Security Assessment of Coastline Market Web Application

December 10, 2016

Abrar Musa, Matas Empakeris, Vincent Chan, Yves Chan

Team 7

Department of Electrical and Computer Engineering
University of British Columbia
Vancouver, Canada

abrar.musa.89@gmail.com (48915086), matas@alumni.ubc.ca (11112142), v.chan36@gmail.com (15327117), chanyves@gmail.com (74546094)

***Abstract* - This report intends to give a description of Coastline Market and analyze the security of the company's web application. Our group performed penetration testing using a variety of popular attacks found in today's security breaches: the directory traversal attack, a password dictionary attack, Man-in-the-Middle attack, parameter tampering, NoSQL injection and cross-site scripting. Our results varied, but ultimately produced compromised user accounts and unexpected effects from parameter tampering and payload modification through a Man-in-the-Middle attack. The vulnerabilities found from our analysis included unauthenticated API endpoints, weak password restrictions and missing parameter verification in forms. Concluding our analysis, we encourage Coastline Market to implement stricter login and password policies, utilize the HTTPS protocol, and move input validation to the server side.**

I. Introduction

Coastline Market is a service that "connect[s] restaurants and chefs with the most traceable, local and fresh seafood, direct from the source." [1] The company acts as a middleman between buyers and fishermen who are looking to sell their catch. Users of the service are given access to a dashboard where they can view their financial analytics, orders, supply chain, customers, and inventory.

As the internet continues to grow, so does the number of web applications. According to McAfee, research indicates that the number of attacks on internet-connected devices and programs continues to grow [2]. Like all web applications, Coastline Market is susceptible to threats and contains several vulnerabilities which we have analyzed in this report. For this reason, we made it our intent to find and report these possible exploits to the company, with the goal of helping protect their customers and financial assets.

Coastline Market's web application uses the MEAN stack which is a commonly used full-stack JavaScript framework.

Having completed the project, our security analysis revealed several flaws and provided insight into major problems found in web applications today such as weak passwords, lack of data obfuscation, and inadequate access control.

To test Coastline Market's web application, a variety of attacks were considered. These attacks included a password dictionary attack, a cross-site scripting (XSS) attack, a man-in-the middle attack to modify payloads, and front-end parameter tampering. Penetration testing succeeded for three of the seven utilized in our analysis: the password dictionary attack successfully revealed one-third of all plaintext passwords Coastline Market customers are using, payload modification allowed our test user to change the profile data of another customer, and tampering with parameters produced changes to information stored on the back-end.

Result from our tests indicated a lack of password restrictions and suggestions allowing for easy password dictionary attacks. Insecure API endpoints exposed sensitive user data such as customer emails and hashed passwords to unauthorized users Insecure verification and input sanitization allowed our test user to successfully tamper with payloads and modify users' personal information. In addition, a lack of back-end authentication and user form verification allows attackers to tamper with front-end code to be able to change user's information with the intention of making unauthorized changes to user data.

Based on the findings above, we recommended that Coastline Market updates their API endpoints to authenticate all requests and restrict access to sensitive user data. We also encouraged the company to implement stricter password guidelines to avoid being vulnerable to password dictionary attacks. In addition, API requests should be authenticated and encrypted to prevent payload modification through man-in-the-middle attacks.

The vulnerabilities mentioned in this report have been reported to Coastline Market, so that they can correct and patch their application to make it more reliable and secure.

II. Analyzed System

A. Technology Stack

Coastline Market uses the commonly-used MEAN stack, which is composed of MongoDB, Express, AngularJS and NodeJS. MongoDB is a NoSQL database which stores its objects as a BSON object. Express is a framework built on top of NodeJS and provides a robust set of features for web and mobile applications. AngularJS is a front-end framework used with JavaScript and supports two way data-binding and HTML templating. NodeJS is server-side JavaScript runtime environment built on the V8 engine.

