma, Shen, Lv, & Zhang, 2019; Lima Filho, Silveira, de Medeiros Brito Junior, Vargas-Solar, & Silveira, 2019). The AST module used for generating a tree with a Control Flow Graph (CFG) to web app source code. Figure 5 shows a simple example of using the AST module.

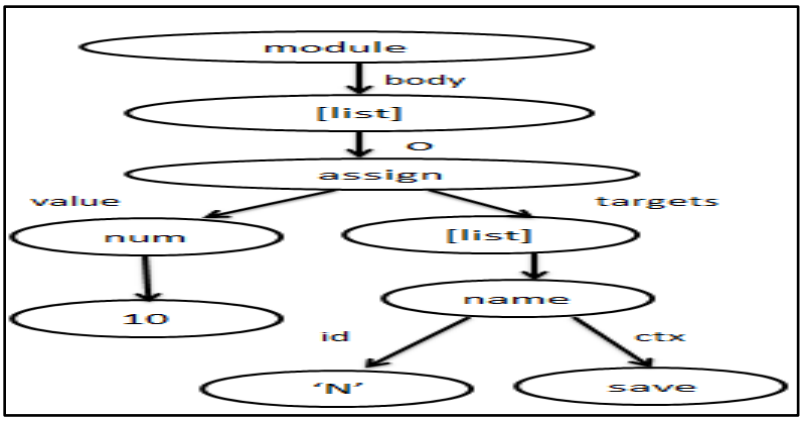

Figure.5 *simple example is used AST module for N=10*

CFG is a mnemonic of Control Flow Graph is a directed graph contains nodes and edges to represent the specific program. The CFG consist of two nodes one for entry and another for exit called entry and exit nodes. A given (n) nodes, pred (n) denote to predecessor set of nodes and succ (n) denote to set of successors nodes. Figure (6) shows example of changing AST to CFG (Pektaş & Acarman, 2017).

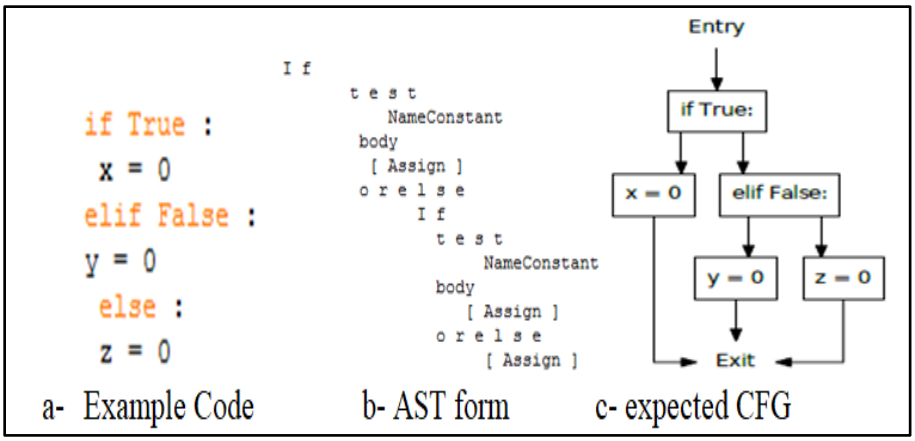

Figure. 6 *Converting to CFG from AST form*

Such example, the AST is imagining as a tree. There are four types of nodes, the first, body, NameConstant and assign nodes. The first node includes the test node while the body nodes is proceeded as a list node. The condition nodes are represented by (NameConstant). While all other statements treated as assign nodes. Python language has rules for representing operators, objects, and functions. The algorithm that applied for producing nodes from the resulting CFG is a Fixed-point algorithm. In designing SCANSCX, converting data from structure to others was used as a Framework Adaptor class, it contains a CFGs list.

The Framework Adapter used with Flask and Django in the implementation of SCANSCX. By using Flask tool all the designed web apps are realized as a decoration tool. The command (@app.route()) in a Flask Adapter tool used as a decorator command for adding nodes to the CFGs list. The meaning of fixed point in this algorithm is iterate the analysis of dataflow until no changing (Qiang et al., 2017). Command injection is an attack. The goal of this attack injects an operating system commands to be executed via a vulnerable web app. These injection attacks happen when an app passes unsafe user input data (cookies, HTTP, etc.) to a shell of the system. The uncontrolled injection reaches the interpreter. It injects commands and running these commands into the server shell. For example, deleting system files, reading password files, and different dangerous operations. The workaround of such a problem is input sanitization (filter input from different symbols like; | >> << ... etc. by parsing it before doing anything) and accept the desired character only. Sanitization is the process of removing all dangerous characters from URL input before passing to the search engine. There are many symbols like (<,>, >>, <<, |…) were used by attacker to inject command attack, at first, these symbols were removed by SCANSCX parser. The parser will escape the special symbols for windows such as '&$|><#....' and other symbols like '()-*? ..' for Linux operating system (Pektaş & Acarman, 2017). There are two categories of command injection: first is the result–based and second is blind command injection.

