Regular Article

An Improving Way for Website Security Assessment

Nguyen Duc Thai, Nguyen Huu Hieu

Faculty of Computer Science and Engineering, Ho Chi Minh City University of Technology (VNUHCM), Ho Chi Minh City, Vietnam

Correspondence: Nguyen Duc Thai, ngdthai@hcmut.edu.vn

Communication: received 10 June 2019, revised 21 June 2019, accepted 18 March 2020
Online publication: 10 August 2020, Digital Object Identifier: 10.21553/rev-jec.239
The associate editor coordinating the review of this article and recommending it for publication was Prof. Tran Manh Ha.

Abstract– **The web-based enterprise applications, such as e-commerce, online banking, online auction, social networks, forums..., have been more and more popular in our society. These applications become the target of security attacks. Hence, securing websites and connection to the users is important. If we own or manage a website, we certainly concern about how secure it is. For assessing the security level of a website, we usually take some action, including testing the website using security scanning tools. Unfortunately, most of scanning tools have limitations and need to be updated frequently for new vulnerabilities. Using only one scanning tool is sometime not enough to determine security level of a website. In this paper we propose a framework supporting website security assessment. The idea of this framework is to integrate different scanning tools into the framework. We then write a program to implement this framework with a real website. We guide the users how to add a new scanning tool to this framework, manage it and generate a final report. In addition, we discuss the problem of security on client-side called *clickjacking attack* that many clients may suffer when accessing the malicious websites, we propose a method to protect them from this attack.**

Keywords– **Security assessment, OWASP standards, website security scanning tools.**

1 Introduction

Along with the improvement of web technology, the security attacks have dramatically increased over the past decade, target on websites and web applications. In consequence, sensitive and confidential data could be stolen. According to Symantec Internet Security Report Vol. 24 [1], web attacks were up to 12.5% in December 2018 in compare with previous months at approximately 150,000 per day.

Data illustrated in Figure 1 describes the web attacks per day in a month, each column presents the number of web attacks in one month from January 2018 to December 2018. We assess the website security based on OWASP standards [2] as criteria. The OWASP is a not-for-profit organization focusing on improving the security of software. It provides the testing guide to find vulnerabilities of a software. As defined, a vulnerability in our context is a flaw or weakness in the website design, implementation, operation or management that could be exploited to compromise the website security objectives.

There are some types of testing:

- Blackbox testing: this is the simplest and most basic form of identifying a web server, it looks at the server field in the HTTP response header. It is known as behaviorial testing in which the internal structure of the website is not known to the tester.
- Whitebox testing: it is known as clear box testing, refers to testing a website with full knowledge and access to all source code and architecture documents.

To identify and patch the vulnerabilities that would be exploited by an attacker, people usually perform penetration testing. The ideal form of penetration testing is blackbox, as most of attackers have no knowledge of the internal features prior to launching their attack.

This paper is concerned with providing a new framework giving users the posibility to use more scanning tools in one running. Section 2 reviews related works with similar model. Some common attacks on websites and web applications are discussed in Section 3. In Section 4 we propose the new framework mentioned above. Section 5 shows experimental results with our software program called *VulScanner* with evaluation. In Section 6 we discuss the problem of Clickjacking attack and conclude the paper in Section 7.

2 Related Works

YASCA [3] is an open-source program, which scans website security vulnerabilites in the program source code called whitebox scanning. It is a command-line tool that generates reports in HTML, CSV, XML, MySQL, SQLite, and other formats. YASCA has many tools for different programming languages, such as .NET, ASP, C/C++, COBOL, ColdFusion, CSS, HTML, Java, JavaScript, Perl, PHP, Python, Visual Basic,... Unfortunately, this program was discontinued, its official website cannot be accessed anymore.

J. Bau et al. [4] combined 8 different blackbox scanning tools for website security assessment and showed results. Their assessment was provided based on a database of vulnerabilities and the combination

