SANS AppSec
APPLICATION & SOFTWARE SECURITY

Securing Web Application Technologies (SWAT) CHECKLIST

The SWAT Checklist provides an easy-to-reference set of best practices that raise awareness and help development teams create more secure applications. It's a first step toward building a base of security knowledge around web application security. Use this checklist to identify the minimum standard that is required to neutralize vulnerabilities in your critical applications.

ERROR HANDLING AND LOGGING

BEST PRACTICE	DESCRIPTION	CWE ID
Display generic error messages	Error messages should not reveal details about the internal state of the application. For example, file system path and stack information should not be exposed to the user through error messages.	CWE-209
No unhandled exceptions	Given the languages and frameworks in use for web application development, never allow an unhandled exception to occur. Error handlers should be configured to handle unexpected errors and gracefully return controlled output to the user.	CWE-391
Suppress framework-generated errors	Your development framework or platform may generate default error messages. These should be suppressed or replaced with customized error messages, as framework-generated messages may reveal sensitive information to the user.	CWE-209
Log all authentication activities	Any authentication activities, whether successful or not, should be logged.	CWE-778
Log all privilege changes	Any activities or occasions where the user's privilege level changes should be logged.	CWE-778
Log administrative activities	Any administrative activities on the application or any of its components should be logged.	CWE-778
Log access to sensitive data	Any access to sensitive data should be logged. This is particularly important for corporations that have to meet regulatory requirements like HIPAA, PCI, or SOX.	CWE-778
Do not log inappropriate data	While logging errors and auditing access are important, sensitive data should never be logged in an unencrypted form. For example, under HIPAA and PCI, it would be a violation to log sensitive data into the log itself unless the log is encrypted on the disk. Additionally, it can create a serious exposure point should the web application itself become compromised.	CWE-532
Store logs securely	Logs should be stored and maintained appropriately to avoid information loss or tampering by intruders. Log retention should also follow the retention policy set forth by the organization to meet regulatory requirements and provide enough information for forensic and incident response activities.	CWE-533

THE MOST TRUSTED SOURCE FOR INFORMATION SECURITY TRAINING, CERTIFICATION, AND RESEARCH

SANS

Security Roadmap POSTER

Securing Web Application Technologies (SWAT) CHECKLIST

Version 1.4

AND

WhatWorks in Application Security

Ingraining security into the mind of every developer.

software-security.sans.org

AppSec_Poster_v1.4_8-17

DATA PROTECTION

BEST PRACTICE	DESCRIPTION	CWE ID
Use HTTPS everywhere	Ideally, HTTPS should be used for your entire application. If you have to limit where it's used, then HTTPS must be applied to any authentication pages as well as to all pages after the user is authenticated. If sensitive information (e.g., personal information) can be submitted before authentication, those features must also be sent over HTTPS. **EXAMPLE:** Firesheep	CWE-311 CWE-319 CWE-523
Disable HTTP access for all protected resources	For all pages requiring protection by HTTPS, the same URL should not be accessible via the insecure HTTP channel.	CWE-319
Use the Strict-Transport-Security header	The Strict-Transport-Security header ensures that the browser does not talk to the server over HTTP. This helps reduce the risk of HTTP downgrade attacks as implemented by the sslsniff tool.	
Store user passwords using a strong, iterative, salted hash	User passwords must be stored using secure hashing techniques with strong algorithms like PBKDF2, bcrypt, or SHA-512. Simply hashing the password a single time does not sufficiently protect the password. Use adaptive hashing (a work factor), combined with a randomly generated salt for each user to make the hash strong. **EXAMPLE:** LinkedIn password leak	CWE-257
Securely exchange encryption keys	If encryption keys are exchanged or pre-set in your application, then any key establishment or exchange must be performed over a secure channel.	
Set up secure key management processes	When keys are stored in your system they must be properly secured and only accessible to the appropriate staff on a need-to-know basis. **EXAMPLE:** AWS Key Management Service (KMS), Azure Key Vault, AWS CloudHSM	CWE-320
Weak TLS configuration	Weak ciphers must be disabled on all servers. For example, SSL v2, SSL v3, and TLS protocols prior to 1.2 have known weaknesses and are not considered secure. Additionally, disable the NULL, RC4, DES, and MD5 cipher suites. Ensure all key lengths are greater than 128 bits, use secure renegotiation, and disable compression. **EXAMPLE:** Qualys SSL Labs	
Use valid HTTPS certificates from a reputable certificate authority	HTTPS certificates should be signed by a reputable certificate authority. The name on the certificate should match the FQDN of the website. The certificate itself should be valid and not expired. **EXAMPLE:** Let's Encrypt (https://letsencrypt.org)	
Disable data caching using cache control headers and autocomplete	Browser data caching should be disabled using the cache control HTTP headers or meta tags within the HTML page. Additionally, sensitive input fields, such as the login form, should have the autocomplete attribute set to off in the HTML form to instruct the browser not to cache the credentials.	CWE-524
Limit the use and storage of sensitive data	Conduct an evaluation to ensure that sensitive data elements are not being unnecessarily transported or stored. Where possible, use tokenization to reduce data exposure risks.	