Result-Based Command Injection Technique

This attack is a common and simpler technique of command injection. The vulnerable app will display the injected command as output. The attacker used the operator of a common Linux shell and directly can infer the command injection succeeded or not. The results of this injection are visible. Figure (7) shows an example of a result-based command injection.

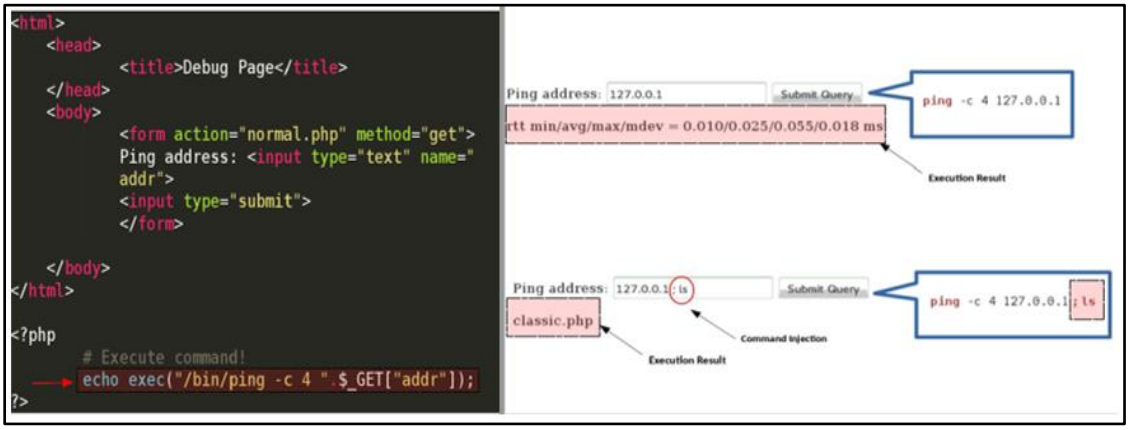

Figure. 7 *(a) Example of result-based (b) Output of command injection*

Blind Command Injection

In this category of injection, the attacker injects a command to a vulnerable app, but it does not display any results into a screen. The injected command output is not visible. Figure (8) displays an example of a blind command injection [29].

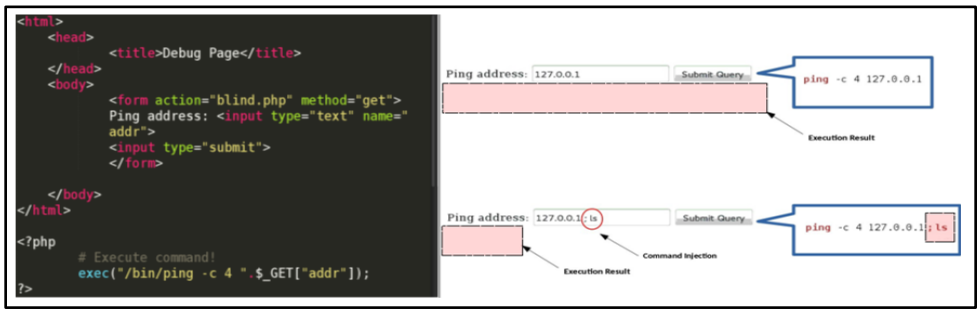

Figure. 8 (a) *Example of blind command injection (b) Output of blind command injection*

There is a difference between blind and result-based command injection techniques lies in the way of retrieving data after execute the injected command. The attacker observes the output of a requested command. In case of no returned result backed to attacker, the last used the following three techniques for inferring output for injected command:

- According to delay time, the attacker can estimate the injected command results. This technique is called “time-based semi-blind injection command”.
- Depending on redirection output, the attacker writes a file with the output of the injected command. This technique is called “file-based blind command injection”.
- The attacker merged the above two techniques of a blind command injection to show a new technique called temple-based semi-blind command.

