Clarity: Analysing Security in Web Applications

Connor J. Potter[1], Neetesh Saxena[1] and Soumyadev Maity[2]
[1]*School of Computer Science & Informatics, Cardiff University,* Cardiff, United Kingdom
[2]*Department of Information technology, IIIT Allahabad,* Prayagraj, India
potterc10@cardiff.ac.uk, nsaxena@ieee.org, soumyadev@iiita.ac.in

***Abstract*—The rapid rise in business' moving online has resulted in e-commerce web applications becoming increasingly targeted by hackers. This paper proposes Clarity, a dynamic black box vulnerability scanner capable of detecting Cross-Site Scripting, SQL Injection, HTTP Response Splitting, and Session Management vulnerabilities in web applications. The developed tool employs the use of Mechanize and Selenium to perform the majority of its web scraping requirements. Clarity was tested against 50 e-commerce web applications, uncovering Session Management flaws as the most prevalent vulnerability, with 36 out of the 50 applications being vulnerable.**

Keywords—Web security, HTTP response, SQL injection, vulnerability

I. INTRODUCTION

More than 1.92 billion people worldwide have used online shopping by 2019, with these numbers expected to continue increasing [1]. This increase in online shopping users has resulted in an intensified demand for e-commerce web applications. These applications allow businesses to target a larger consumer base and in turn increase in sales, resulting in more business's going online. However, there are of course drawbacks too. For example, e-commerce web applications contain sensitive user data such as name, address, and bank details. These must be protected appropriately by the web application, or this sensitive data may very well fall into the wrong hands. For this reason, hackers target such web applications, making it a top priority that we ensure the appropriate security controls are in place. If an e-commerce web application is found vulnerable to attack, there is a possibility for the attacker to execute any of the following: Stealing user or administrator account or personal details, using the web application to distribute malware, using the web application as a base for scamming operations, using it as a base for phishing operations, or defacing the web page. Since 2011, more than 75 percent of all attacks occur on web applications [2]. With applications facing this huge level of threat, the aim of this work is to assess how these attacks are carried out on e-commerce web applications, how we can detect them. We analyse and investigate not only if a web application is using adequate security controls but also that it is using them correctly. This work looks to automate the detection of various vulnerabilities found in OWASP's Top 10 most critical web application security risks [3].

Motivation. Attackers are constantly finding new vulnerabilities and new ways of exposing them for personal gain. This has motivated this work to give consumers cheap and accessible means of detecting these vulnerabilities so they may protect themselves before any harm is done during the visit to a malicious or insecure web application. The motivation behind the cheap alternative is the target audience of this work, which would be people with limited experience in cybersecurity. Therefore, these individuals will likely have no interest in paying a large amount of money for a security scanner like many currently on the market. Given the threat web applications face, this work aims to identify how vulnerable e-commerce web applications are to attack. This aim is broken up into three tasks: investigate attack vectors for e-commerce web applications, investigate the identification and detection of web application vulnerabilities, and create a tool to automate the detection of web application vulnerabilities. In this work, we have tested fifty web applications. This number is reasonably enough to gain a general understanding of how efficient Clarity is as well as generally how secure e-commerce web applications are.

Key Challenges. The challenge to this work is to ensure Clarity has satisfactory coverage of data entry points. It is a well-known fact a black box penetration testing approach to finding vulnerabilities in web applications often encounters false-negatives due to poor coverage of the web application and this is something we hope to overcome. Furthermore, the validation of the web application after the attack has been executed must be as accurate as possible. If this phase is not accurate, neither will be the results.

II. RELATED WORKS

In this work, we focus specifically on four vulnerabilities: SQL Injection, Cross-Site Scripting, Session Management Flaws, and HTTP Response Splitting. The web applications targeted are primarily small stores or sole traders, as these are the applications most at risk due to them having a smaller budget to put into ensuring the security of their web application.

A. Preliminaries

Sessions. HTTP is a stateless mechanism used to handle requests and responses to and from a web application. Yet for many web applications, being stateless cannot offer the desired functionality. Session management is implemented by assigning each user a unique session ID or session token which tells the application who the user is and what data they have permission to access. This token implementation can be broken up into three mechanisms [6]: (1) tokens stored in cookies, (2) tokens sent in hidden fields, and (3) tokens are created by the web server and subsequently added to each link the user clicks on. The first two options are client side, whereas for the third, a session ID is created and stored on the server side.