CONFIGURATION AND OPERATIONS

BEST PRACTICE	DESCRIPTION	CWE ID
Automate application deployment	Automating the deployment of your application, using Continuous Integration and Continuous Deployment, helps to ensure that changes are made in a consistent, repeatable manner in all environments.	
Establish a rigorous change management process	A rigorous change management process must be maintained during operations. For example, new releases should only be deployed after proper testing and associated documentation has been completed. **EXAMPLE:** RBS production outage http://www.computing.co.uk/ctg/analysis/2186972/rbs-wrong-rbs-manager	CWE-439
Define security requirements	Engage the business owner to define security requirements for the application. This includes items that range from the whitelist validation rules all the way to nonfunctional requirements like the performance of the login function. Defining these requirements up front ensures that security is baked into the system.	
Conduct a design review	Integrating security into the design phase saves money and time. Conduct a risk review with security professionals and threat model the application to identify key risks. This helps you integrate appropriate countermeasures into the design and architecture of the application.	CWE-701 CWE-656
Perform code reviews	Security-focused code reviews can be one of the most effective ways to find security bugs. Regularly review your code looking for common issues like SQL Injection and Cross-Site Scripting.	CWE-702
Perform security testing	Conduct security testing both during and after development to ensure the application meets security standards. Testing should also be conducted after major releases to ensure vulnerabilities did not get introduced during the update process.	
Harden the infrastructure	All components of infrastructure that support the application should be configured according to security best practices and hardening guidelines. In a typical web application this can include routers, firewalls, network switches, operating systems, web servers, application servers, databases, and application frameworks.	CWE-15 CWE-656
Define an incident handling plan	An incident handling plan should be drafted and tested on a regular basis. The contact list of people to involve in a security incident related to the application should be well defined and kept up to date.	
Educate the team on security	Training helps define a common language that the team can use to improve the security of the application. Education should not be confined solely to software developers, testers, and architects. Anyone associated with the development process, such as business analysts and project managers, should all have periodic software security awareness training.	

AUTHENTICATION

BEST PRACTICE	DESCRIPTION	CWE ID
Don't hardcode credentials	Never allow credentials to be stored directly within the application code. While it can be convenient to test application code with hardcoded credentials during development this significantly increases risk and should be avoided. **EXAMPLE:** Hard-coded passwords in networking devices https://www.us-cert.gov/control_systems/pdf/ICSA-12-243-01.pdf	CWE-798
Develop a strong password reset system	Password reset systems are often the weakest link in an application. These systems are often based on users answering personal questions to establish their identity and in turn reset the password. The system needs to be based on questions that are both hard to guess and brute force. Additionally, any password reset option must not reveal whether or not an account is valid, preventing username harvesting. **EXAMPLE:** Sara Palin password hack http://en.wikipedia.org/wiki/Sarah_Palin_email_hack	CWE-640
Implement a strong password policy	A password policy should be created and implemented so that passwords meet specific strength criteria. **EXAMPLE:** http://www.pcworld.com/article/128823/study_weak_passwords_really_do_help_hackers.html	CWE-521
Implement account lockout against brute-force attacks	Account lockout needs to be implemented to prevent brute-force attacks against both the authentication and password reset functionality. After several tries on a specific user account, the account should be locked for a period of time or until it is manually unlocked. Additionally, it is best to continue the same failure message indicating that the credentials are incorrect or the account is locked to prevent an attacker from harvesting usernames.	CWE-307
Don't disclose too much information in error messages	Messages for authentication errors must be clear and, at the same time, be written so that sensitive information about the system is not disclosed. For example, error messages which reveal that the user id is valid but that the corresponding password is incorrect confirm to an attacker that the account does exist on the system.	
Store database credentials securely	Modern web applications usually consist of multiple layers. The business logic tier (processing of information) often connects to the data tier (database). Connecting to the database, of course, requires authentication. The authentication credentials in the business logic tier must be stored in a centralized location that is locked down. Scattering credentials throughout the source code is not acceptable. Some development frameworks provide a centralized secure location for storing credentials to the backend database. These encrypted stores should be leveraged when possible.	CWE-257
Applications and middleware should run with minimal privileges	If an application becomes compromised it is important that the application itself and any middleware services be configured to run with minimal privileges. For instance, while the application layer or business layer need the ability to read and write data to the underlying database, administrative credentials that grant access to other databases or tables should not be provided.	CWE-250