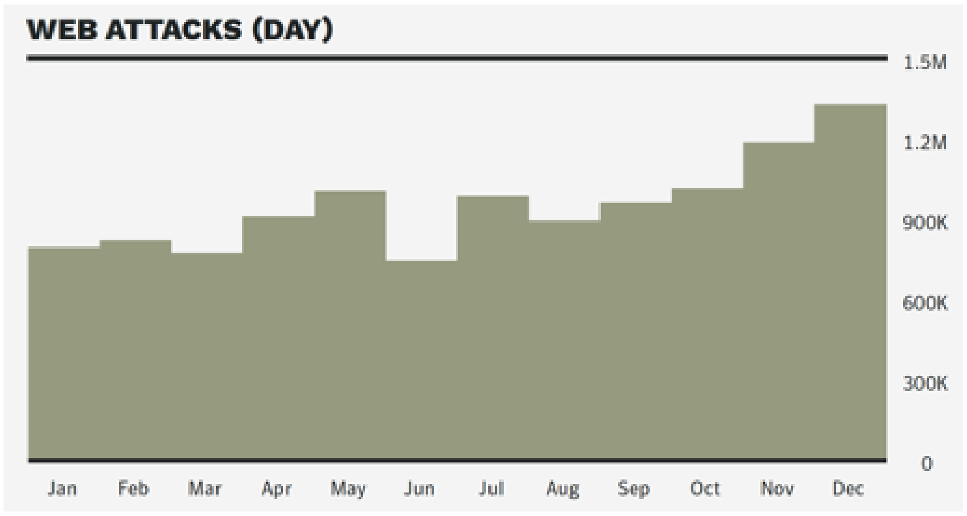


Figure 1. Web attacks per day [1].

of scanning tools was achieved manually, not automatically.

JARVIS is a framework provided in [5], which integrates five different security scanning tools such as Arachni, OWASP ZAP, Skipfish, Wapiti, w3af. JARVIS works as a proxy between scanning tools and web applications, which are scanned. The weakness of this model is that it is unable to integrate scanning tools which are not connected through proxy.

3 Common Attacks on Web Applications

The OWASP provided top 10 web security vulnerabilities, discussed as below.

- Injection: this is a widely known attack technique, rated as the number one problem on the list of top 10 security issues. This attack occurs when the user is able to input untrusted data tricking the website to execute unintended commands or accessing data without proper authorization. One of the reasons is that the user-supplied data is not validated, filtered, or sanitized by the website. The injections can be SQL queries, PHP queries,…
- Broken authentication: this occurs when the application mismanages session related information such that the sensitive information gets compromised. The information can be the password, credit card number, session cookies,…
- Sensitive data exposure: sensitive data could be sniffed or modified if not handled securely by the website. The data could be transmitted or stored in clear text, weak cryptographic algorithms used, or sensitive data in use in default data, such as keys, passwords,… To secure the user data in transmission, we consider to use secure connections, such as HTTPs instead of HTTP.
- XML external entities: this occurs when the website enables users to upload a malicious XML which exploits the vulnerable code and/or dependencies.
- Broken access control: this occurs if a user is able to access unauthorized resources, this can be access to restricted pages, database, directories,… Example: a user can use the permissions of the website admin to edit and modify the data. The countermeasure is usually to apply access control to the server side of the website.
- Security misconfigurations: the developers and IT staff ensure functionality of the website and not the security. This can happen at any level of an application stack, including the network services, platform, web server, application server, database, frameworks, custom code, and pre-installed virtual machines, containers, or storage.
- Cross Site Scripting (XSS): this occurs when an attacker is able to insert untrusted data/scripts into a web page. This data/scripts get executed in the browser and can steal user data, deface websites,… Preventing XSS requires separation of untrusted data/scripts from active browser content.
- Using Components with Known Vulnerabilities: it may occur when we do not know the versions of all components we use (both client-side and server-side). This includes components we directly use as well as nested dependencies, or if we do not scan for vulnerabilities regularly and so on. To prevent this risk, we consider to remove unused dependencies, unnecessary features, components, files, and documentation,…

4 Proposed Framework

Our proposed framework model is shown in Figure 2. Part (1) in the model presents scanning tools, shown as *plugins*, integrated into the proposed framework. These tools could be called to run anytime, return report to the framework. Because the tools are different, the output formats could be different. We need to change all the output results to the desired formats before generating the final report. Part (2) shows *Lexer/Parser* for syntax analysis used for whitebox scanning. We map all scanning results into database in part (3) where we remove duplicated vulnerabilities from different

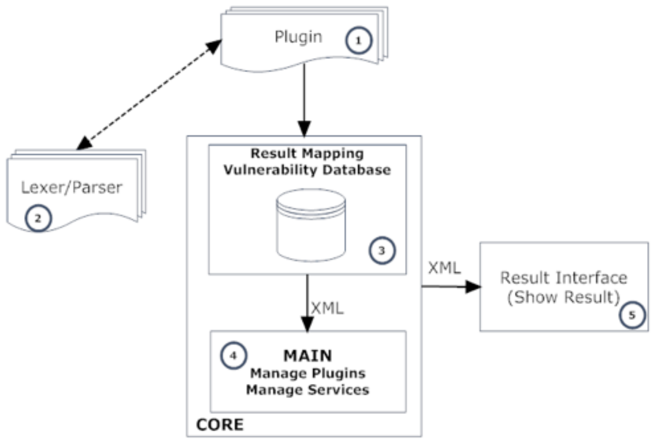

Plugin 1
Lexer/Parser 2
Result Mapping Vulnerability Database 3
XML
MAIN Manage Plugins Manage Services 4
CORE
XML
Result Interface (Show Result) 5

Figure 2. The model of proposed framework.