Moreover, by this technique, the attacker stores the result of the injected command attack with the temporary text file in a directory [25]. The cross-site scripting attack is abbreviated of XSS (or CSS) is a type of very common injection vulnerability found in web app. Each web page contains text and Html markup generated by the server and interpreted by a client browser. There are two types of web site pages static and dynamic pages. The static pages have full control by browser to interpreting these pages, while dynamic pages do not have full control about their interpreted outputs by the client. If the distrusted content used dynamic page neither the client and nor the web sites have enough information to recognize what happened and what to do for protective actions (Feng & Zhang, 2017).

This attack happens in a dynamic web app, when untrusted data send to another user without sanitization, and when input user reaches the HTML pages of a web app. The attacker attacks the website databases, by injects, it is payload through web app vulnerable. The victim browser requests a web app from web server. The server will serve the victim browser with payload of attacker will displayed as a part of the HTML body. The web browser will display the injected attack contained in the HTML body, by this case the cookie of a victim will be sent to the attacker’s server. The attacker will extract this cookie, while the HTTP request arrives server. After that, the attacker can be used the stolen cookie for impersonation. Figure (9) shows the walkthrough of XSS attack.

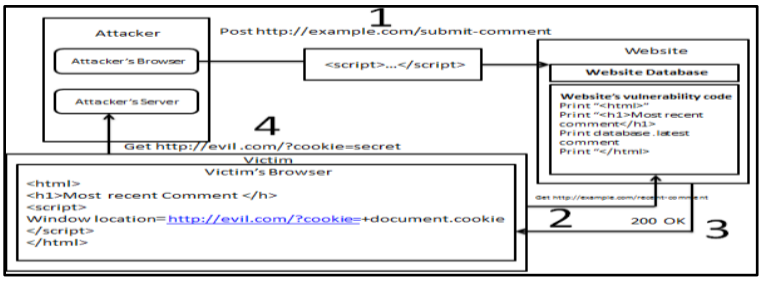

Figure.9 *represents a simple XSS attack*

There are two kinds of XSS attacks: persistent and non-persistent (Ann Zeki, 2020). The user input inserted directly into the resulting XSS. In persistent XSS the malicious input will be stored in a database or web app. The HTML in non-persistent problems that caused by XSS are:

- Change the looking and the contents of web sites by inserting any JavaScript code.
- Redirect the web site to evil steps, by forcing the victims to download different malicious codes like a virus and Trojans.
- The attacker can steal and read user's cookies (The cookie is a small file stored on a user's computer, designed to hold a modest data specified by a particular website and client to be accessed by the web server or client computer). By stolen such a cookie, the attacker has the ability to log in to sites, even to the account of administrator, without knowing the password and user names. This attack is the most dangerous step of XSS attack.
- The attacker changed the DOM (Document Object Model is a programming interface for all documents of HTML) of a website (Ann Zeki, 2020).

To detect such vulnerabilities a special tool called SCANSCX was designed, that connects a user input with a sensitive code to detect dangerous performed operation.

The Proposed System SCANSCX

At first, to demonstrate SCANSCX a simple web app was designed by using the python flask tool (figure 10 represents preparing a simple web app). There are many tools were set up with a different command like: ("pip install flask") for installing flask tool, ("pip install Django") for installing Django tool, ("pip install virtualenvwrapper-win") (Ann Zeki, 2020) for setting Virtual Environment, ("pip install ast") for installing the standard library of Abstract Syntax Tree and ("mkvirtualenv SCANSCX") used to create a virtual environment. Activate instruction used for activating a virtual environment.