SESSION MANAGEMENT

BEST PRACTICE	DESCRIPTION	CWE ID
Ensure that session identifiers are sufficiently random	Session tokens must be generated by secure random functions and must be of sufficient length to withstand analysis and prediction.	CWE-6
Regenerate session tokens	Session tokens should be regenerated when the user authenticates to the application and when the user privilege level changes. Additionally, should the encryption status change, the session token should always be regenerated.	CWE-384
Implement an idle session timeout	When a user is not active, the application should automatically log the user out. Be aware that Ajax applications may make recurring calls to the application, effectively resetting the timeout counter automatically.	CWE-613
Implement an absolute session timeout	Users should be logged out after an extensive amount of time (e.g., 4-8 hours) has passed since they logged in. This helps mitigate the risk of an attacker using a hijacked session.	CWE-613
Destroy sessions at any sign of tampering	Unless the application requires multiple simultaneous sessions for a single user, implement features to detect session cloning attempts. Should any sign of session cloning be detected, the session should be destroyed, forcing the real user to reauthenticate.	
Invalidate the session after logout	When the user logs out of the application, the session and corresponding data on the server must be destroyed. This ensures that the session cannot be accidentally revived.	CWE-613
Place a logout button on every page	The logout button or logout link should be easily accessible to users on every page after they have authenticated.	
Use secure cookie attributes (i.e., HttpOnly and Secure flags)	The session cookie should be set with both the HttpOnly and the Secure flags. This ensures that the session id will not be accessible to client-side scripts and will only be transmitted over HTTPS.	CWE-79 CWE-614
Set the cookie domain and path correctly	The cookie domain and path scope should be set to the most restrictive settings for your application. Any wildcard domain scoped cookie must have a good justification for its existence.	
Set the cookie expiration time	The session cookie should have a reasonable expiration time. Non-expiring session cookies should be avoided.	

Website
software-security.sans.org
Free resources, white papers, webcasts, and more

