

**Identify Vulnerability in a Mobile App for  
the Hackazon Application**

Hayden Eubanks

School of Business, Liberty University

CSIS 486-D01

Prof. Backherms

November 26, 2023

## **Initial Dynamic Security Scan of the Hackazon Application**

### **Introduction:**

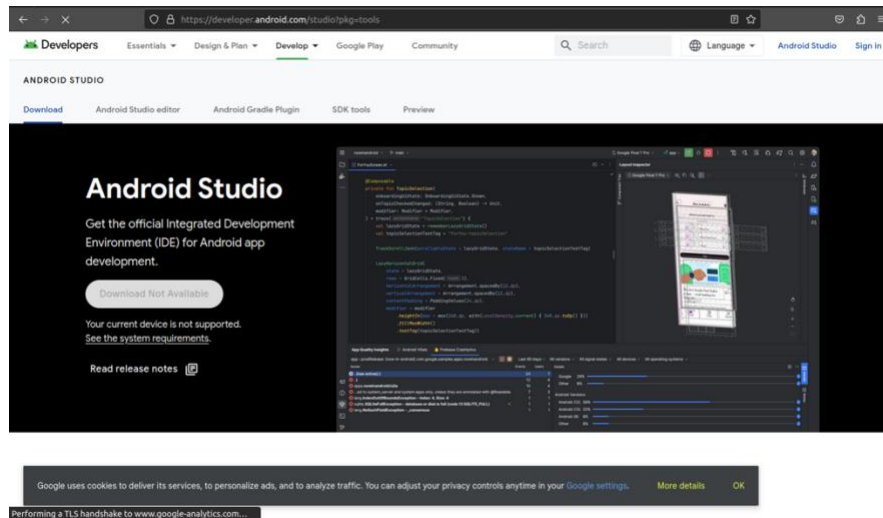
When examining the infrastructure of many modern databases, it can be seen that most web pages utilize a database for storing and retrieving data elements (Bedečković et al., 2022). Often, the data within these databases is retrieved by issuing SQL commands to the database to have the database return the desired values (Simpson & Anthill, 2017). This could include data such as the product listings for an e-commerce application, but also more sensitive info such as login credentials, payment information, or sensitive personal details. With this in mind, it can be understood that the data stored within databases is an attractive target for cybercriminals and as such malicious actors may seek to bypass the security controls of a webpage and have the database return sensitive data to them. Additionally, this attack vector may potentially be used to insert or remove items into the database further emphasizing the severity of this vulnerability. Malicious actors seek to accomplish this by performing SQL injection attacks where SQL code can be inserted into an input field to modify the behavior of the database interaction (Bedečković et al., 2022). SQL injection is one of the most prevalent vulnerabilities found within web applications today (OWASP, 2021) and as such, security professionals must understand how SQL injection attacks are performed as well as mitigation strategies for protecting sensitive data from malicious injection.

When addressing SQL injection vulnerabilities, security professionals may utilize a variety of techniques and tools to effectively mitigate the vector of attack. One of the most effective of these methods is referred to as interface fuzzing and involves entering unexpected input into input fields and examining the output results of that entry (GitLab, n.d.). For example, enumerating a list of common SQL injection commands and then examining the output may

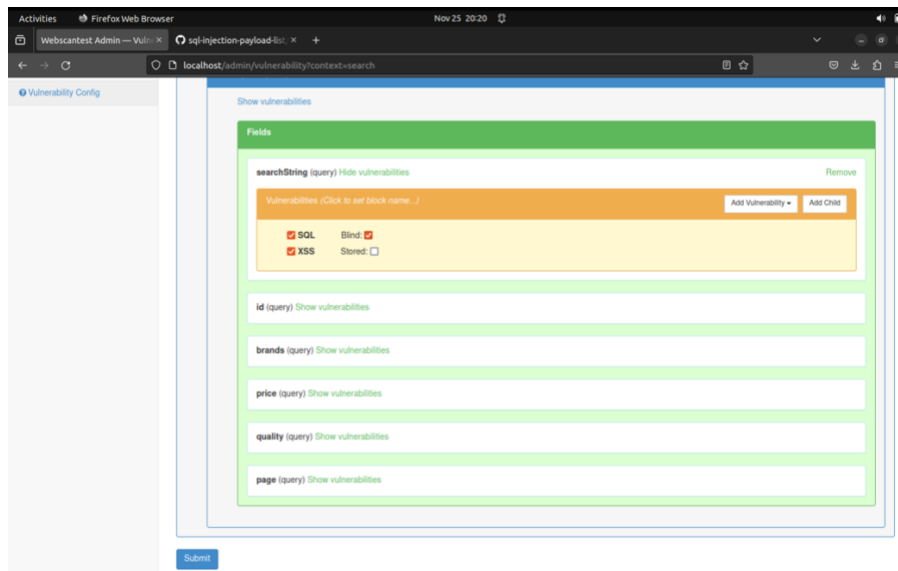
reveal some outputs being significantly larger potentially indicating a SQL injection vulnerability (Zhang et al., 2019). Additionally, exploratory and manual testing are strong candidates when testing for SQL injection as the output from trial and error with SQL injection may reveal details about the underlying architecture leading to more specific attacks (Zhang et al., 2019). This technique can be further utilized in blind SQL injection attacks where the attacker injects SQL code to have the system return true or false as well as error-based SQL injection attacks where the attacker attempts to generate error messages and by examining this output the attacker can glean information regarding how the attack can be further carried out (Bedeković et al., 2022). Fortunately, SQL injection is relatively easily mitigated, and in many cases, the vulnerability can be mitigated by encoding user input and filtering out the special characters involved in injection attacks (Zhang et al., 2019). Within the context of inspecting the Hackazon application for SQL injection vulnerabilities, the Burp suite of tools will be used to automate the inspection process. Burp Suite is a powerful tool for detecting web application vulnerabilities and highlights the importance for security professionals to learn and adopt a tool for the vulnerability scanning process.

## Testing Setup and Configuration:

### Screenshot 1: Android debugger incompatible with my system

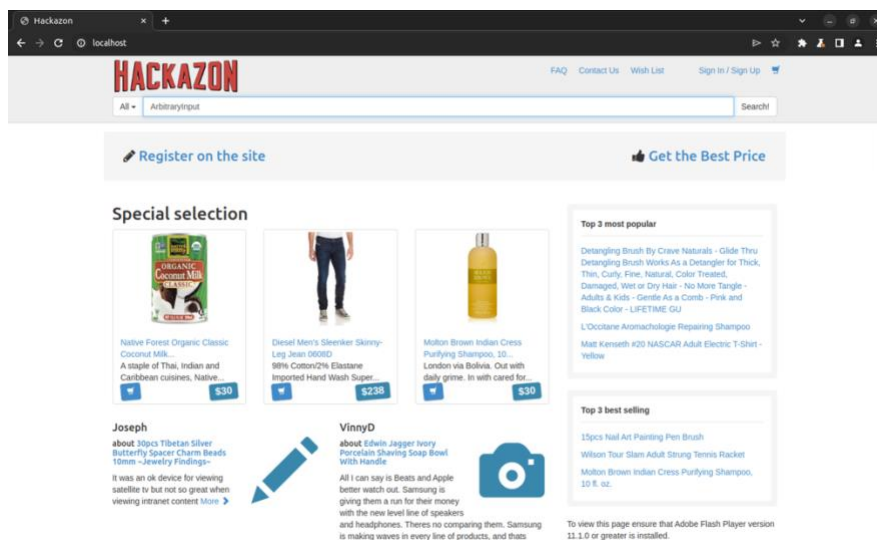


### Screenshot 2: Configuring Hackazon for SQL Injection

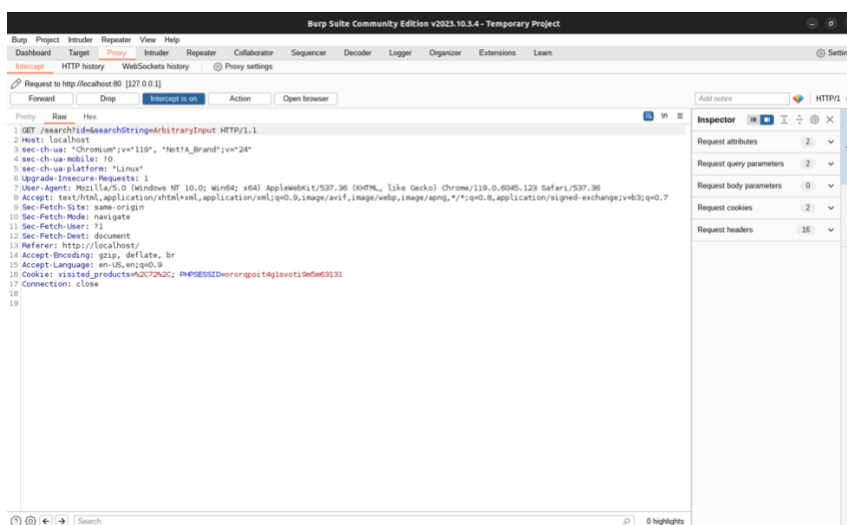


## Evidence of Fuzzing Activity and Vulnerabilities Found:

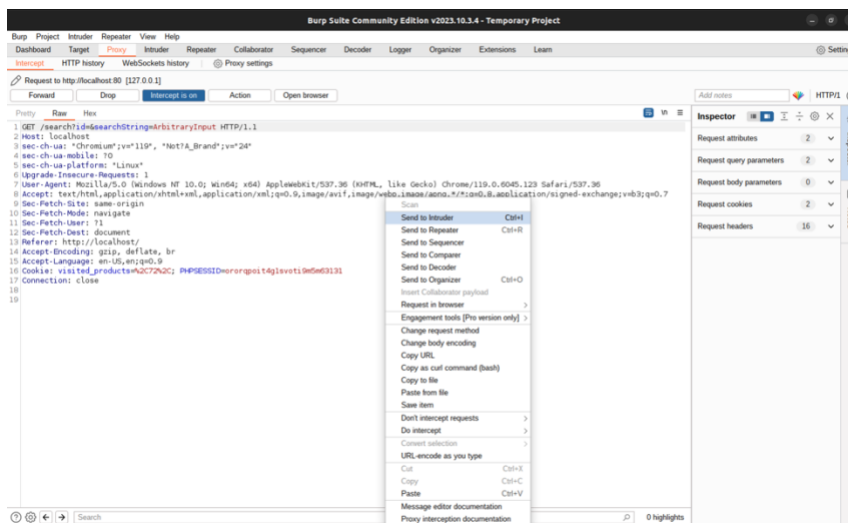
### Screenshot 3: Enter Arbitrary Input into the search field



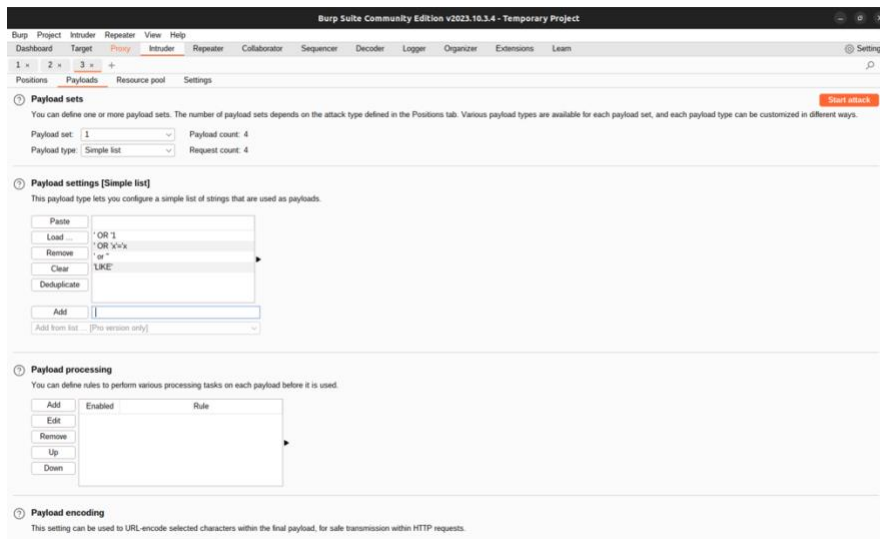
### Screenshot 4: Examine HTTP Request



## Screenshot 5: Identify the input field for replacement with SQL Injection



## Screenshot 6: Input list of SQL Injection commands to be iterated over



## Screenshot 7: Examine the results of the attack

5. Intruder attack of http://localhost - Temporary attack - Not saved to project file

Attack Save Columns

Results Positions Payloads Resource pool Settings

Filter: Showing all items

Request ^	Payload	Status code	Error	Timeout	Length	Comment
0		200	<input type="checkbox"/>	<input type="checkbox"/>	32522	
1	' OR 'X'='X	200	<input type="checkbox"/>	<input type="checkbox"/>	51622	
2	'LIKE'	503	<input type="checkbox"/>	<input type="checkbox"/>	413	
3	'or "	503	<input type="checkbox"/>	<input type="checkbox"/>	413	
4	'OR '1	200	<input type="checkbox"/>	<input type="checkbox"/>	51538	

Finished

## Screenshot 8: All items shown when the attack is entered into the search field

Activities Firefox Web Browser Nov 25 20:24

Hackazon — Search by «' or '1'='1»

localhost/searchId=searchString='or+'1'%3D'1

HACKAZON

FAQ Contact Us Wish List Your account Logout

All «' or '1'='1» Search!

Brands

Apple  
NBA  
InterDesign  
Sony Pictures Entertainment

Price

✓ \$0 – \$100  
\$100 – \$200  
\$200 – \$300  
\$300 – \$500  
\$500 – \$1000

Quality

Brand New  
Used/Preowned  
Refurbished

Filter

Search by «' or '1'='1»

Martha Stewart Crafts Garland, Pink...  
Fun, festive party decorative pop poms. Perfect for any occasion. Pre-cut pieces.  
\$9

Martha Stewart Crafts Modern Festive...  
Decorate for your party with this Modern Festive pennant garland from Martha...  
\$5.6

Martha Stewart Crafts Pom Poms, Pink...  
Fun, festive party decorative pop poms in two sizes. Includes 5 tissue paper pom...  
\$10

Martha Stewart Gift Card Box, White...  
Gift card box for guests to insert gift cards and envelopes. Easy to assemble...  
\$13

Sketch  
Sketch Pads 9...  
a general purpose, medium weight sketch paper intended for technique practices...  
\$9.9

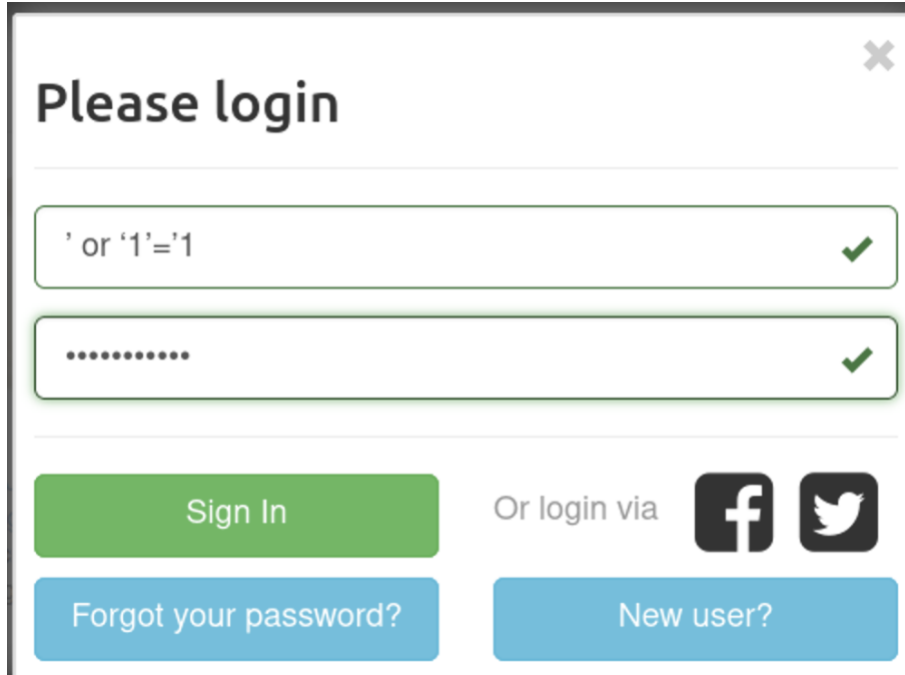
100 Pcs Swarovski Crystal Rondelle...  
Material: Silver-Plated Quantity: 100 pcs Size: 8 mm Inside Size: 2mm (HCT)...  
\$2.25

30pcs Tibetan Silver Butterfly Spacer...  
This item is for 30pcs of these lovely Tibetan Silver Butterfly charm spacer...  
\$1.6

Beadnious 100 Pcs Gold Plated Crystal...  
Material: Lead Free Nickel Free Metal: Gold Plated Crystal Color: 4207 Montana  
\$9.99

localhost/product/viewId=4

**Screenshot 9: During account creation, the field returns that the user already exists**





Please login

' or '1'='1' ✓

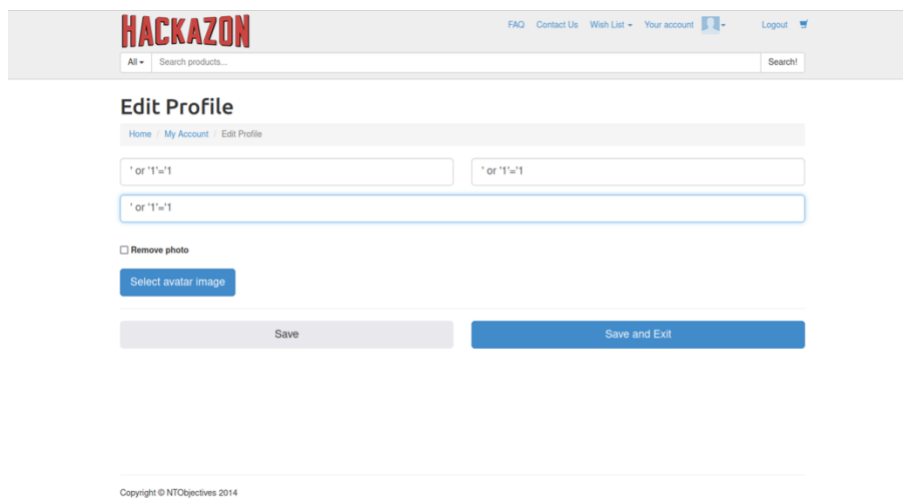
..... ✓

Sign In

Or login via  

Forgot your password? New user?

**Screenshot 10: Profile data can be edited to include injection commands**



HACKAZON

FAQ Contact Us Wish List Your account Logout

All Search products... Search!

### Edit Profile

Home / My Account / Edit Profile

' or '1'='1' ' or '1'='1'

' or '1'='1'

☐ Remove photo

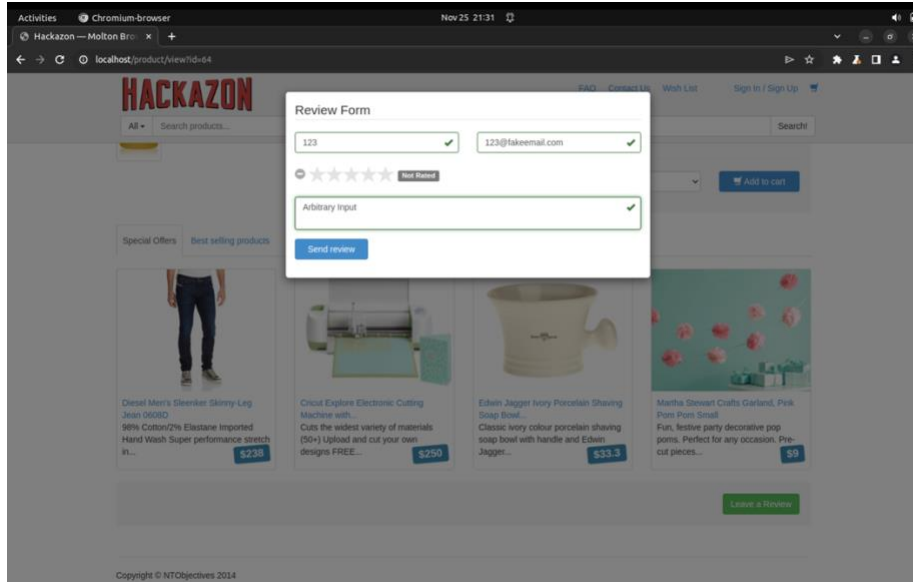
Select avatar image

Save Save and Exit

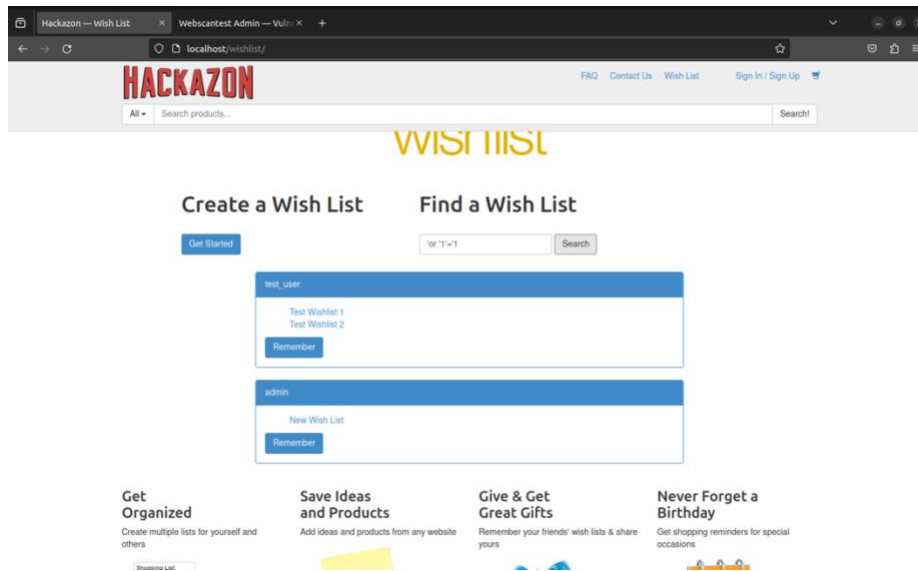
Copyright © NTOjectives 2014

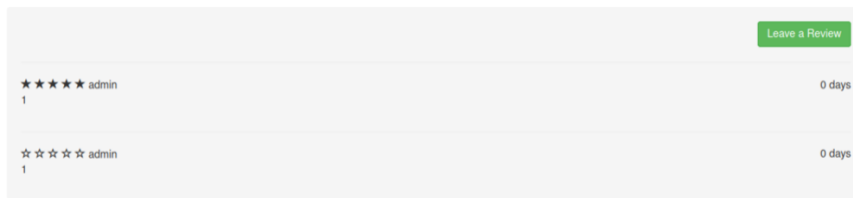
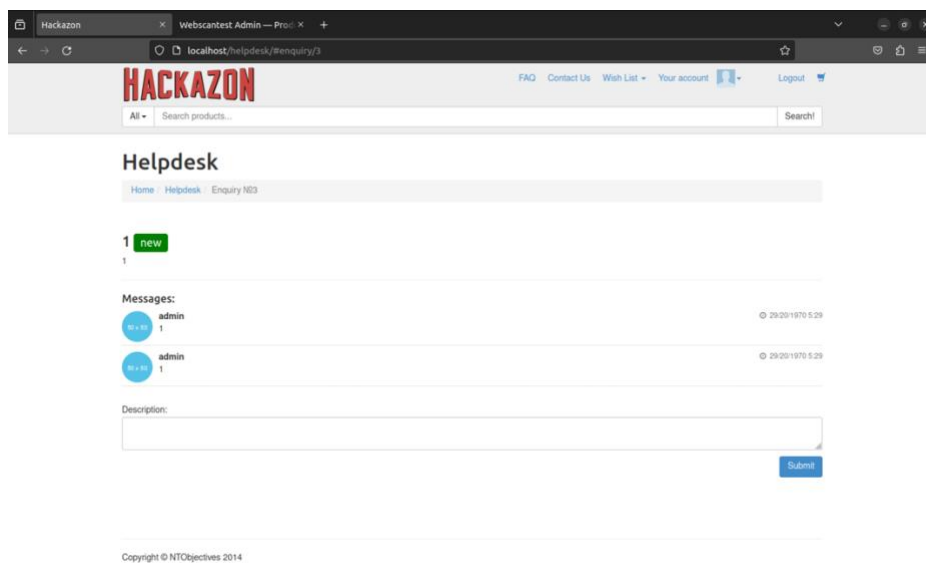


## Screenshot 11: SQL injection can be performed on user review comments



## Screenshot 12: All wish lists can be revealed with an injection command



**Screenshot 13: Review result returning the value of 1 after injection****Screenshot 14: A value of 1 is also returned on the helpdesk page when injection is performed**

### **Discussion of Discovered Vulnerabilities:**

Throughout the Hackazon application, several vulnerabilities to SQL injection were discovered, and performing an analysis of these vulnerabilities can allow a security professional to better understand SQL injection attacks as well as how they can be mitigated. As an e-commerce website, the Hackazon application is heavily reliant on its underlying database in providing its services to users as well as storing user login credentials. This means that throughout the website, several input fields send SQL queries to the database and as such must be tested for vulnerability to SQL injection. Some of the primary fields where vulnerabilities were discovered include the main search function, the leave a review section for a product, the Wishlist search field, and the account creation field. Each of these fields provides an access point to a SQL query interacting with the underlying database and as such must be protected from SQLi (SQL Injection) attacks.

For all of the fields being examined, the Intruder tool of the Burp Suite application was used which allowed for the automation of input fuzzing by iterating over common SQL injections and outputting data regarding the request's return (GitLab, n.d.). This can be accomplished by first setting the Burp Suite application to be a proxy for requests sent to the Hackazon application. This means that all requests intended for the Hackazon application are first passed through Burp Suite allowing the proxy to modify fields of the HTTP request and allowing for the automation of SQL injection input (PortSwigger, 2023). Within this analysis of the Hackazon application, a standard list of simple SQLi commands was used, but more complex or specifically tailored lists of SQLi payloads can be imported into the Intruder tool for increased testing coverage allowing the security professional to better ensure that SQLi vulnerabilities are mitigated. These payload lists can then be modified and used to observe the input field's

response to the differing forms of SQLi attacks including error-based, blind, union-based, or standard SQLi attacks allowing for a more comprehensive overview of the field's security to SQL injection (Bedeković et al., 2022).

The most standard form of SQL injection attack involves inserting special characters such as comment, escape, or quotation characters to terminate the intended input section of the SQL query and then inserting additional query instructions to execute commands on the database (Bedeković et al., 2022). This idea forms the foundation for all SQLi attacks and can allow a malicious actor to bypass access control and perform actions such as database modification, lookup, or even code execution (Zhang et al., 2019). Expanding on this, union-based attacks then additionally utilize the union command of SQL syntax which returns a union between two database tables (Bedeković et al., 2022). Union-based SQLi vulnerabilities could then be exploited to return information from tables outside of the table the field is meant to access highlighting a severe risk of data exposure. However, to perform more sophisticated attacks such as this, more information about the underlying database and SQL query must be understood and this may be accomplished through the error-based and blind SQLi techniques. The first of these, the error-based approach, involves sending values to the database that will generate errors and then observing the error messages to glean information regarding the type of SQL being used as well as how the query itself is structured (Bedeković et al., 2022). Similarly, blind SQLi attacks function by sending truth values to the database such as "and '1' = '1'" and then observing if the value returned is true or false to gain a greater understanding of the remaining elements of the SQL query (OWASP, 2023). In the previous example given, '1' = '1' should always return true so by logically ANDing this expression with the preceding values of the SQL query, more information can be gained regarding the structure of the query. If this SQLi returned false, the

attacker would then know that the portion of the query preceding the injected command results in a value of false allowing for the next input command to follow this discovery (Bedecković et al., 2022). The information gained from these approaches can then allow an attacker to fine-tune their injections and increase the likelihood of their attack succeeding.

The first field where a SQLi vulnerability can be observed is in the primary search field for searching for products on the Hackazon web application. By passing arbitrary input to this field and proxying the HTTP request, the inputs of SQLi commands could then be automated and the outputs observed (PortSwigger, 2023). While observing the output of the SQLi commands and comparing their return values against the baseline, it can be seen that the size of the return request for certain injections resulted in a return request much larger than the baseline potentially indicating a SQLi vulnerability. One of the SQLi payloads that potentially indicated a SQLi vulnerability was “‘or ‘1’ = ‘1’” and upon rendering the return value of the request, it was observed that all of the products for the website were returned as part of the request. From this observation, an assumption could then be made that the SQL query takes the user’s input and then searches the database for input like the inserted value. By terminating this portion of the query and appending a statement that always returns true the database responds by selecting all values for that table and returning them in the return request. This highlights a severe SQLi vulnerability as this field could then be used for further SQL injection attacks to attempt to access other database tables or to modify the values stored in the database.