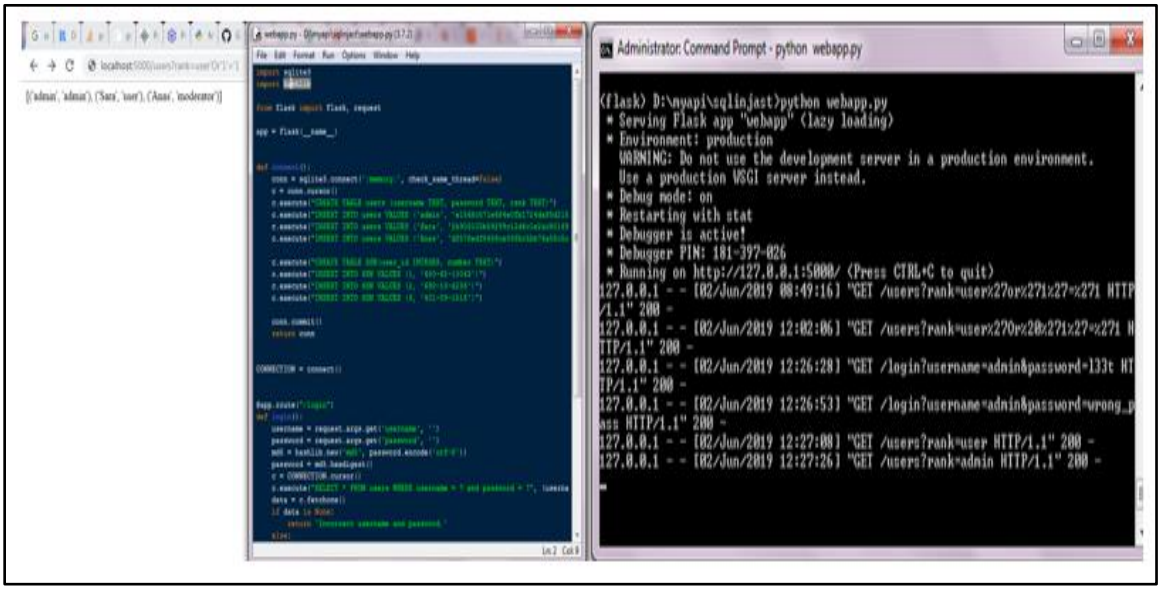

Figure. 10 *(a) Represents code of built web app (b) Status code of HTTP and access results using flask tool and displayed Explorer of the web.*

The proposed system SCANSCX is a software tool written in Python (version 3.7) and it can run in both Windows and Linux. Figure 11 represents an algorithm of prepared parser.

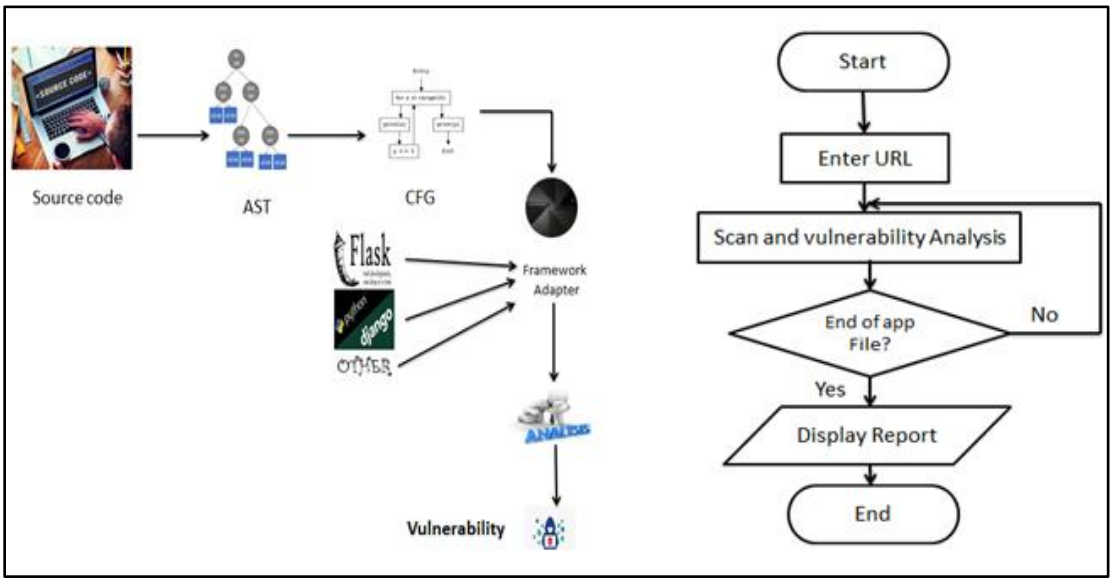

Figure 11 *represents an algorithm of SCANSCX*

The AST is a python module that helps the app for processing trees. The ast.AST is a base class for all nodes of AST class. The AST class has many subclasses such as (generate_ast), (ast_helper),(ast.NodeVisitor), (ast.parse(f.read)), (FrameworkAdapter). Each subclass has it is functions. For e.g. the subclass (ast_helper) used for scanning and splitting the tested web app source code into a list of tokens to produse an abstract tree. The abstract tree of AST is a set of linked nodes by edges that based on the python language grammer [31]. The ouput tree sent to (ast.parse(source.read())) function that responsible for generating a CFG file graph file the to be an object tree. The subclass (ast.NodeVisitor) used for tracking all the nodes of the AST tree. The subclass NodeVisitor used for scanning trees. Figure (12) shows an example of changing to CFG (Control Flow Graph) graph from the AST tree (Ann Zeki, 2020).

