Professor Carl Eyler

SDEV425 6381: Mitigating Software Vulnerabilities

Homework

5 March 2024

University of Maryland Global Campus

The Payment Card Industry Data Security Standard (PCI DSS) has 12 requirements for strengthening software security. Implementing them decreases the chances of a cyberattack, keeping the customers' data safe, and increasing their trust in the company. This document will analyze five PCI requirements that should looked for within the code provided for Homework 4 of the SDEV425 course. Each requirement will be described in some detail, along with the ways in which this requirement can be addressed by an application like the one provided.

Requirement 2: Do not use vendor-supplied defaults for system passwords and other security parameters.

The first issue that must be looked at in an application like this is whether it provides the user with a pre-defined set of credentials, and whether it allows them to create their own. According to PCI DSS, "Malicious individuals (external and internal to an entity) often use vendor default passwords and other vendor default settings to compromise systems. These passwords and settings are well known by hacker communities and are easily determined via public information (PCI DSS, 2015, p. 28, para.1). This happens because the default username and password are usually quite simplistic in their nature, and are only meant to be used temporarily, until the creation of new credentials by the user. This makes them extremely easy for hackers to bypass, opening the door to data theft and other cybercrimes. This is perhaps why hackers often publish these default credentials in their communities, having a wide selection of them to choose from (PCI DSS, 2015). In fact, this vulnerability can easily be seen in the provided code. Specifically, in lines 135-137 of the "Authenticate" file. There, the program provides a default username and password, both of which are "sdev425". Having both credentials match each other is practically an invitation for attackers, especially since the app does not allow the user to change them (See Figure 1).

```
127         boolean status = false;
128         int hitcnt=0;
129
130         try {
131             ClientDataSource ds = new ClientDataSource();
132             ds.setDatabaseName("SDEV425");
133             ds.setServerName("localhost");
134             ds.setPortNumber(1527);
135             ds.setUser("sdev425");
136             ds.setPassword("sdev425");
137             ds.setDataSourceName("jdbc:derby");
138
139             Connection conn = ds.getConnection();
140
141             Statement stmt = conn.createStatement();
142             String sql = "select user_id from sdev_users  where email = '" + this.username + "'";
143             ResultSet rs = stmt.executeQuery(sql);
144             while (rs.next()) {
145                 user_id = rs.getInt(1);
146             }
```

Figure 1: Predefined Credentials given by the program

The completely identical username and password open the door for attackers to strike at any moment.

The first and most obvious way to address this issue is allowing the user to change the credentials. To do this, developers can include an additional prompt within the application can include an additional prompt, which will tell users to come up with their own username and password. As they are entered, they become the user's new credentials, and default ones are disabled, along with any default accounts. However, PCI DSS recommends making these changes even before the system is installed onto a network (PCI DSS, Requirement 2.1, 2015). This will deprive hackers of the opportunity to strike in the few minutes that the default credentials would still be active. This guideline applies to every default password in the system, including those for security and operating systems (PCI DSS, Requirement 2.1, 2015). In addition, the document stresses the importance of changing defaults responsible for cardholder data protection, encryption, and community strings (PCI DSS, Requirement 2.1.1, 2015).

To ensure that the above solution is implemented correctly, the document suggests that companies create a set of well-thought-out standards to detect and resolve vulnerabilities. They

must be put together with common vulnerabilities in mind, and be consistent with industry requirements (PCI DSS, Requirement 2.2, 2015).

Besides the standard solution of changing all default credentials, there are many more regulations that are needed to address this problem. For example, each server must have only one primary function operating on it. This is done so that functions with different security levels don't coexist on the same server, opening the door to an attack (Requirement 2.2.1). Furthermore, each function must have only necessary components working for it, to prevent exploitation (Requirement 2.2.2). Any required parts of the system considered highly vulnerable must have additional security measures implemented, with the use of services like SSH and VPN (Requirement 2.2.3). Finally, correctly configured system parameters, encrypted data, and removal of unnecessary functionality can meet PCI DSS Requirement 2 at a satisfactory level (Requirements 2.2.4, 2.3, and 2.2.5).

