

# OWASP Application Security Verification Standard

February 2019 Update

# Jim Manico



- Former OWASP Board Member
- OWASP ASVS Lead Author
- OWASP Proactive Controls Lead Author
- OWASP Cheatsheet Series Project Manager
- Kauai, Hawaii Resident

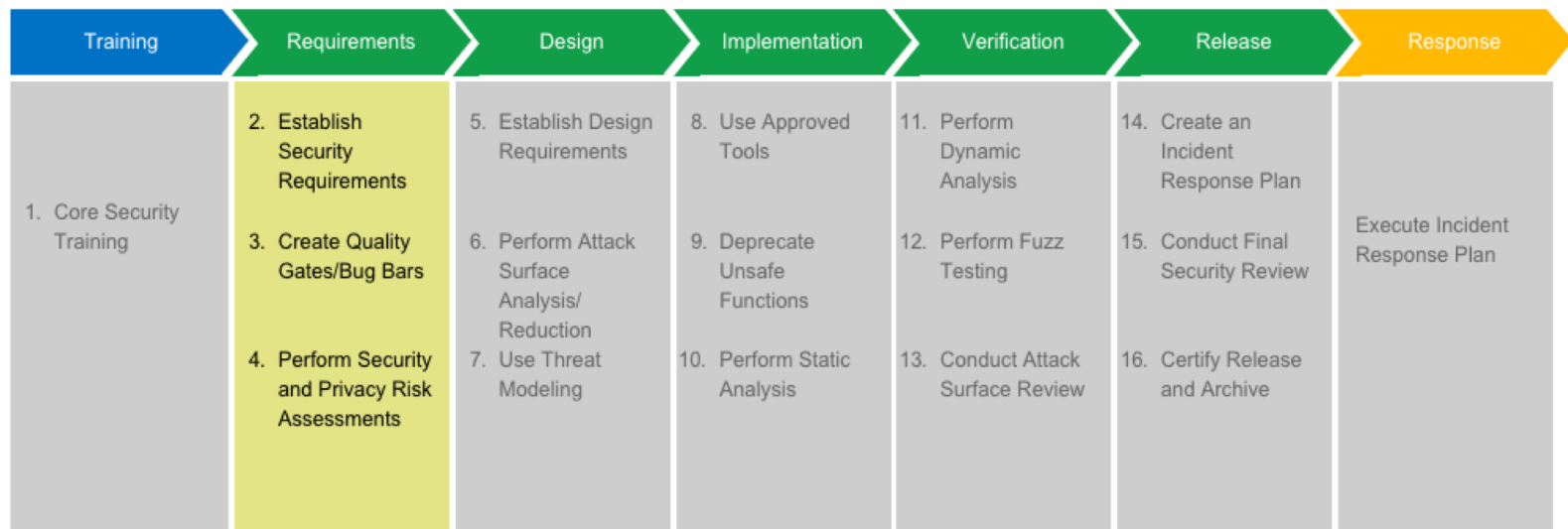


# Secure SDLC

# Microsoft SDL for waterfall

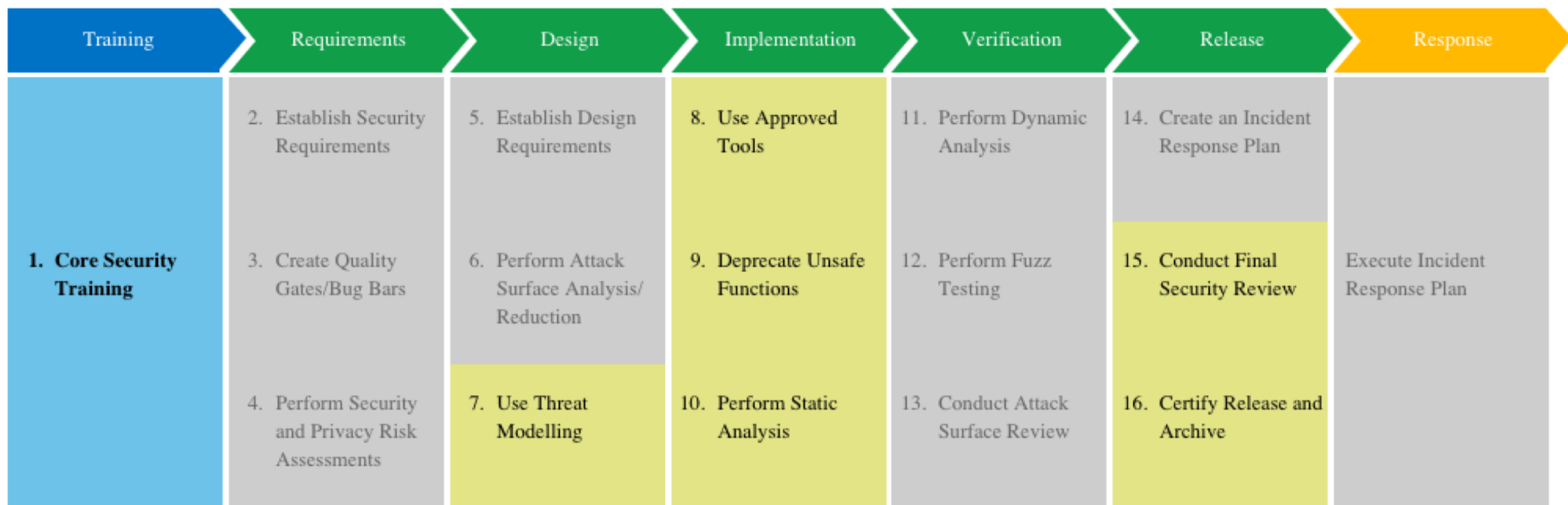
## SDL Process: Requirements

The project inception phase is the best time for a development team to consider foundational security and privacy issues and to analyze how to align quality and regulatory requirements with costs and business needs.



### SDL Practice #2: Establish Security and Privacy Requirements

# Microsoft SDL for Agile





# OWASP

Open Web Application  
Security Project

The Open Web Application Security Project (OWASP) is a **501(c)(3) worldwide not-for-profit charitable organization** focused on improving the security of software.

Our mission is to **make software security visible**, so that individuals and organizations worldwide can make informed decisions about true software security risks.