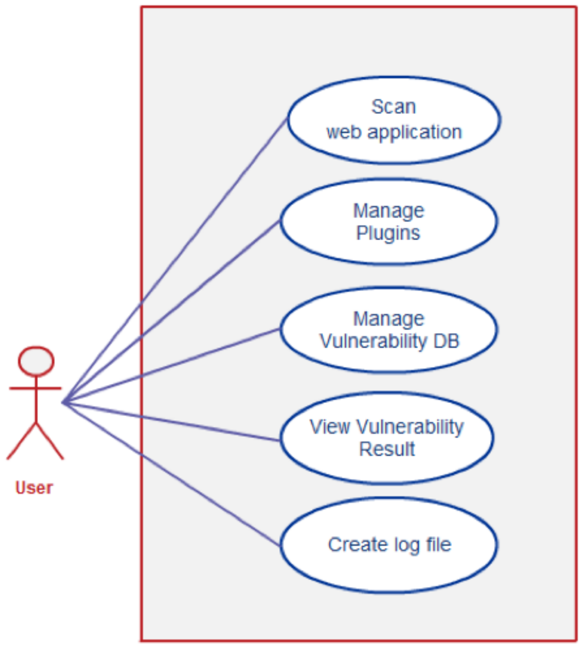


Figure 3. Use-case diagram of proposed framework.

scanning tools. The main part of our framework is shown in part (4), which manages all scanning tools. The final report is created in part (5), given in XML format and printed out by another module in this part, for different purposes, such as for web, for printer, for pdf file,...

In the requirement analysis stage we determine the functionalities of the framework, shown in Figure 3. The users can scan a website by calling the selected plugin(s), can manage plugins by adding a new one, editing its features, removing it from the framework. Further, the users can manage the vulnerability database, view the results after scanning, create log files and so on.

Figure 4 shows flowchart of the framework. The users select plugins to call for execution, input website URL (in case of blackbox testing) or path to web application (in case of whitebox testing) that needs to be scanned. For each plugin it takes some time to complete scanning, so if we select more than one plugin, the time is

