



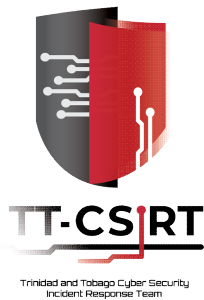
Trinidad And Tobago
Cyber Security
Incident Response Team

APPLICATION SECURITY WEBINAR

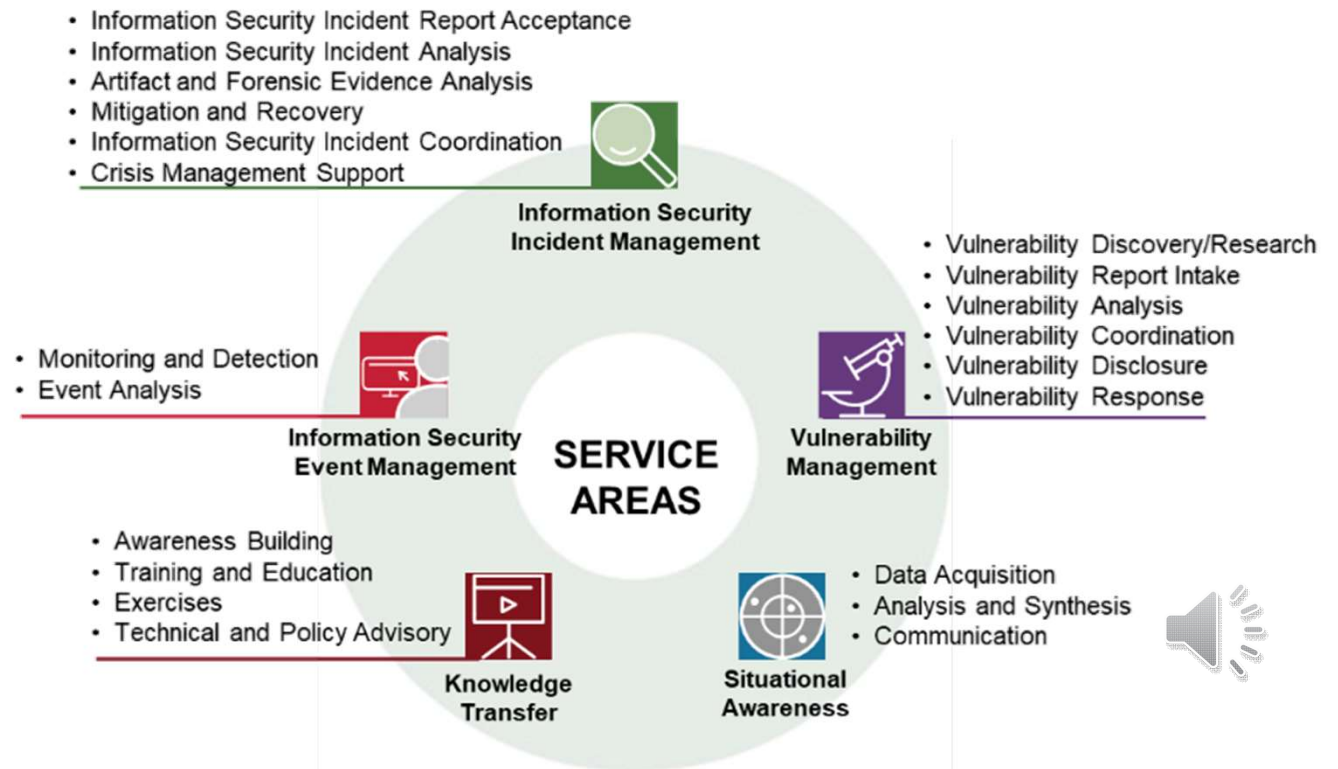
PRESENTED BY THE TRINIDAD AND TOBAGO CYBER SECURITY INCIDENT RESPONSE TEAM (TT-CSIRT)



WHO ARE WE AND WHAT WE DO

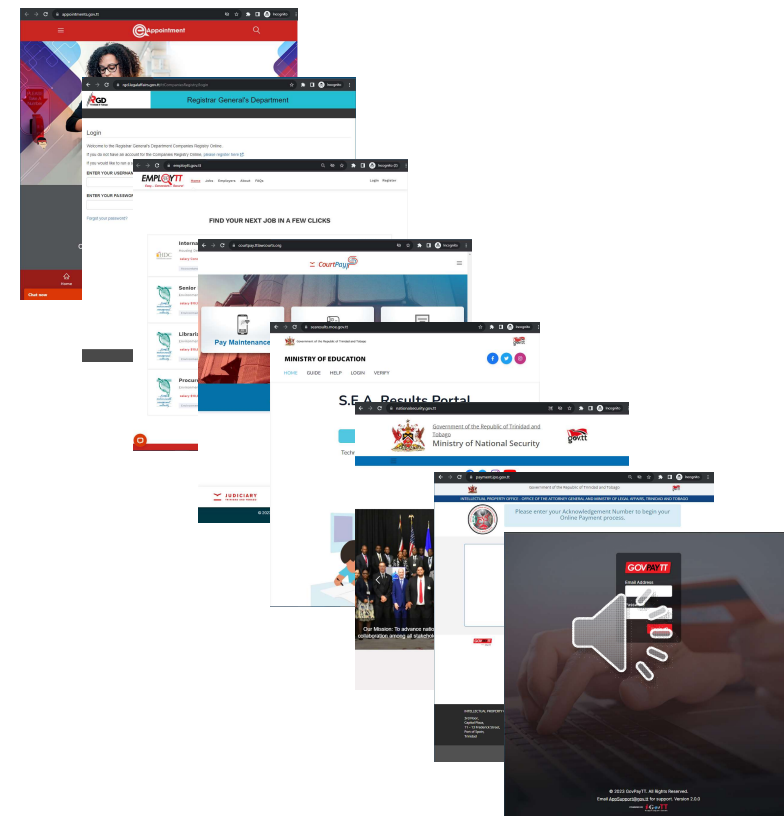


- Unit under the Ministry of National Security
- Established to serve both government and non-government organizations (public and private sectors)
- Services accessible at no cost – funded by the tax payers of T&T



TTCSIRT'S SECURITY ASSURANCE SERVICES

- The TTCSIRT provides security assurance activities including:
 - Static and Dynamic Application Testing
 - Network and Application Penetration Testing
 - Vulnerability Assessments
- As a reminder to MDAs, the TTCSIRT must assess all government applications that require treasury approval to accept online payments according to:
 - The Electronic Funds Transfer (EFT) Financial Instructions for Public Moneys Collected via the Credit Card Online Solution, 2020
 - The Electronic Funds Transfer (EFT) Financial Instructions for Public Moneys Collected via the Payment Service Provider Retail Payment Network, 2022



APPLICATION SECURITY TRENDS

OWASP's Top 10 (2021)

1. Broken Access Control
2. Cryptographic Failures
3. Injection
4. Insecure Design
5. Security Misconfiguration
6. Vulnerable And Outdated Components
7. Identification And Authorization Failures
8. Software And Data Integrity Failures
9. Security Logging And Monitoring Failures
10. Server-side Request Forgery



OWASP

- Educational publications & Training
- Open Web Application Security Projects
 - OWASP Top 10, WSTG, ASVS, SAMM and more!
- Enable developers to write better software and verify their work



LOCALLY OBSERVED APPLICATION SECURITY TRENDS

OWASP's Top 10 (2021)

