Abstract

PatientBank, a platform for individuals to aggregate their own health information, deals with vast amounts of sensitive, personal information. Dealing with such information raises well-deserved concerns regarding the security of the system. Hence, it is of utmost significance to maintain high standards for the security of PatientBank, while preserving the performance of database read and write operations. Health Insurance Portability and Accountability Act (HIPAA) provides a framework for the security of systems that deal with such sensitive information. This paper analyzes HIPAA security and privacy requirements and elucidates how PatientBank meets them. Additionally, the paper provides a clear outlook on good security practices and as future opportunities, explores formal verification methods related to HIPAA compliance.

1 Background

1.1 What is PatientBank?

PatientBank is a platform for individuals to aggregate their own health information. Especially for high utilizers of health care, comprehensive health records are incredibly valuable. While electronic health records have become more prevalent, patients access to their own information remains limited. PatientBank solves this problem for the patient by requesting, gathering, and maintaining a comprehensive and unified record on his or her behalf.

1.2 Technology Stack

The current implementation of the project uses the following technologies:

- Ruby on Rails: API and backend framework
- PostgreSQL: Primary relational database
• Redis: In-memory database for storing events and notifications
• Solr: Indexing data for full-text search
• Angular.js: Frontend framework

1.3 What is HIPAA? What is defined as sensitive information (or PHI)?

HIPAA stands for Health Insurance Portability and Accountability Act. In the words of U.S. Department Health and Human Services (HHS), “[t]he primary goal of the law is to make it easier for people to keep health insurance, protect the confidentiality and security of healthcare information...” [Dreyzehner].

HIPAA describes PHI as “protected health information”, pieces of data that can be used in the identification of an individual. According to HHS, “PHI can relate to past, present or future physical or mental health of the individual. PHI describes a disease, diagnosis, procedure, prognosis, or condition of the individual and can exist in any medium files, voice mail, email, fax, or verbal communications” [Dreyzehner].

Fundamentally, HIPAA requirements are relevant to all organizations/applications that maintain protected health information. Nevertheless, it is possible to reflect on the principles that HIPAA imposes and use them as the basis of reasonable security practices for any application with sensitive information (i.e. Credit card, bank accounts etc.).

2 Method and Implementation

HIPAA mandates three main requirements for compliance and protection of sensitive patient information [HHS, 2003]:

• Sensitive information (PHI or other forms of healthcare information) needs to be stored on HIPAA-compliant servers. The data needs to be fully reproducible at all times and precautions need to be taken against potential data loss due to corruption.

• The application needs to be equipped with logging and auditing capabilities that would allow the extraction of accurate information regarding access to sensitive data.

• Sensitive data needs to be encrypted with at least 128-bit encryption both at rest and in transit.
This section will explain how PatientBank fulfills these requirements and focus on the last two. This section will also shed light on good security practices that are applicable on a broader scale than what HIPAA mandates.

2.1 HIPAA-compliant Storage

This requirement is one of the easiest to fulfill. There are many readily available services that provide access to HIPAA-compliant servers. PatientBank utilizes Amazon Web Services servers with EC2 to securely store sensitive information. Amazon provides basic capabilities of data recovery, availability as well as server-side data encryption, if needed [Amazon Web Services, 2012].

2.2 Logging and Auditing

Logging and auditing capabilities obviously are trickier than HIPAA-compliant storage. Amazon provides a very rudimentary logging service that gives detailed information about deployment and run-time incidents. In the past, we have faced challenges regarding concurrency, which in turn, caused our database connections to experience performance issues. Similarly, some of the Rails migrations that we have implemented to modify our database schema caused deployment problems. With the correct utilization of Amazon services, these issues are easy to identify and fix. However, this form of logging is insufficient, since it gives limited insight into access to sensitive data.

Fortunately, Rails comes with built-in logging and auditing capabilities. Specifically in Debug mode, Rails logging engine produces very detailed logs of database operations. Unfortunately, the density and abundance of information Rails logs make the task of information mining very challenging. Thus, for HIPAA compliance and eventually a better customer experience, PatientBank combines Rails logging and auditing capabilities with a custom Events data model. The data model keeps track of the documents that have been generated/updated/uploaded in any of our portals, as well as the users (patient, admin or provider) who have seen the documents. On the security and HIPAA-compliance ends, this provides the necessary means to produce very human-readable logs. The data model was originally developed in Spring 2014 during CPSC 439: Software Engineering. The team made modifications to the model as needed to make it more flexible and to accommodate the needs of a growing need for better logging capabilities.

2.3 Encryption

The process of achieving at rest and in transit encryption required a lot of thinking and research. For at rest encryption, the process was admittedly
simpler. By utilizing the *attr_encrypted* Ruby gem, it was straightforward to implement secure at rest encryption that combines symmetric and asymmetric encryption mechanisms [Faulkner and Huber].

Nevertheless, for in transit encryption, the options were varied and we had to put the necessary effort to make the best decision. As known, SSL is one of the most common and tested ways of achieving in transit encryption. The team of security experts behind the creation of SSL went through numerous iterations to create this sophisticated security protocol. Yet, the PatientBank team explored other options that helped us better explore the underlying magic of SSL. By trying to replicate some of the functionality that SSL provides, we got insight into encryption key exchange problems, learned about the complexity of server-side and client-side authentication and discovered numerous ways of system vulnerabilities, including but not limited to man-in-the-middle, replay and reflection attacks. In the face of such challenges, the PatientBank team decided to use SSL for in transit encryption.