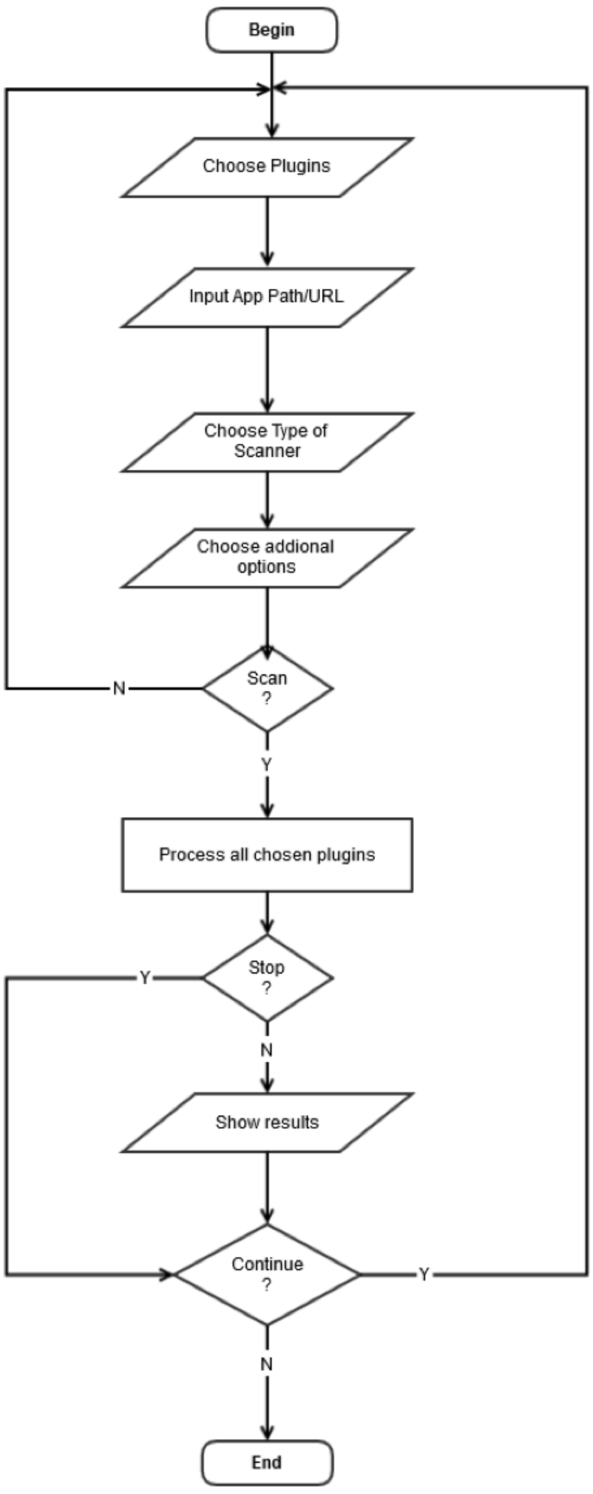


Figure 4. Framework flowchart.

expected to be larger. If we cancel scanning, we receive incomplete report.

4.1 Call plugin(s) for scanning

The framework itself cannot scan any website. It just calls the plugins to do that and receives the results from them. If the website source code is not available, only blackbox scanning type could be used.

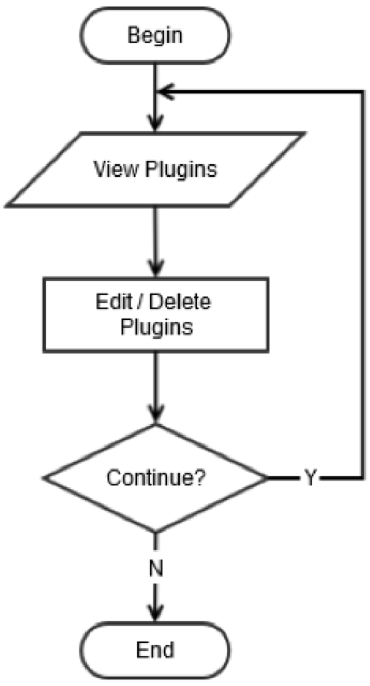


Figure 5. Plugin management.

4.2 Plugin Management

We can manage plugins using different ways. As shown in Figure 5, we can edit a plugin by changing the plugin name, version, path to web application (whitebox) or remove it from the framework.

4.3 Vulnerability Database Management

Information about website vulnerabilities according to OWASP standards can be found in Vulnerability Database. The users can manage this database by adding new kinds of vulnerabilities, updating them or removing them from database. We manage this database based on most recent OWASP standards, where we can find information about different testing attacks, vulnerabilities and countermeasures. See diagram in Figure 6.

4.4 Vulnerability Mapping

Vulnerabilities found by plugins may be the same but appear with different names and we want to show them once in the report. We identify this difference and map the vulnerabilities onto desired names. After one simple step of processing we get the results without duplicates, example shown in Figure 7.

4.5 Data Formats

Data containing vulnerabilities and data in the final report are stored in XML files. The advantage of using XML is easier retrieving data and sharing between framework features. Example is shown in Listing 1.

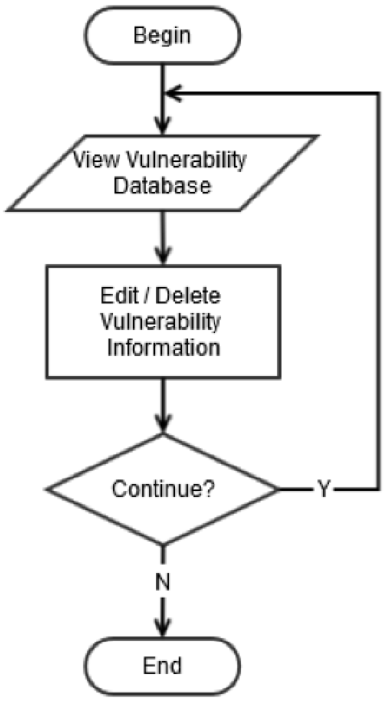
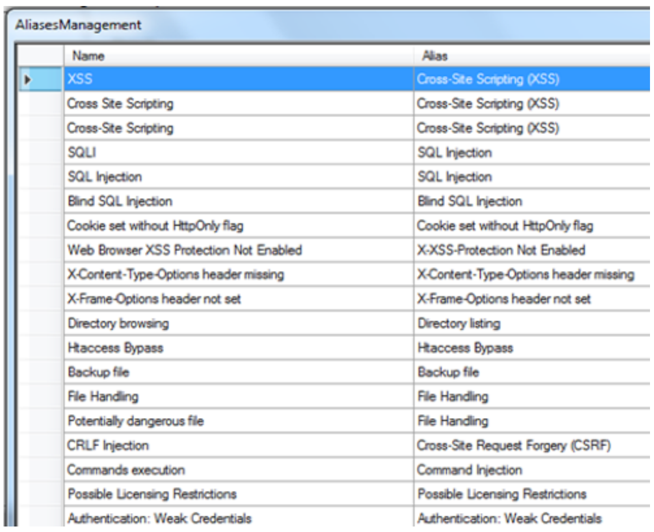