B. Data Flow

1. Log In/Sign up

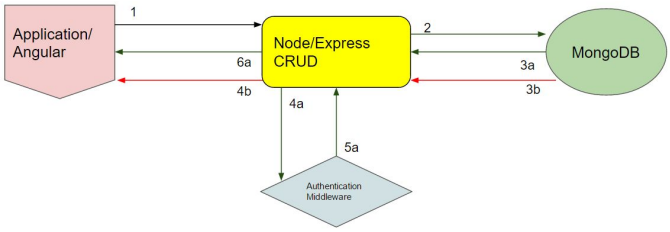

Fig 1. Log in Flow of Coastline Market

TABLE 1. Log In Flow Legend

Step	Application Instruction
1	Log in with username and password
2	Send username and encrypted password to MongoDB
3	a. User authenticated b. Reject user authentication
4	a. Generate JWT b. Return invalid input parameters to user
5	a. Give JWT to client
6	a. Store JWT in local storage

When a user first logs in, their username and password are sent to the server. During this process, the password is hashed with a known salt using the bcrypt algorithm and compared with a value that had been previously stored in the database. If the user credentials from the application match that from the database, then the user data is retrieved and passed onto the authentication middleware. From this point, the authentication middleware generates a JSON web token (JWT) associated with the user ID. It is then passed to the user so that it can be used for further authorization throughout the session. If, however, the credentials do not match, the user is prevented from accessing any other part of the dashboard or the rest of the application.

The signup process is less restrictive, and a new user will receive a JWT for the session as long as input fields of username, email, password, Paypal account, and company name are correctly inputted.

2. Application Usage

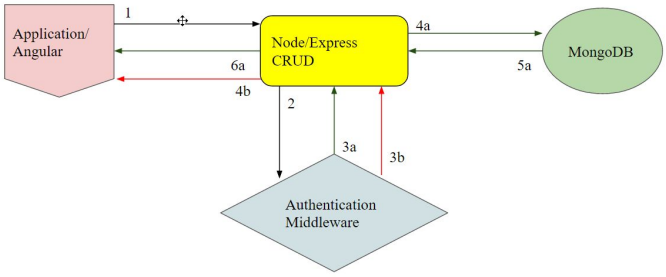

Fig. 2. User Flow of Coastline Market

TABLE 2. User Flow Legend

Step	Application Instruction
1	RESTFUL call. send JWT
2	Process JWT
3	a. Accept JWT b. Reject JWT
4	a. Request data from DB b. Return error to User
5	a. Process data from DB
6	a. Provide data to user

Figure 2 shows the user flow of Coastline market and how users obtain data. A user makes a REST call (GET, POST, PUT, or DELETE request) which gets processed by the server-side code containing the authentication middleware. At the same time, the server also gets a JWT. This JWT is composed of a header, containing information that states it is a JWT, a payload. a hash, and an encryption algorithm used for the hash. The payload contains the user ID, the timestamp indicating when the JWT was issued, and a JWT expiry timestamp. The header and payload is base64 encoded and hashed with HMAC using SHA-256 and a secret. When the authentication middleware receives the JWT, it decodes the JWT ensuring its integrity is maintained. If the web token expires or the user ID after decoding is not correct, the user is prevented from performing that REST call.

III. Related Work

AngularJS access control is not necessarily supported because it is a client-side framework and all the source code can be seen and inspected. There is nothing preventing users from tampering with the code to gain access to certain views or elements [3]. If real authorization is needed, then it should be done server-side, not with AngularJS [3].

Since the Coastline application uses MongoDB, a NoSQL database, certain vulnerabilities have been found in other independent analyses. NoSQL injections exist for MongoDB in providing bypass to login scenarios [4]. To mitigate NoSQL injections, AppSec suggests that libraries be used to support parameterized quotes and to use input validation through means of type validation, and regular expression checks.

IV. Methodology

A. System Analysis

1. Accessing unauthorized APIs

Figure 2 shows that the JWT is the only authorizing token that is used to process requests. Therefore, we investigated the possibility of manipulating and forging the token. We first investigated the structure of the JWT and discovered that the hash used in HMAC was SHA-256. The only identifying information was the user ID stored in the payload. Because the header and payload are encoded in base64, we decided to generate different secret keys to see if we could retrieve the same HMAC value. Our first approach was to research common libraries utilizing JWT in the MEAN stack. and confirmed our original assumption that the developers used a JWT library such as Passport.js, simple-jwt, or jsonwebtoken. We then proceeded to look up boilerplate code for setting up JWT in which the secret string was a generic message, such as "secret", "my_secret", "MY_SECRET", and gathered common variations of these for use in in our dictionary of the most common 100, 000 passwords. Finally, we generated many HMAC hashes to see if we could generate an identical hash to our client's. In addition, the dictionary words were encoded to base64, in case that type of encoding was used as the secret.

