OWASP Top 10 Proactive Controls	IEEE Top 10 Software Security Design Flaws	OWASP Top 10 Vulnerabilities Mitigated	OWASP Mobile Top 10 Vulnerabilities Mitigated
C1: Define Security Requirements A security requirement is a statement of needed security functionality that ensures one of many different security properties of software is being satisfied. Security requirements are derived from industry standards, applicable laws, and a history of past vulnerabilities. Security requirements define new features or additions to existing features to solve a specific security problem or eliminate a potential vulnerability. Security requirements provide a foundation of vetted security functionality for an application. Instead of creating a custom approach to security for every application, standard security requirements allow developers to reuse the definition of security controls and best practices. Those same vetted security requirements provide solutions for security issues that have occurred in the past. Requirements exist to prevent the repeat of past security failures.	• **Earn or give, but never assume, trust** • **Use an authentication mechanism that cannot be bypassed or tampered with** • **Authorize after you authenticate** • **Strictly separate data and control instructions, and never process control instructions received from untrusted sources** • **Define an approach that ensures all data are explicitly validated** • **Use cryptography correctly** • **Identify sensitive data and how they should be handled** • **Always consider the users** • **Understand how integrating external components changes your attack surface** • **Be flexible when considering future changes to objects and actors**	• **A1:2017-Injection** • **A2:2017-Broken Authentication** • **A3:2017-Sensitive Data Exposure** • **A4:2017-XML External Entities (XXE)** • **A5:2017-Broken Access Control** • **A6:2017-Security Misconfiguration** • **A7:2017-Cross-Site Scripting (XSS)** • **A8:2017-Insecure Deserialization** • **A9:2017-Using Components with Known Vulnerabilities** • **A10:2017-Insufficient Logging & Monitoring**	• **M1: Improper Platform Usage** • **M2: Insecure Data Storage** • **M3: Insecure Communication** • **M4: Insecure Authentication** • **M5: Insufficient Cryptography** • **M6: Insecure Authorization** • **M7: Client Code Quality** • **M8: Code Tampering** • **M9: Reverse Engineering** • **M10: Extraneous Functionality**

C2: Leverage Security Frameworks and Libraries Secure coding libraries and software frameworks with embedded security help software developers guard against security-related design and implementation flaws. A developer writing an application from scratch might not have sufficient knowledge, time, or budget to properly implement or maintain security features. Leveraging security frameworks helps accomplish security goals more efficiently and accurately.	• **Earn or give, but never assume, trust** • **Use an authentication mechanism that cannot be bypassed or tampered with** • **Authorize after you authenticate** • **Strictly separate data and control instructions, and never process control instructions received from untrusted sources** • **Define an approach that ensures all data are explicitly validated** • **Use cryptography correctly** • **Always consider the users** • **Understand how integrating external components changes your attack surface** • **Be flexible when considering future changes to objects and actors**	• **A1:2017-Injection** • **A2:2017-Broken Authentication** • **A3:2017-Sensitive Data Exposure** • **A4:2017-XML External Entities (XXE)** • **A5:2017-Broken Access Control** • **A7:2017-Cross-Site Scripting (XSS)** • **A8:2017-Insecure Deserialization**	• **M1: Improper Platform Usage** • **M2: Insecure Data Storage** • **M3: Insecure Communication** • **M4: Insecure Authentication** • **M5: Insufficient Cryptography** • **M6: Insecure Authorization** • **M7: Client Code Quality** • **M8: Code Tampering** • **M9: Reverse Engineering** • **M10: Extraneous Functionality**
C3: Secure Database Access This section describes secure access to all data stores, including both relational databases and NoSQL databases. Some areas to consider: 1. Secure queries 2. Secure configuration 3. Secure authentication 4. Secure communication	• **Earn or give, but never assume, trust** • **Strictly separate data and control instructions, and never process control instructions received from untrusted sources** • **Define an approach that ensures all data are explicitly validated** • **Use cryptography correctly** • **Identify sensitive data and how they should be handled** • **Always consider the users** • **Be flexible when considering future changes to objects and actors**	• **A1:2017-Injection** Injection flaws, such as SQL, NoSQL, OS, and LDAP injection, occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization.	• **M1: Improper Platform Usage** This category covers misuse of a platform feature or failure to use platform security controls. It might include Android intents, platform permissions, misuse of TouchID, the Keychain, or some other security control that is part of the mobile operating system.