Blog
software-security.sans.org/blog

Twitter
@sansappsec
Latest news, promos, and other information

AppSec CyberTalent Assessment
sans.org/appsec-assessment

INPUT AND OUTPUT HANDLING

BEST PRACTICE	DESCRIPTION	CWE ID
Conduct contextual output encoding	All output functions must contextually encode data before sending the data to the user. Depending on where the output will end up in the HTML page, the output must be encoded differently. For example, data placed in the URL context must be encoded differently than data placed in JavaScript context within the HTML page. **EXAMPLE:** Resource: https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet	CWE-79
Prefer whitelists over blacklists	For each user input field, there should be validation on the input content. Whitelisting input is the preferred approach. Only accept data that meet a certain criteria. For input that needs more flexibility, blacklisting can also be applied where known bad input patterns or characters are blocked.	CWE-159 CWE-144
Use parameterized SQL queries	SQL queries should be crafted with user content passed into a bind variable. Queries written this way are safe against SQL injection attacks. SQL queries should not be created dynamically using string concatenation. Similarly, the SQL query string used in a bound or parameterized query should never be dynamically built from user input. **EXAMPLE:** Sony SQL injection hack http://www.infosecurity-magazine.com/view/27930/lulzsec-sony-pictures-hackers-were-school-chums)	CWE-89 CWE-564
Use tokens to prevent forged requests	In order to prevent Cross-Site Request Forgery attacks, you must embed a random value that is not known to third parties into the HTML form. This CSRF protection token must be unique to each request. This prevents a forged CSRF request from being submitted because the attacker does not know the value of the token.	CWE-352
Set the encoding for your application	For every page in your application set the encoding using HTTP headers or meta tags within HTML. This ensures that the encoding of the page is always defined and that the browser will not have to determine the encoding on its own. Setting a consistent encoding like UTF-8 for your application reduces the overall risk of issues like Cross-Site Scripting.	CWE-172
Validate uploaded files	When accepting file uploads from the user make sure to validate the size of the file, the file type, and the file contents, and ensure that it is not possible to override the destination path for the file.	CWE-434 CWE-616 CWE-22
Use the nosniff header for uploaded content	When hosting user uploaded content that can be viewed by other users, use the X-Content-Type-Options: nosniff header so that browsers do not try to guess the data type. Sometimes the browser can be tricked into displaying the data type incorrectly (e.g., showing a GIF file as HTML). Always let the server or application determine the data type.	CWE-430
Validate the source of input	The source of the input must be validated. For example, if input is expected from a POST request do not accept the input variable from a GET request.	CWE-20 CWE-346
Use the X-Frame-Options header	Use the X-Frame-Options header to prevent content from being loaded by a foreign site in a frame. This mitigates Clickjacking attacks. For older browsers that do not support this header add framebusting Javascript code to mitigate Clickjacking (although this method is not foolproof and can be circumvented). **EXAMPLE:** Flash camera and mic hack http://jeremiahgrossman.blogspot.com/2008/10/clickjacking-web-pages-can-see-and-hear.html	CAPEC-103 CWE-693
Use secure HTTP response headers	The Content Security Policy (CSP), X-XSS-Protection, and Public-Key-Pins headers help defend against Cross-Site Scripting (XSS) and Man-in-the-Middle (MITM) attacks. **EXAMPLE:** OWASP Secure Headers Project https://www.owasp.org/index.php/OWASP_Secure_Headers_Project	CWE-79 CWE-692

ACCESS CONTROL

BEST PRACTICE	DESCRIPTION	CWE ID
Apply access control checks consistently	Always apply the principle of complete mediation, forcing all requests through a common security "gate keeper." This ensures that access control checks are triggered whether or not the user is authenticated.	CWE-284
Apply the principle of least privilege	Use a Mandatory Access Control system. All access decisions will be based on the principle of least privilege. If not explicitly allowed then access should be denied. Additionally, after an account is created, rights must be specifically added to that account to grant access to resources.	CWE-272 CWE-250
Don't use direct object references for access control checks	Do not allow direct references to files or parameters that can be manipulated to grant excessive access. Access control decisions must be based on the authenticated user identity and trusted server-side information.	CWE-284
Don't use unvalidated forwards or redirects	An unvalidated forward can allow an attacker to access private content without authentication. Unvalidated redirects allow an attacker to lure victims into visiting malicious sites. Prevent this from occurring by conducting the appropriate access control checks before sending the user to the given location.	CWE-601

SANS APPSEC CURRICULUM

PLATFORM SECURITY

DEV531 Defending Mobile Applications Security Essentials

DEV541 Secure Coding in Java/JEE *GSSP-JAVA*

DEV544 Secure Coding in .NET *GSSP-NET*

CORE

STH.DEVELOPER Application Security Awareness Modules

DEV522 Defending Web Applications Security Essentials *GWEB*

DEV534 Secure DevOps: A Practical Introduction

SEC540 Secure DevOps and Cloud Application Security

SPECIALIZATION

SEC542 Web App Penetration Testing and Ethical Hacking *GWAPT*

SEC642 Advanced Web App Penetration Testing and Ethical Hacking

ASSESSMENT

AppSec CyberTalent Assessment
sans.org/appsec-assessment

WhatWorks IN Application Security

software-security.sans.org

Why is application security so important?