Requirement 3: Protect Stored Cardholder Data

Since the program we are examining is a banking application, protection of stored cardholder data is a priority. A cyberattack on a bank's systems can lead to everything from stolen funds to identity theft, not only damaging customer trust and company reputation, but also allowing attackers to potentially use the newly obtained data for future criminal activities. As explained in the PCI DSS, "encryption, truncation, masking, and hashing are critical components of cardholder data protection." (PCI DSS, 2015, p.36, para.1). The first one is especially important, since encrypted data is obsolete to a hacker even if they somehow manage to access it.

There are a few recommendations to protect the integrity of cardholder data. Firstly, the program must only store such data if completely necessary. The need for storage must be determined by

company procedures and policies regarding data retention. Overall, banks must only store data that is legally required to be stored, and have specific processes to securely delete data once they have no further use for it (PCI DSS, Requirement 3.1, 2015). The document especially warns against storing highly sensitive cardholder data, such as data from the magnetic stripe at the back of a card, card verification codes, and Personal Identification Number (PIN) (Requirement 3.1, 2015). Some of the data mentioned can actually be seen in the application examined here, such as the PIN, and track data. The program even specifies the latter variable as "FullTrackData". Since PCI DSS prohibits the storage of both the PIN full track data, this is a serious security vulnerability (See Figure 2).

```
26  * @author jim
27  */          C:\Users\Mark Kardash\AppData\Local\Temp\3b4716f4-1f84-45
28 public class ShowAccount extends HttpServlet {
29
30     // Variable
31     private HttpSession session;
32     // Database field data
33     private int user_id;
34     private String Cardholdername;
35     private String CardType;
36     private String ServiceCode;
37     private String CardNumber;
38     private int CAV_CCV2;
39     private Date expiredate;
40     private String FullTrackData;
41     private String PIN;
42
```

Figure 2: Storage of PIN and full track data violates PCI DSS requirements.

Although they are set as "private", the sole act of storing the customer PIN and full data from a track goes in direct violation of PCI DSS Requirements 3.2.1 and 3.2.3.

In addition, the Standard suggests keeping only the last few digits of Permanent Account Numbers (PANs) visible, never send them by means like SMS and email (Requirement 4.1,

2015) and store data encryption keys in the fewest possible locations, and in protected formats (Requirements 3.4 and 3.5, 2015).

From a practical standpoint, an application like the one examined can implement this solution by, firstly, setting all required stored data to “private”. This will make it visible only to developers, minimizing risks of attacks. The application being examined here has already done this, so the other thing that could be done is setting up an encryption mechanism to make the data unreadable in case of a breach. The keys for that mechanism must be strong as determined by company policies, and able to be retired/replaced when they are weakened. While these are measures specific to the app’s source code, there are also requirements that would apply to those handling its safety, such as encryption key custodians being informed, and understanding the responsibility given to them. Split knowledge must be enacted for manual encryption keys (meaning multiple people must have knowledge of the keys (Requirements 3.4-3.7, 2015) to prevent a scenario where a single disgruntled employee perpetrates an attack. Encryption will be talked more about in the next requirement examined, while hashing is a good alternative to this. A hash mechanism will greatly strengthen the password's integrity, assuming it has been changed from default to a more secure one (Requirement 3, 2015).

Requirement 4: Encrypt transmission of cardholder data across open, public networks

Once again, because of its nature as a financial app, encryption would be a significant improvement for the program examined here. Especially since the network it would be used on would most likely be public and open, just as the title describes. The PCI Standard warns that “Misconfigured wireless networks and vulnerabilities in legacy encryption and authentication protocols continue to be targets of malicious individuals who exploit these vulnerabilities to gain privileged access to cardholder data environments (PCI DSS, Requirement 4, 2015, para.1).

When a system lacks encryption, the data it stores becomes clearly visible to an attacker as soon as all other security measures are bypassed. This would leave the app with a significant disadvantage compared to encrypted software, as it would miss an opportunity to significantly decrease chances of a breach.