Broken Access Control

Cryptographic Failures

Injection

Insecure Design

Security Misconfiguration

Vulnerable And Outdated Components

Identification And Authorization Failures

Software And Data Integrity Failures

Security Logging And Monitoring Failures

Server-side Request Forgery

TTCSIRT's Top 10

1. Vulnerable And Outdated Components

2. Identification And Authorization Failures

3. Broken Access Control

4. Security Misconfiguration

5. Insecure Design

6. Security Logging And Monitoring Failures

7. Software And Data Integrity Failures

8. Injection

9. Cryptographic Failures

10. Server-side Request Forgery



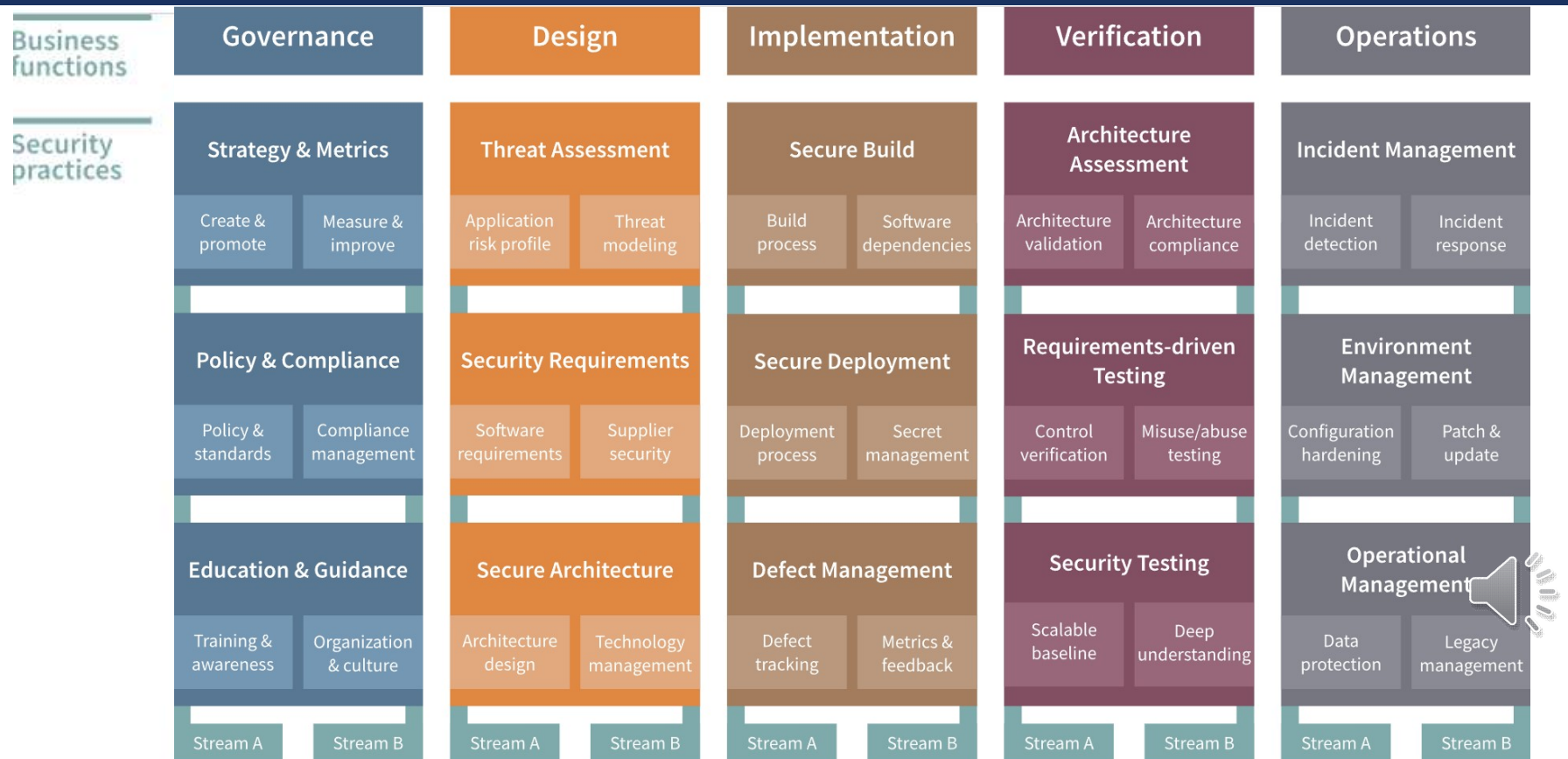
BUILDING SECURITY INTO SOFTWARE DEVELOPMENT

- Establish development policies, standards and guidelines
 - Data Classification & Handling, Compliance & Regulations, Coding Standard, 3rd Party Component Management
- Consider security at every phase of the SDLC
 - Planning – identify potential security risks and requirements, define security goals
 - Design – considerations for secure architecture, least privilege, defence in depth, resiliency
 - Coding – Make use of secure coding standards, best practices, validation, encryption, code reviews, error handling
 - Testing – Unit testing, configuration testing, vulnerability scanning, SAST & DAST, penetration testing
 - Deployment – secure production environment, secure configurations on application, host and network, monitoring, IRP
 - Maintenance – continuous monitoring, patch management, periodic security assessments.
- Threat Modelling
- Developer security awareness and training
- Leverage the Software Assurance Maturity Model (SAMM)




SOFTWARE ASSURANCE MATURITY MODEL

<https://owaspsamm.org/>



SAMM: THREAT ASSESSMENT

<https://owasp samm.org/model/design/threat-assessment/>

Maturity Level	Description	Application Risk Profile	Threat Modelling
1	Best-effort identification of high-level threats to the organization and individual projects.	A basic assessment of the application risk is performed to understand likelihood and impact of an attack.	Perform best-effort, risk-based threat modeling using brainstorming and existing diagrams with simple threat checklists.
2	Standardization and enterprise-wide analysis of software-related threats within the organization.	Understand the risk for all applications in the organization by centralizing the risk profile inventory for stakeholders.	Standardize threat modeling training, processes, and tools to scale across the organization.
3	Proactive improvement of threat coverage throughout the organization.	Periodically review application risk profiles at regular intervals to ensure accuracy and reflect current state.	Continuously optimization and automation of your threat modeling methodology. 

SAMM: THREAT ASSESSMENT – APPLICATION RISK PROFILE

<https://owaspsamm.org/model/design/threat-assessment/stream-a/>

- Describes how to assess and achieve the 3 Maturity Levels
 - Level 1: Ability to classify applications according to risk
 - Level 2: Solid understanding of the risk level of your application portfolio
 - Level 3: Timely update of the application classification in case of changes
- Questions to also consider:
 - Are there any specific compliance requirements for the application?
 - What is the sensitivity of the data?
 - What is the reputation impact of the application to your organization?
 - What is the reputation impact of the application to your customers using this application?
 - What is the security knowledge and skills of the developers working on this application?
 - Can external users register and use the application?
 - What are the availability requirements?

Model | Design | Threat Assessment | Application Risk Profile

MATURITY LEVEL 1 MATURITY LEVEL 2 MATURITY LEVEL 3

Benefit

Ability to classify applications according to risk

Activity

Use a simple method to evaluate the application risk, per application, estimating the potential business impact that it poses for the organization in case of an attack. To achieve this, evaluate the impact of a breach in the confidentiality, integrity and availability of the data or service. Consider using a set of 5-10 questions to understand important application characteristics, such as whether the application processes financial data, whether it is internet facing, or whether privacy-related data is involved. The application risk profile tells you whether these factors are applicable and if they could significantly impact the organization.