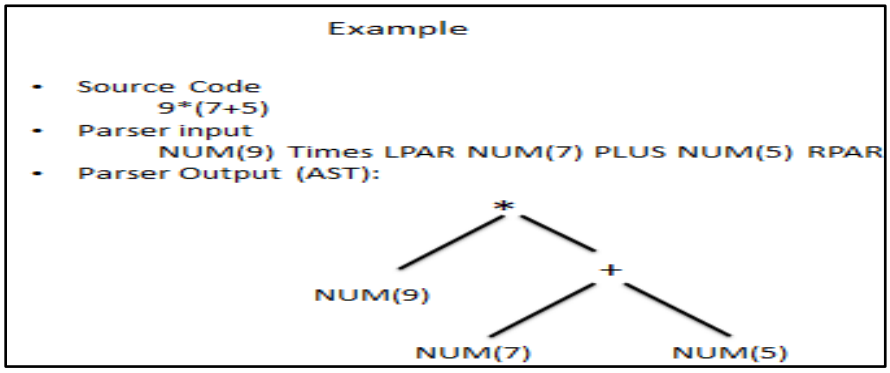

Figure.12 *displays example of changing to CFG from the AST tree*

In figure (12), the source code means the parsed user input, to the AST tool for traversing to CFG form. The generating of the CFG tool sends to the Flask adapter. The Flask Adapter is an effective python tool used for testing client requests without running the server of Flask. The algorithm fixed-point will be applied to CFG for producing a new nodes and iterate analysis of dataflow using flexible of the fixed-point algorithm until nothing changing, that why is called a fixed point. Through analysis the produced node of test web application a potential vulnerability will found (Ann Zeki, 2020). Through designing SCANSCX a Framework Adapter class is used (list of CFGs) for converting data from form to other. Figure 13 represents a part of SCANSCX codes.

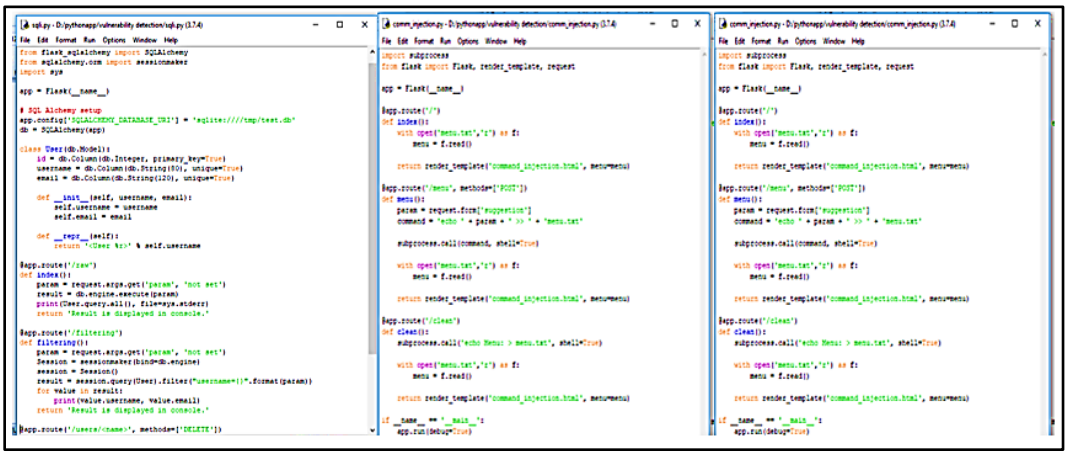

Figure. *13 shows a part of SCANSCX*

Results of Scansex

This part represents running the SCANSCX. SCANSCX accepts a path of a web app to be tested, such as "python comm_injection.py webapp_namepath". Due to the Cross-Site Scripting injection (XSS) and command execution vulnerability dangerous, a proposed developed tool was prepared, a parser for detecting Cross-Site Scripting injection (XSS) and command execution vulnerability. This tool is easy and smart to load a web app, analyze it, and detect all types of Cross-Site Scripting injection (XSS) and command execution vulnerability. The steps of running SCANSCX and generating the parsing output report of a web app to declaring if there is Cross-Site Scripting injection (XSS) and command execution vulnerability of web app as follow:

- Scanning the input command line and it is argument.
- Generate AST tree (Abstract Syntax Tree).
- Generate CFG (Control Flow Graph) for AST tree.
- The outcome of producing CFG will be passed to the Framework Adapter.The Framework Adapter will sign the argument as tainted sources to detect Cross-Site Scripting injection (XSS) and command execution vulnerability.
- Detect all suspicious XSS and command vulnerability.
- Output a report for all detecting vulnerability.

The output report of SCANSCX like in figure (14) (15) (16) (17) represents the testing of web applications sites http://sqlfiddle.com, http://facebook.com, , https://www.scriptalert1.com, https:/tck.ntu.edu.iq respectively.

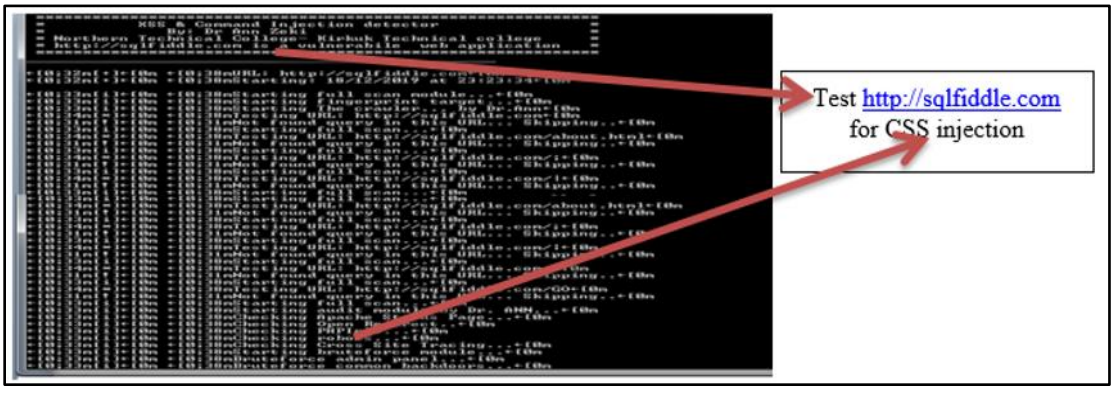

Figure. 14 *represents the output of SCANSCX in the testing of http://sqlfiddle.com*

Figure. 15 *represents the output of SCANSCX in the testing of http://facebook.com*

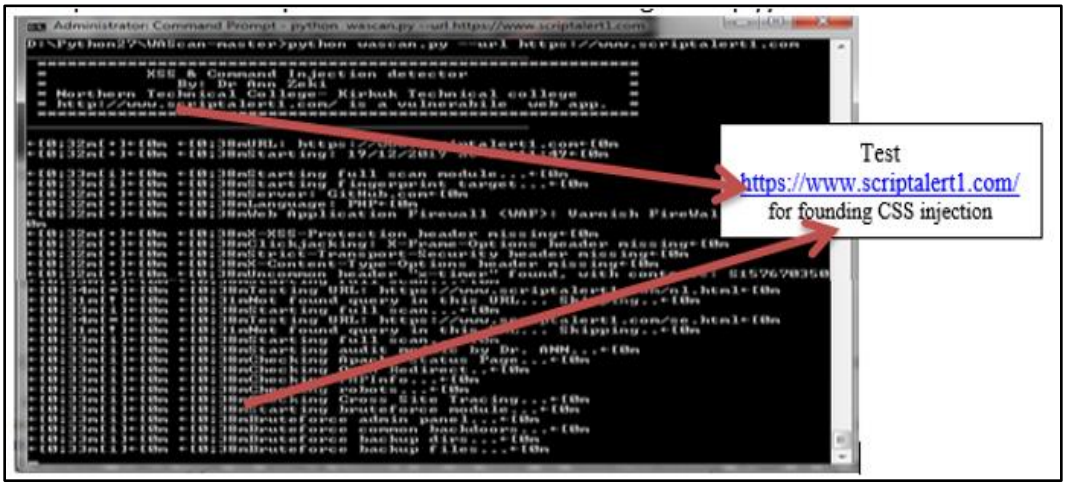

Figure. 16 *represents the output of SCANSCX in the testing of https://www.scriptalert1.com*

Figure. 17 *represents the output of SCANSCX in the testing of https://tck.ntu.edu.iq*

After testing a huge web app to find XSS and Command vulnerability it shows that Cross-Site Scripting forms (45%), Command Injection (30%) and others form 25%. Figure (18) displays a pie chart of SCANSCX in testing.