The DSS offers a few ways to implement this requirement. The development team must first ensure that they have strong protocols for both cryptography and security. This is done to prevent cardholder data from being attacked during transmission. Some of the policies recommended by the Standard are IPSEC and SSH. The encryption methodologies used must have appropriate encryption strength, with the protocol only accepting secure versions, and encrypted data only using trusted key to be unlocked (Requirement 4.1, 2015). As mentioned in the previous section, if the code of the particular banking app examined here was to be modified, developers would have to make sure that a secure encryption mechanism is added, and that it complies with company standards. Other encryption-related measures to be taken were mostly discussed in the previous section, such as using industry best practices for authentication and encryption, not sending PANs via email or text, as well as making sure that all policies regarding the security of cardholder data are known, used, and properly documented (Requirements 4.1.1-4.1.3, 2015).

Requirement 7: Restrict access to cardholder data by business need to know

One of the most important aspects of protecting customer data is ensuring it can only be accessed by authorized persons. Thus, certain rules must be implemented to ensure only those in appropriate positions have such “need to know” access. The PCI DSS defines “need to know” as “when access rights are granted to only the least amount of data and privileges needed to perform a job (PCI DSS, 2015, p. 64, para.1). The consequences of giving certain individuals privileges they should not have are quite obvious, as the risk of them abusing these privileges is very real.

Having such unique access to the application, they can either sell data to criminals for monetary gain, or, in the case of disgruntled employees, use it to intentionally damage the system themselves. To prevent these things from happening, the DSS once again gives certain regulations. As said, the personal data of cardholders, as well as specific components of the system, must only be accessible to those employees with positions that require them. The access needs for every role must be clearly defined, including the privilege level every employee would have (user, administrator, etc.) In addition to this, all access must be confirmed via documented permission from a higher up (Requirement 7.1, 2015). To implement this requirement on the application given for this project, the best solution may be to follow section 7.2. That means modifying the existing code by embedding a function that restricts access to system components based on the level and ID of the user (Requirement 7.2, 2015). For example, as the user selects to make a transaction, there could be a prompt asking them for their authorization level (client, administrator, etc.), and a unique ID number. Based on the data received, either a "Welcome to the system" message, or an "Access denied" message would be displayed. The PCI DSS also mentions that the program must be set to a "deny all" setting by default. Thus, all users must be automatically denied all privileges, unless specified otherwise by the individuals in charge of the system (Requirement 7.2.3, 2015). This will make the system even more difficult for attackers to access.

Requirement 8: Identify and authenticate access to system components

In the previous section, the need to restrict cardholder data based on user roles and privileges was discussed. However, before restricting privileges to individuals with certain roles, one must be able to correctly identify them. According to PCI DSS, one of the best ways to ensure each user is indeed who they claim to be is to assign a unique ID number to each one. With this

requirement in place, administrators will be able to know exactly who performed each action, tracing any suspicious activity to concrete individuals (PCI DSS, 2015 p. 67). The consequences of unauthorized users accessing the system have been discussed many times over this document, so this requirement is extremely important. In the case of the app provided, a prompt could be written into the original code, asking the user for their ID. According to the PCI requirements, the ID must be unique to every user (Requirement 8.1.1, 2015), with there being a limited number of access attempts (with a maximum of 6) before the user is locked out (Requirement 8.1.6, 2015). The time of the lockout must be at least 30 minutes (Requirement 8.1.7, 2015) to delay the progress of any ongoing attacks. In addition, the ID must be strictly monitored, and disabled when it is not being used (Requirement 8.1.5, 2015). The PCI DSS also dictates that administration should remove any inactive user accounts within 90 days (Requirement 8.1.4, 2015). Another useful addition to the card-handling app would be a function that would require the user to reauthenticate if they leave the program idle for more than 15 minutes during a session (Requirement 8.1.8, 2015).

Finally, something the app already has, is a unique password that would ensure secure access to the system as a complement to the unique user ID (Requirement 8.2, 2015).

Conclusion:

Banking apps like the one discussed in this document are some of the most lucrative targets for hackers, due to the nature and sensitivity of the information they carry. As such, special attention must be paid to them when it comes to security enhancements. For the purposes of this assignment, the five enhancements were analyzed very briefly, highlighting only their definitions. implementation procedures, and consequences of not implementing them. They were then applied to the program in theory, and with screenshots of their violations within the

original code, whenever possible. Overall, the PCI DSS has very detailed explanations of specific rules, that are extremely useful when it comes to enhancing software security.

References

Payment Card Industry (PCI) Data Security Standard Requirements and Security Assessment Procedures (3.1). (2015). PCI Security Standards Council.