### 2.4 Adequacy of HIPAA Requirements

With HIPAA-compliant server storage and basic encryption techniques, HIPAA requirements lay a strong foundation for ensuring the privacy of sensitive information. Nonetheless, basic requirements set forth by HIPAA might not be sufficient to cover all vulnerabilities. HIPAA’s emphasis on logging/auditing capabilities provides the necessary tools to identify individuals (patient, admin or provider) who access sensitive information, as well as to detect potential data breaches. Yet, it also shows that HIPAA does not take a very strong preventative approach to security of sensitive information.

At PatientBank, we actively endeavor to take a different stance on this issue. While basic encryption provides a foundation for security, we try to create the right tools to allow greater control over more granular access to information. One of these important tools is our *restrict_access* method that we use as a helper to our API controller methods. Essentially, this method let’s us limit access to information in a very granular way. The individuals who has the right privileges to have access to specific types of information are specified in the method via *options*. The method checks these options against the *current_user* and returns 403: *Forbidden* if there is not a match. As you can see in the code sample below, the acquisition of new information via the *params_hash* throughout the controller, let’s us filter access to right individuals as we step through the code.
def restrict_access(options)
    @api_key = APIKey.find_by(access_token: token)
    @current_user = @api_key.try(:user)
    # raise PermissionDeniedError if authentication resulted in a render
    raise Exceptions::PermissionDeniedError if performed?
    return true if options.empty?
    classes = options.select { |option| option.is_a? Class }
    if options.include?(@current_user) ||
        classes.any? { |c| @current_user.is_a? c }
        return true
    raise Exceptions::PermissionDeniedError
end

def handle_permission_denied
    unless performed?
        render_errors "Access denied.", :forbidden
    end
end

def update
restrict_access Admin, .grantee, .patient
update_params = .params
if approval_params.any?
    if .authenticate(approval_params[:approval_code])
        restrict_access Admin, .approver, .creator
    else
        restrict_access Admin, .approver
    end
update_params.merge(approval_params)
end

if render_json
    .update_attributes(update_params)
else
    render_errors .errors.full_messages, :unprocessable_entity
end

Figure 1: Implementation of the restrict_access method

Figure 2: Use of restrict_access in one of our API controller methods. Some fields are blocked for security reasons.
2.5 Testing

Since the inception of PatientBank in CPSC 439: Software Engineering, both backend and frontend testing has been a priority for our team. Our backend testing routine built with RSpec, a testing framework for Rails applications, extensively exercises our API controllers and model verifications. Although it is difficult to perform penetration tests without specialized tools, our testing suite provides convenient mechanisms for ensuring the validity of our main restrict_access method. On top of in transit and at rest encryption, this level of testing provides an additional layer of insurance. The results of these tests and any security flaws that are found are collected in a monthly risk analysis report.

In addition to the backend tests, the frontend tests combined with manual penetration tests and static tests provide comprehensive feedback about security, design and implementation flaws in our system. As revealed by the comprehensive survey and analysis of [Austin and Williams, 2011], one methodology of testing is not sufficient to reveal security or implementation problems. Thus, we try to take a more holistic approach to testing to ensure the security of the application and the data that resides in it.

3 Future Direction

At the beginning of the semester, one of the goals of this project was set to explore more academia oriented aspects of privacy, security and HIPAA. With Professor Piskac’s guidance and expertise, we endeavored to scrutinize formal verification methods to measure the compliance of the PatientBank platform, given HIPAA security and privacy rules as a set of constraints. Throughout the semester, we had several chances to see what can be done on that end. Yet, our lack of knowledge of formal verification methods made these efforts less fruitful than our more practical, implementation-oriented efforts. Thus, this part will explore two academic areas that can be pursued in the future both for PatientBank and in general for the set of applications that deal with sensitive information:

- Enforcing HIPAA privacy and security rules in applications through formal methods: This area is pioneered by Anupam Datta of Carnegie Mellon University. Datta et al. developed various techniques through different frameworks that they themselves delineated to enforce compliance with HIPAA requirements. In the paper, Formalizing and Enforcing Purpose Restrictions in Privacy Policies, Datta et al. describes a new method that relies on Markov Decision Processes. This method defines actions as “purposeful” if they serve medical needs and enforces privacy requirements by restricting redundant, “purposeless” actions [Tschantz, Datta and Wing, 2012].
• Interpreting privacy from a modern point of view and addressing the privacy needs of our society: This area approaches the issues of security and privacy from a broader scope than HIPAA privacy requirements do. Helen Nissenbaum of New York University made several contributions to this area by scrutinizing privacy policies and trying to shed light on discrepancies between theoretical privacy policies and their applications in the real world [Nissenbaum, 2004]. This clearly does not define a purely computer science oriented research field, but one that also engages fields of political science, law and philosophy.

Note: While these two areas are intriguing, there are many other security-focused research projects conducted by scholars around the world. While not directly linked to privacy rules, David Kotz of Dartmouth University explores new, distributed authorization and authentication schemes [Kazuhiro and Kotz, 2005].

4 Demo Opportunities

Codebase of PatientBank is extensive and spans 5 distinct repositories and it is difficult to extract security specific code snippets to present to the reader. However, if you would like to see a demo of the product, you can always contact me at feridun.celebi@yale.edu or the PatientBank team at info@patientbank.us.

5 Acknowledgements

I would like to take this opportunity to thank Ruzica Piskac for her endless support and patience. She was extremely generous with providing guidance throughout the semester. I also would like to thank the PatientBank team for their work and dedication.

6 References

• Amazon Web Services. Creating Healthcare Data Applications to Promote HIPAA and HITECH Compliance. August 2012.

• Austin, Andrew and Williams Laurie. One Technique is Not Enough: A Comparison of Vulnerability Discovery Techniques. International Symposium on Empirical Software Engineering and Measurement. 2011