Input validation. According to the Tainted Mode Model all data received from the client to the server through HTTP requests is untrustworthy (tainted) [7]. However, in order to perform operations such as generating custom SQL queries, the user input must be concatenated with the underlying code. In

order to do so securely, we must first validate the input. This validation phase is often executed through means of sanitisation. Sanitisation routines turn untrusted user input into trusted data by filtering out any potentially harmful characters from the input. The vulnerabilities which can occur as a result of incorrectly implemented input validation are injection attacks. Cross-Site Scripting and SQL Injection are two of such attacks. Additionally, other injection attacks may ensue as a result such as Command Injection.

B. Analysing Web Application Attacks

In this work, we discuss two possible models to detect vulnerabilities in the web applications.

Tainted Mode Model. The Tainted Mode model can be used to analyse web applications in both static and dynamic environments to detect security vulnerabilities caused by improper form validation. Several assumptions are made by the Tainted Mode model [8]: (1) all data received from the client via HTTP requests is untrustworthy (tainted), (2) all data being local to the web application is trustworthy (untainted), and (3) any untrustworthy data can be made trustworthy by sanitisation. Based on these assumptions, the following security policies are then defined: (1) tainted data must not be used in HTTP response construction, (2) tainted data must not be written into local web application storage, and (3) tainted data must not be used in system command construction.

Penetration Testing. Penetration testing is based on simulating attacks against a web application to determine whether the application has any vulnerabilities. This is a black-box testing approach, since an attacker would not normally have access to the underlying code of the web application. According to [9], [10], black box penetration testing follows these steps: (1) identify all pages being a part of the web application, (2) extract all Data Entry Points (DEPs) from each page visited by the first step. Both steps can be automated by using a web crawler. Retrieving all DEPs is an important stage as these are the vectors from which a hacker can attack the application, (3) simulate various attacks by fuzzing, which is a method of attacking web applications where DEPs are filled out (or fuzzed) with a mixture of malicious and innocent string patterns and sent as a HTTP request to the web application, and (4) analyse all HTTP responses for indications of vulnerabilities.

C. Vulnerabilities

The reasoning behind choosing these vulnerabilities was SQL Injection and Cross-Site Scripting have been the most prevalent vulnerabilities in the wild for many years [3]. Additionally, almost all existing security scanners researched also detected these vulnerabilities, therefore in order to compete it felt necessary to provide this functionality. Session Management flaws however were frequently left out of these tools, so providing this functionality sets Clarity apart from the rest. Furthermore, HTTP Response Splitting, as a less common and mostly fixed in modern web applications, is overlooked. In fact, no scanners could be found that offer this functionality, once again giving this tool an edge on existing scanners. All vulnerabilities outlined are aimed to be detected using the Clarity tool.

D. Existing Vulnerability Scanners

Many other security scanners have a vast number of different options and information a layman would not comprehend. Overwhelming the user with information may very well put people off from using their software. For example, the open source vulnerability scanner Wapiti [21] and Vega [22] are command-line based tools which do not provide rich interface to use them, especially to those who do not have any experience this can be intimidating and confusing. Some of the larger more renowned vulnerability scanners such as Netsparker [23] and Acunetix [24] require paid subscriptions in order to use their services. Most of the general public will not willingly pay for such services, giving Clarity and edge over these more expensive options.

Anagandula et al. [25] analysed black-box web application scanners in detecting SQL injection and XSS vulnerabilities. Tyagi et al. [26] evaluated two static web application vulnerability analyses tools, OWASP WAP and RIPS using the deliberately vulnerable web application and found that OWASP WAP offers better results over RIPS. Anhar and Suryanto [27] evaluated web application vulnerability scanners such as OWASP ZAP, Wapiti, Arachni, and Burp Suite Professional. They have found that the four WAVS have an average f-measured value, Burp Suite Professional had the best true positive and recall values, while Arachni has perfect Precision valued. Mburano et al. [28] evaluated web vulnerability scanners based on OWASP Benchmark and provide some recommendations. Alptekin et al. [29] analysed different vulnerabilities using vulnerability scanners and reported results from top vulnerabilities. Chen et al. [30] proposed a scanner with vulnerability detection is proposed to verify whether the target web application is vulnerable.

Here, we look at existing vulnerability scanners and the vulnerability detection capabilities they offer. Table I gives five popular open source vulnerability scanners, followed by a subset of the vulnerability detection they offer. As seen, most of these scanners offer detection of both Cross-Site Scripting and SQL Injection. These are two of the most common, and arguably the most dangerous vulnerabilities to web applications in the wild. Consequently, there is no surprise they are largely sought after in vulnerability detection software. However, none of these popular scanners are capable of detecting Session Management flaws or HTTP Response Splitting. For that reason, Clarity has been given the functionality to detect these two vulnerabilities in addition to both XSS and SQLi. This sets Clarity aside from the rest, making it a viable option to those needing to cover a different range of vulnerabilities.

