Security Report on Global Health System Application

Name: Trupti Kulkarni(x15012948)
Name: Apurv Deshpande(x15001687)

***Abstract*—the Report is aimed at providing security to Global Health System Application .The Application is about that patients can make online appointments with doctors irrespective of their locations and also all the patient reports are saved in the cloud so that they can access their reports anytime at any place.**

Keywords—Patient; Doctor; Appoitnment; Reports

I. INTRODUCTION

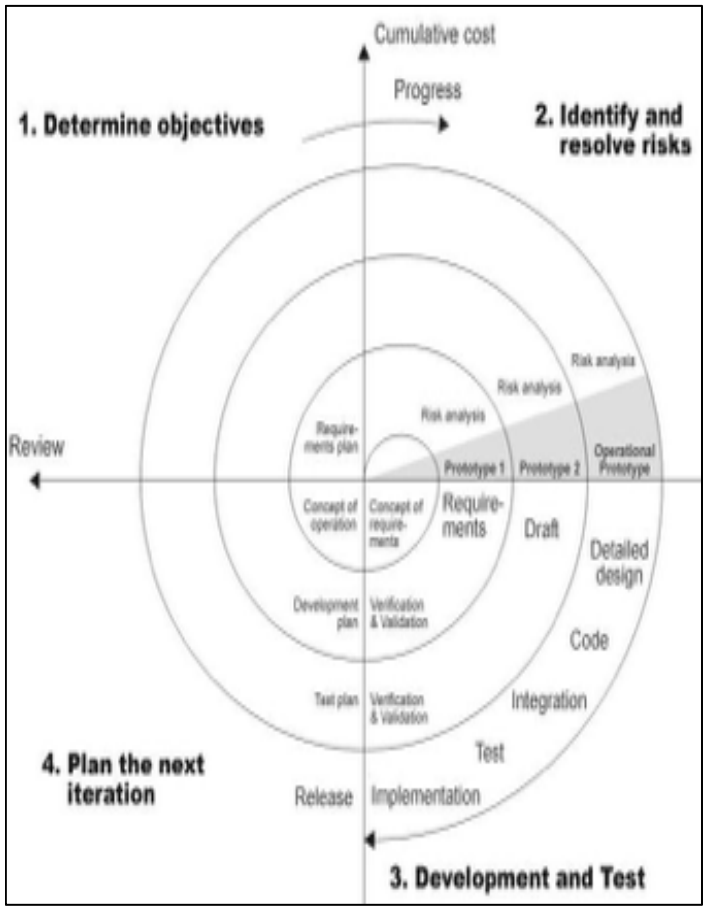
The project is about developing cloud-based solution to facilitate easy appointment system for users. Also this Application can be used to store information about patient their profile and reports and also to maintain profile of doctors. Hence as this application contains sensitive data it is important to provide security for the application.

II. S-SDLC METHODOLOGY

The Security Development Lifecycle (SDL) consists of seven phases: training, requirements, design implementation, verification, release and response. To implement this we use spiral model in our project.

SPIRAL MODEL: Steps of Implementation

1. All the security requirements for our application were defined in details considering aspects of existing application.
2. Using the design of application we created a prototype of security.
3. Taking into consideration the strengths, weakness, risks and defining the additional requirements we created the second prototype.
4. Then we tested the second prototype and iterated it to get the desired security for the application.
5. Based on the refined prototype, the final system was constructed, tested and evaluated.



III. SELECTION OF TOOLS, METHADOLOGIES, FRAMEWORKS AND BENCHMARKING

The possible threats that have been taken into consideration and the related precautions taken to protect the system are listed further below:
Ruby on rails provide certain build in functionalities to harden the application against some critical well-known security attacks. To enable the security feature some configurationally changes needs to be done to make the application more secure.
To protect Global Health Application from some known attacks following are the configurations done using in built functionality of Ruby on Rails ::

- **SQL Injection:**
  SQL injection is the technique of manipulating the database of the system in an unintended fashion. SQL injection's outcome is a vulnerable system that ranges from data leaks, direct access to database,
  Preventing the system from SQL Injection
  Rails provide the build in measure to avoid SQL injection by inheriting "Active Record" which is by a very convenient and safer ORM abstraction layer that Rails framework have provided. ORM layer is designed in such a way to help developer to code easily and in more secure manner. ORM facilitates developers to avoid manually build SQL queries by incorporating the best practices and avoid direct user input handling. In turn which avoid the basic idea of SQL injection.
  In Global Health System, we have used this functionality by inheriting "Active Record".

- **Code Injection :**
  Code Injection is quite similar to SQL injection. However here instead of attacking the data directly attacker tries to manipulate the code or the script written to run the application. The attacker tries to embed the untrusted content in the database and eventually it comes to HTML then CSS and then Javascipts written to run the application. This injected code is runs under the same privileges as that of the developer's which is extremely hazardous to the system.

