Complete Security Analysis

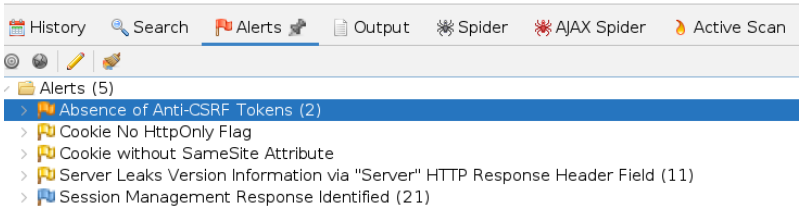
"Vanishing Notes": The application is a way for people to exchange information using a third-party service and ensure that no trace remains. The user uses the website to create the note, grab the link to the note after its creation, and send the note to the receiver. Only one person can view the note: after a person has accessed the note, the record is deleted and all traces of it are removed from the website. The web service does not keep any logs of the interaction. No personal information whatsoever is collected from the users.

Assignment Tasks:

1. **Automated Vulnerability Scanner:**

    a. **Start with an automated vulnerability scan using ZAP Proxy. Look at the list of vulnerabilities that were reported. Grab a screenshot of the list of alerts and paste the screenshot into your word document.**

The screenshot below shows the 5 alerts reported during the ZAP Proxy automated vulnerability scan.



    b. **Analyze the two most serious vulnerabilities that were reported. With a few simple lines, explain what those vulnerabilities mean and why the scanner is reporting them.**

The two most serious vulnerabilities found in this automated scan were the absence of Anti-CSRF Tokens. The scan found that this vulnerability was present in two different parts of the application, namely, create.php and preview.php. This was a medium alert, the highest of all alerts that the scan generated. These vulnerabilities mean that in these two parts of the application, there is an html form submission that is susceptible to cross-site request forgery. This vulnerability allows an attacker to trick an authenticated user into performing unwanted actions on a web application in which they're currently authenticated. The attacker sends a request to a web application that the user is logged into, and the web application can't distinguish between legitimate requests and forged requests. This means the

attacker can make the user's browser perform unwanted actions without the user's knowledge. The scanner is reporting this vulnerability because, without any preventative techniques against this attack, such as CSRF tokens, this application has no way of validating requests to ensure they are coming from the authenticated user, and not some attacker. This poses an immediate threat, and is a major vulnerability in this application. Other vulnerabilities that were given a 'low-priority' include 'Cookie No HttpOnly Flag', 'Cookie without SameSite Attribute' and 'Server Leaks Version Information via "Server" HTTP Response Header Field'. All of these alerts are no doubt important, however the ones given the highest priority are the two listed under the 'Absence of Anti-CSRF Tokens' alert.

c. **The ZAP Proxy report will list a "Low Priority Alert" for "Cookie No HttpOnly". Explain what this vulnerability is and how to fix it.**

This vulnerability is present in the application since there have been Cookies set within the application without a flag called HttpOnly. This basically means that the cookies set within the application can easily be accessed using JavaScript. If an attacker uses malicious script on this page, the cookie will be easily accessible and can be transmitted to another site. If these cookies are session cookies, that would mean that the attacker is able to conduct session hijacking through the use of these cookies. There is a simple solution for this alert, simply to ensure that this flag, 'HttpOnly' is set for all cookies in the script. This would ensure that the cookies are not easily accessible by JavaScript and would avoid this issue altogether.

2. **Database Interaction:**

a. **The application is using basic requests (INSERT, UPDATE, DELETE) to modify the data in the database. Suggest another way that the application could perform these operations in a more tightly controlled fashion.**

While there is nothing inherently wrong with using basic SQL requests to modify Database data, there are better ways to ensure a more secure application. For instance, instead of using direct SQL statements to modify the database, using stored procedures and parameterized queries would be a more secure, efficient alternative. Using stored procedures means encapsulating the SQL code, and optionally including additional logic to validate and sanitize inputs. This ensures that only authorized operations are performed on the database, and the data within. Additionally, combined with parameterized queries is a very effective way of preventing SQL injection attacks. This is done using placeholders for parameters