Begin
View Vulnerability Database
Edit / Delete Vulnerability Information
Continue?
Y
N
End

Figure 6. Database management.


AliasesManagement

Name	Alias
XSS	Cross-Site Scripting (XSS)
Cross Site Scripting	Cross-Site Scripting (XSS)
Cross-Site Scripting	Cross-Site Scripting (XSS)
SQLI	SQL Injection
SQL Injection	SQL Injection
Blind SQL Injection	Blind SQL Injection
Cookie set without HttpOnly flag	Cookie set without HttpOnly flag
Web Browser XSS Protection Not Enabled	X-XSS-Protection Not Enabled
X-Content-Type-Options header missing	X-Content-Type-Options header missing
X-Frame-Options header not set	X-Frame-Options header not set
Directory browsing	Directory listing
Htaccess Bypass	Htaccess Bypass
Backup file	Backup file
File Handling	File Handling
Potentially dangerous file	File Handling
CRLF Injection	Cross-Site Request Forgery (CSRF)
Commands execution	Command Injection
Possible Licensing Restrictions	Possible Licensing Restrictions
Authentication: Weak Credentials	Authentication: Weak Credentials

Figure 7. Mapping vulnerability names.

5 Experimental Results

We create a program called *VulScanner* to implement the proposed framework, written in C#. The interface of VulScanner shown in Figure 8, where the users can select plugin(s) and specify the target website to perform testing. As mentioned above, whitebox testing can be used only if the website source code is available.

For demonstration, we integrate a couple of scanning tools into the framework, as listed in Table I and Table II.

The first experiment was conducted with localhost DVWA with 3 plugins Arachni, SQLMap and Wapiti, as in Table III. After selecting all 3 plugins, the results are shown in Table V.

Listing 1
DATA STORED IN XML FILE

```
<vulcategories>
  <vul>
    <name>Vulnerability name</name>
    <description>Vulnerability description</description>
    <environment>Environment of this vulnerability</environment>
    <example>Some example</example>
    <determination>How to detect</determination>
    <protection>How to protect your web app</protection>
  </vul>
...
</vulcategories>
```

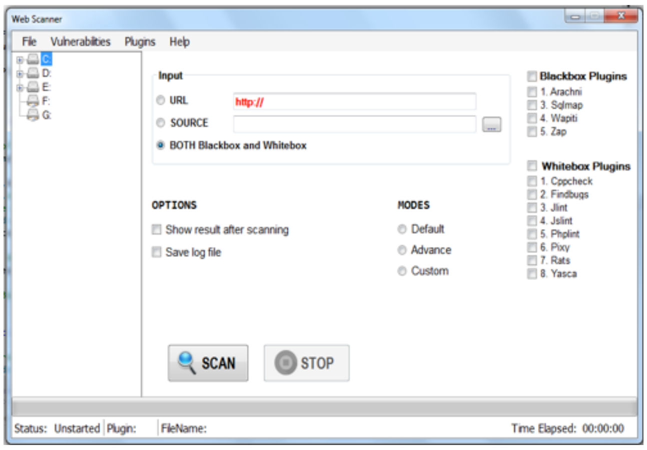


Figure 8. The interface of VulScanner.

Table I
BLACKBOX PLUGINS

Plugins	Search for vulnerability type
SQLMap	SQL Injections
Wapiti	Different types of vulnerabilities
Zap	
Arachni	

Table II
WHITEBOX PLUGINS

Plugins	Search for vulnerability type
CPPCheck	C++ programming code
Findbugs, Jlint	Java source code
Jslint	JavaScript source code, syntax only
PHPlint	PHP source code, syntax only
Pixy	XSS and SQL injections
Yasca	Open source program, discontinued

From the results displayed in the VulScanner interface, we can follow the link to find more details of each vulnerability such as name of plugin, which found the vulnerability, from what URL, vulnerability description, severity, and the link out where to see more details about the vulnerability.

Similarly, we demonstrated VulScanner with white-box type scanning and received experimental results as expected.

Further test was performed on a website containing different vulnerabilities, described as OWASP Top 10, in Table IV.

VulScanner runs on Windows 7 of a computer 2 GB RAM, CPU core 2 duo e8500. We call each plugin separately, compare total scanning time, number of vulnerabilities to the results received from the VulScanner:

Some vulnerabilities could be found by more than

Table III
DEMONSTRATION WITH VULSCANNER

Plugins	**Found vulnerabilities**
Arachni SQLMap Wapiti	Cross-Site Scripting (XSS)
	Blind SQL Injection (timing attack)
	Cross-Site Scripting (XSS)
	SQL Injection
	HTTP TRACE
	X-Frame-Options header not set
	HTTPOnly Cookie
	Interesting response

Table IV
VULNERABILITIES IN OWASP TOP 10

Vulnerability name	**OWASP Top 10**
Command Execution	A1 (2017)
CSRF	A8 (2013)
File Inclusion	A3 (2007)
SQL Injection	A1 (2017)
XSS	A7 (2017)
Path Traversal	A5 (2017)
Sensitive Data Exposure	A3 (2017)
Weak Credentials	A2 (2017)

Table V
Scanning Results with DVWA

Plugins	Scanning time (hh:mm:ss)	Number of vulnerabilities
Arachni	05:28:12	293
SQLMap	00:01:46	1
Wapiti	00:21:53	5
ZAP	00:12:14	141
Pixy	00:00:38	0
YASCA	00:01:09	323
Framework	06:18:42	386

one scanning tool, so the number of vulnerabilities found by VulScanner is smaller than total vulnerabilities found by the tools called separately. The VulScanner calls the plugins sequentially, the time it requires for completion of the experiment is a little bit longer in compare with the total time of all individual scannings.

One more experiment was performed with the website provided by Acunetix as shown in [6]. We used the same system configuration as mentioned above, the tested website is http://testphp.vulnweb.com/. The website source code is not available, so only the blackbox scanning method could be used. We choose SQLMap, Wapiti and ZAP as the plugins in VulScanner.

We repeated the test 3 times and received the results as in the Table VI. The scanning time in different tests are different due to network state.

The number of vulnerabilities are shown in Table VII. They are different because of network state, some of the connection requests are failed or got timeout.

The proposed framework was implemented and provided results as we expected. The accuracy of the testing depends on the plugins selected and their accuracy. The more plugins selected the longer it takes to complete the scanning. The blackbox tests are usually the fastest tests, however the limited information available to the testers increases the probability that vulnerabilities will be overlooked and decreases the efficiency of the tests. Whitebox tests are usually slow, and large amount of data available to the testers requires time to process.

6 Clickjacking

In general, securing the websites is not enough. The websites' users can still become the victims of a kind if security attacks, called *Clickjacking*. To protect the users from this attack, we developed a script running on client-side system, automatically detects clickjacking and lets the users continuing their surfing securely.

Clickjacking, also called *web framing attack* [7], was reported by Jeremiah Grossman and Robert Hansen in 2008 [8]. This attack uses a transparent frame or a tiny frame to hijack user's clicks. Clickjacking attack is based on the simple idea: attackers construct a web page containing an invisible (or rarely visible) frame,

Table VI
Scanning Time [in seconds]

Tests	SQLMap	Wapiti	ZAP	VulScanner
1	26	1:8:20	1:25	1:10:11
2	32	1:18:55	1:45	1:21:12
3	18	1:12:52	1:22	1:14:32

Table VII
Number of Vulnerabilities Found

Tests	Number of Vulnerabilities Found by VulScanner
1	93
2	96
3	106

the position of the frame is set so that it tricks users into clicking on elements in frame while they think they are acting with the parent page. In practice, clickjacking attacks are usable for the purpose of tricking users into clicking on banner ads, "Like" button in social networks, button that shares their webcam, initiates money transfers, or performs any action caught by a user's mouse click [9, 10].

Recently, clickjacking attack becomes popular and many prevention techniques have been proposed, provided in [11], X-FRAME-OPTIONS headers in [12], frame bursting in [13],.... Unfortunately, these techniques could not help to protect users from some kinds of Clickjacking attacks using sharing widgets, such as the Like button of Facebook [14] or Plus One button provided by Google Plus social network. These social widgets allow users to interact with the network by one-click without leaving the context of the current page.