  In Ruby on rails, such scenarios can be much avoided as in "Views" the attributes of the Model are accessed in following syntax:
  :
  Doctor ::="<%= Doctor.name %>">
  :

  The HTML output is ::
  :
  Data-Doctor="<script>Doctor.name</script>">
  :

- **Logging:**
  By Default, Rails tracks all the user interaction actions in the log file that it maintains internally. However, this can be very dangerous if log file captures all the data that user inputs in the system. In this case especially when user is entering sensitive information such as passwords and credit card numbers or PINs. In order to avoid such situations in Global Health Application, we have filtered such request parameters from the log files by making changes in application configuration file as mentioned below:
  config.filter_parameters << :password
  By making this configuration password would be marked as [FILTERED] in the log file.

- Cross-Site Request Forgery (CSRF)
  In Cross-site Request Forgery , victim is tricked into submitting a malicious request . By doing this, it identifies the privileges and rights that any user has got in the system which can be useful to make entry point in the system to do the undesired operations in the system on behalf of victim.
  Preventing Cross-Site Request Forgery (CSRF)
  Although enabled by default, you can double check that seeing if the protect_from_forgery method is within the main Application Controller enables it.
  Findings & Limitations:
  The interaction is more like a question (i.e., it is a safe operation such as a query, read operation, or lookup).
- Session Hijacking:
  For any "http:/" website, Session Hijacking is the most common attack in which the user logs into the website and any one can intercept their passwords in the clear text.
  Prevention to Session Hijacking:
  To avoid session Hijacking, we have used HTTP Secure on login page in order to encrypt the traffic between the end user (patient/doctor) and Global Health website.
  Rails provide build in functionality to avoid this issue by making following changes in application.rb
  config.force_ssl = true
  For setting SSL feature on feature on heroku :
  heroku addons:add ssl:endpoint
  heroku certs:add my_cerficate.crt site.key

- **Storing Data in Session**:

Session stores data in various ways. By default in Rails since version 2, Session uses cookies for storing the data. However there are few limitations to cookies as well as the size of cookies is limited to 4KB that means one cannot store much of the data in it. Also cookies are stored on client side, which implies that stored data can be viewed by anyone having access to them.

Rails include SHA512 technique to store session data on server data which prevents from anyone to tamper it however the stored data can be viewed. In conclusion it risky to store the sensitive information into session as by default you are exposing the data to the world.

Prevention measures followed to avoid this :

In Global Health application we have used Rails build in controller "session_migration" to store the session data in the database. Also we have done changes as follows in config/initializers/session_store.rb :

Security:Application.config.session_store :active_record_store

External Gem used to maintain the security:

- **Devise :**

Devise is the gem used basically to achieve secure authentication to any Ruby application. Devise is complete CMV solution which includes different 10 different modules namely Database Authentication, Traceable, Confirmable, Timeout,Locable,Validatable.

Installation steps ::

$ rails generate devise:install
$ rails generate devise User

After performing the above commands, migrate the database. As there was need to customize the Devise views and controllers in Global Health Application , we have installed the same and customize them accordingly to achieve our functionality.

IV.Technical Testing Approach

It is not advisable to rely on built in functionality of Ruby on Rails to protect the applications from different threats and vulnerabilities. So in order to prevent Global Health system from different security attacks we have used few of the external gems and done some configurational changes in application so as to harden the system.

- **Bundler audit:**

Bundler audit is the tool that checks for any vulnerability in the Gemfiles. Basic intention to use this gem is that we use gem, which is developed by others. Due to this we cannot always rely on other developers code when it comes to security. So there is a natural lag between security best practices implemented by the other developer to access the database.

Following are the steps to install bundler audit:

1. gem install bundler-audit

Test result:

No vulnerabilities found

- **Brakeman:**

Brakeman is the tool that scans your application and checks for the common security vulnerabilities that your application is holding. We have used Brakeman in Global Health System, as it is a static code analyzer to find the problems and the possible security threats before deploying the application on Heroku cloud.

During the process of development, as it is possible for the properly running application into the vulnerable condition we have done brakeman scan preferably prior to each time we commit or push our application to the Heroku cloud.

Following are the steps to install brakeman:

1. gem install brakeman
2. Include the following snippet into gemfile:

```
group: development do
  gem 'brakeman', :require => false
end
```

Test results:

Summary:

Scanned/Reported	**Total**
Controllers	11
Models	5
Templates	41
Errors	0
Security Warnings	8 (5)

Warning Type	**Total**

Attribute Restriction	5
Mass Assignment	3

+SECURITY WARNINGS+

Confidence	Class	Method	Warning Type	Message
Weak	AppointmentsController	create	Mass Assignment	Unprotected mass assignment near line 46: Appointment.new(appointment_params)
Weak	DoctorsController	create	Mass Assignment	Unprotected mass assignment near line 33: Doctor.new(doctor_params)
Weak	PatientsController	create	Mass Assignment	Unprotected mass assignment near line 28: Patient.new(patient_params)

Benchmarking: Benchmarking is tool to find out the performance of a system, identify the bottlenecks and to improve the algorithm of the system. Benchmarking can be used in our application to find about the performance. If the application is running slow are there any security compromises in our system.