TABLE I. COMPARISON OF EXISTING VULNERABILITY SCANNERS

	XSS Reflected	***XSS Stored***	***XSS DOM***	***SQLi Reflected***	***Session Management***	***HTTP Response Splitting***
Grabber	Yes	Yes	Yes	Yes	No	No
Vega	Yes	Yes	Yes	Yes	No	No
SQLMap	No	No	No	Yes	No	No
Wapiti	Yes	Yes	Yes	No	No	No

Netsparker Hawk	Yes	Yes	Yes	Yes	No	No
Clarity	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**

III. Our Approach and Clarity Tool

In this work, we aim to analyse how many web applications suffer from various vulnerabilities and to what degree. The developed tool named 'Clarity' takes a dynamic black box penetration testing in order to scan the target web applications for vulnerabilities.

A. *Quantitative research through a survey*

First, we carried out a survey that was completed by 40 people. Two key questions and their responses are presented in Figure 1 and Figure 2. Primarily targeting those with little knowledge in the field of cybersecurity since these are people most likely to be victim to such crimes. This is clear when looking at how many web application vulnerabilities participants have heard of. 65% of people have not so much as heard the names of any of the vulnerabilities focused. The primary age group who completed the survey was 18–24-year-olds, at a rate of 55%. Astonishingly, 74% of participants had encountered some form of attack while shopping online, ranging from having personal details stolen to encountering fake websites. Furthermore, 3 of these participants suffered a financial loss as a result. These numbers demonstrate that even though web application security has improved over the years, there is still much room for improvement for such people.

When questioned how likely they will be to use a tool which would detect for them if a web application contained any vulnerabilities, a staggering 75% of participants said they would be either likely or very likely to do so. With 20% being neither likely nor unlikely and only 5% stating they are unlikely to benefit from the proposed tool. With these numbers, we clearly see a market for Clarity. In addition, 45% said they would be willing to pay between £5-9 for this tool, 7.5% would be willing to go as high as £9-13. With the remaining 47.5% not wanting to spend any money at all. The importance of efficiency to the participants was very clear. When asked how important it is that the tool does not take a long time to run, 45% said it was extremely important, and 37.5% thought it was important. This did not come as a surprise however it gives the work more direction in the sense that we know this to be of high importance and thus there will be a greater focus put on Clarity's runtime.

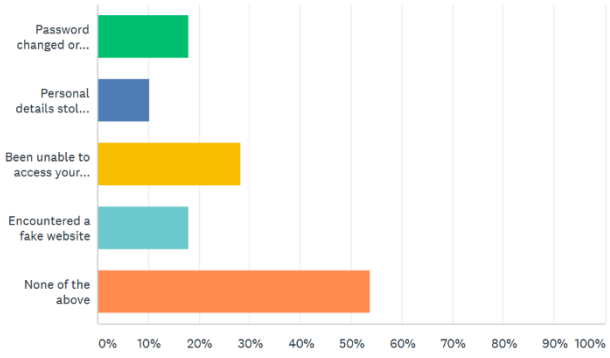


Fig. 1. Survey response to the question: "Have you experienced any of the following security issues while using a shopping website?".

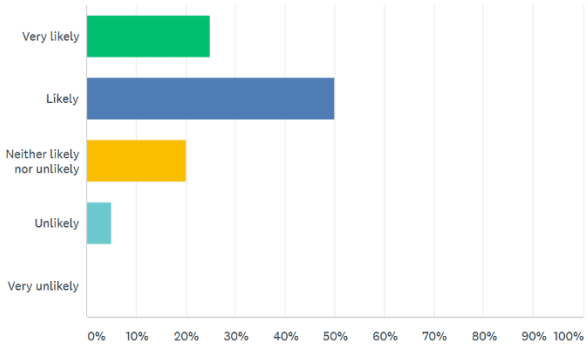


Fig. 2. Survey response to the question: "How likely would you be to use a tool which checks for you if the web application is secure before you use it?"

B. *System Model and Clarity Execution*

A system model is created to aid the defining of Clarity's system structure, as seen in Fig. 3. In this model, the online shopping user is the individual wanting to test for vulnerabilities in a specific e-commerce web application.

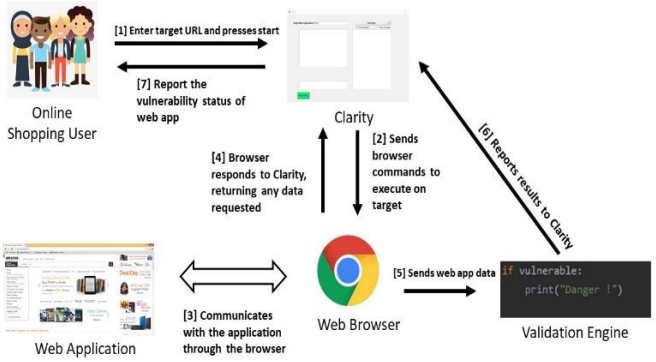


Fig. 3. Clarity system model and scan process.

They interact with Clarity, entering the target web page and starting the scan. Once the scan starts, the tool communicates with the web browser, in this case Google Chrome, which acts as a middle man in communicating with the target web application itself. The browser communicates with Clarity, sending it information collected by the crawler component. When an attack is simulated, the web browser sends data through Clarity's validation engine, the results of which are then passed back to Clarity and returned to the user.

IV. Experimental Design and Execution

Python version 3.8 is used to implement Clarity tool using an extensive number of external libraries for web scraping, including Mechanize, BeautifulSoup4, Selenium, and TKinter. The vulnerabilities explored and identified by Clarity are SQL Injection, XSS, Session Management Flaws, and HTTP Response Splitting. All vulnerability scans except from session management flaws require two phases: the Attack Phase and the Validation Phase. The attack phase performs an attack on the web application and the validation phase reads the response and checks for evidence the attack was successful. For each attack,

the Crawler method is first run in order to retrieve all web pages found in the application. This gives us a set of URLs for the scripts to target. The source code of the Clarity tool is available online (https://github.com/cycislab/clarity).

A. SQL injection

Attack Phase. In order to enact an SQL Injection vulnerability test, we must first attack the target page. Clarity takes the first page from the web application found by the Crawler, and retrieves all DEPs found within it. Secondly, DEPs are filled with the first payload. Finally, the form is submitted, and the results are validated to check whether a vulnerability has been found. This is then carried out for all payloads until all have been tested. The first string containing just an apostrophe has the purpose of prompting a syntax error from the database. A web application should not display detailed errors relating to the database to the user, as this could be taken advantage of by an attacker. All other strings have the purpose of bypassing the login screen and gaining illegitimate access using various syntaxes of different backend databases such as MySQL and Oracle.

Validation Phase. Now that the attack has been performed, we must check to see whether the attack was successful. To do this, we read the web page for strings which would suggest this. It looks for strings such as "stack trace", "error", "SQL", and "database" to detect if the attack string has resulted in an SQL error. Additionally, Clarity also looks for strings which would suggest a login was successful, such as a "logout" or "sign out" button would be present.

B. Cross-site scripting

Attack Phase. Only one payload is used by Clarity for cross-site scripting attacks. This script is:

<ScRiPt>alert("XSS Vulnerable")</ScRiPt>

The *alert* function creates an alert box in the browser containing the string "XSS Vulnerable". The script tag uses a combination of upper and lower case as an evasion technique in order to throw off certain input validation which may only be looking for "script" in all lower case. This attack checks the input type of all form fields. If it comes across a form of type "email" it enters a dummy email address, the same applies for "password" type form fields. This once again bypasses certain security measures. Finally, any form field of type "text" is fuzzed with the attack string, then the form is submitted.

Validation Phase. The validation phase for cross-site scripting is somewhat more complicated than it is for SQL injection. This is due to Clarity needing to handle JavaScript in the form of the alert box in the event of a successful attack. Therefore, Selenium had to be used in the place of Mechanize, as the latter is unable interact with JavaScript. Once the form has been submitted containing the malicious input, Clarity waits for a moment and then looks for an alert box. If one is found, Clarity reads it and looks for the string "XSS Vulnerable". If this string is found, we know the script had been executed and the attack has been successful.

C. Session management

The tool currently only searches for two types of session management flaw: missing Secure flag and missing HTTPOnly flag. These are not the only two ways in which a web application can be vulnerable, but also the two which could realistically be completed in the given timeframe.

The detection of these vulnerabilities is followed by Clarity that had to visit each page, look at the cookies the page uses, then move onto the next. Whenever a vulnerability was found, it was added to a counter which was then returned to the user. To offer perspective, this test was written into less than 20 lines of code.

D. HTTP response splitting

Attack Phase. The attack phase of HTTP response splitting is very similar to that of XSS. There is just one minor difference in the attack string. Instead of entering the script as was done for XSS, we must first add a carriage return, line feed. The attack string used by Clarity is:

Test%0d%0a%0d%0a<ScRiPt>alert("HTTP Splitting Vulnerable")</ScRiPt>

Validation Phase. The validation phase for this attack is virtually identical to that of XSS. The only difference between the two is the string to search for within the alert box. In this case the string is "HTTP Spitting Vulnerable".

V. Results and Evaluation

All testing was carried out on a Microsoft Surface Pro laptop running Windows 10 OS, 2.5GHz Intel Core i7 processor, and 16GB LPDDR3 RAM. The internet service provider was Virgin Media using M100 Fibre Broadband, with 66.69Mbps download and 9.88Mbps upload speeds. 50 web applications were tested using the Clarity tool. The same settings were used for each application. The scan type was "Full Scan" and "Scan all Pages" was selected, aiming to have the most thorough results achievable.

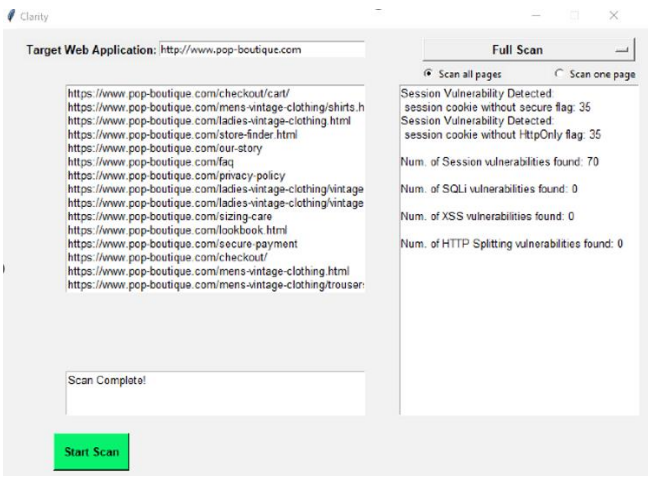


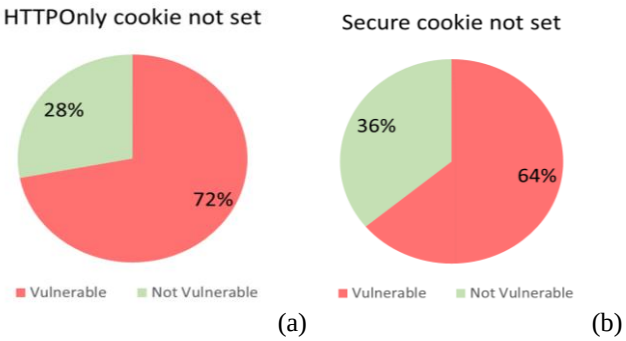
Fig. 4. Clarity's full scan on "www.pop-boutique.com", one of the 50 target web application tested. The results show 35 pages missing the Secure cookie, and 35 pages missing the HTTPOnly cookie.

Table II contains a conclusive list of all web applications scanned and their results. It provides a table containing the names of each application scanned, the time taken in seconds, and the number of vulnerable pages detected for each

vulnerability. Followed by Fig. 4, demonstrating a scan on a target e-commerce web application, displaying several session management flaws.

A. Session management

The most common security vulnerability found in this study was session management flaws.



(a) (b)

Fig. 5. Percentage of web applications do not have (a) HTTPOnly cookie set, and (b) Secure cookie set.

In total, 72% of the 50 web applications scanned were vulnerable – having either a missing HTTPOnly cookie or a missing Secure cookie. Figure 5(a) and (b) displays the percentages for each flaw scanned for by Clarity.

B. SQL Injection

As can been seen in Fig. 6, very few of the scanned web applications were vulnerable to SQL injection. In fact, only two were discovered: one with only one vulnerable page, and the other with as many as 16 detected vulnerabilities.

C. XSS and HTTP Response Splitting

There were zero results found for both XSS and HTTP Response Splitting vulnerabilities. This is a good sign that businesses and developers do pay attention to these vulnerabilities.

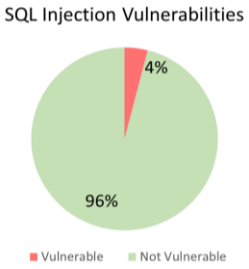


Fig. 6. Percentage of the web applications found to be vulnerable to SQL Injection attacks.

D. Time Efficiency

Figure 7 presents the time taken to scan all 50 web applications. The total time taken was almost 1000 minutes, approximately 16 hours. The initial Crawler function takes the least amount of time to process, followed by session management, then SQL injection attacks, with XSS and HTTP response splitting attacks taking the longest time to execute. This is likely due to needing to wait roughly 3 seconds for the alert box to appear before it can be validated.