6.1 Proposed Algorithm

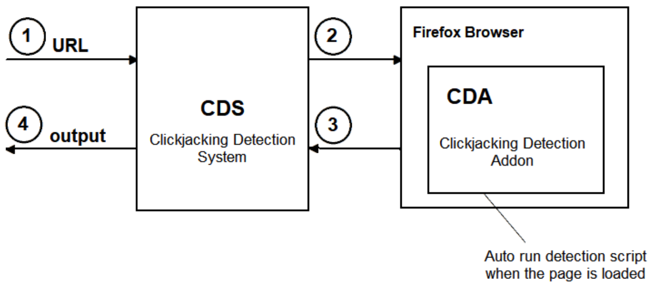
We proposed an algorithm to detect hidden frame that can perform Clickjacking attack in a web page, based on Attack Variants [10] and demonstrations collected from Internet. Using this algorithm, we implemented Clickalert, a Firefox browser extension to protect users from clicking on an element that they do not totally see. There are different browsers such as Microsoft Internet Explorer, Google Chrome, Mozilla Firefox, Safari, Opera,... but we choose to use Mozilla Firefox for this research. The model is shown in Figure 9.

From the model above, the main program receives the input as a list of URLs, sends it to the Firefox browser, receives the results of scanning process and returns the final report.

Our extension can be used by all Internet users to protect themselves from this kind of attack, it warns them when they click on a hidden element in an iframe. Furthermore, the extension can help security experts to test a large number of web pages for clickjacking attack.

Listing 2
Pseudo Code for Checking Visibility of An Element

```
Set parent to iframe (the iframe need to check for visiblity)
While parent is not null
    Check visibility of parent
    If visibility attribute is met
        Set parent to parent element of parent
    else
        Return True
    Endif
Endwhile
Return False
```



1 URL
4 output
CDS
Clickjacking Detection System
2
3
Firefox Browser
CDA
Clickjacking Detection Addon
Auto run detection script when the page is loaded

Figure 9. Clickjacking Detection Model.

The key point of our approach is to check all elements' attributes in a web page to find those match criteria defined above by examine all HTML/CSS attributes of the checking web page. Listing 2 shows the pseudo code we use to check the visibility of an iframe in the current web page. We start checking from the frame element in the DOM tree [15, 16]. Then we check the visibility property of the selected element and all of its preceding elements. If any of these elements does not pass the visibility checker function, then this is the indication of an invisible element used for Clickjacking attack.

For checking the visibility of an HTML element, as mentioned in [15], we verify the attributes of its element if they meet the following criteria:

- The opacity attribute value is a CSS attribute that describes the opacity-level of an element, its value is in the range 0 (transparent) to 1.0 (completely visible). The condition is that the value of the opacity attribute must be higher than a certain threshold. Checking these attributes can be done by using Javascript language (used to build the Browser addon), we get it by the Javascript API *window.getComputedStyle()* that will compute all final CSS values of an element after the web page is loaded completely in the browser.
- Dimensions of an element can also influence its visibility, e.g. we can set the width and height of an element to zero to make it hidden. To satisfy this condition, we define the minimum threshold for width and height, and dimensions of an element must be higher than these thresholds.

Table VIII
Experimental Results of Clickjacking Attack

	Number of websites checked	Percentage
Visible	695	95.3
Hidden	34	4.7
Total	729	100

Using the algorithm described above, we develop the Clickalert extension that can protect users from Clickjacking attack when they surf the Web. When users have our extension integrated with their Firefox browser, whenever they click on an element in the browser, the extension will check whether that element belongs to an iframe and is visible to users. They will receive a warning if clicking on an element in an frame that is not totally visible. Our extension also has a feature allowing users to remove hidden frame when they click on it accidentally.

6.2 Experimental Results

We created the sample pages containing malicious codes for Clickjacking attack. We try to access those pages and see that the browser addon works correctly and sends an alert whenever user click on a hidden iframe. It can also detect all hidden iframes and their source codes.

We choose 729 websites, collected from [17], [18] and use our addon to scan and find out websites that may trick users by Clickjacking attack, as in Table VIII.

6.3 Discussion

Clickjacking is the web-based attack which is the subject of many security reports recently. As an appendix to the framework proposed for website security asssessment, we created an extension that can be integrated into users' browsers as an ADD-ON, to protect them from Clickjacking attack. Experimental results showed that the extension works properly in the case when users click on an invisible element, intentionally created by the attackers.