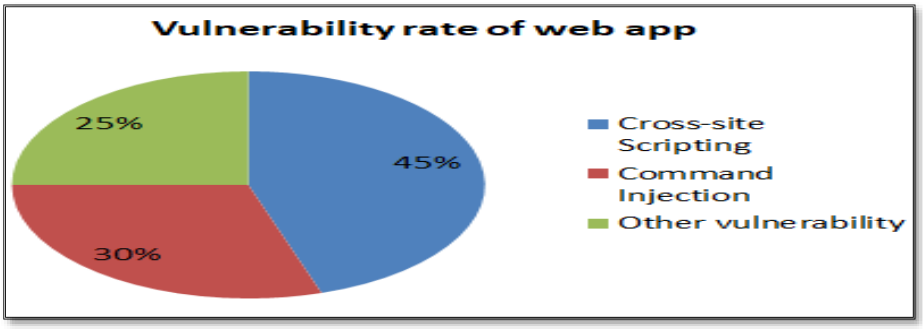

Figure. 18 *pie chart represents the penetration rate of SCANSCX testing.*

Conclusion

Cross-Site Scripting injection (XSS) and command execution vulnerability are web application modern attacks. These attacks are not commonly known to all worlds. Each type of attack consumes lot of time in understanding the behavior and the way of detecting them. Because of the wide and the problems caused by both Cross-Site Scripting injections (XSS), command execution vulnerability of web applications, a SCANSCX tool was proposed for detecting the vulnerability of them. A new version of python (3.7) programming language with Flask and Django was used in preparing this tool for analyzing many different web applications to detect these attacks. A SCANSCX tool was evaluated, to find all types of Cross-Site Scripting injection (XSS) and command execution attacks. It is easy to apply (in each web app) and flexible in updating.

Reference

Abdulsahib, G. M., & Khalaf, O. I. (2018). Comparison and evaluation of cloud processing models in cloud-based networks. International Journal of Simulation--Systems, Science & Technology, 19(5).

Alazab, A., & Khresiat, A. (2016). New strategy for mitigating of SQL injection attack. International Journal of Computer Applications, 154(11).

Ann Zeki, S. A. (2020). Using Flask for SQLIA Detection and Production. TJES Tikrit Joural Engineering and Science.

Backes, M., Bugiel, S., & Derr, E. (2016). Reliable third-party library detection in android and its security applications. Paper presented at the Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security.

Cenzic. (2014). Application Vulnerability Trends Report :2014.

Chen, Y., Jin, B., Yu, D., & Chen, J. (2018). Malware Variants Detection Using Behavior Destructive Features. Paper presented at the 2018 IEEE Symposium on Privacy-Aware Computing (PAC).

Feng, C., & Zhang, X. (2017). A static taint detection method for stack overflow vulnerabilities in binaries. Paper presented at the 2017 4th International Conference on Information Science and Control Engineering (ICISCE).

Henderson, A., Yan, L. K., Hu, X., Prakash, A., Yin, H., & McCamant, S. (2016). Decaf: A platform-neutral whole-system dynamic binary analysis platform. IEEE Transactions on Software Engineering, 43(2), 164-184.

Hunt, J. (2019). Regular Expressions in Python Advanced Guide to Python 3 Programming (pp. 257-271): Springer.

Khairnar, T. S. (2017). Next Generation Black-Box Web Application Vulnerability Analysis Framework. Arizona State University.

Khalaf, O. I., & Abdulsahib, G. M. (2019). Frequency Estimation by the Method of Minimum Mean Squared Error and P-value Distributed in the Wireless Sensor Network. Journal of Information Science and Engineering, 35(5), 1099-1112.

Khalaf, O. I., Abdulsahib, G. M., Kasmaei, H. D., & Ogudo, K. A. (2020). A New Algorithm on Application of Blockchain Technology in Live Stream Video Transmissions and Telecommunications. International Journal of e-Collaboration (IJeC), 16(1), 16-32.

Khalaf, O. I., Abdulsahib, G. M., & Sabbar, B. M. (2020). Optimization of Wireless Sensor Network Coverage using the Bee Algorithm. Journal of Information Science and Engineering, 36(2), 377-386.