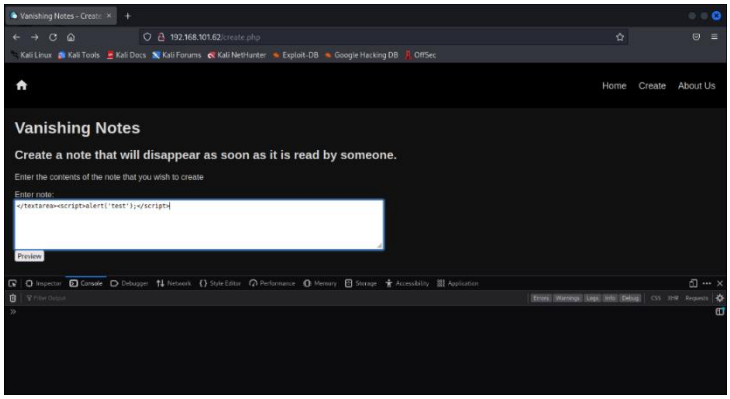
within the SQL statements and then binding the real values after. This ensures that data is correctly formatted and sanitized before inputting it into the database. Using stored procedures and parameterized queries instead of simple, direct SQL requests can ensure that these operations are done in a more tightly controlled fashion.

b. **The application has created a user to be the administrator of the database for the application. Explain in a few lines if you agree with this approach as opposed to using the root account for the interaction instead.**

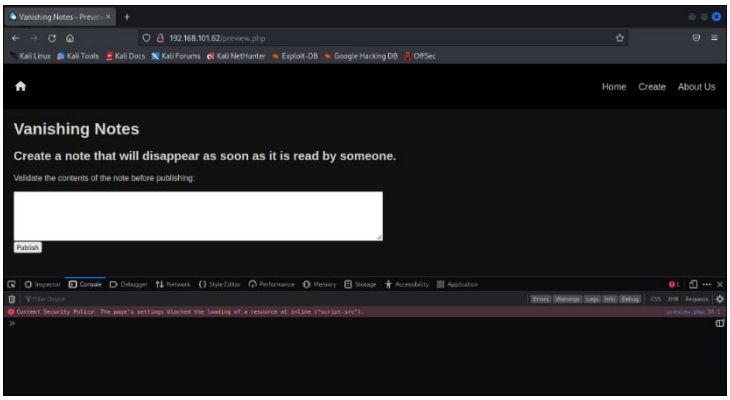
Although, yes, it is better for the user to be the administrator of the database as opposed to the root account, this still gives the user too much power. If an attacker tries to manipulate the database, they will have all the privileges granted for this database, since the user is the administrator. It is surely much better than them being the root account, however, it would be better if there was a 'user' account. This would allow only necessary privileges for the application to work, while limiting everything else. This would ensure that if an attacker wishes to manipulate the database, their actions would be limited, and they would not get very far,

3. **Cross-Site-Scripting (XSS) analysis:**

a. **ZAP proxy is reporting that the application might be susceptible to XSS. The application does let potentially malicious content go through. For instance, this injection could trick the application to close the text area input element, and then activate injected JavaScript:**



b. **However, if we attempt the injection, the process fails, and the console provides this message:**



c. **In a few simple lines, explain what is happening in the application that is preventing the injection from being successful.**

Here, the console provides the reasoning for which the malicious script cannot be executed, for it is within the Content Security Policy. In the configuration files, namely '000-default.conf' there is a Content Security Policy header configuration, which looks like the following line: "*Header always set Content-Security-Policy "default-src 'self'; frame-src 'self'; frame-ancestors 'none'; form-action 'self'; script-src 'self'; style-src 'self';""* This line is a security feature embedded within the application, included on all pages of the application that limits possible attacks, such as data injection as well as cross-site scripting attacks by controlling the location from where different types of content can be loaded, reducing the risk of vulnerabilities. For instance, in the given example, the injected JavaScript was not executed because of the line in the configuration file *script-src 'self';* this line restricts the sources from which scripts can be loaded to only the same origin. This prevents the execution of malicious scripts, or JavaScript injection such as seen in this example.

