Penetration Test Report

February 23th, 2020

Table of Contents

Introduction

Nonrepudiation

Nonrepudiation means to ensure that a transferred message has been sent and received by the parties claiming to have sent and received the message. Nonrepudiation is a way to guarantee that the sender of a message cannot later deny having sent the message and that the recipient cannot deny having received the message.In this case with the Api, data is sent and saved into a dattabase of which the records of all transactions are kept. Thus Non repudiation is properly catered for.

Confidentiality

Confidentiality is roughly equivalent to privacy of the data. Measures undertaken to ensure confidentiality are designed to prevent sensitive information from reaching the wrong people, while making sure that the right people can, in fact, get it: Access must be restricted to those authorized to view the data in question. It is common, as well, for data to be categorized according to the amount and type of damage that could be done should it fall into unintended hands. More or less stringent measures can then be implemented according to those categories.

Integrity

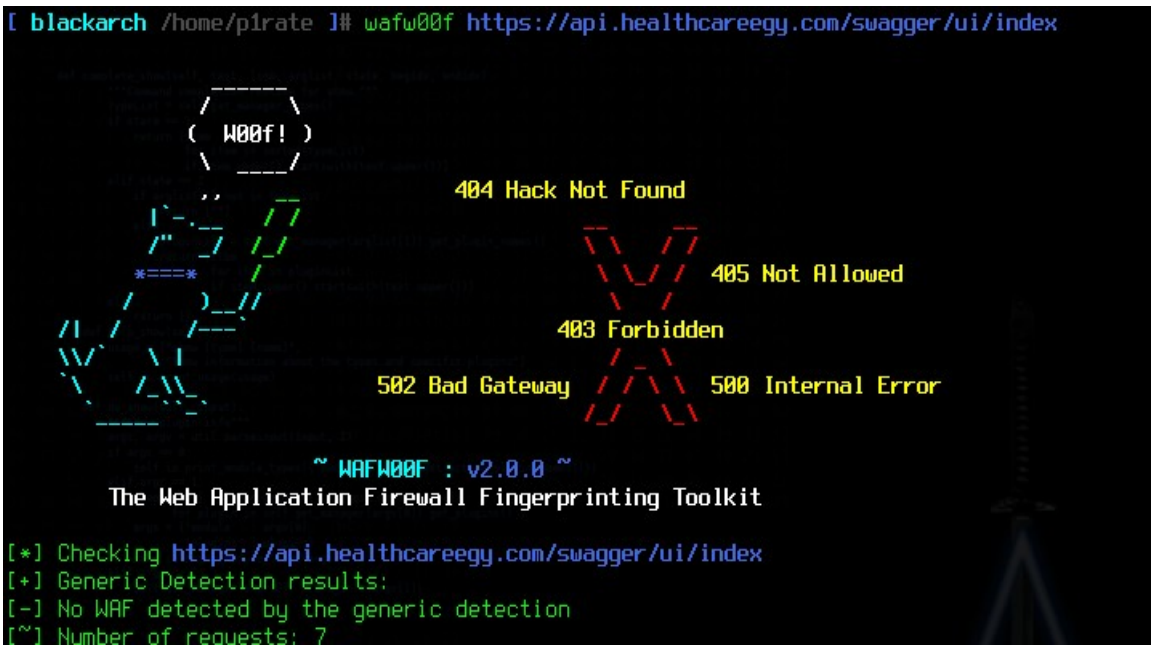
Integrity involves maintaining the consistency, accuracy, and trustworthiness of data over its entire life cycle. Data must not be changed in transit, and steps must be taken to ensure that data cannot be altered by unauthorized people (for example, in a breach of confidentiality). These measures include file permissions and user access controls. Version control may be used to prevent erroneous changes or accidental deletion by authorized users from becoming a problem. In addition, some means must be in place to detect any changes in data that might occur as a result of non-human-caused events such as an electromagnetic pulse or server. Some data might include checksums, even cryptographic checksums, for verification of integrity. Backups or redundancies must be available to restore the affected data to its correct state.

Availability

Availability is best ensured by rigorously maintaining all hardware performing hardware repairs immediately when needed and maintaining a correctly functioning operating system environment that is free of software conflicts. It’s also important to keep current with all necessary system upgrades. Providing adequate communication bandwith and preventing the occurrence of problems are equally important.