Next, use a scheme to classify applications according to this risk. A simple, qualitative scheme (e.g. high/medium/low) that translates these characteristics into a value is often effective. It is important to use these values to represent and compare the risk of different applications against each other. Mature highly risk-driven organizations might make use of more quantitative risk schemes. Don't invent a new risk scheme if your organization already has one that works well.

Question

Do you classify applications according to business risk based on a simple and predefined set of questions?

Quality criteria

An agreed-upon risk classification exists

The application team understands the risk classification

The risk classification covers critical aspects of business risks the organization is facing

The organization has an inventory for the applications in scope

Answers

No

Yes, some of them

Yes, at least half of them

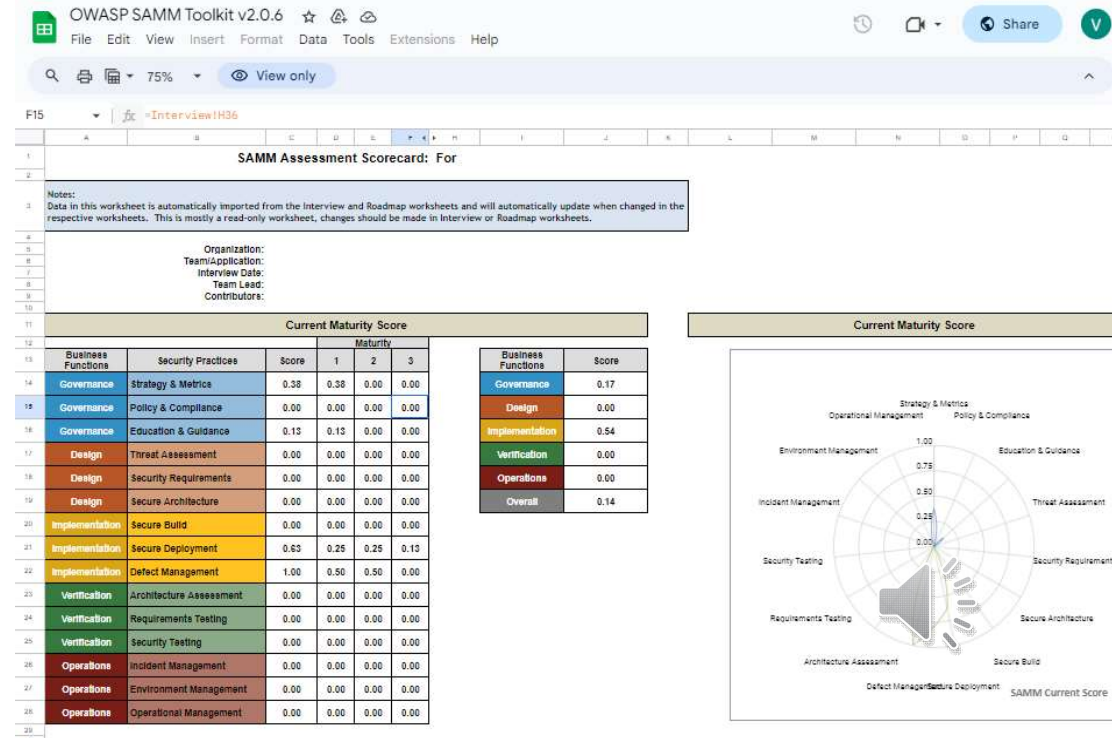
Yes, most or all of them



SAMM ASSESSMENT

<https://owasp samm.org/assessment/>

- SAMM Assessment can be completed:
 - Offline on a spreadsheet
 - SAMM Toolbox, a [Microsoft Excel Toolbox](#) and a [Google Spreadsheet Toolbox](#)
 - A localhost web application
 - <https://github.com/owasp samm/sammwise>
 - An online website (SAMMY)
 - <https://sammy.codific.com/>



THE FIVE W'S OF TESTING

■ What is Testing?

- A procedure intended to establish the quality, performance, or reliability of something, especially before it is taken into widespread use
- There are several criteria to test against based on the application

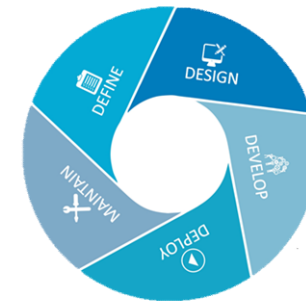


■ Why Perform Testing?

- To discover vulnerabilities that may cause loss of “CIA”
- To determine the gap between existing practices and industry best practices
- To understand the magnitude of resources required to test and maintain software, or to prepare for an audit.

■ When to Test?

- Throughout the Software Development Life Cycle – include security in each of its phases since there is an increased cost to fix security issues later on.



■ Where to Test?

- People – to ensure that there is adequate education and awareness;
- Process – to ensure that there are adequate policies and standards and that people know how to follow these policies;
- Technology – to ensure that the process has been effective in its implementation.

THE FIVE W'S OF TESTING

Who should Test?

- Most importantly, **YOU**, the developer or internal testing team.

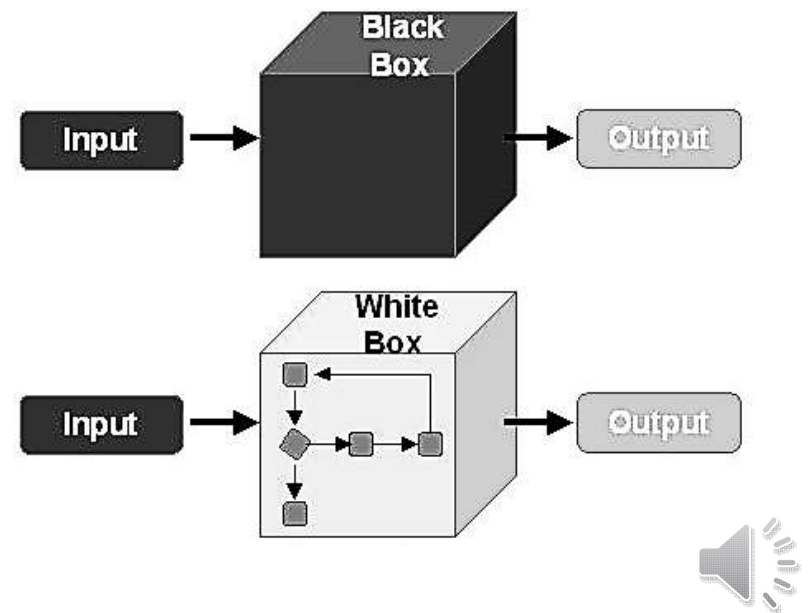


- TT-CSIRT – for applications that facilitate payments online as mandated by the Treasury Under Regulation 31 Of The Exchequer And Audit (Electronic Funds Transfer) Regulations, 2015.
- Other non-payment application assessments can also be requested by the TTCSIRT.
- Third party assessors – there are other private companies that offer services to ensure security.



IMPORTANCE OF INTERNAL TESTING

- Developers and internal teams have a greater understanding about the application and therefore can **quicker** identify relevant concerns
- **Easier** access to proper documentation without the concern of releasing sensitive information to 3rd party assessors; architecture, source code, data-flow diagrams, use cases, etc.
- Access to source code allows increased **visibility** for detecting bugs or issues which may not be obvious when creating test cases.
- Security can be integrated into each phase of the SDLC to allow **earlier** detection and resolution
- Establish **repeatable** tests which can be used to verify application upon change or on future developments



SOFTWARE DEVELOPMENT SECURITY TESTING FRAMEWORK

- The software development security testing framework does not specify the use of any particular development methodology
- The framework comprises of techniques and tasks that are appropriate at various phases of the software development life cycle (SDLC).
- The framework facilitates testing in the early cycles of application development, such as during definition, design, and development
- Testing during the definition, design, development and maintenance phases saves on the costly strategy of waiting until code is completely built.



PHASE I: BEFORE DEVELOPMENT BEGINS

■ Define a SDLC

- Before application development starts, a SDLC must be defined where security is inherent at each stage.

■ Review Policies and Standards

- Ensure that there are appropriate policies, standards, and documentation in place to give development teams guidelines and policies that they can follow.
- For example, PCI-DSS when dealing with credit card information, application's acceptable use policy, privacy policy, ISO, secure coding standards for specific programming languages.
- Document common and predictable issues so there will be fewer decisions that need to be made during the development process.

■ Develop Measurement and Metrics Criteria and Ensure Traceability

- Define criteria that need to be measured to provides visibility into defects in both the process and product.
- For example, consider what interaction on the application needs to be logged, successful/ unsuccessful logins, traffic analysis, incidents, violations, etc.



PHASE 2: DURING DEFINITION AND DESIGN

■ Review Security Requirements

- Define how an application works from a security perspective and test the assumptions made in the requirements to see if there are any gaps in the requirements definitions.
- For example: If there is a security requirement that states that users must be registered before they can get access to an informational section of a website, does this mean that the user must be registered with the system or should the user be authenticated?
- Consider looking at security mechanisms such as:

- User Management
- Accountability
- Authentication
- Session Management
- Authorization
- Transport Security
- Data Confidentiality
- Tiered System Segregation
- Integrity
- Legislative and Standards Compliance



PHASE 2: DURING DEFINITION AND DESIGN (CONT'D)

■ Review Design and Architecture

- Applications should have a documented design and architecture which include items such as models and textual documents. This ensures that the design and architecture includes security functions as defined in the requirements. If flaws are identified at this stage, the design can be easily changed to mitigate the flaws.
- For example, having a firewall illustrated in the network design or a verification process drawn on a flowchart

■ Create and Review UML Models

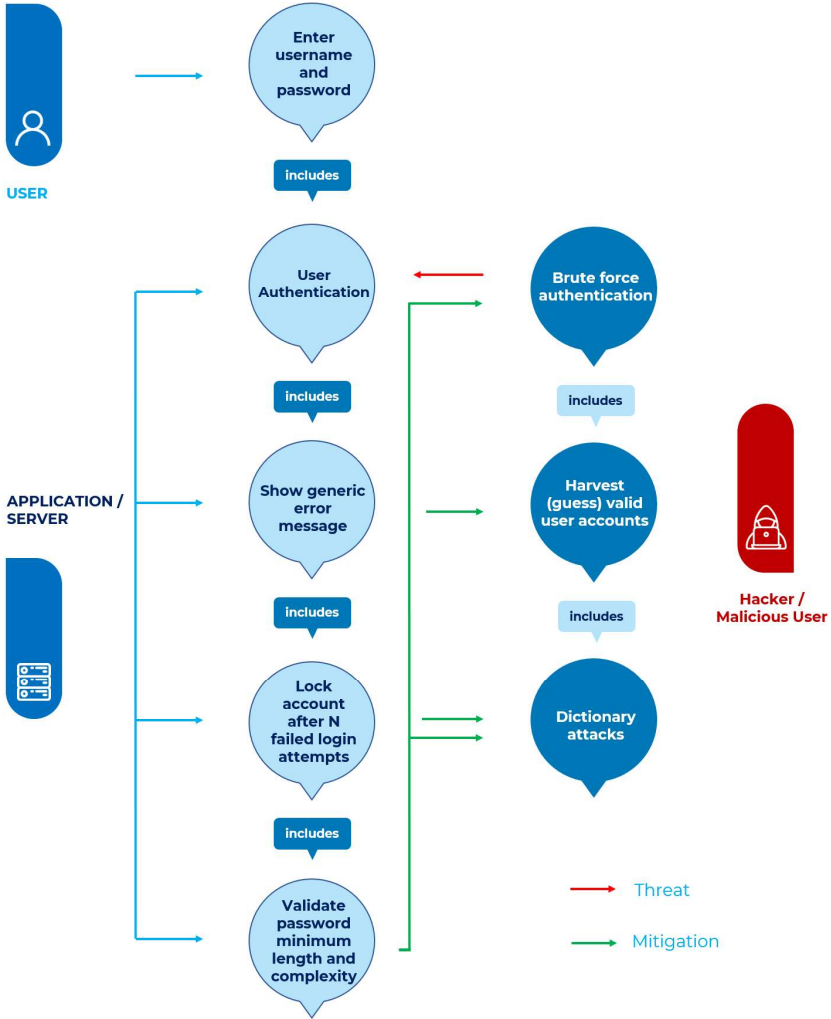
- These models describe how the application works in an easy to follow method and can be distributed to stakeholders who are required to understand processes without reviewing source code or processes in dept.

■ Create and Review Threat Models

- Develop realistic threat scenarios to conduct a threat modeling exercise.
- Analyze the design and architecture to ensure that these threats have been mitigated, accepted by the business, or assigned to a third party
- If threats are identified and there is no mitigation strategies, revisiting the design and architecture would be necessary



Phase 2: During Definition and Design (Cont'd)



PHASE 3: DURING DEVELOPMENT

Development is the implementation of a design. However, many design decisions are made during code development since these are often smaller decisions that were either too detailed to be described in the design, or issues where no policy or standard guidance was offered. To ensure that the developers' decisions were inline with the business interests, the following steps can be done:

■ Code Walkthrough

- A code walkthrough is a high-level look at the code during where the developers can explain the logic and flow of the implemented code.
- The developers would conduct a code walkthrough with the necessary stakeholders; testers, senior developers or managers.
- Providing this walkthrough may uncover issues in terms of process flow or business logic flaws.


■ Code Reviews

- Static code reviews validate the code against a set of checklists, including: Business requirements for CIA, OWASP WSTG or Top 10 checklist, language or framework specific best practices, industry specific checklists.



PHASE 3: DURING DEVELOPMENT (CONT'D)

■ Unit Testing

- The main objective of security tests is to validate that code is being developed in compliance with secure coding standards requirements
- Developers' own coding artifacts (such as functions, methods, classes, APIs, and libraries) need to be functionally validated before being integrated into the application build.
- Unit tests should be defined for use and misuse cases to security test functions, methods and classes.
- Security test suite might include security test cases to validate both positive and negative requirements for security controls such as: identity, authentication & access control, Input validation & encoding, Encryption, User and session management, Error and exception handling, Auditing and logging
- At the component level, security unit tests can validate positive assertions as well as negative assertions, such as errors and exception handling should be caught without leaving the system in an insecure state or state of constant processing. 

PHASE 4: DURING DEPLOYMENT

■ Application Penetration Testing

- Penetration testing the application after it has been deployed provides an additional check to ensure that nothing has been missed.
- The deployed application may be within a development environment which does not mirror the production environment in terms of infrastructure, software or policy & procedures. As a result, the production environment must be tested to ensure that there are no security gaps in the final application.

■ Configuration Management Testing

- It is important to review configuration aspects to ensure changed default setting that may be vulnerable to exploitation.
- For example, change default admin passwords, certificates, configurations with low security settings.



PHASE 5: DURING MAINTENANCE AND OPERATIONS

■ Conduct Operational Management Reviews

- Implement and follow a process which details how the operational side of the application and infrastructure is managed
- This would include such as: checking that the load the systems experience doesn't exceed its capacity, checking uptime, faults/errors which may have caused an outage and similar checks which monitor operations disturbance. IAM changes. Any findings should then be followed up with accordingly

■ Conduct Periodic Health Checks

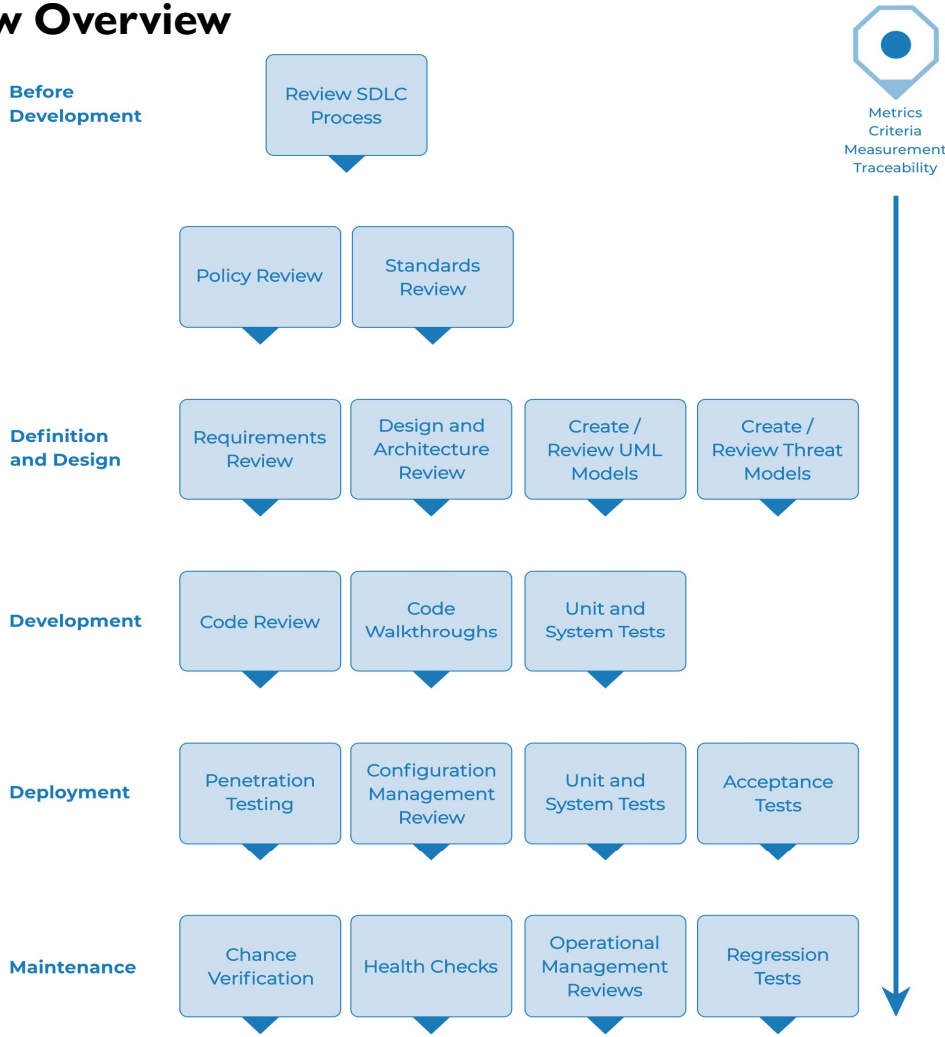
- Ensure that no new security risks have been introduced to either the application or infrastructure
- Periodically check for software updates from vendors and examine the changelogs for information on vulnerabilities or bugs which may affect your product.
- Subscribe to threat feeds which relate to your systems in use

■ Ensure Change Verification

- Changes should go through a testing and approval process to ensure that the change doesn't degrade the security of the system or introduce any interoperability issues.



SDLC Testing Workflow Overview



APPLICATION SECURITY TESTING



OWASP APPLICATION SECURITY VERIFICATION STANDARD (ASVS)

- The OWASP Application Security Verification Standard (ASVS) Project provides a basis for testing web application technical security controls and also provides developers with a list of requirements for secure development.
- **Used as a metric** - Provide application developers and application owners with a yardstick with which to assess the degree of trust that can be placed in their Web applications,
- **Used as guidance** - Provide guidance to security control developers as to what to build into security controls in order to satisfy application security requirements, and
- **Used during procurement** - Provide a basis for specifying application security verification requirements in contracts.

URL: <https://owasp.org/www-project-application-security-verification-standard/>



Application Security Verification Standard 4.0.3

Final

October 2021



OWASP APPLICATION SECURITY VERIFICATION STANDARD (ASVS)

1. Architecture, Design, And Threat Modeling
2. Authentication
3. Session Management
4. Access Control
5. Validation, Sanitization And Encoding
6. Stored Cryptography
7. Error Handling And Logging
8. Data Protection
9. Communications
10. Malicious Code
11. Business Logic
12. Files And Resources
13. API And Web Services
14. Configuration



Application Security Verification Standard 4.0.3

Final

October 2021



OWASP ASVS: FILE AND RESOURCES

Control Objectives

- File Upload
- File Integrity
- File Execution
- File Storage
- File Download
- SSRF Protection

Verify that the application will not accept large files that could fill up storage or cause a denial of service.

Verify that the application checks compressed files (e.g. zip, gz, docx, odt) against maximum allowed uncompressed size and against maximum number of files before decompressing the file.

Verify that a file size quota and maximum number of files per user is enforced to ensure that a single user cannot fill up the storage with too many files, or excessively large files.

Verify that files obtained from untrusted sources are validated to be of expected type based on the file's content.

Verify that user-submitted filename metadata is not used directly by system or framework filesystems and that a URL API is used to protect against path traversal.

Verify that user-submitted filename metadata is validated or ignored to prevent the disclosure, creation, updating or removal of local files (LFI).

Verify that user-submitted filename metadata is validated or ignored to prevent the disclosure or execution of remote files via Remote File Inclusion (RFI) or Server-side Request Forgery (SSRF) attacks.

Verify that the application protects against Reflective File Download (RFD) by validating or ignoring user-submitted filenames in a JSON, JSONP, or URL parameter, the response Content-Type header should be set to text/plain, and the Content-Disposition header should have a fixed filename.

Verify that untrusted file metadata is not used directly with system API or libraries, to protect against OS command injection.

Verify that the application does not include and execute functionality from untrusted sources, such as unverified content distribution networks, JavaScript libraries, node npm libraries, or server-side DLLs.

Verify that files obtained from untrusted sources are stored outside the web root, with limited permissions.

Verify that files obtained from untrusted sources are scanned by antivirus scanners to prevent upload and serving of known malicious content.

Verify that the web tier is configured to serve only files with specific file extensions to prevent unintentional information and source code leakage. For example, backup files (e.g. .bak), temporary working files (e.g. .swp), compressed files (.zip, .tar.gz, etc) and other extensions commonly used by editors should be blocked unless required.

Verify that direct requests to uploaded files will never be executed as HTML/JavaScript content.

Verify that the web or application server is configured with an allow list of resources or systems to which the server can send requests or load data/files from.

OWASP ASVS: ACCESS CONTROL

Control Objectives

- General Access Control Design
- Operational Level Access Control
- Other Access Control Considerations

Verify that the application enforces access control rules on a trusted service layer, especially if client-side access control is present and could be bypassed.

Verify that all user and data attributes and policy information used by access controls cannot be manipulated by end users unless specifically authorized.

Verify that the principle of least privilege exists - users should only be able to access functions, data files, URLs, controllers, services, and other resources, for which they possess specific authorization. This implies protection against spoofing and elevation of privilege.

[[C7](https://owasp.org/www-project-proactive-controls/#div-numbering))

[DELETED, DUPLICATE OF 4.1.3]

Verify that access controls fail securely including when an exception occurs. [[C10](https://owasp.org/www-project-proactive-controls/#div-numbering))

Verify that sensitive data and APIs are protected against Insecure Direct Object Reference (IDOR) attacks targeting creation, reading, updating and deletion of records, such as creating or updating someone else's record, viewing everyone's records, or deleting all records.

Verify that the application or framework enforces a strong anti-CSRF mechanism to protect authenticated functionality, and effective anti-automation or anti-CSRF protects unauthenticated functionality.

Verify administrative interfaces use appropriate multi-factor authentication to prevent unauthorized use.

Verify that directory browsing is disabled unless deliberately desired. Additionally, applications should not allow discovery or disclosure of file or directory metadata, such as Thumbs.db, .DS_Store, .git or .svn folders.

Verify the application has additional authorization (such as step up or adaptive authentication) for lower value systems, and / or segregation of duties for high value applications to enforce anti-fraud controls as per the risk of application and past fraud.



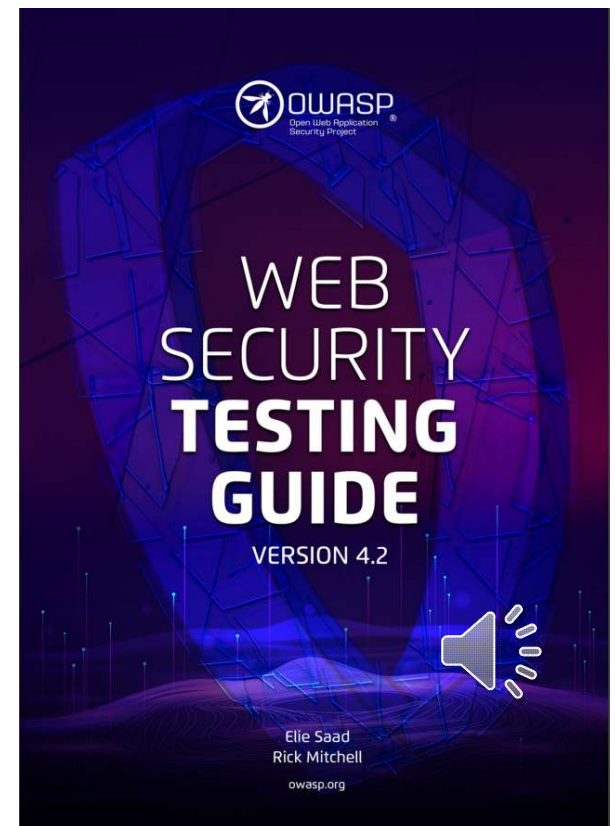
OWASP WEB SECURITY TESTING GUIDE (WSTG)

- The Web Security Testing Guide (WSTG) Project produces the premier cybersecurity testing resource for web application developers and security professionals.
- The WSTG is a comprehensive guide to testing the security of web applications and web services. Created by the collaborative efforts of cybersecurity professionals and dedicated volunteers, the WSTG provides a framework of best practices used by penetration testers and organizations all over the world.
- The TT-CSIRT has adopted the use of this guide and it is fundamental in our approach to assessing web applications.

URLs

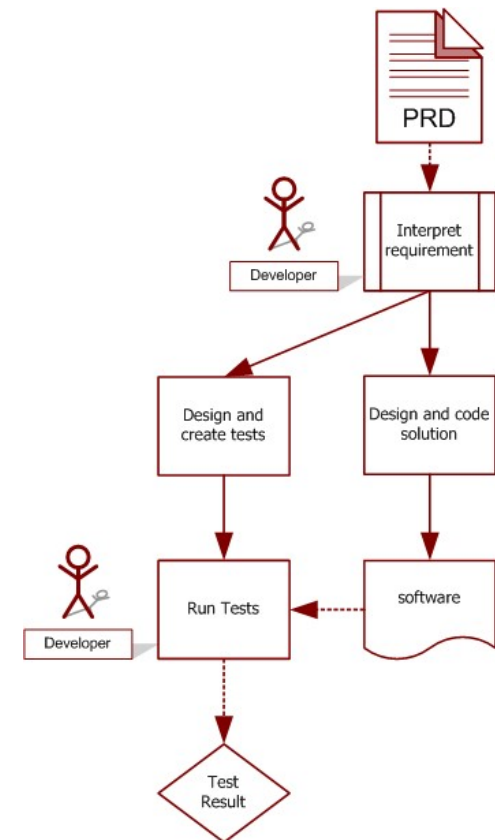
Web view: <https://owasp.org/www-project-web-security-testing-guide/stable/>

PDF Download: <https://github.com/OWASP/wstg/releases/download/v4.2/wstg-v4.2.pdf>



TESTING STRATEGY

- While the SDLC is in progress, testing or threat modeling can be done
- You know what you need to test based on the Application Security Verification Standard
- You can effectively go through the test cases specified in the OWASP WSTG to verify that it is secure.
- OWASP WSTG contains **11** categories with a total of 109 test cases for web application security testing.



OWASP'S WSTG CATEGORIES

OWASP WSTG contains **12** categories covering over 100 test cases.

1. Information Gathering
2. Configuration and Deployment Management Testing
3. Identity Management Testing
4. Authentication Testing
5. Authorization Testing
6. Session Management Testing
7. Input Validation Testing
8. Error Handling
9. Cryptography
10. Business Logic Testing
11. Client-side Testing
12. API Testing



TESTING

- Testing Techniques
 - Passive Testing
 - Active Testing
 - Manual Inspections & Reviews
 - Threat Modeling
 - Code Review
 - Penetration Testing



PASSIVE TESTING

Goals

- To understand the application as a user
- Perform information gathering
- Observe points of entry or critical sections of the application
 - Forms, URLs, features, functions, etc.

For example:

- A tester may find a page at the following URL:
https://www.example.com/login/auth_form
 - This may indicate an authentication form where the application requests a username and password.
- URL parameters also represent access points to the application:
<https://www.example.com/appx?uid=123&permission=admin>



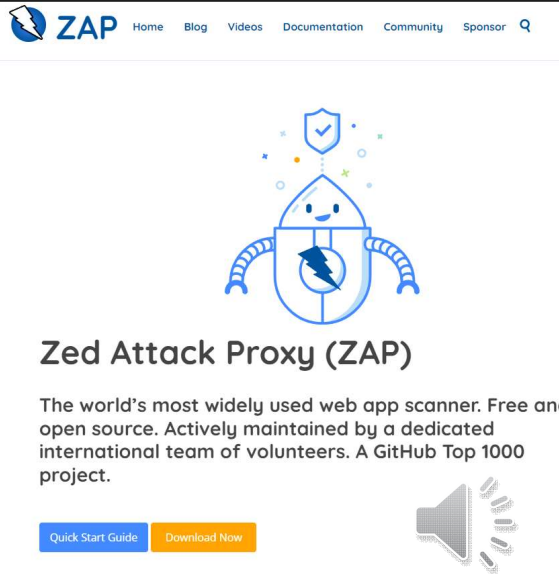
ACTIVE TESTING

Goals

- Active testing aims to probe and evaluate the application actively, simulating potential attacks and identifying vulnerabilities.
- Evaluate security controls, assess attack vectors, pinpoint gaps and weaknesses.
- It involves interacting with the application to assess its security controls, including authentication, authorization, input, error handling and the other categories from OWASP's WSTG

For example:

- In active testing, a tester may attempt to exploit an authentication form by attempting SQL injection or XSS.
- Submitting malicious payloads and attempting to create a loss of CIA



The screenshot shows the homepage of the Zed Attack Proxy (ZAP) website. At the top, there is a navigation bar with the ZAP logo and links for Home, Blog, Videos, Documentation, Community, and Sponsor. The main content area features a large, stylized blue robot character with a shield on its chest and a lightning bolt on its head. Below the robot, the text reads "Zed Attack Proxy (ZAP)" followed by a description: "The world's most widely used web app scanner. Free and open source. Actively maintained by a dedicated international team of volunteers. A GitHub Top 1000 project." At the bottom of the page, there are two buttons: "Quick Start Guide" and "Download Now". A speaker icon is also visible in the bottom right corner.

THREAT MODELING

■ Threat Modeling

- A type of risk assessment for applications that all applications undergo.
- Helps system designers think about the security threats that their systems and applications might face.
- This technique involves:
 - Decomposing the application – use a process of manual inspection to understand how the application works, its assets, functionality, and connectivity.
 - Defining and classifying the assets – classify the assets into tangible and intangible assets and rank them according to business importance.
 - Exploring potential vulnerabilities - whether technical, operational, or managerial.
 - Exploring potential threats – develop a realistic view of potential attack vectors from an attacker’s perspective by using threat scenarios or attack trees.
 - Creating mitigation strategies – develop mitigating controls for each of the threats deemed to be realistic.



THREAT MODELING

■ Threat Modeling

Advantages

- Practical attacker view of the system
- Flexible
- Early in the SDLC

Disadvantages

- Good threat models don't automatically mean good software



MANUAL INSPECTION & REVIEW

■ Manual Inspections & Reviews

- Manual inspections are human reviews.
- It tests the security implications of people, policies, and processes.
- Manual inspections can also include inspection of documentation, secure coding policies, security requirements, and architectural designs.
- Manual inspection and reviews are conducted by analyzing documentation or performing interviews with the designers or system owners.
- Manual reviews are particularly good for testing whether people understand the security process, have been made aware of policy, and have the appropriate skills to design or implement secure applications.

Advantages

- Requires no supporting technology
- Can be applied to a variety of situations
- Flexible
- Promotes teamwork
- Early in the SDLC

Disadvantages

- Can be time-consuming
- Supporting material not always available
- Requires significant human thought and skill to be effective



SOURCE CODE REVIEW

■ Source Code Review

- This is the process of manually checking the source code of a web application for security issues.
- “if you want to know what’s really going on, go straight to the source.”
- Many unintentional but significant security problems are extremely difficult to discover with other forms of analysis or testing, such as penetration testing.
- This remove the guess work of black-box testing.
- Source code reviews can highlight concurrency problems, flawed business logic, access control problems, and cryptographic weaknesses, backdoors, Trojans, Easter eggs, time bombs, logic bombs, and other forms of malicious code.

Advantages

- Completeness and effectiveness
- Accuracy
- Fast (for competent reviewers)

Disadvantages

- Requires highly skilled security aware developers
- Can miss issues in compiled libraries
- Cannot detect runtime errors easily
- The source code actually deployed might differ from the one being analyzed



SAST & DAST

■ **SAST - Static Application Security Testing:**

- **Overview:** SAST is a white-box testing technique that examines the source code, bytecode, or binary code of an application for vulnerabilities without executing it.
- **Advantages:**
 - **Early Detection:** Identifies vulnerabilities during the development phase.
 - **Source Code Analysis:** Analyzes source code for potential security issues.
 - **Reduces Risk:** Mitigates security threats before deployment.
- **Challenges:**
 - **False Positives:** May generate false alarms.
 - **Limited Runtime Data:** Lacks insight into runtime behavior.

■ **DAST - Dynamic Application Security Testing:**

- **Overview:** DAST is a black-box testing technique that assesses an application's security by simulating real-world attacks during runtime.
- **Advantages:**
 - **Real-World Scenarios:** Mimics how an actual attacker would interact with the application.
 - **Accurate Testing:** Identifies vulnerabilities in the deployed application.
 - **No Access to Source Code:** Requires no access to source code or internal structures.
- **Challenges:**
 - **Limited Early Detection:** Typically identifies vulnerabilities post-development.
 - **May Not Cover All Code Paths:** May miss vulnerabilities in rarely used functions.



PENETRATION TESTING

■ Penetration Testing

- Penetration testing is testing a system or application to find security vulnerabilities, without knowing the inner workings of the target itself
- Penetration testing has proven to be effective in network security and applications can be used as a point of entry.
- It can be effective in finding, and then exploiting, known vulnerabilities in specific web technologies

Advantages

- Can be fast
- Requires a relatively lower skill-set than source code review
- Tests the code that is being exposed

Disadvantages

- Too late in the SDLC
- Front-impact testing only



WHICH TESTING TECHNIQUE TO USE?

- A combination of the techniques should be used to cover testing in all phases of the SDLC



WALK BEFORE YOU RUN

- Conducting assessments based on OWASP's Software Assurance Maturity Model, Application Security Verification Standard and Web Security Testing Guide can be overwhelming or challenging
- Challenging based on lacking resources (staff, tools, expert knowledge, etc.)
- Aim to cover OWASP's Top 10 at minimum.



TOP 10




#1: BROKEN ACCESS CONTROL

What is Access Control?

- Access control enforces policy such that users cannot act outside of their intended permissions.
- Access control is only effective in trusted server-side code or server-less API, where the attacker cannot modify the access control check or metadata.
- Failures typically lead to unauthorized information disclosure, modification, or destruction of all data or performing a business function outside the user's limits.

Prevention

- Focus on manual testing
- Have well defined business rules that translate into access control rules
 - Role-based access controls matrix
- Implement default deny – except for public resources
- Disable web server directory listing 
- Log and alert on access events

#2 CRYPTOGRAPHIC FAILURES

Dealing with Cryptographic Failures

- 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,
- Does any data fall 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).
- Is any data transmitted in clear text?
- Is the received server certificate and the trust chain properly validated?
- Encrypt all sensitive data at rest and in transit.
- Use up-to-date and strong standard algorithms, protocols, and keys
- Encrypt all data in transit with secure protocols such as TLS 1.3
- Enforce encryption using directives like HTTP Strict Transport Security (HSTS).
- Disable caching for response that contain sensitive data.
- Do not use legacy protocols such as FTP and SMTP for transporting sensitive data.
- Store passwords using strong adaptive and salted hashing functions such as Argon2, scrypt, bcrypt or PBKDF2. Avoid MD5 and SHA1!




#3 INJECTION

Injection can occur when:

- User-supplied data is not validated, filtered, or sanitized by the application.
- Dynamic queries or non-parameterized calls without context-aware escaping are used directly in the interpreter.
- Maliciously supplied data is used within search parameters to extract additional, sensitive records or to bypass access controls.
- Classic SQLi payload: `' or 1=1;--`

Prevention:

- Perform proper input validation. Positive or "allow list" input validation
- Use a safe API which avoids the use of the interpreter entirely and provides a parameterized interface.
- If a parameterized API is not available, encode/escape special characters in input and output using the specific escape syntax for that interpreter. 
- Validate All User Supplied Input

#4 INSECURE DESIGN

- Secure design is a culture and methodology that constantly evaluates threats and ensures that code is robustly designed and tested to prevent known attack methods.
- Threat modeling should be integrated into refinement sessions (or similar activities); look for changes in data flows and access control or other security controls.
- In the user story development determine the correct flow and failure states, ensure they are well understood and agreed upon by responsible and impacted parties.
- Analyze assumptions and conditions for expected and failure flows, ensure they are still accurate and desirable.
- Determine how to validate the assumptions and enforce conditions needed for proper behaviors.
- Secure design is neither an add-on nor a tool that you can add to software



#4 PREVENTING INSECURE DESIGN

- Establish and use a secure development lifecycle with AppSec professionals to help evaluate and design security and privacy-related controls
- Establish and use a library of secure design patterns or security approved components
- Use threat modeling for critical authentication, access control, business logic, and key flows
- Integrate plausibility checks at each tier of your application (from frontend to backend)
- Write unit and integration tests to validate that all critical flows are resistant to the threat model. Compile use-cases *and* misuse-cases for each tier of your application.
- Segregate layers on the system and network layers depending on the exposure and protection needs
- Limit resource consumption by user or service



#5 SECURITY MISCONFIGURATION

The application might be vulnerable if the application is:

- Missing appropriate security hardening or improperly configured permissions.
- Unnecessary features are enabled or installed
- Default accounts and their passwords are still enabled and unchanged.
- Error handling reveals stack traces or other overly informative error messages
- Security features are disabled or not configured securely.
- The server does not send security headers or directives, or they are not set to secure values.