A second part of the Hackazon application where a vulnerability to SQL injection was observed was the user registration page. By performing a similar exploration as outlined in the previous example, the fields of the user registration page were injected with SQL commands to observe the behavior of the database. When inserting a command that always returned true, such

as the injection outlined in the previous step for the username field an interesting error occurred stating that a user already existed with this username. This then potentially highlights another vector of attack where a malicious actor could inject the truth value for the username field in an effort to break access control and infiltrate accounts other than their own. During the analysis of this vulnerability, truth values were attempted to be injected into the username field of the application while iterating over passwords to observe the behavior of the database during the sign-in process. This attempted attack did not allow access to user accounts potentially indicating a difference between the SQL query utilized on the registration page and the one used for the sign-in page. However, the understanding that some username fields do accept a truth value formed the foundation for the next attack vector and discovered SQLi vulnerability.

With the understanding that some username fields accept a truth value for matching input, the next field that was inspected was the username search field of the wish list page. This field accepts user input and then returns the public wish lists of users who match the input username. However, by injecting a value that always returns true into the SQL query all user wish lists can be revealed and returned indicating a major breach of access control. This again highlights a major vulnerability as further SQL commands could then be injected into this field in an attempt to return values from other database tables or to modify the values existent within this table. However, the search field was not the only field of interest discovered on this page, and another interesting response was returned when modifying the value of a wish list's name through the edit name field. By modifying the name of a wish list to append a SQL injection value such as a value that always returns true or false the name of the wish list will be set to the Boolean value of either a '1' or a '0'. This same vulnerability was additionally discovered in most fields that stored user input such as the helpdesk page, the fields for editing personal

information, and the fields for product information. The fact that these fields return a Boolean value indicates that they could be a further target for exploration using blind SQLi techniques (OWASP, 2023).

Throughout the Hackazon application, several vulnerabilities to SQL injection were discovered and these vulnerabilities could potentially allow a malicious actor to break access control or execute malicious code on the database (Simpson & Anthill, 2017). While SQL injection remains one of the most prominent forms of attack against web applications, it is relatively easy to mitigate through the application of strong security practices during development (Bedečković et al., 2022). This could include implementing the principle of least privilege on input fields so that only the SQL features needed for an input field are enabled as well as limiting the special characters allowed (Bedečković et al., 2022). Additionally, special characters could be encoded so that they are not in a form that could be executed as part of the SQL command. In addition to these mitigation strategies, limiting the information provided by error messages or blind SQL injection return values could mask the underlying database implementation making it more difficult for a malicious actor to discover vulnerabilities (OWASP, 2023). SQL injection is a serious vulnerability facing web applications that utilize a database and as such it is essential that strong security practices be enforced during development as well as security testing be performed to ensure the vulnerability is mitigated. This highlights the importance for security professionals to understand SQL injection vulnerabilities and be able to inform developers of the dangers of SQL injection.

## References

- Bedeković, N., Havaš, L., Horvat, T., & Crčić, D. (2022). The importance of developing preventive techniques for SQL injection attacks. *Tehnički Glasnik*, 16(4), 523-529. <https://doi.org/10.31803/tg-20211203090618>
- GitLab. (n.d.). *Web API fuzz testing*. GitLab.  
[https://docs.gitlab.com/ee/user/application\\_security/api\\_fuzzing/](https://docs.gitlab.com/ee/user/application_security/api_fuzzing/)
- OWASP. (2021). *OWASP top ten: Top 10 web application security risks*. OWASP.  
<https://owasp.org/www-project-top-ten/>
- OWASP. (2023). *Blind SQL injection*. OWASP. [https://owasp.org/www-community/attacks/Blind\\_SQL\\_Injection](https://owasp.org/www-community/attacks/Blind_SQL_Injection)
- PortSwigger. (October, 2023). *Vulnerabilities detected by Burp scanner*. PortSwigger.  
<https://portswigger.net/burp/documentation/scanner/vulnerabilities-list>
- Simpson, M. T., Anthill, N. (2017). *Hands-on ethical hacking (3<sup>rd</sup> ed.)*. Cengage Learning.  
<https://ng.cengage.com/static/nb/ui/evo/index.html?deploymentId=5681612456081340358553076923&eISBN=9781337271721&id=1937025370&snapshotId=3720027&>
- Zhang, L., Zhang, D., Wang, C., Zhao, J., & Zhang, Z. (2019). ART4SQLi: The ART of SQL injection vulnerability discovery. *IEEE Transactions on Reliability*, 68(4), 1470-1489. <https://doi.org/10.1109/TR.2019.2910285>