Executive Summary

The Penetration test was done in order to determine how secure and how reliable data is on the data api https://api.healthcareegy.com/swagger/ui/index. The api system which was hosted on amazon aws.A simple scan on the site shows that the site runs of no WAF(Web Application Firewall) as discovered using ***wafw00f.***



Summary of Results

Based on my finding, I am quite sure that the api’s data is Confidential, Its integrity is witheld and availability guaranteed. Nonrepudiation is guaranteed though authentication using the api keys thus making sure data is available to the right person who has been granted access to.

Attack Details

In order to determine if the api was secure, we carried out a series of attacks mostly on the Open Web Applications Security Project top 10 vulnerabilities which are:

- Injection
- Broken Authentication
- Sensitive data exposure
- XML External Entities
- Broken Access Control
- Security Misconfigurations
- Cross Site Scripting
- Insecure Desirialization
- Using Components with known vulnerabilities
- Insufficient Logging and Monitoring.

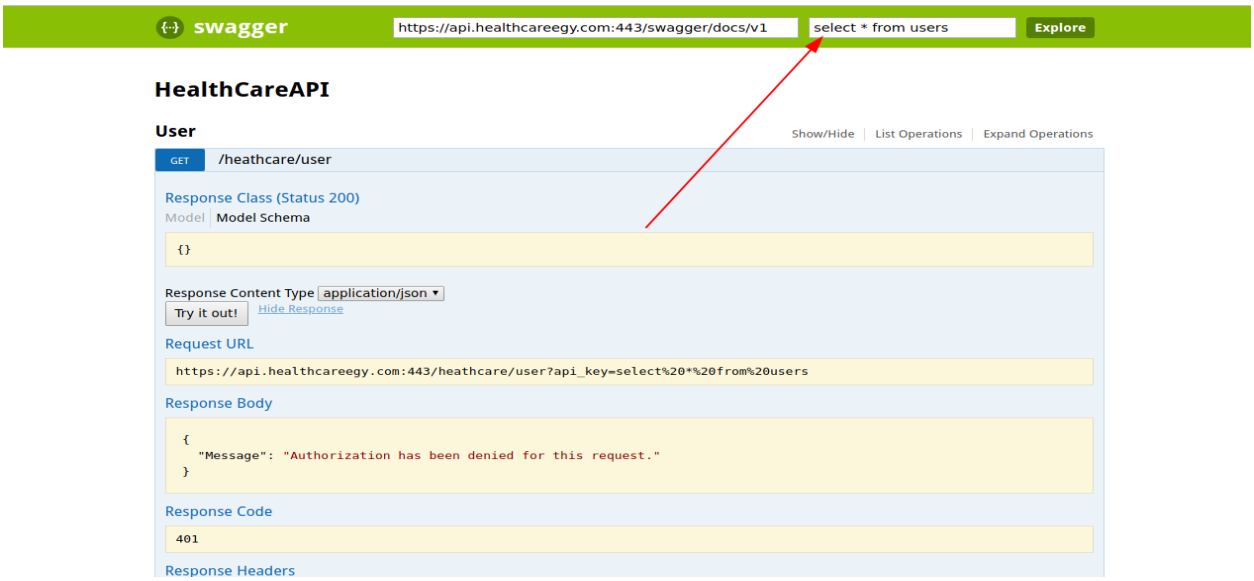
The penetration testing tools used was mainly burpSuite professional edition which is mostly used for web applications penetration testing. In some instances, I used Sqlmap to perform injection attacks.

Injection

Injection attacks happen when untrusted data is sent to a code interpreter through a form input or some other data submission to a web application.It poisons dynamic SQL statements to comment out certain parts of the statement or appending a condition that will always be true. It takes advantage of the design flaws in poorly designed web applications to exploit SQL statements to execute malicious SQL code.

The Api system responds negatively to simple sql queries posted and thus this shows that data can't be easily manipulated using such technique. Thus the integrity of data holds.