Prevention:

- Follow a hardening process and establish a secure base line. CIS benchmarks provide resources for most applications and infrastructures.
- This process can minimize the effort required to set up a new secure environment.
- Remove or do not install unused features and frameworks.
- Review and update the configurations appropriate to all security notes, updates, and patches as part of the patch management process
- Segment application architecture to reduce the impact of a breach.
- Sending security directives to clients, e.g., Security Headers.




#6 VULNERABLE AND OUTDATED COMPONENTS

You are likely vulnerable:

- If you do not know the versions of all components you use
- If the software is vulnerable, unsupported, or out of date.
- If you do not scan for vulnerabilities regularly and subscribe to security bulletins related to the components you use.
- If you do not fix or upgrade the underlying platform, frameworks, and dependencies in a risk-based, timely fashion.
- If software developers do not test the compatibility of updated, upgraded, or patched libraries.

Prevention:

- Remove unnecessary components
- Maintain an inventory of assets and their versions
- Monitor assets for vulnerabilities
- Obtain software from trusted sources only
- Have a plan that includes prioritization of critical security patches 

#7 IDENTIFICATION AND AUTHENTICATION FAILURES

There may be authentication weaknesses if the application:

- Permits automated attacks such as credential stuffing, brute force and others
- Permits default, weak, or well-known passwords, such as "Password1" or "admin/admin".
- Uses weak or ineffective credential recovery and forgot-password processes, such as "knowledge-based answers," which cannot be made safe.
- Uses plain text, encrypted, or weakly hashed passwords data stores
- Has missing or ineffective multi-factor authentication.
- Exposes session identifier in the URL.
- Reuse session identifier after successful login.
- Does not correctly invalidate Session IDs.

Prevention:

- implement multi-factor authentication
- Do not use default credentials, particularly for admin users.
- Implement weak password checks,
- Align password length, complexity, and rotation policies with modern password policies.
- Ensure registration and credential recovery, are hardened against account enumeration attacks.
- Limit or increasingly delay failed login attempts, but be careful not to create a denial of service scenario.
- Log all failures and alert administrators when attacks are detected.
- Use server side built-in session management



#8 SOFTWARE AND DATA INTEGRITY FAILURES

Software and data integrity failures relate to

- Code and infrastructure that does not protect against integrity violations.
- An insecure CI/CD pipeline can introduce the potential for unauthorized access, malicious code, or system compromise.
- Auto-updates are downloaded without sufficient integrity verification and applied to the previously trusted application.
- Attackers could potentially upload their own updates to be distributed and run on all installations.
- Another issue is insecure deserialization where objects or data are encoded or serialized into a structure that an attacker can see and modify.

Prevention:

- Use digital signatures or similar mechanisms to verify the software or data is from the expected source and has not been altered.
- Ensure libraries and dependencies, such as npm or Maven, are consuming trusted repositories.
- Ensure that a software supply chain components do not contain known vulnerabilities
- Have a review process for code and configuration changes to minimize the chance that malicious code or configuration.
- Ensure that unsigned or unencrypted serialized data is not sent to untrusted clients without some form of integrity check or digital signature to detect tampering or replay of the serialized data

#9 SECURITY LOGGING AND MONITORING FAILURES

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 security testing tools (such as OWASP ZAP) do not trigger alerts.
- The application cannot detect, escalate, or alert for active attacks in real-time or near real-time.

Prevention

- Ensure all login, access control, and server-side input validation failures can be logged with sufficient user context to identify suspicious or malicious accounts and held for enough time to allow delayed forensic analysis.
- Ensure that logs are generated in a format that log management solutions can easily consume.
- Ensure log data is encoded correctly to prevent injections or attacks on the logging or monitoring systems.
- Ensure high-value transactions have an audit trail with integrity controls to prevent tampering or deletion, such as append-only database tables or similar.
- DevSecOps teams should establish effective monitoring and alerting such that suspicious activities are detected and responded to quickly.
- Establish or adopt an incident response and recovery plan, such as National Institute of Standards and Technology (NIST) 800-61r2 or later.



#10 SERVER-SIDE REQUEST FORGERY

- SSRF flaws occur whenever a web application is fetching a remote resource without validating the user-supplied URL.
- It allows an attacker to coerce the application to send a crafted request to an unexpected destination, even when protected by a firewall, VPN, or another type of network access control list (ACL).
- As modern web applications provide end-users with convenient features, fetching a URL becomes a common scenario.
- Port scan internal servers – If the network architecture is unsegmented, attackers can map out internal networks and exploit or abuse internal services
- Sensitive data exposure – Attackers can access local files or internal services to gain sensitive information such as `file:///etc/passwd`

Prevention

From Network layer

- Segment remote resource access functionality in separate networks to reduce the impact of SSRF
- Enforce “deny by default” firewall policies or network access control rules to block all but essential intranet traffic.

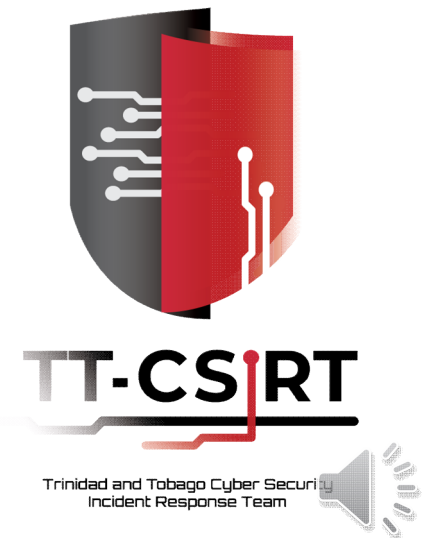
From Application layer:

- Sanitize and validate all client-supplied input data
- Enforce the URL schema, port, and destination with a positive allow list
- Do not send raw responses to clients
- Disable HTTP redirections



WHAT NEXT?

- Start testing your own applications!
- Use the knowledge you've gained for your next development project
- Completing a full OWASP based assessment can be overwhelming but TTCSIRT is here to help
- Use tools like vulnerability scanners, Zed Attack Proxy (ZAP), OpenVAS, Code scanners (GitHub's CodeQL, Snyk, etc)
- Remember automated scans are not always accurate and lacks context. Manual testing fills those gaps



TTCSIRT ASSISTANCE

- We encourage all Trinidad and Tobago Ministries, Divisions and Agencies to utilize our security assurance services.
- Other organization within Trinidad and Tobago may also access our services based on available resources
- Engagements can be initiated by emailing contacts@ttcsirt.gov.tt



TT-CSIRT

Trinidad and Tobago Cyber Security
Incident Response Team



Thank you!