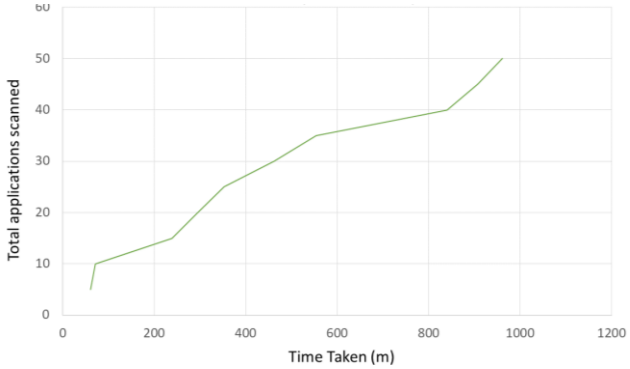


Fig. 7. Time taken to complete the scanning of all 50 target web applications using Clarity.

E. Key Discussion

In this section, we discuss the key takeaways from this work from the results obtained using Clarity tool. Table II shows the results obtained from the scans performed by Clarity on 50 e-commerce web applications. Most of these web applications are vulnerable to secure cookie and HTTPOnly cookie attacks.

1) SQL injection: SQL injection received very few results. This came as a surprise due to SQL Injection being the number one most critical threat to web applications for many years. The occurrence of this anomaly could have been a result of the small sample size examined; however, it can also be speculated Clarity has encountered some false negatives which have been overlooked. Despite this, it can be expected that if many more web applications were scanned, we would begin to see a higher percentage of web applications being vulnerable.

2) Session management: Having a missing HTTPOnly flag alone is not a threat, it is considered a bad practice. A missing HTTPOnly flag only poses a threat if the web page is also vulnerable to XSS attacks, in which case the page could be vulnerable to XSS cookie sniffing. A cookie missing the secure flag on the other hand poses a much larger threat. This could leave a web application's users vulnerable to traffic interception as well as man-in-the-middle attacks. Additionally, missing the secure flag can result in session hijacking in some situations. Scans for session management flaws ran the quickest of the four vulnerabilities.

3) Cross-site scripting & HTTP response spitting: Both XSS and HTTP Response Splitting having zero positive results came as a big surprise. However, there are known flaws in the fuzzing phase of testing for XSS vulnerabilities which, due to the nature of the process for testing for both vulnerabilities, will also affect HTTP Response Splitting. The flaws talked about here are Clarity not currently being able to detect or process when regex is being used for form validation. This results in a false-negative when some fields use regex form validation and others do not. For example, it was discovered a web application "www.castlewelshcrafts.co.uk" was vulnerable to XSS when testing for such vulnerabilities in the early phases of the work. However, when scanned by Clarity, no such vulnerability is

found. In this situation on the "register" page where the vulnerability was found, the "name" field used no form validation whatsoever. This would make the web application vulnerable, but due to other fields using regex form validation such as "email address", Clarity's fuzzing strings do not meet the criteria and thus returns the web page as not vulnerable.