2. Cross-site scripting/Angular Template injection

To perform XSS attacks, we first scrutinized the JavaScript code available in the web pages to find missing input sanitization. Once candidates were found, we issued reflected XSS attacks on the input by setting the fields to JavaScript code wrapped in a <script> element.

Since AngularJS was used, we also attempted a similar attack called template injection. If framework-relevant attributes are not properly sanitized, then we can issue a template injection attack to override a native function located inside a template.

3. Directory traversal

To successfully attempt a directory traversal attack, we had to access directories meant to be hidden from unauthorized users. Typically, a web application is structured to have a directory tree no different from a typical Unix file structure found on Unix-based operating systems. With the advent of front-end and back-end web frameworks, however, web application directory structures might not be reflected in the url.

Knowing that the back-end framework used was Express, which runs on top of NodeJS, and the front-end framework was AngularJS, we were aware that a traditional directory traversal attack would most likely fail due to the way these frameworks organize URLs.

4. NoSQL Injection

Coastline Market uses MongoDB which is a NoSQL database and is possibly vulnerable to a NoSQL injection attack. We attempted a simple login form injection to try to gain unauthorized access to sensitive data. Using NoSQL specific syntax such as '{"$gt": ""},' meaning greater than, it is theoretically possible to force the middleware to perform a search for any user whose username were greater in value than an empty string (similar to using 'OR 1=1' in SQL injections). We tried to perform the same injection in the password field as well.

With success, this attack should allow an attacker to gain unauthorized access to a different user's dashboard or password, depending on the passed parameter.

5. Man-in-the-Middle and Modifying Payloads

Although the main coastlinemarket.com web address is protected with HTTPS, the dashboard subaddress is not. When a user logs in, the username and password are stored as plaintext in a JSON object but available in plaintext. For users on unprotected wireless networks, the packet containing username and password information would be sent during login and possibly exposed to eavesdroppers. Using Wireshark or another packet sniffer, an intruder would be able to retrieve both those fields and use this information to access the user's account. Therefore, a man-in-the-middle attack can be performed to collect information about the user and pass user authentication checks.

Furthermore, we analyzed the potential of intercepting API calls, investigating request headers, and changing parameters before they are sent to the server. There are many web debugging proxies available, such as Fiddler, that allow users to set breakpoints or pause POST/PUT requests. Using this type of program, the user input or payload data can be modified before it is sent to the server.

6. Parameter tampering

User fields can be validated on the client-side, meaning the browser is responsible for checking if user input is valid. Client-side validation is efficient because it does not require a roundtrip to the server. Unfortunately, client-side validation can be easily bypassed. If an input field is only validated via a CSS class, an attacker can remove the class ID from the HTML tag and manipulate the input field. Any kind of client-side sanitization or character escaping can also be bypassed using a web debugging proxy. Using the proxy, a request can be paused and the sanitized input changed before sending it back to the server.

7. Password Dictionary

As mentioned earlier in this report, during our initial inspection of the website, we discovered that the site relies on AngularJS to manage states, API calls and pages. This was an indicator that most of the data was likely held as state variables in the front-end of the application. Making a standard GET API call, we were able to get all user data from our test account.

The user data contained an important piece of information for our analysis: the hashed password of each user. Since we had already assumed that the web application used the MEAN stack, we decided to test whether bcrypt had been used to hash the passwords, as it is a popular cryptographic library used in the MEAN stack.

We obtained a list of 100,000 most commonly used passwords and tested it against the hashed passwords retrieved from the API response using a compare function available in the NodeJS bcrypt library. This method proved our hypothesis that the passwords had been hashed using bcrypt. We used the same procedure against other users' retrieved account data to find some of the passwords using our password hash cracker. [6]

B. Ethical Considerations

When conducting analysis on the Coastline system, there were some ethical considerations to follow. Any methodology conducted was to provide benefit to society or stakeholders while preventing any harm. The penetration testing methods were also disclosed to system owners prior. Any results obtained will respect individual privacy and confidentiality by ensuring information is not leaked or disclosed to unauthorized personnel. If something was unclear, then guidance and advice will be sought after to ensure potential harms are anticipated.