Everyone is free to participate in OWASP and all of our materials are **available under a free and open software license**. You'll find everything about OWASP linked from our wiki and current information on our OWASP Blog.

<https://owasp.org>

# The OWASP Top Ten The Good

- Project members include a **variety of security experts from around the world** who have shared their expertise to produce this list.
  - Andrew van der Stock
  - Neil Smithline
  - Torsten Gigler
  - Brian Glas
- Significant public comments and conversation on Top Ten 2017 choices  
<https://github.com/OWASP/Top10/tree/master/2017>
- Frequently cited application security awareness document

# The OWASP Top Ten Struggles

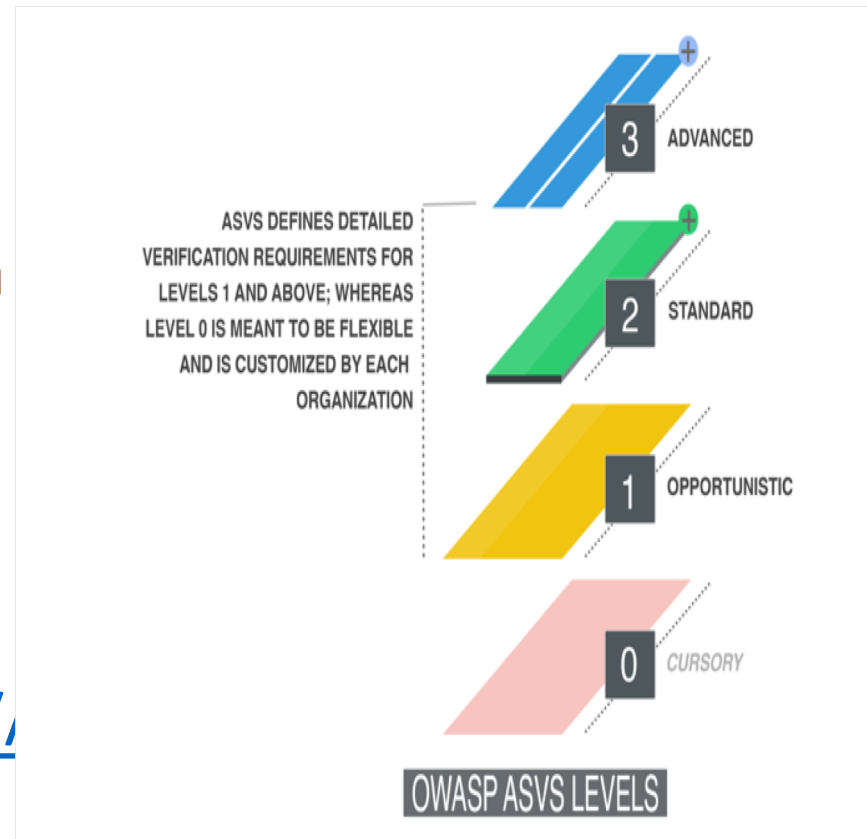
- The OWASP Ten has a history of vendor shenanigans that dates back a decade.
- The OWASP Top 10 list is meant to **spread awareness** regarding Web Security issues. It is not a standard. I'm looking at you *PCI-DSS and others who incorrectly list it as so.*
- The OWASP Top Ten is not a comprehensive list of web security risks.
- The OWASP Top Ten does not go into detail about how to fix or prevent these issues.



**So what standard can we use for  
web applications and webservice  
security?**

# OWASP ASVS 4.0!

- First application security standard by developers for developers!
- Defines three risk levels with 200+ controls.
- Similar to ISO 27034
- To be released in under 2 weeks!
- <https://github.com/OWASP/ASVS/tree/master/4.0/en>



# What will be new in 4.0?

- Moving all of authentication requirements inline with NIST 800-63-3
- Changing session requirements to acknowledge new world of JWT's and stateless mechanisms
- Removing mobile requirements due to MASVS
- Lots of small edits and clarifications on requirement language
- Collapsing many requirements that duplicate concepts
- Triaging hundreds of comments from the field
- Moving to markdown for primary text
- Moar GDPR
- <https://github.com/OWASP/ASVS/tree/master/4.0/en>

# Application Security Verification Standard 4.0

| #          | Description  | L1 | L2 | L3 | CWE | NIST    | &sect; |
|------------|--|----|----|----|-----|---------|--------|
| **2.1.1**  | Verify that user set passwords are at least 12 characters in length.   | ✓  | ✓  | ✓  | 521 |         |        |
| 5.1.1.2    |  |    |    |    |     |         |        |
| **2.1.2**  | Verify that passwords 64 characters or longer are permitted.   | ✓  | ✓  | ✓  | 521 | 5.1.1.2 |        |
| **2.1.3**  | Verify that passwords can contain spaces and truncation is not performed. Consecutive multiple spaces MAY optionally be coalesced.   | ✓  | ✓  | ✓  | 521 | 5.1.1.2 |        |
| **2.1.4**  | Verify that Unicode characters are permitted in passwords. A single Unicode code point is considered a character, so 8 emoji or 64 kanji characters should be valid and permitted.   | ✓  | ✓  | ✓  | 521 |         |        |
| 5.1.1.2    |  |    |    |    |     |         |        |
| **2.1.5**  | Verify users can change their password.  | ✓  | ✓  | ✓  | 620 | 5.1.1.2 |        |
| **2.1.6**  | Verify that password change functionality requires the user's current and new password.  | ✓  | ✓  | ✓  | 620 | 5.1.1.2 |        |
| **2.1.7**  | Verify that passwords submitted during account registration, login, and password change are checked against a set of breached passwords either locally or via API, either using a zero knowledge proof or otherwise ensuring that the plain text password is not sent or used in verifying the breach status of the password. If the password is breached, the application must require the user to re-choose a non-breached password. | ✓  | ✓  | ✓  | 521 | 5.1.1.2 |        |
| **2.1.8**  | Verify that a password strength meter is provided to help users set a stronger secret.   | ✓  | ✓  | ✓  | 521 | 5.1.1.2 |        |
| **2.1.9**  | Verify that there are no password composition rules limiting the type of characters permitted. There should be no requirement for upper or lower case or numbers or special characters.  | ✓  | ✓  | ✓  | 521 | 5.1.1.2 |        |
| **2.1.10** | Verify that there are no periodic credential rotation or password history requirements.  | ✓  | ✓  | ✓  | 263 | 5.1.1.2 |        |

# Application Security Verification Standard 4.0

## Level 1: Opportunistic



- Minimum required for all software
- Mostly automatable
- Easy to discover
- Straight forward developer fixes
- Not enough for high risk apps

# Application Security Verification Standard 4.0

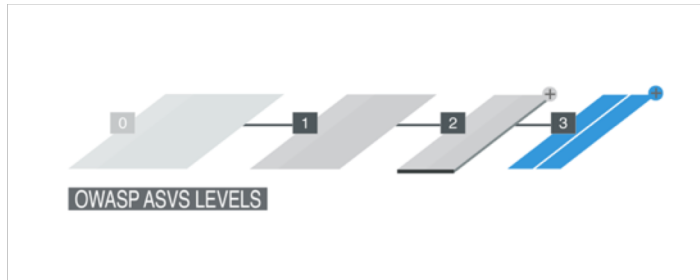
## Level 2: Standard



- Suitable for sensitive data
- About 75% testable
- Somewhat test automatable
- Moderate to complex developer security challenges

# Application Security Verification Standard 4.0

## Level 3: Advanced



- Suitable for critical apps
- Mostly testable, but many more manual verifications required
- Not amenable to automation
- Significant developer challenges

# How to fork and use ASVS

Building the Application Security  
Verification Standard into your SDLC

**May the Fourth  
be with you!**





# ASVS Anti-Patterns

- Security leaders mailing the ASVS document to the development teams telling them "security says you have to follow this now. Good luck!"
- Avoid engaging developers on ASVS items before making it policy
- Using ASVS out of the box without customizing it for your organization
- Setting ASVS as a standard in way where it's never used, never read or never considered. But given to customers.

# ASVS Effective Adoption

- The goal of ASVS adoption is for developers to actively use it in their development and architectural work every day.
- Work with developers early on in forking ASVS.
- Let developers lead in version 1 as to what requirements will be accepted by the team.
- Like any complex legislation, just get it in there and modify it over time after version 1.

# Writing Unit Tests Using ASVS

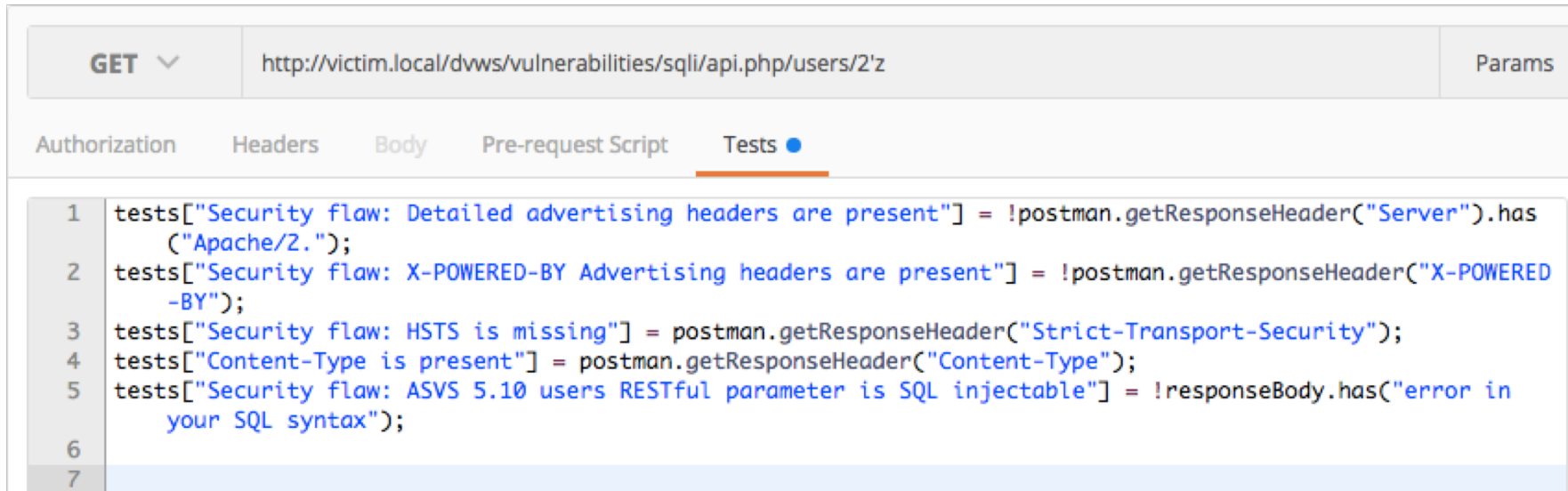
Write unit tests to validate your application each and every build