TABLE II. RESULTS OF CLARITY'S SCANS PERFORMED ON 50 E-COMMERCE WEB APPLICATIONS

Name of Site	Time Taken (s)	XSS	SQL	HTTP Splitting	Secure cookie	httponly cookie
www.castlewelshcrafts.co.uk	99	0	0	0	7	7
www.hull-lightingonline.co.uk	1636	0	1	0	1	7
www.cloudokids.com	722	0	0	0	153	155
www.featherandstitch.com	934	0	0	0	36	36
www.belleepoque.me.uk	637	0	0	0	0	0
www.woodieandmorris.co.uk	101	0	0	0	0	0
www.the-stitchery.co.uk	341	0	0	0	1	3
www.peppercorn.net	2211	0	0	0	146	291
www.jagotenby.co.uk	449	0	0	0	0	0
www.troutmarkbooks.com	13	0	0	0	0	0
www.derricksmusic.co.uk	185	0	0	0	0	0
www.diversevinyl.com	8165	0	16	0	1	4
www.palasprint.com	322	0	0	0	0	0
www.uneeka.com	975	0	0	0	121	244
www.fishboypz.co.uk	345	0	0	0	1	14
www.beaunidol.co.uk	182	0	0	0	0	0
www.cliquecustoms.com	255	0	0	0	75	75
www.st-eval.com	705	0	0	0	215	218
www.aldridges.co.uk	2104	0	0	0	0	0
www.butisitart.co.uk	296	0	0	0	74	74
www.eclecticgames.co.uk	1754	0	0	0	0	0
www.jacobsthejewellers.com	771	0	0	0	0	6
www.rosieswoolemporium.com	59	0	0	0	0	0
www.coloursmayvary.com	314	0	0	0	71	77
www.villagebooks.co	534	0	0	0	125	128
www.okcomics.co.uk	130	0	0	0	15	15
www.framecraftonline.com	2103	0	0	0	51	51
www.juliadavey.com	621	0	0	0	168	170
www.chaniibshoes.com	511	0	0	0	113	121
www.thesilvershopofbath.co.uk	3201	0	0	0	0	47
www.morston.net	334	0	0	0	3	5
www.lostartshop.com	207	0	0	0	33	36
www.transalpino.co.uk	2861	0	0	0	0	0
www.pop-boutique.com	540	0	0	0	35	35
www.69aliverpool.co.uk	1500	0	0	0	70	70
www.thecardingshed.co.uk	198	0	0	0	0	0
www.uniquelylocal.co.uk	1323	0	0	0	1	3
www.rivetandhide.com	14443	0	0	0	110	220
www.utilitygift.co.uk	985	0	0	0	310	466
www.form-shop.com	252	0	0	0	0	2
www.magma-shop.com	854	0	0	0	369	370
www.oipolloi.com	1070	0	0	0	314	417
www.abebooks.co.uk	751	0	0	0	1	5
www.lemonfizzgifts.co.uk	550	0	0	0	0	0
www.moshulu.co.uk/petersfield	784	0	0	0	0	103
www.rainbowspetersfield.com	15	0	0	0	0	0
www.regattaroom.co.uk	118	0	0	0	0	0
www.hopelessrecords.eu/hlruk	162	0	0	0	2	4
www.lego.com	1700	0	0	0	6	8
www.next.co.uk	477	0	0	0	32	36
www.newlook.co.uk	718	0	0	0	64	319

VI. CONCLUSIONS

This paper analyses and discovers security vulnerabilities in e-commerce web applications and verifies whether the applications are implementing adequate security controls. Furthermore, the work proposed a new tool web application vulnerability scanning tool named Clarity. Clarity took a dynamic black box penetration testing approach to automatically detect security vulnerabilities in web applications. It was able to successfully detect Cross-Site Scripting vulnerabilities, SQL Injection, HTTP Response Splitting and Session Management flaws. Clarity aims to be an affordable, user-friendly alternative to existing tools, with the primary focus being targeting individuals with little experience in the field of cybersecurity. With this proposed tool, this work investigated the security of 50 e-commerce web applications in order to identify how secure these applications are on the web. We also provided a comprehensive description of these vulnerabilities including how they occur.

The results of the web applications analysed by Clarity uncovered an abundance of Session Management flaw, several SQL injection vulnerabilities, and no XSS or HTTP Response Splitting vulnerabilities. However, it was discovered that one web application suffered from an XSS vulnerability which could potentially cause significant damage to the application. This false negative could be an anomaly, but it could also be possible that there are more vulnerabilities in the scanned applications which have been overlooked. This instance resulting in a false negative can be overcome by increasing how thoroughly Clarity understands the input validation used in form fields and how efficiently it abides by these rules. The results show that although many applications do use the appropriate security controls, there is still much room for improvement. Session Management flaws can be relatively minor, XSS vulnerabilities on the other hand cannot be overlooked.

Though there is still much progress to be made to make online shopping a safer experience, this work has highlighted some of the key issues that need attention. By enabling individuals to make themselves aware of the dangers on the web, we offer them the opportunity to put measures in place to prevent themselves from falling victim to malicious users looking to take advantage of these vulnerable applications.

REFERENCES

[1] J. Clement, Number of Digital Buyers Worldwide from 2014 to 2021, Jul. 2019. [Online]. https://www.statista.com/statistics/251666/ number-of-digital-buyers-worldwide/

[2] H. Atashzar, A. Torkkaman, M. Bahrololum, M. H. Tadayon, "A Survey on Web Application Vulnerabilities and Countermeasures," *6th International Conference on Computer Sciences and Convergence Information Technology (ICCIT)*, 2011.

[3] OWASP Top 10 – 2017. Available: https://owasp.org/www-project-top-ten/

[4] X. Li and Y. Xue, "A Survey on Web Application Security", 2011. [Online]. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.434.7174&rep=rep1&type=pdf

[5] R. Nixon, "The Benefits of PHP, MySQL, JavaScript, and CSS", in Learning PHP, MySQL, JavaScript, and CSS, 2nd ed: O'Reilly Media, 2012, pp. 5-8.

[6] C. A. Vlsaggio and L. C. Blasio, "Session management vulnerabilities in today's web," *IEEE Security & Privacy*, vol. 8, no. 5, pp. 48-56, Sept.-Oct. 2010.