Any and all methodologies, exploits and results were first released to system owners to follow responsible disclosure. This gives owners an opportunity to patch vulnerabilities before details are released to the public. By publicizing the findings, it will help security analysts and other developers using the same technology stack to further investigate their own systems for vulnerabilities. Less vulnerabilities means less threats and risks to assets, which only benefits society.

C. Risk Management

By reaching an agreement with the system administrator and system owner, we were able to avoid much, if not all, of the risk associated with penetration testing. This agreement allowed us to test the system in various ways without being at risk for prosecution or legal action by the system owner, and each part of our security analysis was performed under complete permission of the company. With this being said, we were not allowed to issue a denial-of-service attack on the website, nor were we allowed to alter any user information already stored on Coastline Market's server.

V. Results

A. Failed Attacks

1. Accessing unauthorized APIs

When trying to gain access to unauthorized APIs, we tried to regenerate a valid JWT token by going through a dictionary of common passwords and common MEAN application boilerplate secrets. We were unable to resolve the secret used to generate the JWT used throughout the application for authorization.

2. *XSS (Cross-site Scripting) Attack*

We attempted to use a reflected XSS attack to glean information stored in the browser, specifically fields of the document cookie. While we were able to slightly change the expected behaviour of the web application by modifying JavaScript code, a traditional reflected XSS attack failed due to the built-in input sanitization AngularJS provides for developers. We found that when we attempted to inject a script tag into all the fields of Coastline Market's account settings page, the output was sanitized.

3. Directory Traversal Attack

Like we had hypothesized, the directory traversal attack failed. As mentioned in the *Methodologies* section, the Coastline Market web application is built using AngularJS and the Express back-end framework for NodeJS. Neither of these frameworks organize urls traditionally, and use routes instead to indicate the page the user is currently on. Because of this, attempting to access a server directory with a url not directly correlated to a file path rendered this attack useless.

B. Successful Attacks

1. Password Dictionary Attack

The password dictionary attack on the discovered password hashes compared against a list of 100,000 most common user passwords yielded a list of 27 user passwords. This in addition to observing the forms on the web application itself allows us to conclude that weak password restrictions and suggestions for stronger password use are not enforced.

2. Man-in-the-Middle Attack

Since the dashboard page was not secured through HTTPS, we were able to attempt a man in the middle attack. The open API calls gave us access to the entire user list, along with their user IDs, we are able to use this information to modify another user's existing data. We know that their technology stack uses MongoDB and MongoDB generates a random ID for every new document that is submitted. If the web application uses the generated ID to do user lookups and user updates, we can modify the payload data so that any updates we want, will be posted to any victim of our choosing. Once we changed the ID of the JSON data with Fiddler, we were able to update another user's account. Even without access to all the user IDs, an

attacker can simply generate many random IDs until a successful POST response is returned. The attacker may not know which account was modified, but that some account was modified successfully.

3. Parameter Tampering

During the man in the middle attack and editting another user's information, we noticed a field that was disabled for the user also propagated to the victim account. This lead us to believe that perhaps the field is simply disabled on the client side. Parameters like drop down menus could have new inputs inserted, along with being able to update disabled fields.

VI. Discussion

A. Interpretation of Results

The success of the payload modification and allowing users to update another user's information was largely due to the fact that all MongoDB generated IDs were obtained. A simple assumption that any updates to a user profile would be matched with the user ID.

Having access to the entire user list also made it possible for the dictionary password attack. The success rate of getting an account password is heavily dependent on how often people use the most common passwords.

Overall, by inadvertently allowing users see the entire user list lead to the two successful attacks, which can cause a lot of damage not only to the users of the system, but also the reputation of Coastline Market.

With respect to the 3 security principles of confidentiality, integrity and availability, we discovered vulnerabilities which violated the first two principles.

Confidentiality was reduced. This was due to the fact that some of the api endpoints were unauthorized giving anyone access to the entire list of customer emails and password hashes. Also poor password standards allowed us to obtain a fair number of user passwords through the password dictionary attack.

Integrity of user data was also compromised since we were able to modify payload data and change a different user's information using a man in the middle attack.

The third principle was availability was not affected since none of the attacks performed were able to successfully disrupt service of the web application itself. A denial of service attack may have possibly allowed us to overload the servers with requests causing them to crash but this is a rather trivial attack and was not within the scope of our particular analysis.