7 Conclusions

The contribution of our paper is not in creation of a new scanning tool, but we suggest the way to integrate different plugins to get better results. We showed that it is very easy to add a new plugin to the framework, easy to configure the framework to work with the new plugin.

Different tools give results in different format, so we need to map them onto desired format as provided by OWASP. The results given by our framework can be acceptable, however we can speedup the scanning by running the tools in parallel, we consider this issue in the future work.

Regard the clickjacking, we provided an effective way to detect hidden frames, then we implemented it in Firefox. This tool can protect users from clickjacking on an element that they do not see.

Acknowledgment

This research is funded by Vietnam National University Ho Chi Minh City, under grant number C2017-20-19.

References

[1] Symantec Internet Security Report, vol. 24, 2019.

[2] OWASP Top 10 Most Critical Web Application Security Risks. [Online]. Available: https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project (accessed Jan. 25, 2019)

[3] YASCA, official website. [Online]. Available: yasca.org, or on Github: https://github.com/scovetta/yasca (accessed Jul. 2018)

[4] J. Bau, E. Bursztein, D. Gupta, and J. Mitchell, "State of the art: Automated black-box web application vulnerability testing," in *Proceedings of the IEEE Symposium on Security and Privacy*. IEEE, 2010, pp. 332–345.

[5] D. Esposito, M. Rennhard, L. Ruf, and A. Wagner, "Exploiting the potential of web application vulnerability scanning," in *Proceedings of the Thirteenth International Conference on Internet Monitoring and Protection*, 2018, pp. 22–29.

[6] Home of Acunetix Art. [Online]. Available: http://testphp.vulnweb.com (accessed Aug. 2019)

[7] Cursorjacking again. [Online]. Available: http://blog.kotowicz.net/2012/01/cursorjacking-again.html (accessed Jan. 2012)

[8] R. Hansen and J. Grossman. Clickjacking. [Online]. Available: http://www.sectheory.com/clickjacking.htm (accessed Jan. 25, 2019)

[9] M. Johns and S. Lekies, "Tamper-resistant likejacking protection," in *Proceedings of the International Workshop on Recent Advances in Intrusion Detection*. Springer, 2013, pp. 265–285.

[10] L.-S. Huang, A. Moshchuk, H. J. Wang, S. Schecter, and C. Jackson, "Clickjacking: Attacks and defenses," in *Proceedings of the 21st USENIX Security Symposium*, 2012, pp. 413–428.

[11] A. S. Narayanan, "Clickjacking vulnerability and countermeasures," *International Journal of Applied Information Systems*, vol. 4, no. 7, pp. 7–10, 2012.

[12] D. Ross, T. Gondrom, and T. Stanley. HTTP Header Field X-Frame-Options (RFC 7034) (2013) InternetEngineering Task Force (IETF). [Online]. Available: https://www.ietf.org/rfc/rfc7034.txt (accessed Jan. 2018)

[13] G. Rydstedt, E. Bursztein, D. Boneh, and C. Jackson, "Busting frame busting:a study of clickjacking vulnerabilities on popular sites," in *Proceedings of the IEEE Oakland Web 2.0 Security and Privacy (W2SP'10)*.

[14] Facebook worm - "likejacking". [Online]. Available: http://nakedsecurity.sophos.com/2010/05/31/facebook-likejacking-worm/ (accessed May 2010)

[15] P. Thiemann, "A type safe DOM API," in *Proceedings of the International Workshop on Database Programming Languages*. Springer, 2005, pp. 169–183.

[16] M. Heiderich, T. Frosch, and T. Holz, "Iceshield: Detection and mitigation of malicious websites with a frozen dom," in *International Workshop on Recent Advances in Intrusion Detection*. Springer, 2011, pp. 281–300.

[17] Top 1000 website. [Online]. Available: https://www.alexa.com/topsites

[18] Free web stats counter from motigo webstats - catalogue/top 1000 sites. [Online]. Available: https://trends.builtwith.com/websitelist/Motigo-Stats

Nguyen Duc Thai received BSc. (MSc.) and Ph.D. from Slovak University of Technology in Bratislava, Slovakia, in 1996 and 2005 respectively. He is currently Head of Department of Computer Systems and Networking, Faculty of Computer Science and Engineering, Ho Chi Minh City University of Technology. He has been actively involved as a researcher and teacher in the area of computer networks and network security for many years.

Nguyen Huu Hieu is currently a full time researcher at Ho Chi Minh City University of Technology. He received his B.Sc. and M.Sc. degree from Ho Chi Minh City University of Technology in 2012 and 2015, respectively. His current research interests include network security and website security.