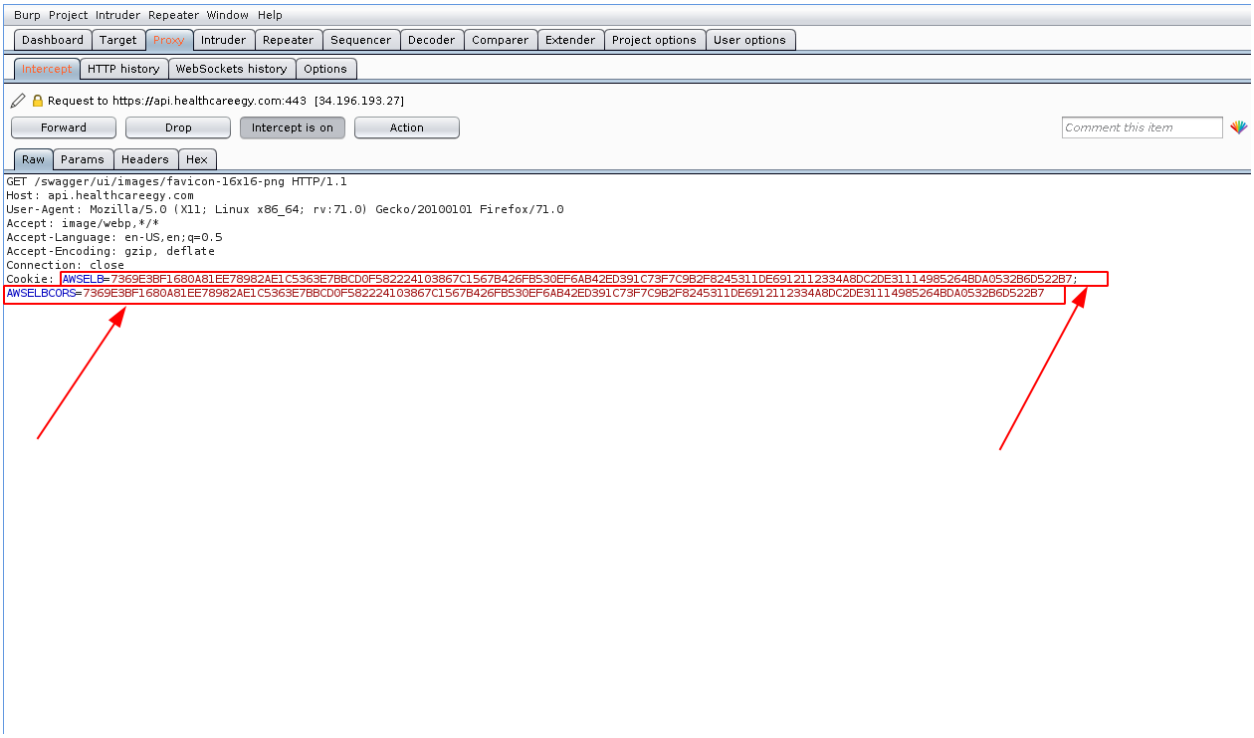
I used a query *Select * from users,* and the system responded negatively. The query is placed in the form field where one is to input the api_key as seen below.



After placing that, then the system is unable to execute that and gives the response demonstrated below in the screenshot below with the message *Authorisation has been denied for the request* meaning the server was unable to execute the query.

Broken Authentication

When authentication functions related to the application are not implemented correctly, it allows hackers to compromise passwords or session ID's or to exploit other implementation flaws using other users credentials. Since the Cookie values in the api are random and is alphanumeric, the api seems safe from broken authentication attacks and all bruteforce techniques.



Broken Authentication allow an attacker to either capture or bypass the authentication methods that are used by a web application.

- Permits automated attacks such as credential stuffing, where the attacker has a list of valid usernames and passwords.
- Permits brute force or other automated attacks.

- Permits default, weak, or well-known passwords, such as "Password1" or "admin/admin“.
- Uses weak or ineffective credential recovery and forgot-password processes, such as "knowledge-based answers", which cannot be made safe.
- Uses plain text, encrypted, or weakly hashed passwords.
- Has missing or ineffective multi-factor authentication.
- Exposes Session IDs in the URL (e.g., URL rewriting).
- Does not rotate Session IDs after successful login.
- Does not properly invalidate Session IDs. User sessions or authentication tokens (particularly single sign-on (SSO) tokens) aren't properly invalidated during logout or a period of inactivity

Sensitive Data Exposure

Sensitive Data Exposure vulnerabilities can occur when a web application does not adequately protect sensitive information from being disclosed to attackers. This can include information such as credit card data, medical history, session tokens, or other authentication credentials.Sensitive Data Exposure occurs when an application does not adequately protect sensitive information. The data can vary and anything from passwords, session tokens, credit card data to private health data and more can be exposed.

A few examples would be exposed data that someone mistakenly uploaded somewhere, weak crypto that means an attacker would be able to read the data if they successfully compromised the target and the lack of headers that prevent browser caching. In short, every possible way where it would have been possible to better protect the sensitive data.When building an application, many are going to down-prioritise protection of sensitive data, and even if the developer is aware of the fact that they should.It is often said that the most common flaw is failing to encrypt data. One example of this vulnerability is the cleartext submission of a password. This is mostly due to lack of **T**ransport **L**ayer **S**ecurty(TLS) certificate. Since the Api is secured its transport layer with ssl encryption, its a guarantee that data transmitted over the network is safe from man in the middle attacks.Thus Sensitive Data Exposure wont work against the API.

XML External Entities

This is an attack against a web application that parses XML* input. This input can reference an external entity, attempting to exploit a vulnerability in the parser. An ‘external entity’ in this context refers to a storage unit, such as a hard drive. An XML parser can be duped into sending data to an unauthorized external entity, which can pass sensitive data directly to an attacker.

The best ways to prevent XEE attacks are to have web applications accept a less complex type of data, such as JSON**, or at the very least to patch XML parsers and disable the use of external entities in an XML application. The Api accepts data input in json format making it open to XEE attacks. However,XXE attacks are mostly based on objects that are not properly serialized. The input is properly screened for malicious codes and some special symbols escaped. Thus, these measures makes out the api safe from xxe.

Broken Access Control

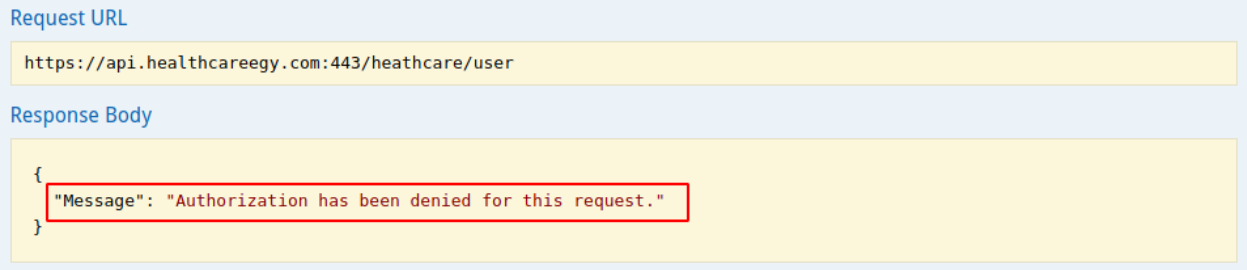
Access control refers a system that controls access to information or functionality. Broken access controls allow attackers to bypass authorization and perform tasks as though they were privileged users such as administrators. For example a web application could allow a user to change which account they are logged in as simply by changing part of a url, without any other verification.

Access controls can be secured by ensuring that a web application uses authorization tokens* and sets tight controls on them.The api is safe since one only requires an Api key to perform operations.

Security Misconfigurations

Security misconfiguration is the most common vulnerability on the list, and is often the result of using default configurations or displaying excessively verbose errors. For instance, an application could show a user overly-descriptive errors which may reveal vulnerabilities in the application. This can be mitigated by removing any unused features in the code and ensuring that error messages are more general.

In general, the errors the system displays are short and precise. This makes the api proof to this vulnerability.