4. **SQL Injection analysis:**

a. **The automated scanner did not report any vulnerability to SQL Injection, but it does not mean that no SQL Injection vulnerability remains.**
b. **In the code for the script "preview.php", the note is inserted into the database. There is a mechanism applied to the content of the note (the only user-provided entry) that makes it virtually impossible to inject SQL into the interaction. In one or two lines, explain what is being done on the server side to prevent the vulnerability.**

The mechanism being applied to the note content in 'preview.php' that mitigates the vulnerability of SQL injection is that the note content is being encoded. In the

script of this file, the value of the note is subject to Base64 encoding before it is inserted into the database. This means that the value of the note is not legible or understandable to the database itself. Because of its encoding, the database would never know the real value of the message; therefore it could not harm the database.

   c. **In the code for the script "view.php", the parameter "note_id" is provided by the user and its use creates a potential vulnerability for SQL Injection. However, there is a security mechanism applied to the parameter to prevent the vulnerability. In one line, explain how this is done.**

The security mechanism is that the parameter must be an integer, this prevents any SQL injection vulnerability since the application will convert the parameter to an integer and refuse any parameter that is not a number.

5. **The application does not enforce a sequence from the script "preview.php" and "publish.php". These two scripts should always follow each other, i.e., it does not make sense to invoke "publish.php" directly. Briefly explain how to enforce the flow of operations between "preview.php" and "publish.php".**

One way that this can be enforced is using session variables to track which page the user has already visited. For instance, a session variable can be set to true when the user visits the 'preview.php' page. Then, on 'publish.php' that same session variable can be evaluated, if the value is true, the user may be allowed to view the page, if the session variable is not set, or the value is anything else, then the user should be redirected back to the preview page. This would ensure that the user is following the proper flow of operations of this application.

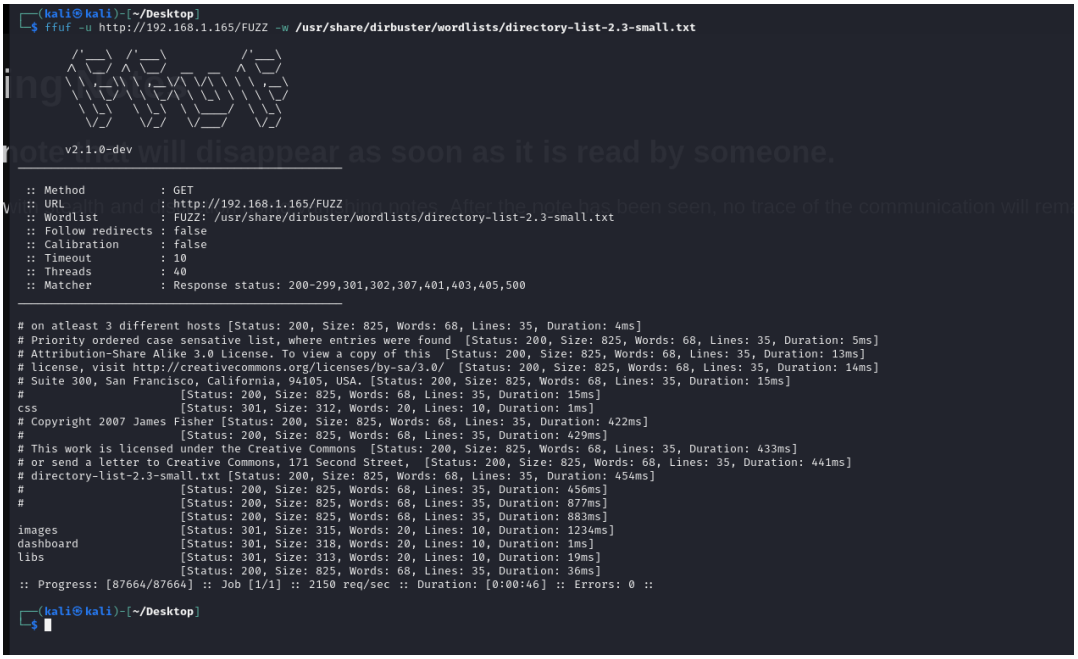
6. **In your brief discussion with the website operators, you asked if there was an administration panel for the application. They said that it was not reachable by the users and there was no need to assess its security. As part of your analysis, you will use a web crawler to test this claim.**

   a. **In Kali Linux, use the command-line utility "ffuf" to do a brute-force search on existing directories of the web application. The utility works with a list of common names that you provide and will test each name to see if there is a response from the web server. For example, if the target would be at the IP address 192.168.101.62, you could search for existing folders with a command such as:**

```
ffuf -u http://192.168.101.62/FUZZ -w /usr/share/dirbuster/wordlists/directory-list-2.3-small.txt
```