Allows penetration testers to concentrate on difficult to automate tests, such as business logic flaws, access control issues, and things you forgot in the unit tests



# Writing Integration Tests

Integration tests can be written using Postman, Selenium, OWASP Zap API



The screenshot shows the Postman interface for a GET request. The URL is `http://victim.local/dvws/vulnerabilities/sql/2/z`. The 'Tests' tab is selected, displaying the following JavaScript test script:

```
1 tests["Security flaw: Detailed advertising headers are present"] = !postman.getResponseHeader("Server").has("Apache/2.");
2 tests["Security flaw: X-POWERED-BY Advertising headers are present"] = !postman.getResponseHeader("X-POWERED-BY");
3 tests["Security flaw: HSTS is missing"] = postman.getResponseHeader("Strict-Transport-Security");
4 tests["Content-Type is present"] = postman.getResponseHeader("Content-Type");
5 tests["Security flaw: ASVS 5.10 users RESTful parameter is SQL injectable"] = !responseBody.has("error in your SQL syntax");
6
7
```

# v1 Architecture, design and threat modelling

## Design security in

Ensure that a verified application satisfies the following high level requirements:

- **At level 1**, components of the application are identified and have a reason for being in the app
- **At level 2**, the architecture has been defined and the code adheres to the architecture
- **At level 3**, the architecture and design is in place and is effective at achieving necessary security goals

# v2 Authentication

Authentication is the act of establishing identity.

Ensure that a verified application satisfies the following high level requirements:

- **Verifies the digital identity of the sender of a communication.**
- Ensures that only those authorized are able to authenticate and credentials are transported in a secure manner.
- Changed in 4.0 to be in alignment with NIST 800-63b

# v3 Session management

Ensure that a verified application satisfies the following high level session management requirements:

- Sessions are unique to each individual and cannot be guessed or shared
- Sessions are invalidated when no longer required and timed out during periods of inactivity
- This category was changed significantly to address new forms of stateless session management (JWT's)



# Access control

Acting as admin since 1998



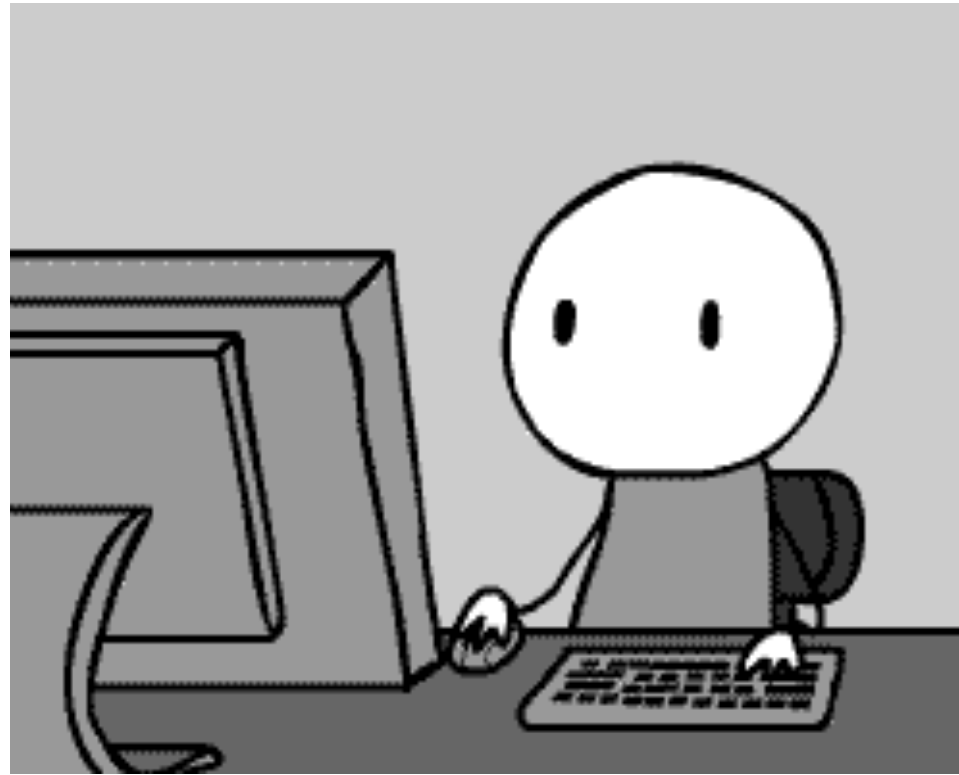
# v4 Access Control

Ensure that a verified application satisfies the following high level requirements:

- Business requirements for access control policy are understood.
- Users are associated with a well-defined set of roles and privileges.
- Role and permission metadata is protected from replay or tampering.
- Post 4.0 release new access control methods such as capabilities need to be addressed.

# Malicious input handling

Inject all the things



# v5 Validation, Sanitization and Encoding

Ensure that a verified application satisfies the following high level requirements:

- All input is validated to be correct and fit for the intended purpose.
- The various and complex forms of XSS defense are addressed for UI security.
- The various forms of injection are handled correctly.

# v6 Cryptography at rest

Ensure that a verified application satisfies the following high level requirements:

- Proper data classification is achieved
- Strong cryptographic architectures and algorithms are in use
- Suitable random number generator is used when randomness is required.
- That access to keys is managed in a secure way with secrets management.

# v7 Error handling and logging

High quality logs will often contain sensitive data, and must be protected as per local data privacy laws or directives. This should include:

- Not collecting or logging sensitive information if not specifically required.
- Ensuring all logged information is handled securely and protected as per its data classification.
- Ensuring that logs are not forever, but have an absolute lifetime that is *as short as possible*.

# v8 Data protection

Ensure that a verified application satisfies the following high level data protection requirements:

- **Confidentiality:** Data should be protected from unauthorised observation or disclosure both in transit and when stored.
- **Integrity:** Data should be protected being maliciously created, altered or deleted by unauthorized attackers.
- **Availability:** Data should be available to authorized users as required

# v9 Communications security

Ensure that a verified application satisfies the following high level requirements:

- TLS or strong encryption is always used, regardless of the sensitivity of the data being transmitted
- The most recent, leading configuration advice is used to enable and order preferred algorithms and ciphers
- Weak or soon to be deprecated algorithms and ciphers are ordered as a last resort
- Deprecated or known insecure algorithms and ciphers are disabled

# v10 Malicious Code

Ensure that a verified application satisfies the following high level requirements:

- Malicious activity is handled securely and properly as to not affect the rest of the application.
- Do not have time bombs or other time based attacks built into them
- Do not “phone home” to malicious or unauthorized destinations
- Applications do not have back doors, Easter eggs, salami attacks, or logic flaws that can be controlled by an attacker



# v11 Business Logic

Ensure that a verified application satisfies the following high level requirements:

- The business logic flow is sequential, processed in order, and cannot be bypassed.
- Business logic includes limits to detect and prevent automated attacks, such as continuous small funds transfers.
- High value business logic flows have considered abuse cases and malicious actors, and have protections against **spoofing, tampering, repudiation, information disclosure, and elevation of privilege attacks.**

# v12 Files and Resources

Ensure that a verified application satisfies the following high level requirements:

- Untrusted file data should be handled accordingly and in a secure manner
- Obtained from untrusted sources are stored outside the webroot and limited permissions.

# v13 API Security

Ensure that a verified application that uses RESTful or SOAP based web services has:

- Adequate authentication, session management and authorization of all web services
- Input validation of all parameters that transit from a lower to higher trust level
- JSON and XML handling

# v14 Configuration

Ensure that a verified application has:

- A secure, repeatable, automatable build environment.
- Hardended third party library, dependency and configuration management such that out of date or insecure components are not included by the application.
- A secure-by-default configuration, such that administrators and users have to weaken the default security posture.