B. Principles of Designing the Secure System

Following our findings, we were able to determine three main security principles that were violated by Coastline Marketplace. The first violated principle is least privilege, which indicates that any user should be given the least amount of privilege to accomplish a task. The company's web application allows the general public to access their list of customers, which should only be seen by internal members of the team. Coastline Market customers and regular internet users should not be able to access the personal information of other customers.

The second main principle violated is fail-safe defaults. This privilege indicates that a system should not reveal any more sensitive information about itself when it fails than if it works as expected. In our analysis, we found that tampering with the JavaScript code and changing the states the application was supposed to be in at a given time resulted in unexpected behaviour that temporarily removed data stored related to our account. By definition, this indicated that when the system failed, it resulted in behaviour that was not only unexpected, but possibly dangerous as well.

The last main unfollowed security principle we had discussed was separation of duties. Given the type of information we were able to successfully retrieve from the web application, we were able to determine that this information should not be accessed by anyone outside an internal team, specifically customer service or development. However, similar to the incident discussed in least privileges, any unauthorized user had unobscured and unencrypted access to all customer data stored in the company's database. This finding indicated to us that proper precautions were not taken to explicitly separate unauthorized and authorized users from internal company employee roles, thus violating this principle.

VII. Recommendations

From our security analysis we proposed a couple of recommendations that would make Coastline Market more secure.

A. Proper use of HTTPS

Although the main website is protected with HTTPS, the dashboard component of Coastline Market is still using HTTP. We recommend using HTTPS on all webpages because then the users would not be able to modify payload data and change fields of other users as described earlier. HTTPS encrypts all traffic from user's web browsers to the server of Coastline Market.

Another danger of not using HTTPS is that confidential information such as usernames and passwords are sent over the network in plain text. This means that if a user in Coastline Market was logging in to the website from a public wifi network, an adversary may listen in on all the packets transferred over the local network and therefore retrieve the information from the Coastline Market user. This would compromise the user's account and would therefore tarnish their relationship with Coastline Market.

B. Protect API endpoints

From our analysis, we discovered that although most endpoints are protected with the authentication middleware, some of them are not. Unprotected endpoints is a huge

vulnerability in that it allows users to retrieve information from the application which is unnecessary and could be detrimental. One of the endpoints in particular; http://api.coastlinemarket.com/api/fisheries, provide a list of all the fisheries with the users information such as name, e-mail and hashed passwords. An adversary with this information could then do potential harm by exposing or selling user information to parties, or in our case by trying to match the password hashes by using a password dictionary attack to gain access into their Coastline Market accounts.

C. Strong Passwords

One of the vulnerabilities that allowed us to perform a dictionary attack on the users was that the passwords were too simple. By performing a common password dictionary attack, we were able to compromise the confidentiality of 33% of the users. Using a standard password scheme that follows the guidelines:

- 8 characters to 20 characters
- contain one number
- contain one lowercase
- contain one uppercase
- contain one symbol

would greatly improve the security of the user accounts.

D. Timeout after too many failed attempts

Ensuring customer data integrity is very important, and that is why we also recommend a timeout after failing to login with too many failed attempts. If an adversary had unlimited access to perform a dictionary attack on a user account by trying all combinations using a script, this has several consequences. The server hosting Coastline Market may slow down overtime because many POST requests when trying to test passwords in the dictionary attack accounts through the web interface. Another consequence is that the user account could potentially be compromised given the adversary has a certain amount of time to guess the password. By implementing a timeout, this prevents exhaustive search by expanding the time it takes for them to crack user passwords past their own life time.

Another way to prevent automated attacks against Coastline Market is the implement a CAPTCHA system in which only users should be detected and have access to.

E. Move user input validation to server side

There were a couple of forms on the dashboard which has client-side validation. For example, in the update settings page, users were able to tamper with the parameters to update fields they should not have access to. We recommend sanitizing all inputs on server side, as well as performing to appropriate checks so that all information that is allowed to be updated on the database is as expected.

F. Use RSA Encryption with JWT

Although we were unable to re-generate a valid JWT as we were unable to resolve the secret used to generate the JWT, we recommend using RSA encryption with a private key. By using a public and private key encryption scheme, this relies on a secure framework that is deemed more secure that the standard symmetric key framework that is currently used by HMAC-SHA-256. The JWT is the most important validator for Coastline Market and we believe by switching to RSA encryption, Coastline Market can have more peace of mind when it comes to the security of their customers and the application.

VIII. Conclusions

The integrity and confidentiality of Coastline's customer data is important to maintain a trusted relationship. Coastline uses the MEAN stack to build their web application which provides it's own security against the most common web attacks such as XSS and directory traversals. We found information leaks (entire user information) through other channels, which allowed for password dictionary attacks and also modifying another user's information using the hidden user ID. A total of 27 of 82 accounts were compromised and the potential for a malicious user to do a lot of damage with the user database. It is up to the developers to follow secure design principles like separation of duties, giving user accounts with the least privilege and fail-safe defaults. Following the guidelines and implementing the recommended protocols, it can prevent information leaks and account compromises.

References

[1] C. Market, "Coastline Market", Coastlinemarket.com, 2016. [Online]. Available: https://www.coastlinemarket.com/. [Accessed: 08- Nov- 2016].

[2] McAfee Labs, "2016 Threats Predictions", McAfee Labs, 2016.

[3] G. Heyes, "XSS without HTML: Client-Side Template Injection with AngularJS", Blog.portswigger.net, 2016. [Online]. Available: http://blog.portswigger.net/2016/01/xss-without-html-client-side-template.html. [Accessed: 14- Oct- 2016].

[4] "Techniques for authentication in AngularJS applications – Opinionated AngularJS", Medium, 2014. [Online]. Available: https://medium.com/opinionated-angularjs/techniques-for-authentication-in-angularjs-applications-7bbf0346acec#.fro925uea. [Accessed: 14- Oct- 2016].

[5] "AppSec Labs", AppSec Labs, 2016. [Online]. Available: https://www.owasp.org/images/e/e3/AppSecIL_2014_In-Secure_Mongo_And_Angular_Code_-_Israel_Chorzevski.PDF. [Accessed: 14- Oct- 2016

[6] https://gist.github.com/abrarmusa/b8fd595fabcebbf4ad2e25372092bf04

Appendix A: ACM Code of Conduct

PRINCIPLES

Principle 1: PUBLIC

Software engineers shall act consistently with the public interest. In particular, software engineers shall, as appropriate:

1.01. Accept full responsibility for their own work.

1.02. Moderate the interests of the software engineer, the employer, the client and the users with the public good.

1.03. Approve software only if they have a well-founded belief that it is safe, meets specifications, passes appropriate tests, and does not diminish quality of life, diminish privacy or harm the environment. The ultimate effect of the work should be to the public good.

1.04. Disclose to appropriate persons or authorities any actual or potential danger to the user, the public, or the environment, that they reasonably believe to be associated with software or related documents.

1.05. Cooperate in efforts to address matters of grave public concern caused by software, its installation, maintenance, support or documentation.

1.06. Be fair and avoid deception in all statements, particularly public ones, concerning software or related documents, methods and tools.

1.07. Consider issues of physical disabilities, allocation of resources, economic disadvantage and other factors that can diminish access to the benefits of software.

1.08. Be encouraged to volunteer professional skills to good causes and contribute to public education concerning the discipline.

Principle 2: CLIENT AND EMPLOYER

Software engineers shall act in a manner that is in the best interests of their client and employer, consistent with the public interest. In particular, software engineers shall, as appropriate:

2.01. Provide service in their areas of competence, being honest and forthright about any limitations of their experience and education.

2.02. Not knowingly use software that is obtained or retained either illegally or unethically.

2.03. Use the property of a client or employer only in ways properly authorized, and with the client's or employer's knowledge and consent.

2.04. Ensure that any document upon which they rely has been approved, when required, by someone authorized to approve it.

2.05. Keep private any confidential information gained in their professional work, where such confidentiality is consistent with the public interest and consistent with the law.

2.06. Identify, document, collect evidence and report to the client or the employer promptly if, in their opinion, a project is likely to fail, to prove too expensive, to violate intellectual property law, or otherwise to be problematic.

2.07. Identify, document, and report significant issues of social concern, of which they are aware, in software or related documents, to the employer or the client.

2.08. Accept no outside work detrimental to the work they perform for their primary employer.

2.09. Promote no interest adverse to their employer or client, unless a higher ethical concern is being compromised; in that case, inform the employer or another appropriate authority of the ethical concern.

Principle 3: PRODUCT

Software engineers shall ensure that their products and related modifications meet the highest professional standards possible. In particular, software engineers shall, as appropriate:

3.01. Strive for high quality, acceptable cost and a reasonable schedule, ensuring significant tradeoffs are clear to and accepted by the employer and the client, and are available for consideration by the user and the public.

3.02. Ensure proper and achievable goals and objectives for any project on which they work or propose.

3.03. Identify, define and address ethical, economic, cultural, legal and environmental issues related to work projects.

3.04. Ensure that they are qualified for any project on which they work or propose to work by an appropriate combination of education and training, and experience.

3.05. Ensure an appropriate method is used for any project on which they work or propose to work.

3.06. Work to follow professional standards, when available, that are most appropriate for the task at hand, departing from these only when ethically or technically justified.

3.07. Strive to fully understand the specifications for software on which they work.

3.08. Ensure that specifications for software on which they work have been well documented, satisfy the users' requirements and have the appropriate approvals.

3.09. Ensure realistic quantitative estimates of cost, scheduling, personnel, quality and outcomes on any project on which they work or propose to work and provide an uncertainty assessment of these estimates.

3.10. Ensure adequate testing, debugging, and review of software and related documents on which they work.

3.11. Ensure adequate documentation, including significant problems discovered and solutions adopted, for any project on which they work.

3.12. Work to develop software and related documents that respect the privacy of those who will be affected by that software.

3.13. Be careful to use only accurate data derived by ethical and lawful means, and use it only in ways properly authorized.

3.14. Maintain the integrity of data, being sensitive to outdated or flawed occurrences.

3.15 Treat all forms of software maintenance with the same professionalism as new development.

Principle 4: JUDGMENT

Software engineers shall maintain integrity and independence in their professional judgment. In particular, software engineers shall, as appropriate:

4.01. Temper all technical judgments by the need to support and maintain human values.

4.02 Only endorse documents either prepared under their supervision or within their areas of competence and with which they are in agreement.

4.03. Maintain professional objectivity with respect to any software or related documents they are asked to evaluate.

4.04. Not engage in deceptive financial practices such as bribery, double billing, or other improper financial practices.

4.05. Disclose to all concerned parties those conflicts of interest that cannot reasonably be avoided or escaped.

4.06. Refuse to participate, as members or advisors, in a private, governmental or professional body concerned with software related issues, in which they, their employers or their clients have undisclosed potential conflicts of interest.

Principle 5: MANAGEMENT

Software engineering managers and leaders shall subscribe to and promote an ethical approach to the management of software development and maintenance . In particular, those managing or leading software engineers shall, as appropriate:

5.01 Ensure good management for any project on which they work, including effective procedures for promotion of quality and reduction of risk.

5.02. Ensure that software engineers are informed of standards before being held to them.

5.03. Ensure that software engineers know the employer's policies and procedures for protecting passwords, files and information that is confidential to the employer or confidential to others.

5.04. Assign work only after taking into account appropriate contributions of education and experience tempered with a desire to further that education and experience.

5.05. Ensure realistic quantitative estimates of cost, scheduling, personnel, quality and outcomes on any project on which they work or propose to work, and provide an uncertainty assessment of these estimates.

5.06. Attract potential software engineers only by full and accurate description of the conditions of employment.

5.07. Offer fair and just remuneration.

5.08. Not unjustly prevent someone from taking a position for which that person is suitably qualified.

5.09. Ensure that there is a fair agreement concerning ownership of any software, processes, research, writing, or other intellectual property to which a software engineer has contributed.

5.10. Provide for due process in hearing charges of violation of an employer's policy or of this Code.

5.11. Not ask a software engineer to do anything inconsistent with this Code.

5.12. Not punish anyone for expressing ethical concerns about a project.

Principle 6: PROFESSION

Software engineers shall advance the integrity and reputation of the profession consistent with the public interest. In

particular, software engineers shall, as appropriate:

6.01. Help develop an organizational environment favorable to acting ethically.

6.02. Promote public knowledge of software engineering.

6.03. Extend software engineering knowledge by appropriate participation in professional organizations, meetings and publications.

6.04. Support, as members of a profession, other software engineers striving to follow this Code.

6.05. Not promote their own interest at the expense of the profession, client or employer.

6.06. Obey all laws governing their work, unless, in exceptional circumstances, such compliance is inconsistent with the public interest.

6.07. Be accurate in stating the characteristics of software on which they work, avoiding not only false claims but also claims that might reasonably be supposed to be speculative, vacuous, deceptive, misleading, or doubtful.

6.08. Take responsibility for detecting, correcting, and reporting errors in software and associated documents on which they work.

6.09. Ensure that clients, employers, and supervisors know of the software engineer's commitment to this Code of ethics, and the subsequent ramifications of such commitment.

6.10. Avoid associations with businesses and organizations which are in conflict with this code.

6.11. Recognize that violations of this Code are inconsistent with being a professional software engineer.

6.12. Express concerns to the people involved when significant violations of this Code are detected unless this is impossible, counterproductive, or dangerous.

6.13. Report significant violations of this Code to appropriate authorities when it is clear that consultation with people involved in these significant violations is impossible, counterproductive or dangerous.

Principle 7: COLLEAGUES

Software engineers shall be fair to and supportive of their colleagues. In particular, software engineers shall, as appropriate:

7.01. Encourage colleagues to adhere to this Code.

7.02. Assist colleagues in professional development.

7.03. Credit fully the work of others and refrain from taking undue credit.

7.04. Review the work of others in an objective, candid, and properly-documented way.

7.05. Give a fair hearing to the opinions, concerns, or complaints of a colleague.

7.06. Assist colleagues in being fully aware of current standard work practices including policies and procedures for protecting passwords, files and other confidential information, and security measures in general.

7.07. Not unfairly intervene in the career of any colleague; however, concern for the employer, the client or public interest may compel software engineers, in good faith, to question the competence of a colleague.

7.08. In situations outside of their own areas of competence, call upon the opinions of other professionals who have competence in that area.

Principle 8: SELF

Software engineers shall participate in lifelong learning regarding the practice of their profession and shall promote an ethical approach to the practice of the profession. In particular, software engineers shall continually endeavor to:

8.01. Further their knowledge of developments in the analysis, specification, design, development, maintenance and testing of software and related documents, together with the management of the development process.

8.02. Improve their ability to create safe, reliable, and useful quality software at reasonable cost and within a reasonable time.

8.03. Improve their ability to produce accurate, informative, and well-written documentation.

8.04. Improve their understanding of the software and related documents on which they work and of the environment in which they will be used.

8.05. Improve their knowledge of relevant standards and the law governing the software and related documents on which they work.

8.06 Improve their knowledge of this Code, its interpretation,

and its application to their work.

8.07 Not give unfair treatment to anyone because of any irrelevant prejudices.

8.08. Not influence others to undertake any action that involves a breach of this Code.

8.09. Recognize that personal violations of this Code are inconsistent with being a professional software engineer.

This Code was developed by the ACM/IEEE-CS joint task force on Software Engineering Ethics and Professional Practices (SEEPP):

Executive Committee: Donald Gotterbarn (Chair), Keith Miller and Simon Rogerson;

Members: Steve Barber, Peter Barnes, Ilene Burnstein, Michael Davis, Amr El-Kadi, N. Ben Fairweather, Milton Fulghum, N. Jayaram, Tom Jewett, Mark Kanko, Ernie Kallman, Duncan Langford, Joyce Currie Little, Ed Mechler, Manuel J. Norman, Douglas Phillips, Peter Ron Prinzivalli, Patrick Sullivan, John Weckert, Vivian Weil, S. Weisband and Laurie Honour Werth.

Appendix B: Responsible Disclosure

The findings of the project were reported to the owners of Coastline Market through a comprehensive final report. The owners will receive this report no later than December 2nd 2016. The owners names, emails, and phone numbers are Joseph Lee, joseph@coastline.com, and 7789196070. Due to geographical constraints, we are unable to schedule a face-to-face meeting with the system owner. However we will discuss our findings with the system owners through email and/or phone. The main objectives of this meeting should be (1) to debrief the system owner about the findings of the project, (2) to discuss the project team or the course professor can collaborate with the system owner on developing countermeasures for the discovered vulnerabilities, and (3) to determine the timeline for making the project findings public. We will send an e-mail message (CC-ed to Kosta) to the system owner, describing outcomes of the meeting and any resolutions/decisions/agreements made during the meeting.