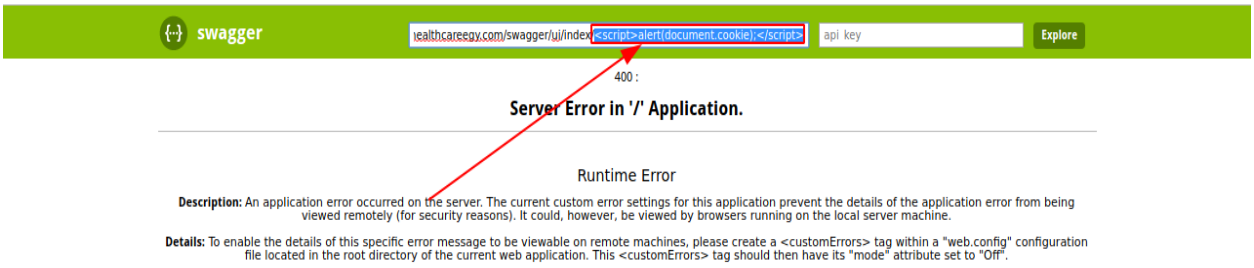


Cross Site Scripting

XSS vulnerabilities arise when data is copied from a request and echoed in to the application’s immediate response in an unsafe way. An attacker can use the vulnerability to construct a request which, if issued by another application user, will cause JavaScript code supplied by the attacker to execute within the user's browser in the context of that user's session with the application. The attacker-supplied code can perform a wide variety of actions, such as stealing the victim's session token or

login credentials, performing arbitrary actions on the victim's behalf, and logging their keystrokes.

The site is vulnerable to cross site scripting. After Adding the javascript code to the api url *<script>alert(document.cookie);</script>* . The url was then *https://api.healthcareegy.com/swagger/ui/index/<script>alert(document.cookie);</script>*



I received a response and the site executed the javascipt code and gave the following feedback. The site also displayed the error show in the screenshot below.However, the api is safe from stored XSS

Insecure Desirialization

This threat targets the many web applications which frequently serialize and deserialize data. Serialization means taking objects from the application code and converting them into a format that can be used for another purpose, such as storing the data to disk or streaming it. Deserialization is just the opposite: converting serialized data back into objects the application can use.Serialization is sort of like packing furniture away into boxes before a move, and deserialization is like unpacking the boxes and assembling the furniture after the move. An insecure deserialization attack is like having the movers tamper with the contents of the boxes before they are unpacked.The Api is safe from this since the input of the api key is properly serialized and escaped. The output is also escaped.

Using Components With Known Vulnerabilities

Many modern web developers use components such as libraries and frameworks in their web applications. These components are pieces of software that help developers avoid redundant work and provide needed functionality; common example include front-end frameworks like React and smaller libraries that used to add share icons or a/b testing. Some attackers look for vulnerabilities in these components which they can then use to orchestrate attacks. Some of the more popular components are used on hundreds of thousands of websites; an attacker finding a security hole in one of these components could leave hundreds of thousands of sites vulnerable to exploit For example, Using ***Microsoft-IIS/10.0*** makes one’s open to the following vulnerabilities.

However, once after installing updates, the vulnerabilities will have already be patched. I would recommend your to be updating the server if possible weekly.

Insufficient Logging and Monitoring

Many web applications are not taking enough steps to detect data breaches. The average discovery time for a breach is around 200 days after it has happened. This gives attackers a lot of time to cause damage before there is any response. OWASP recommends that web developers should implement logging and monitoring as well as incident response plans to ensure that they are made aware of attacks on their applications.

Conclusion

From the tests done, the Confidentiality of data, integrity and availability is guaranteed. However, with the vulnerability found,this maybe undone. I would like you to follow the recommendations on how to prevent such.Based on the system, nonrepudiation was achieved through the use of:

- ✓ SSL or TLS authentication making sure that the data is sent and retrieved from the correct server.
- ✓ Digital Signatures which are used to introduce the qualities of uniqueness which are signed digitally by a trusted certificate Authority.

We also had Confidentiality,Integrity and Availability achieved by the api in the following ways:

- ✓ Confidentiality was enhanced by the use of the api key making sure that authorised users with an api key gets access to the data.
- ✓ The penetration test was mainly done to determine the integrity of data. Since no data was modified during the pentest, the injections attacks never worked, then this assured data integrity as no data was altered during the test.
- ✓ With the api key, one is able to access the data. If the user’s api key is invalid, then the system would alert the user that there was a n error in authentication. This makes the information available to the right users thus allowing availability

Recommendations

- ✓ From the reflected XSS vulnerability, please make sure that the input into the system is thoroughly filtered and any special character escaped. An example, if a character is to be escaped a case one input a < character, the system should escape it into ***< &*** for **&** and others.

  Otherwise, the system is 99% hackproof. Good Luck

Risk

The overall risk of the vulnerability found is rated Low.