C4: Encode and Escape Data Encode data before passing to an interpreter or parser (JS, CSS , XML) and encode special characters so that that they are interpreted as text and not as closing a string.	**Earn or give, but never assume, trust** Designs that place authorization, access control, enforcement of security policy, or embedded sensitive data in client software thinking that it won't be discovered, modified, or exposed by clever users or malicious attackers are inherently weak. **Strictly separate data and control instructions, and never process control instructions received from untrusted sources** Co-mingling data and control instructions in a single entity, especially a string, can lead to injection vulnerabilities. **Define an approach that ensures all data are explicitly validated** Software systems and components commonly make assumptions about data they operate on. It is important to explicitly ensure that such assumptions hold: Vulnerabilities frequently arise from implicit assumptions about data, which can be exploited if an attacker can subvert and invalidate these assumptions.	**A1:2017- Injection** Injection flaws, such as SQL, NoSQL, OS, and LDAP injection, occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization. **A7:2017- Cross-Site Scripting (XSS)** XSS flaws occur whenever an application includes untrusted data in a new web page without proper validation or escaping, or updates an existing web page with user-supplied data using a browser API that can create HTML or JavaScript. XSS allows attackers to execute scripts in the victim's browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites.	**M7: Poor Code Quality** Threat Agents include entities that can pass untrusted inputs to method calls made within mobile code. These types of issues are not necessarily security issues in and of themselves but lead to security vulnerabilities.

C5: Validate All Inputs Input validation is a programming technique that ensures only properly formatted data may enter a software system component.	**Earn or give, but never assume, trust** Designs that place authorization, access control, enforcement of security policy, or embedded sensitive data in client software thinking that it won't be discovered, modified, or exposed by clever users or malicious attackers are inherently weak. Such designs will often lead to compromises. **Strictly separate data and control instructions, and never process control instructions received from untrusted sources** Co-mingling data and control instructions in a single entity, especially a string, can lead to injection vulnerabilities. **Define an approach that ensures all data are explicitly validated** Software systems and components commonly make assumptions about data they operate on. It is important to explicitly ensure that such assumptions hold: Vulnerabilities frequently arise from implicit assumptions about data, which can be exploited if an attacker can subvert and invalidate these assumptions.	**A1:2017- Injection** Injection flaws, such as SQL, NoSQL, OS, and LDAP injection, occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization. **A7:2017- Cross-Site Scripting (XSS)** XSS flaws occur whenever an application includes untrusted data in a new web page without proper validation or escaping, or updates an existing web page with user-supplied data using a browser API that can create HTML or JavaScript. XSS allows attackers to execute scripts in the victim's browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites.	**M7: Poor Code Quality** Threat Agents include entities that can pass untrusted inputs to method calls made within mobile code. These types of issues are not necessarily security issues in and of themselves but lead to security vulnerabilities.

C6: Implement Digital Identity Digital Identity is the unique representation of a user (or other subject) as they engage in an online transaction. Authentication is the process of verifying that an individual or entity is who they claim to be. Session management is a process by which a server maintains the state of the user's authentication so that the user may continue to use the system without re-authenticating.	**Use an authentication mechanism that cannot be bypassed or tampered with** The ability to bypass an authentication mechanism can result in an unauthorized entity having access to a system or service that it shouldn't.	**A2:2017-Broken Authentication** Confirmation of the user's identity, authentication, and session management are critical to protect against authentication-related attacks.	**M4: Insecure Authentication** There are many different ways that a mobile app may suffer from insecure authentication: • If the mobile app is able to anonymously execute a backend API service request without providing an access token. • If the mobile app stores any passwords or shared secrets locally on the device. • If the mobile app uses a weak password policy to simplify entering a password • If the mobile app uses a feature like TouchID.
C7: Enforce Access Controls Access Control (or Authorization) is the process of granting or denying specific requests from a user, program, or process. Access control also involves the act of granting and revoking those privileges. It should be noted that authorization (verifying access to specific features or resources) is not equivalent to authentication (verifying identity).	**Authorize after you authenticate** Authorization should be conducted as an explicit check, and as necessary even after an initial authentication has been completed. Strictly separate data and control instructions, and never process control instructions received from untrusted sources Co-mingling data and control instructions in a single entity, especially a string, can lead to injection vulnerabilities. Lack of strict separation between data and code often leads to untrusted data controlling the execution flow of a software system.	**A5:2017-Broken Access Control** Access control enforces policy such that users cannot act outside of their intended permissions. Failures typically lead to unauthorized information disclosure, modification or destruction of all data, or performing a business function outside of the limits of the user.	**M6: Insecure Authorization** It is important to recognize the difference between authentication and authorization. Authentication is the act of identifying an individual. Authorization is the act of checking that the identified individual has the permissions necessary to perform the act. The two are closely related as authorization checks should always immediately follow authentication of an incoming request from a mobile device.