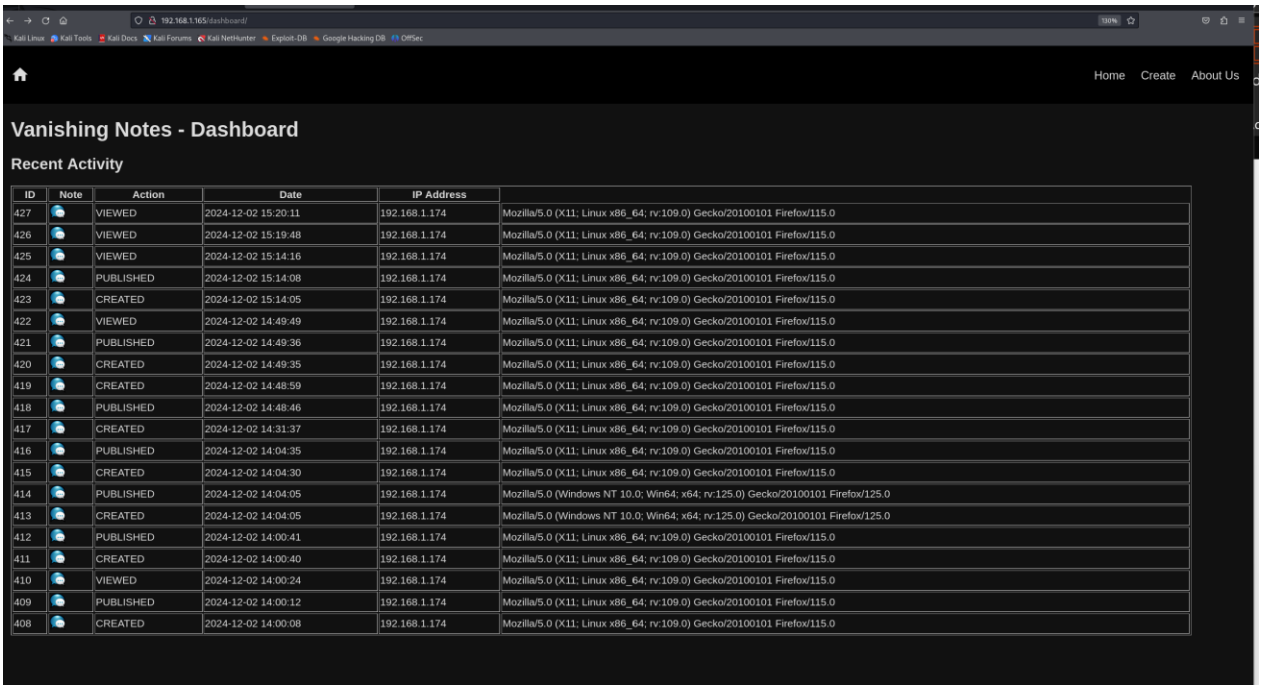
b. In the sample command, the word “FUZZ” will be replaced by each word in the list that was provided. Results which give a response will be listed in the screen. Perform the ffuf brute-force analysis for existing directories in the web server. Copy the execution of the script and paste it into your word document. Which hidden directory have you uncovered?

The screenshot below shows the output of this command. Evidently, as seen in the screenshot, the hidden directory “dashboard” has been uncovered, using this command.



c. Using a browser, explore the newly discovered folder. In a few lines, explain what the purpose of the script in this folder is. Grab a screenshot of the script output and paste into your word document.

The purpose of this script is to track the use of this application. Evidently, the output shows all the use of this application, it shows the id, has a link to the note created, the date the ip address of the user as well as the user agent information of the one using the application. This script is possibly purposed to be used by an admin to track the usage of the application and log its usage. The screenshot below shows the browser output of this script.