[7] A. Petukhov and D. Kozlov, "Detecting Security Vulnerabilities in Web Applications Using Dynamic Analysis with Penetration Testing". *OWASP Application Security Conference*, Ghent, Belgium, 2008.

[8] Y.-W. Huang and D. T. Lee, "Web Application Security - Past, Present, and Future", *Computer Security in the 21st Century*, Springer US, pp. 183-227 (2005).

[9] Y.-W. Huang, S.-K. Huang, T.-P. Lin, Ch.-H.. Tsai, "Web application security assessment by fault injection and behavior monitoring", *12th international conference on World Wide Web*, May 2003.

[10] A. Wiegenstein, F. Weidemann, M. Schumacher, S. Schinzel, "Web Application Vulnerability Scanners - a Benchmark", Virtual Forge GmbH, 2006.

[11] OWASP Top 10 – 2013. Available: https://www.owasp.org/images/f/f8/OWASP_Top_10_-_2013.pdf

[12] V. B. Livshits, and M. S. Lam, "Finding Security Vulnerabilities in Java Applications with Static Analysis", *14th Usenix Security Symposium*, 2005, pp. 273-274.

[13] D. Gol and N. Shah, "Detection of Web Application Vulnerability Based on RUP Model", *National Conference on Recent Advances in Electronics & Computer Engineering (RAECE)*, 2015.

[14] M. Liu and B. Wang, "A Web Second-Order Vulnerabilities Detection Method", *IEEE Access*, vol. 6, pp. 70983-70988, 2018.

[15] T. Farah, M. Shojol, M. Hassan and D. Alam, "Assessment of vulnerabilities of web applications of Bangladesh: A case study of XSS & CSRF," *International Conference on Digital Information and Communication Technology and its Applications (DICTAP)*, Konya, 2016, pp. 74-78.

[16] M. Liu, B. Zhang, W. Chen and X. Zhang, "A Survey of Exploitation and Detection Methods of XSS Vulnerabilities", *IEEE Access*, vol. 7, pp. 182004-182016, 2019.

[17] Y. K. Malviya, S. Saurav and A. Gupta, "On Security Issues in Web Applications through Cross Site Scripting (XSS)", *20th Asia-Pacific Software Engineering Conference (APSEC)*, 2013.

[18] H. Atashzar, A. Torkaman, M. Bahrololum and M. H. Tadayon, "A survey on web application vulnerabilities and countermeasures", *6th International Conference on Computer Sciences and Convergence Information Technology (ICCIT)*, Seogwipo, 2011, pp. 647-652.

[19] Bwapp, the deliberately insecure web application - https://www.mmebvba.com/sites/bwapp/index.htm

[20] Damn Vulnerable Web Application (DVWA) - http://www.dvwa.co.uk/

[21] Wapiti open source vulnerability scanner – http://wapiti.sourceforge.io

[22] Vega open source vulnerability scanner – http://subgraph.com/vega/

[23] Netsparker vulnerability scanner - https://www.netsparker.com/

[24] Acunetix vulnerability scanner - https://www.acunetix.com/

[25] K. Anagandula and P. Zavarsky, "An Analysis of Effectiveness of Black-Box Web Application Scanners in Detection of Stored SQL Injection and Stored XSS Vulnerabilities," *2020 3rd International Conference on Data Intelligence and Security*, 2020, pp. 40-48.

[26] S. Tyagi and K. Kumar, "Evaluation of Static Web Vulnerability Analysis Tools," *2018 Fifth International Conference on Parallel, Distributed and Grid Computing (PDGC)*, 2018, pp. 1-6.

[27] A. Al Anhar and Y. Suryanto, "Evaluation of Web Application Vulnerability Scanner for Modern Web Application," *2021 International Conference on Artificial Intelligence and Computer Science Technology (ICAICST)*, 2021, pp. 200-204.

[28] B. Mburano and W. Si, "Evaluation of Web Vulnerability Scanners Based on OWASP Benchmark," *2018 26th International Conference on Systems Engineering (ICSEng)*, 2018, pp. 1-6.

[29] H. Alptekin, S. Demir, Ş. Şimşek and C. Yilmaz, "Towards Prioritizing Vulnerability Testing," *IEEE International Conference on Software Quality, Reliability and Security Companion*, 2020, pp. 672-673.

[30] H. Chen, J. Chen, J. Chen, S. Yin, Y. Wu and J. Xu, "An Automatic Vulnerability Scanner for Web Applications," *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, 2020, pp. 1519-1524.