C8: Protect Data Everywhere Sensitive data such as passwords, credit card numbers, health records, personal information and business secrets require extra protection, particularly if that data falls under privacy laws (EU's General Data Protection Regulation GDPR), financial data protection rules such as PCI Data Security Standard (PCI DSS) or other regulations.	**Use cryptography correctly** Cryptography is one of the most important tools for building secure systems. **Identify sensitive data and how they should be handled** Data are critical to organizations and to users. One of the first tasks that systems designers must do is identify sensitive data and determine how to protect it appropriately. Many deployed systems over the years have failed to protect data appropriately. This can happen when designers fail to identify data as sensitive, or when designers do not identify all the ways in which data could be manipulated or exposed.	**A3:2017-Sensitive Data Exposure** The first thing is to determine the protection needs of data in transit and at rest. For example, passwords, credit card numbers, health records, personal information and business secrets require extra protection, particularly if that data falls under privacy laws, e.g. EU's General Data Protection Regulation (GDPR), or regulations, e.g. financial data protection such as PCI Data Security Standard (PCI DSS).	**M2: Insecure Data Storage** This category insecure data storage and unintended data leakage. **M3: Insecure Communication** This risk covers all aspects of getting data from point A to point B, but doing it insecurely.

C9: Implement Security Logging and Monitoring Logging is a concept that most developers already use for debugging and diagnostic purposes. Security logging is an equally basic concept: to log security information during the runtime operation of an application. Monitoring is the live review of application and security logs using various forms of automation. The same tools and patterns can be used for operations, debugging and security purposes.	**Always consider the users** **Be flexible when considering future changes to objects and actors**	**A10:2017-Insufficient Logging & Monitoring** Insufficient logging, detection, monitoring and active response occurs any time: • Auditable events, such as logins, failed logins, and high-value transactions are not logged. • Warnings and errors generate no, inadequate, or unclear log messages. • Logs of applications and APIs are not monitored for suspicious activity. • Logs are only stored locally. • Appropriate alerting thresholds and response escalation processes are not in place or effective. • Penetration testing and scans by tools do not trigger alerts. • The application is unable to detect, escalate, or alert for active attacks in real time or near real time.	While implementing security logging and monitoring results in reducing the impact of many types of vulnerabilities, but does not prevent any of the OWASP Mobile Top Ten vulnerabilities.
C10: Handle All Errors and Exceptions Exception handling is a programming concept that allows an application to respond to different error states (like network down, or database connection failed, etc) in various ways. Handling exceptions and errors correctly is critical to making your code reliable and secure.	**Define an Approach that Ensures all Data are Explicitly Validated** **Identify Sensitive Data and How They Should Be Handled**	While handing all errors and exceptions results in reducing the impact of many types of vulnerabilities, but does not prevent any of the OWASP Top Ten vulnerabilities.	While handing all errors and exceptions results in reducing the impact of many types of vulnerabilities, it does not prevent any of the OWASP Mobile Top Ten vulnerabilities.

References:

- OWASP Proactive Controls/ OWASP Top 10 Vulnerabilities Mapping [https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=Top_Ten_Mapping]
- IEEE Top 10 Software Security Design Flaws [http://cybersecurity.ieee.org/center-for-secure-design/avoiding-the-top-10-security-flaws.html]
- OWASP Top Ten 2017 [https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project]
- OWASP Top Ten Mobile Risks 2016 [https://www.owasp.org/index.php/OWASP_Mobile_Security_Project#tab=Top_10_Mobile_Risks]
- OWASP Top 10 Vulnerabilities Mapping – 2015 - [https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=Top_Ten_Mapping]

Checklist

- Enter OWASP Proactive Controls - **Completed**
- Enter OWASP Top 10 Vulnerabilities prevented/ defended - **Completed**
- Enter IEEE Top 10 Software Security Design Flaws prevented/ defended - **Completed**
- Mapping Top 10 Proactive Controls/ Top Ten Risks/ IEEE Top 10 - **Completed**
- Spellcheck - **Completed**
- Formatting check – - **Completed**