Web applications remain the proverbial punching bag of the Internet, with web application attacks accounting for 35% of breaches, according to the Verizon Data Breach Investigation Report.

Why should you have an application security program?

- **LinkedIn**
  LinkedIn reported that the password hashes of 6.5 million members were extracted from their servers. The social media provider used a weak password storage strategy, allowing unsalted passwords to be hashed using SHA1. It was estimated that 4 million passwords were reversed within a few days.
  www.computerworld.com/s/article/9227869/Hackers_crack_more_than_60_of_breached_LinkedIn_passwords
- **AT&T**
  AT&T reported that the e-mail addresses of 114,000 iPad owners were extracted from its website, including the addresses of some notable celebrities. The cell phone provider used a unique cellular ID number in the URL, which could be modified to access another user's email address.
  https://threatpost.com/man-convicted-illegally-accessing-att-servers-impersonating-ipad-112012/77236
- **Heartland Payment Systems**
  Heartland Payment Systems disclosed that a SQL Injection vulnerability resulted in the loss of an estimated 130 million credit card numbers. Reports were later released that Heartland spent an estimated $140 million on breach-related expenses.
  www.computerworld.com/s/article/9136805/SQL_injection_attacks_led_to_Heartland_Hannaford_breaches

Reasons to fund an application security program

- Reduce the number and severity of security incidents
- Avoid data loss and regulatory penalties (e.g., PCI, SOX, HIPAA)
- It is up to 30x cheaper to fix security defects in development vs. production*
- Avoid incident costs, which average $5.4 million per incident in the U.S.**
- Implement appropriate due diligence in relation to peer organizations (based on comparable spending, maturity models, etc.)
- Conform to industry standards (Critical Security Controls, ITIL, COBIT, etc.)

* NIST, 2002, The Economic Impacts of Inadequate Infrastructure for Software Testing
** Poneman Institute, 2013, Cost of Data Breach Report

DESIGN

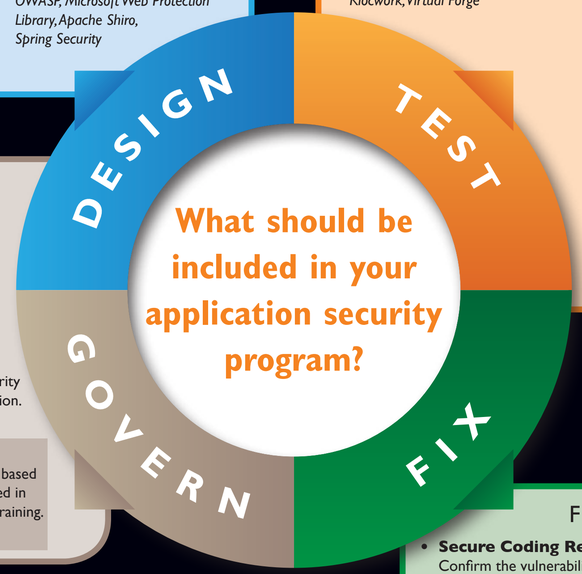
- **Security Requirements**
  Assign a security expert to the project team, inventory sensitive data that will be stored and processed by the application, and work with business partners to create security-specific use cases and development stories.
  SD Elements
- **Security Architecture**
  Perform an attack surface analysis and build a threat model to identify vulnerable areas of the design and architecture.
  Microsoft Threat Modeling Tool
- **Secure Coding Standards**
  Create standards that can be used by development teams to build secure-by-default applications.
  CERT Secure Coding Standards, Apple Secure Coding Guide
- **Secure Coding Libraries**
  Leverage security frameworks and APIs to provide critical security features.
  OWASP, Microsoft Web Protection Library, Apache Shiro, Spring Security

TEST

- **Dynamic Analysis**
  Identifies vulnerabilities in a live application often using automated scanners.
  IBM AppScan, HP Web Inspect, Acunetix, WhiteHat, NT Objectives, Qualys, Cenzic
- **Fuzz Testing**
  Testing technique that provides an application with invalid or corrupted input values with the intention of producing a vulnerability.
  FuzzDB, OWASP SecLists, Codenomicon
- **Static Analysis**
  Automated scanning of an application's source code to locate vulnerabilities in the application.
  HP Fortify, IBM AppScan Source, Checkmarx, Veracode, Coverity, Klocwork, Virtual Forge
- **Bug Bounty Programs**
  Program set up by a participating company to encourage security research and responsible vulnerability disclosure. These programs often reward participants with gifts or monetary prizes.
  Facebook, Google, Microsoft, Mozilla, PayPal, etc.
  https://bugcrowd.com/list-of-bug-bounty-programs
- **Code Reviews**
  Review source code changes with a subject-matter expert for security implications. Manually search the application's source code for vulnerabilities using customized scripts.
  Crucible, Gerrit, BitBucket, GitLab
- **Component Lifecyle Management**
  Implement a solution to find the use of third-party components with known published vulnerabilities. This phase typically involves taking an inventory of the components being used by an application and monitoring vulnerability intelligence feeds.
  Sonatype, Black Duck, Palamida, OWASP Dependency Check
- **Penetration Testing**
  Simulate how an attacker would exploit the vulnerabilities identified during the dynamic analysis phase.
  Burp Suite, W3AF, Skipfish, OWASP ZAP
- **Continuous Integration**
  Include security unit testing along with integration, functional, and smoke tests on every code check-in. Execute light-weight static and dynamic analysis in the Continuous Integration (CI) workflow.
  Jenkins, Bamboo, Travis, Circle CI

GOVERN

- **SDLC Integration**
  Ensure security is integrated into all phases of the development process through continuous business, development, and security collaboration.
  Microsoft SDL, NIST 800-64, BSIMM, Open SAMM
- **Metrics & Reporting**
  Provide dashboards and scorecards with qualitative feedback about the application security program and operational health of the application.
  ThreadFix, Jenkins
- **Application Security Training**
  Attend software security training with content based on the role in the development process. Offered in formats from modular CBT to live, in-person training.
  SANS



FIX

- **Secure Coding Remediation**
  Confirm the vulnerabilities identified in the test phase and use application security best practices to fix those issues. Create security unit-tests to validate the fixes, and add the unit tests to the Continuous Integration (CI) library to prevent a recurrence.
- **Virtual Patching**
  Apply a countermeasure that runs outside of the application and attempts to block vulnerabilities. Common examples are Web Application Firewalls (WAF) and Runtime Application Self-Protection (RASP) technologies.
  F5, ModSecurity, Imperva, Waratek, HP, Contrast, Prevoty
- **Patch Management (e.g., upgrading third-party components)**
  Confirm the vulnerabilities identified in the component lifecycle management phase and upgrade the third-party components to the safe patched version.

DevOps Security Best Practices

- **Continuous Integration**
  Building security into continuous integration starts with creating security-specific unit tests for critical sections of code such as authentication, password management, validation routines, and access control. Execute fast, accurate scans (e.g., static and dynamic analysis) for dangerous functions, OWASP Top 10 issues, and vulnerable dependencies. Failed tests and high-risk vulnerabilities found in these scans should fail the build and require immediate patches before deploying into a testing environment.
  - Jenkins https://jenkins-ci.org/
  - Bamboo https://www.atlassian.com/software/bamboo/
  - Travis https://travis-ci.org/
- **Continuous Delivery**
  Wiring automated security scanning into the continuous delivery process allows in-depth static and dynamic testing to be completed. The scan results are fed through pre-defined acceptance criteria, with failures resulting in an unsuccessful build and deployment.
  - Gauntlt http://gauntlt.org/
  - Mozilla Minon https://wiki.mozilla.org/Security/Projects/Minion
  - Yahoo Gryffin https://github.com/yahoo/gryffin
- **Continuous Deployment**
  Allowing developers to automate the delivery of changes into production requires additional compensating controls. Perform comprehensive audit logging to track the actions made and files viewed in production. Implement detective change control to log and review what files are modified. Encrypt confidential data in production and enable data loss prevention mechanisms to stop unauthorized information disclosure.
  - Go http://www.go.cd/
  - DeployIT http://gallery.xebia.com/component/deployit
- **Infrastructure as Code**
  Writing code to manage the server's infrastructure, configuration, and environment allows new servers to be quickly configured. Store infrastructure code and data in version control to track the history of changes made to the environment. Perform security reviews of the manifests and cookbooks to ensure hardened baseline requirements are met.
  - Puppet https://puppetlabs.com/
  - Chef https://www.chef.io/
  - Ansible http://www.ansible.com/
- **Container Security**
  Containers allow isolated images with an application and its dependencies to be quickly installed and executed. Use signed containers and verify the signatures to prevent tampered images from entering the environment. Isolate container images using a virtual machine or separate server, and ensure the image runs as a non-privileged user account.
  - Docker https://www.docker.com/
  - Rocket https://coreos.com/blog/rocket/
  - Twistlock https://www.twistlock.com/
  - Windows Containers https://msdn.microsoft.com/en-us/virtualization/windowscontainers/containers_welcome

Mobile Application Security Best Practices

- **Security Assessment**
  Ensure that development and security teams perform a security assessment prior to publishing a mobile application.
- **Client-side Data Storage**
  Avoid storing sensitive information (e.g. PCI, PHI, session tokens) permanently on the mobile device file system in locations such as a SQLite database, log file, XML file, or HTML5 database. Attackers can easily steal devices or use malware to extract information stored in these locations. Consider retrieving sensitive information temporarily from a secure web service and removing it from the device when it is no longer needed.
- **Weak Cryptography**
  If sensitive data must be stored on the device, apply an additional layer of encryption using an approved cryptographic algorithm. Do not hard-code encryption keys in source code or store them unprotected on the file system. Instead, retrieve encryption keys from a secure web service and store them in the keychain.
- **Information Disclosure**
  Disable mobile operating system features that cache sensitive information on the device. Common locations include the the keyboard autocomplete cache, copy paste buffer, and screenshots taken during application transitions. These global storage areas are accessible by malicious applications and could result in the exposure of sensitive information.
- **Transport Layer Protection**
  Mobile applications that rely on backend web services to send and receive sensitive information (e.g., customer data and session tokens) should enforce transport layer encryption. Performing certificate chain validation, using approved protocols and ciphers, and certificate pinning ensures that data is sent over secure channels.
- **Server-side Security**
  Web services that support mobile applications are vulnerable to the same types of attacks that have plagued web applications for years. Protect web service endpoints with strong authentication and authorization controls. Perform fuzz testing for injection issues (e.g., SQL, HTML, OS commands, and XML). Secure authentication cookies using the secure and HTTPOnly flags.
- **Mobile Application Security Assessment Tools**
  - QARK (Android) https://github.com/linkedin/qark
  - Needle (iOS) https://github.com/mwrlabs/needle

STH.Developer Training

Online CBT Modules
securingthehuman.org/developer

Critical software systems designed to enable business functions are often at the root of many headlines about data breaches and corporate hacks. **STH.Developer** provides the pinpoint software security awareness training that your team needs to prevent the mistakes that occur while software is being developed and deployed.

This framework, a proven and validated industry standard for secure web development, consists of an in-depth series of modules that provides developers with clear technical information about these security vulnerabilities and how to prevent them within their own code.

Video Courseware Demo available at
securingthehuman.org/developer/demo-training-lab

SANS SECURITY AWARENESS

How to evaluate developer security training

- ☑ **Flexible Delivery**
  Before selecting a training provider, it is important to understand how the training material will be delivered. The availability of online training allows students to take the same course as a live event, without the hassle or costs of traveling. This can be a flexible way to take a longer course from home or the office, without affecting work availability. Another option to consider is taking several CBT modules, which are small consumable training videos intended to raise awareness on one topic at a time.
- ☑ **Course Content**
  The application security landscape is constantly evolving, which means training material also needs to be updated frequently. Make sure you ask the training provider when the last time the course has been updated, and consider asking for the name and bio of the course author.
- ☑ **Certifications**
  Regardless of how training is delivered, it is important for management to understand any subject matter that may need more attention. Make sure the training provider provides knowledge checks in the form of quizzes, pre- and post-assessments, or certifications.

SANS AppSec Poster Contributors:

Ben Allen
Jim Bird
David Deatherage
Eric Johnson
Frank Kim
Jason Lam
Gregory Leonard
Johannes Ullrich, PhD