ID	Note	Action	Date	IP Address	
427		VIEWED	2024-12-02 15:20:11	192.168.1.174	Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
426		VIEWED	2024-12-02 15:19:48	192.168.1.174	Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
425		VIEWED	2024-12-02 15:14:16	192.168.1.174	Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
424		PUBLISHED	2024-12-02 15:14:08	192.168.1.174	Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
423		CREATED	2024-12-02 15:14:05	192.168.1.174	Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
422		VIEWED	2024-12-02 14:49:49	192.168.1.174	Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
421		PUBLISHED	2024-12-02 14:49:36	192.168.1.174	Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
420		CREATED	2024-12-02 14:49:35	192.168.1.174	Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
419		CREATED	2024-12-02 14:48:59	192.168.1.174	Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
418		PUBLISHED	2024-12-02 14:48:46	192.168.1.174	Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
417		CREATED	2024-12-02 14:31:37	192.168.1.174	Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
416		PUBLISHED	2024-12-02 14:04:35	192.168.1.174	Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
415		CREATED	2024-12-02 14:04:30	192.168.1.174	Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
414		PUBLISHED	2024-12-02 14:04:05	192.168.1.174	Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:125.0) Gecko/20100101 Firefox/125.0
413		CREATED	2024-12-02 14:04:05	192.168.1.174	Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:125.0) Gecko/20100101 Firefox/125.0
412		PUBLISHED	2024-12-02 14:00:41	192.168.1.174	Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
411		CREATED	2024-12-02 14:00:40	192.168.1.174	Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
410		VIEWED	2024-12-02 14:00:24	192.168.1.174	Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
409		PUBLISHED	2024-12-02 14:00:12	192.168.1.174	Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
408		CREATED	2024-12-02 14:00:08	192.168.1.174	Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0

d. Now that you know how the administrative panel is implemented, what are your recommendations to the operators to make this safer? You can use arguments pertaining to security architecture, access controls, or whatever other relevant recommendation. Provide at least two relevant recommendations.

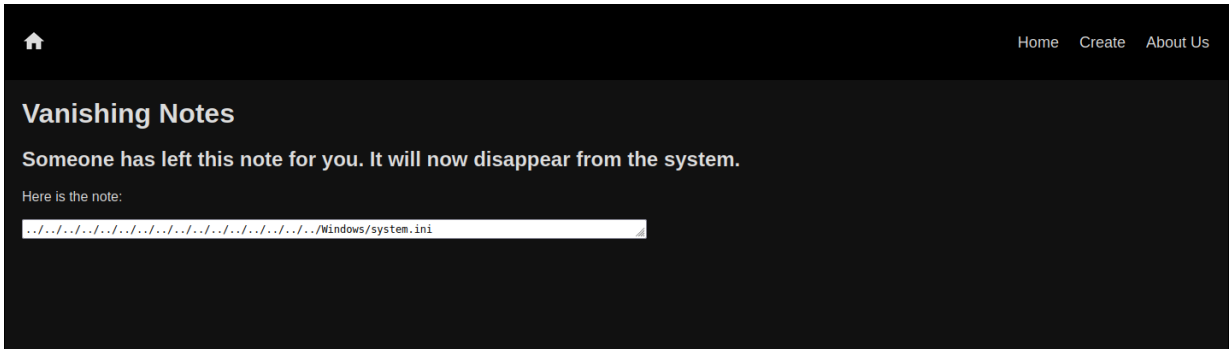
The following two security recommendations are provided in detail, explaining why they should be incorporated in this application.

i. The first recommendation to increase the security of this administrative panel is to set up some sort of access control within the application, preferably role-based access control. This would involve a login method at the very beginning, before the user is allowed to interact with the application. Normal users would be given a role like ‘user’ upon log in, which would prohibit their access to such sensitive information, such as the dashboard. Other users, perhaps members of the organization that would need access to the dashboard can be given a role such as ‘employee’ or ‘admin’, this would allow them higher access to the dashboard and other sensitive information. This would ensure that not just anyone could access the dashboard, only authorised users could.

ii. The second recommendation, in line with the first is to set up secure authentication, mainly, MFA, multi-factor authentication. Simply

using a password is not enough to authenticate a user, giving them access to such sensitive information, there should be more than one factor. Potentially using an authenticator app, which sends a randomly generated one-time password to that user's phone would be an effective example of this security measure. This would ensure that only the right person has the right privileges to access this information.

7. **One of the automated tests run by the vulnerability scanner has injected the following content into a note:**



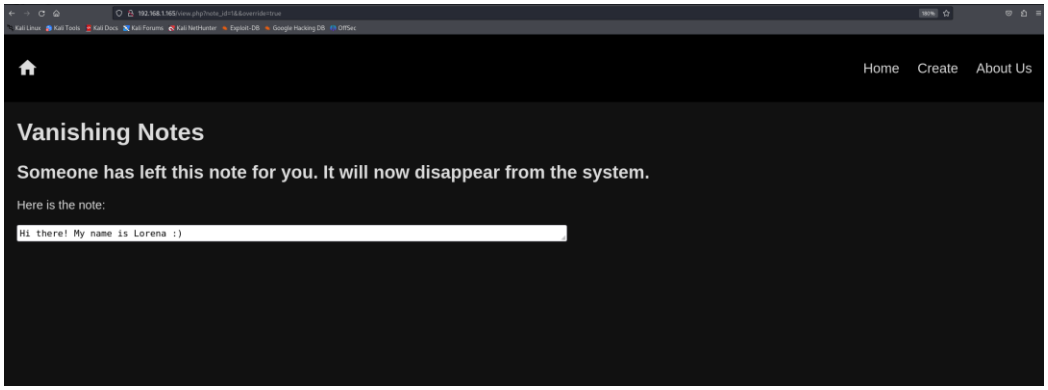
   a. **The scanner is attempting to test for a type of vulnerability. Which one? Write the answer in your word document.**

   The scanner is attempting to test for path traversal vulnerability. Using the Path Traversal vulnerability, the attacker attempts to exploit the web application through its underlying layers.

8. **In your analysis so far, you should have noted a "backdoor" access that was programmed into the script "view.php".**

   a. **Explain what a user needs to do to visualize any note for which he or she knows the ID number?**

   Very simply, the user needs to use the override method that was programmed in view.php. When the override method is set to true, the user can visualize any note for which they know the ID. Here's an example, if the user wanted to see the first note written, they would simply write in the search bar: "http://192.168.1.165**/view.php?note_id=1&&override=true**" This would produce the output of the first note, as seen in the following screenshot.



b. This backdoor will have two very specific effects as it pertains to reading notes. State what they are.

The effects this backdoor has are that this backdoor provides unrestricted access to any user, and there is a potential data breach. Since all the user needs to do to view any message recorded in this application is set the override to true, they have unrestricted access to all messages in the Database. Additionally, this poses a very serious threat of a potential data breach, since the user may share sensitive information through this application thinking it is safe, when in reality, their message can be seen by anyone.

9. Beyond the technical analysis of the web application, you can provide additional insight into the application that is beyond the capacity of an automated scanner. Answer the following questions and put your answers in your word document:

a. You know from your analysis that the claim from the operators that "no logs are kept" is not true. State which user data is being collected they retain besides the notes and what does this information mean? Is this data private?

Besides the notes that are being collected, user data such as IP address, as well as the user agent. The IP address, while self-explanatory is the IP address of the device the user is using to manipulate the web application. The User Agent is a string created by the browser of the user which identifies itself and provides relevant information about its capabilities. This string typically includes details like the browser name and version, the operating system, and other information about itself and the device. These are evidently seen in the dashboard of the site, and even worse, are stored in the database. These two pieces of information, while seemingly innocent are actually private pieces of information that should not be stored, especially without the user's consent. User agent

strings and IP addresses can be used by an experienced attacker to pinpoint a specific user and track them down. Since IP addresses give geolocation away and user agent strings provide specific information to a device, an attacker can use this information to 'fingerprint' a user. This is certainly private data and should not be logged the way that it is.

b. The notes are created with a sequential number. Based on the purpose of this application, explain why this is a security risk?

Since the purpose of this application is to allow users to send "vanishing" messages in a supposedly secure way, without worrying about them being seen later by other, unauthorized people, the fact that notes are created with a sequential number is incredibly risky. As already discovered, there is a 'backdoor' for any user to view any message for which they know the ID number. Since the ID numbers are sequential, starting with 1, any user can see any message by just incrementing the ID number by 1 each time. This completely disregards the purpose of this application, since anyone can see any message at any time.