V. FINDINGS AND RISK RATING

After embedding the security, It has been found that most vulnerable and high risk file is the gemfile. There has been a lot of external gems used in application and it is easy to create a malicious gem and install in the application as users don't look at source from where the gem is provided. Also a YAML file is a huge risk file in our application. YAML files holds the data of the gem so there are huge risks of security associated with these files. In such cases , attacker sends specially designed XML request to the application which in turn is contains encoded YAML object. Now, depending on the type and the structure of the injected Ruby object namely threat agents, Exploitability, Prevalence , Detectability , Technical Impacts , Business Impacts. In case of Technical Impacts and Business Impacts severity is very more as the critical aspects like Ruby code will get maliciously attacked.

The Risk associated in Rails Framework is Denial Of Service. The attack surface on rails is in routes file as it is redirects path between the controller and the views. There is a risk in rails framework because binding activities involve addresses like (0.0.0.0) which means all interfaces are allowed while it should only allow local interface only (starting with 127.0.0.0). Hence there is a huge risk of IP spoofing attack. Risks are also attached while it is directing to a particular file or URL. Also there is big threat that patient data should not fall into the wrong hands. The data is not encrypted hence it is huge risk that data may fall into the wrong hands. Ruby on Rails generates a lot of files. Hence if there is a single malicious file it is very difficult to find that file. Hence there is risk attached with automatically generated ruby on rails files. The risk associated in Ruby is with XML. There is risk associated with sharing of data in our application.

VI.CHALLENGES AND LIMITATIONS

The mail send to users after there registration is vulnerable as there is no encryption technique used in sending those mails. This area needs to be focused, as we don't know whether the particular mail has reached the user successfully. The application is vulnerable to Cross Site Scripting Attack, this attack will have partial influences as the rails have inbuilt security for XSS attack. The huge challenge for the application is to survive against the HTML file attacks and YAML file attack. As we are using external gem in order to develop the application which is in turn developed by third party developer . It is hard to rely completely on external gem to ensure security . Even in order to ensure security check in Global health application , we are using gems like "bundler-audit" which is also third party gem. Hence Security for the external gems used in the application is unknown.

Also we are deploying the application on Heroku Platform, so security at PaaS level is only the security provided by Heroku. Also as the technology advances there are will be new types of attacks, so application is vulnerable to such attacks. Users of the application are responsible for end security. They are responsible for their browser and device security. Also it is user responsibility to keep their password secret. There is no control over the traffic in shared deployment platform. In case of successful security attack there is no particular incident response plan decided. There is no intrusion detection system involved in our application. There are certain flaws in XML parser in ruby on rails.

VII.Conclusion And Findings

The application security has been tested and implemented successfully taking into consideration all the basic security concerns. There is always a risk in deploying cloud application in a shared platform because not many security features can be implemented in shared platform. Administrator should keep an eye on all the users that are using the application. The end users should have anti-virus software installed. The source code should be reviewed. Both manual and automated testing should be used for reviewing. There is no such thing as complete security as the new technology emerges in web applications there will be new security challenges. Application is most vulnerable to Denial Of Service (DOS) attack and Cross Scripting Attack (XSS) and YAML file attack. The Rails framework is complicated to include all the security features. By finding out the weakest link of our application we can find about the point of attack in our application. It is very difficult to provide security considering the privacy and confidentiality of medical data practices.

References:

1) Turner, S. 2014, "Security vulnerabilities of the top ten programming languages: C, Java, C++, Objective-C, C#, PHP, Visual Basic, Python, Perl, and Ruby", *Journal of Technology Research,* vol. 5, pp. 1.
2) http://www.cs.umd.edu/~avik/papers/ssarorwa.pdf
3) https://www.sans.org/reading-room/whitepapers/analyst/health-care-cyberthreat-report-widespread-compromises-detected-compliance-nightmare-horizon-34735
4) https://www.netsparker.com/blog/web-security/ruby-on-rails-security-basics/
5) http://www.sitepoint.com/techniques-to-secure-your-website-with-ruby-on-rails-part-1/
6) http://www.sitepoint.com/devise-authentication-in-depth/
7) http://blog.codeclimate.com/blog/2013/01/10/rails-remote-code-execution-vulnerability-explained/
8) https://blog.sucuri.net/2014/09/quick-analysis-of-a-ddos-attack-using-ssdp.html