Khalaf, O. I., Abdulsahib, G. M., & Sadik, M. (2018). A modified algorithm for improving lifetime WSN. Journal of Engineering and Applied Sciences, 13, 9277-9282.

Li, Y., Ma, L., Shen, L., Lv, J., & Zhang, P. (2019). Open source software security vulnerability detection based on dynamic behavior features. PloS one, 14(8).

Lima Filho, F. S. d., Silveira, F. A., de Medeiros Brito Junior, A., Vargas-Solar, G., & Silveira, L. F. (2019). Smart Detection: An Online Approach for DoS/DDoS Attack Detection Using Machine Learning. Security and Communication Networks, 2019.

Lin, J.-C., & Chen, J.-M. (2007). The automatic defense mechanism for malicious injection attack. Paper presented at the 7th IEEE International Conference on Computer and Information Technology (CIT 2007).

Medhane, M. (2013). R-WASP: real time-web application SQL injection detector and preventer. International Journal of Innovative Technology and Exploring Engineering (IJITEE), 2(5), 327-330.

Natarajan, S., & Moh, M. (2016). Recommending news based on hybrid user profile, popularity, trends, and location. Paper presented at the 2016 international conference on collaboration technologies and systems (CTS).

Nithya, R. R. (2013). A survey on SQL injection attacks, their detection and prevention techniques. International Journal of Engineering and Computer Science, 2(04).

Ogudo, K. A., Muwawa Jean Nestor, D., Ibrahim Khalaf, O., & Daei Kasmaei, H. (2019). A Device Performance and Data Analytics Concept for Smartphones' IoT Services and Machine-Type Communication in Cellular Networks. Symmetry, 11(4), 593.

OWASP, T. (2018). 10 2017. The Ten Most Critical Web Application Security Risks. Release Candidate, 2.

Papagiannis, I., Migliavacca, M., & Pietzuch, P. (2011). PHP Aspis: using partial taint tracking to protect against injection attacks. Paper presented at the 2nd USENIX Conference on Web Application Development.

Pektaş, A., & Acarman, T. (2017). Malware classification based on API calls and behaviour analysis. IET Information Security, 12(2), 107-117.

Pietraszek, T., & Berghe, C. V. (2005). Defending against injection attacks through context-sensitive string evaluation. Paper presented at the International Workshop on Recent Advances in Intrusion Detection.

Qiang, W., Liao, Y., Sun, G., Yang, L. T., Zou, D., & Jin, H. (2017). Patch-related vulnerability detection based on symbolic execution. IEEE Access, 5, 20777-20784.

Rajeh, W., & Abed, A. (2017). A novel three-tier SQLi detection and mitigation scheme for cloud environments. Paper presented at the 2017 International Conference on Electrical Engineering and Computer Science (ICECOS).

Razaghpanah, A., Niaki, A. A., Vallina-Rodriguez, N., Sundaresan, S., Amann, J., & Gill, P. (2017). Studying TLS usage in Android apps. Paper presented at the Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies.

Salman, A. D., Khalaf, O. I., & Abdulsahib, G. M. (2019). An adaptive intelligent alarm

system for wireless sensor network. Indonesian Journal of Electrical Engineering and Computer Science, 15(1), 142-147.

Stock, B., Pellegrino, G., Rossow, C., Johns, M., & Backes, M. (2016). Hey, you have a problem: On the feasibility of large-scale web vulnerability notification. Paper presented at the 25th {USENIX} Security Symposium ({USENIX} Security 16).

Tian, Y., Zhao, Z., Zhang, H., & LI, X.-s. (2014). Second-order SQL Injection Atteck Defense Model. Netinfo Security, 11.

Valeur, F., Mutz, D., & Vigna, G. (2005). A learning-based approach to the detection of SQL attacks. Paper presented at the International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment.

Wassermann, G., & Su, Z. (2007). Sound and precise analysis of web applications for injection vulnerabilities. Paper presented at the Proceedings of the 28th ACM SIGPLAN Conference on Programming Language Design and Implementation.

WU, S.-h., CHENG, S.-b., & HU, Y. (2015). Web attack detection method based on support vector machines. Computer Science, S1.

Zheng, Y., Zhang, X., & Ganesh, V. (2013). Z3-str: A z3-based string solver for web application analysis. Paper presented at the Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering.