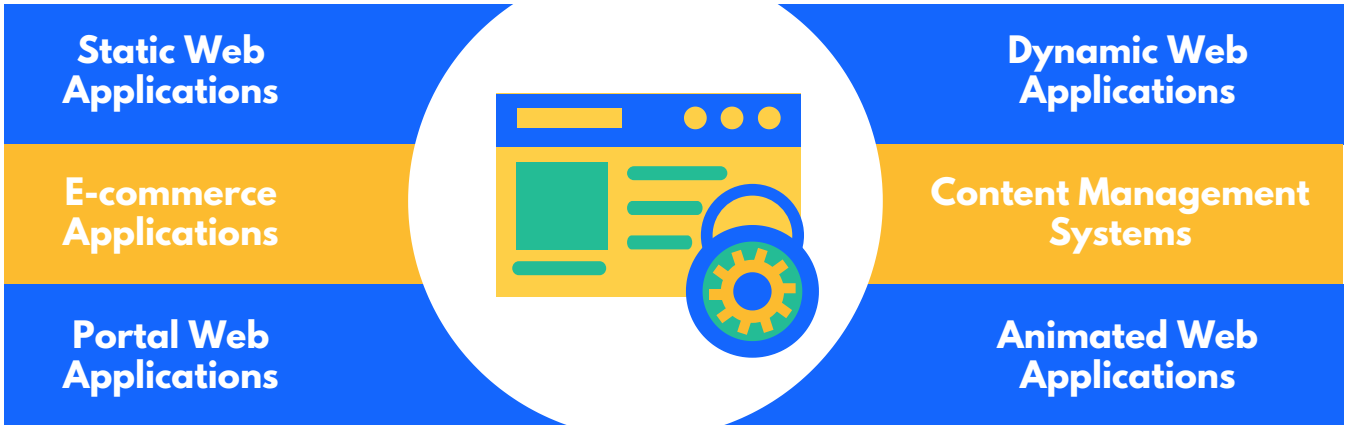
astra

The Ultimate SaaS Security Checklist

In today's cyber threats landscape scenario, cybersecurity enthusiasts have been highlighting the need for comprehensive security audits for the humongous number of SaaS applications on the internet. But it is only now that organizations having SaaS products/solutions started acknowledging it as a necessity for their business due to the increased risk of cyber attacks targeting their poorly secured SaaS applications.

While developing SaaS platforms, developers are provided objectives based solely on functionality, and very little if any security measures are taken into account. After all, if it's not written into their job description, why should it concern them? This in turn produces SaaS applications containing multiple vulnerabilities ranging from weaknesses around input validation, error handling, session management, and failure to implement proper access controls.



Sometimes it takes an exploited vulnerability resulting in a data breach or application defacement for developers and managers to realize the impact of having these weaknesses in their application.

In this checklist, we will discuss steps to take to perform a detailed security audit and penetration testing for your SaaS platform and its security standards for finding and fixing such security vulnerabilities & loopholes in your web applications.

SaaS Security Audit Checklist

Data Validation	1
1	Ensure the data received is what is expected and the data returned to the users is of an expected output
3	Perform Input Validation for All Inputs (prevent “buffer overflow” errors)
4	Ensure asynchronous consistency to avoid timing and sequence errors, race conditions, deadlocks, order dependencies, synchronization errors, etc.
5	Use Commit-or-Rollback semantics for exception handling
6	Use multitasking and multithreading safely
7	Set initial values for data
8	Avoid or eliminate “backdoors”
9	Avoid or eliminate shell escapes
10	Validate configuration files before use
11	Validate command-line parameters before use
12	Validate environment variables before use
13	Ensure communication connection queues cannot be exhausted

Data Protection 2

1	Examine all functions requiring user input or providing user output to ensure proper data protection for any sensitive information being utilized
2	Resource connection strings must be encrypted
3	Sensitive data should never be in code
4	Cookies containing sensitive data should be encrypted
5	Sensitive data should not be passed from page to page on client side
6	Pages containing sensitive data must only retrieve the sensitive data on demand when it is needed instead of persisting or caching it in memory

Error Handling 3

1	Examine the error handling of all functions to ensure the errors are sanitized to the point of providing “need to know” information back to the user
2	Production applications should not provide the same error messages as used in development
3	For any functions that return output to the user, ensure all error messages provide the user with the minimal amount of information needed to correct the error
4	Never display any errors to the user that would reveal information about the system or application itself
5	Ensure that your application has a “safe mode” which it can return if something truly unexpected occurs. If all else fails, log the user out and close the browser window. Production code should not be capable of producing debug messages
6	Assign log files the highest security protection, providing reassurance that you always have an effective 'black box' recorder if things go wrong
7	Simply be aware of different attacks. Take every security violation seriously, always get to the bottom of the cause event log errors and don't just dismiss errors unless you can be completely sure that you know it to be a technical problem.

Communication Security 4

1	Ensure sensitive or personal data is never be passed in clear text through the URL String
2	Ensure message freshness; even a valid message may present a danger if utilized in a “replay attack”
3	Work with the developer to trace the data flow from the functions identified in “Where to Start” to ensure that proper encryption is taking place
4	Calls to the database should use parameterized stored procedures
5	Protect message confidentiality

Authentication 5

1	Authentication of an individual should be validated prior to disclosing or providing any information specific to that individual
2	Determine all areas where the user interface changes between public and privately visible information
3	Re-authenticate user for high value transactions and access to protected areas
4	Authenticate the transaction, not the user
5	With form-based authentication, use a trusted access control component rather than writing your own
6	Use strong authentication (tokens, certificates, etc)

Authorization 6

1	You should follow the Principle of Least Privilege while implementing Authorization in your SaaS application
2	Only restricted and authorized users must have access to administrative interfaces used to manage site content and configuration
3	Always start Access Control Lists using “deny all” and then adding only those roles and privileges necessary
4	Once the application is certain a person is who they claim to be (authenticated), now we need to decide what permissions or visibility this user has access to. Following up on the interfaces identified during the “Authentication” stage, now you want to examine the level of visibility or control the user will have and to what areas of the application. For larger applications, typically roles (or groups) are defined and users are assigned to a specific role within the application
5	Example roles could be: Administrator, HelpDesk, Customer, Reporting Analyst, etc.
6	Work with the developer to identify the roles specific to the application you are auditing and gather the expected authorization (permissions, visibility, access) given to each of these roles
7	follow the transaction flow after the authentication process to examine how authorization is determined

Session Management 7

1	Sessions should be user specific and time specific as well
2	Authorization and role data should be stored on server side only
3	Session identifiers should be as strong as possible
4	Query strings should not contain session identifiers that represent authenticated users
5	A session timeout length should be defined
6	Form data should not contain hidden fields – if it is hidden, it probably needs to be protected and only available on the server side.

SaaS Application Penetration Testing Checklist

Information Gathering

1

- [ ] Conduct Search Engine Discovery Reconnaissance for Information Leakage
- [ ] Fingerprint Web Server
- [ ] Review Webserver Metafiles for Information Leakage
- [ ] Enumerate Applications on Webserver
- [ ] Review Webpage Content for Information Leakage
- [ ] Identify Application Entry Points
- [ ] Testing for Common Libraries and Fingerprinting
- [ ] Identify Application Entry Points
- [ ] Map Execution Paths Through Application
- [ ] Fingerprint Web Application Framework
- [ ] Fingerprint Web Application
- [ ] Map Application Architecture

Configuration and Deployment Management Testing

2

- Test Application Platform Configuration
- Test File Extensions Handling for Sensitive Information
- Review Old Backup and Unreferenced Files for Sensitive Info.
- Enumerate Infrastructure and Application Admin Interfaces
- Test HTTP Methods
- Test HTTP Strict Transport Security
- Test RIA Cross Domain Policy
- Test File Permission
- Test for Subdomain Takeover
- Test Cloud Storage

Identity Management Testing

3

- Test Role Definitions
- Test User Registration Process
- Test Account Provisioning Process
- Testing for Account Enumeration and Guessable User Account
- Testing for Weak or Unenforced Username Policy

4 Authentication Testing

- Testing for Credentials Transported over an Encrypted Channel
- Testing for Default Credentials
- Testing for Weak Lock Out Mechanism
- Testing for Bypassing Authentication Schema
- Testing for Vulnerable Remember Password
- Testing for Browser Cache Weaknesses
- Testing for Weak Password Policy
- Testing for Weak Security Question Answer
- Testing for Weak Password Change or Reset Functionalities
- Testing for Weaker Authentication in Alternative Channel

5 Authorization Testing

- Testing Directory Traversal File Include
- Testing for Bypassing Authorization Schema
- Testing for Privilege Escalation
- Testing for Insecure Direct Object References

Session Management Testing

6

- Testing for Session Management Schema
- Testing for Cookies Attributes
- Testing for Session Fixation
- Testing for Exposed Session Variables
- Testing for Cross Site Request Forgery
- Testing for Logout Functionality
- Testing Session Timeout
- Testing for Session Puzzling

Input Validation Testing

7

- Testing for Reflected Cross Site Scripting
- Testing for Stored Cross Site Scripting
- Testing for HTTP Verb Tampering
- Testing for HTTP Parameter Pollution
- Testing for SQL Injection
- Testing for Oracle
- Testing for MySQL
- Testing for SQL Server

- Testing PostgreSQL
- Testing for MS Access
- Testing for NoSQL Injection
- Testing for ORM Injection
- Testing for Client-side
- Testing for LDAP Injection
- Testing for XML Injection
- Testing for SSI Injection
- Testing for XPath Injection
- Testing for IMAP SMTP Injection
- Testing for Code Injection
- Testing for Local File Inclusion
- Testing for Remote File Inclusion
- Testing for Command Injection
- Testing for Buffer Overflow
- Testing for Heap Overflow
- Testing for Stack Overflow
- Testing for Format String
- Testing for Incubated Vulnerability
- Testing for HTTP Splitting Smuggling
- Testing for HTTP Incoming Requests
- Testing for Host Header Injection
- Testing for Server-side Template Injection

8 Testing for Error Handling

- Testing for Error Code
- Testing for Stack Traces

9 Testing for Weak Cryptography

- Testing for Weak SSL TLS Ciphers Insufficient Transport Layer Protection
- Testing for Padding Oracle
- Testing for Sensitive Information Sent via Unencrypted Channels
- Testing for Weak Encryption

10 Business Logic Testing

- Introduction to Business Logic
- Test Business Logic Data Validation
- Test Ability to Forge Requests
- Test Integrity Checks
- Test for Process Timing

- Test Number of Times a Function Can Be Used Limits
- Testing for the Circumvention of Work Flows
- Test Defenses Against Application Misuse
- Test Upload of Unexpected File Types
- Test Upload of Malicious Files

Client-side Testing

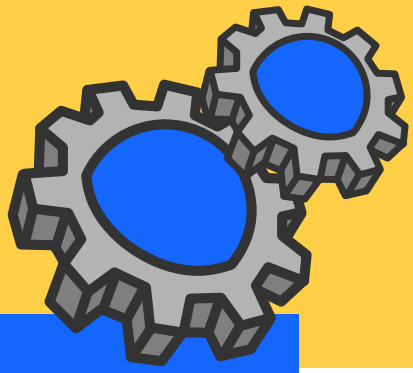
- Testing for DOM-Based Cross Site Scripting
- Testing for JavaScript Execution
- Testing for HTML Injection
- Testing for Client-side URL Redirect
- Testing for CSS Injection
- Testing for Client-side Resource Manipulation
- Testing Cross Origin Resource Sharing
- Testing for Cross Site Flashing
- Testing for Clickjacking
- Testing WebSockets
- Testing Web Messaging
- Testing Browser Storage
- Testing for Cross Site Script Inclusion

Infrastructure Testing

12

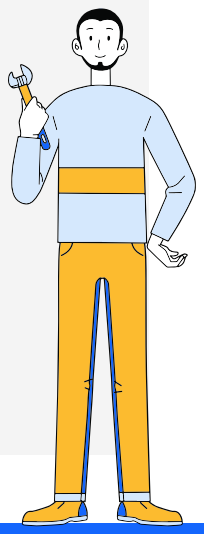
- Online based application testing on a cloud
- SaaS based Integration testing
- Testing for AWS/Azure/GCP infrastructure
- Availability / Accessability of the Cloud infrastructure
- Security/Privacy Testing for the Cloud infrastructure

SaaS Security Audit & Penetration Testing Tools



Tools

1. Zed Attack Proxy
2. Nikto
3. NMap (Network Mapper)
4. BurpSuite
5. Netsparker
6. SQLMap
7. W3af
8. TestSSL
9. Arachni
10. Wapiti
11. Metasploit
12. Acunetix
13. Grabber
14. Ratproxy
15. Wfuzz



Basic SaaS Security Measures for Organizations

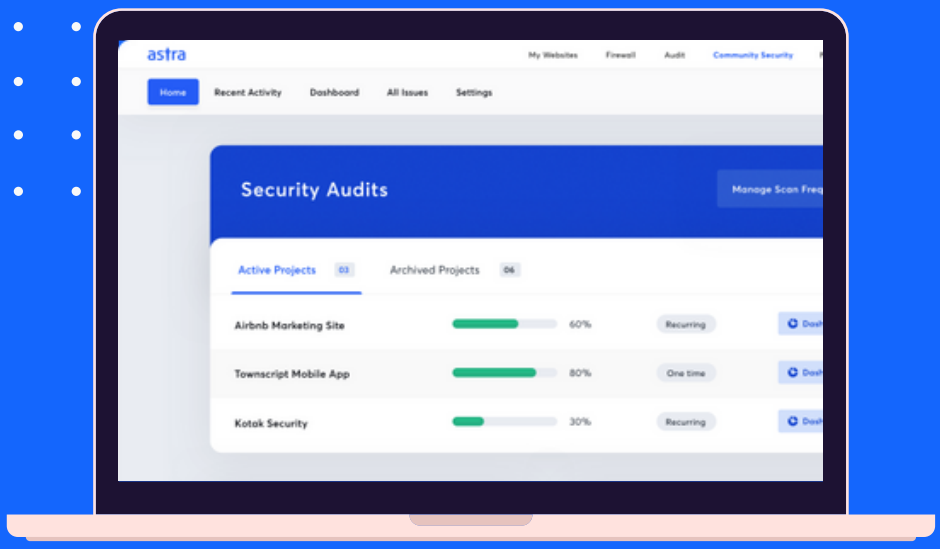
- Ensure security of your domains
- Make sure all critical integrations are updated and secure
- Do not share your internal Wi-Fi passwords
- Have internal and public security policies
- Set up a bug-bounty program for your SaaS applications
- Have a security incident response plan for your application/network
- Provide security awareness training to your employees
- Encrypt all the devices such as laptops and mobile phones
- Use password managers to store passwords
- Use centralized user account management
- Use SSL for your site
- Check your platform's basic security using Malware & Backdoor Scanners
- Secure your network perimeter
- Keep your systems/OS up to date.
- Take regular backups of every important asset
- Restrict internal services with the IP addresses
- Do periodic security audits for your SaaS applications

- **Enforce Secure Code Review Checklist - See here**
- **Perform DAST and SAST for your SaaS application**
- **Use a real-time application firewall for your SaaS**
- **Hire an expert penetration testing team - See here**

Check this article to know more on DIY SaaS Security Audit:
https://www.getastra.com/blog/security-audit/how-to-conduct-saas-security-audit/

Looking for a professional Security Audit & VAPT for your SaaS Platform?

Astra Security can help.



astra

Security audit based on industry leading practices such as OWASP, OSSTMM, WASC, CREST, NIST etc.

Astra Security's vulnerability management dashboard comes with a birds eye view for management keeping you always on the top of security assessment status.

Video PoCs, selenium scripts & collaboration with security team enables your developers to fix the vulnerabilities in record time. With Astra Security, VAPT takes 40% less time than other solutions.

Contact us to get a free demo

hello@getastra.com

fb.com/getAstra

Schedule a Call

@getastra

www.getastra.com

linkedin.com/company/getastra

References: 1) SANS 2) OWASP