Foreword

The primary objective of this document is to establish a robust and consistent framework for client verification processes. It addresses the challenges posed by the ever-evolving digital landscape, where threats of fraud, identity theft, and cybercrime are constantly evolving. By implementing the guidelines and protocols outlined herein, organizations can mitigate risks, enhance security measures, and foster trust among their client base.

This standard encompasses a wide range of topics, including identity proofing techniques, data validation methodologies, risk assessment strategies, and compliance with relevant regulations and industry standards. It provides detailed guidance on the implementation of multi-factor authentication, biometric verification, and other advanced security measures to ensure the highest levels of client verification.

Furthermore, the document emphasizes the importance of maintaining data privacy and adhering to relevant data protection laws and regulations. It outlines best practices for handling sensitive client information, ensuring that the verification processes are conducted in a secure and ethical manner.

Regular updates and revisions to this standard will be undertaken to reflect the latest advancements in technology, emerging threats, and evolving regulatory requirements. This approach ensures that the client verification processes remain current, effective, and aligned with industry best practices.

By adopting and adhering to the guidelines set forth in this thick client verification standard documentation, organizations can demonstrate their commitment to maintaining the integrity of their operations, safeguarding client data, and fostering a secure and trustworthy digital environment.

About the Standard

The OWASP Thick Client Application Security Verification Standard (TASVS) is the industry standard for thick client application security. It provides a comprehensive set of security controls that can be used to assess the security of thick client applications. The standard covers the key components of a thick client's attack surface including threat modelling, code quality, configuration, storage, cryptography and network communication.

Authors

Dave Hanson

Dave is a senior application security engineer at Bentley Systems with hands-on testing experience of a plethora of tech stacks across web, mobile and thick clients.

John Cotter

John is a distinguished security architect at Bentley Systems with many years of experience in the security industry with a passion for system architecture, threat modelling and sharing his knowedge to educate others.

Contributors

- Einaras Bartkus
- Thomas Chauchefoin

Special Thanks

We'd also like to thank the authors and contributors of the ASVS and MASVS standards who showed us the way, we liked their work so much that we "borrowed" some of their ideas!

Sponsors

We would like to express our gratitude to Bentley Systems, their support and allocation of dedicated work time enabled the development of this verification standard.

Bentley Systems is dedicated to advancing the world's infrastructure through software solutions and services. Established in 1984, the company serves engineers and professionals responsible for designing, constructing, and operating sustainable infrastructure essential for quality of life globally.

Learn more about Bentley Systems

Copyright and License

The Thick Client Application Security Verification Standard

This verification standard should be useful to anyone trying to:

- Develop and maintain secure applications.
- Evaluate the security of applications.

Thick Client Application Security Model

The standard is divided into various groups labelled `TASVS-{word}` that represent the most critical areas of a thick clients attack surface. These control groups are divided into sub groups, labeled `TASVS-{word}-{digit}`. Each of these control groups contains individual controls labeled `TASVS-{word}-{digit}.{digit}`, which provide specific guidance on the particular security measures that need to be implemented to meet the standard. For example `TASVS-ARCH-1.1`.

Top Group

- TASVS-ARCH - Architecture & threat modelling.
- TASVS-CODE - Code quality and exploit mitigation.
- TASVS-CONF - Configuration and building.
- TASVS-CRYPTO - Cryptography.
- TASVS-NETWORK - Communication and privacy.
- TASVS-STORAGE - Data storage considerations.
- TASVS-FUTURE - Coming soon General considerations for future cloud adoption strategies.

Sub Group

- TASVS-ARCH-1 - Threat Modeling
- TASVS-CODE-1 - Server Side
- TASVS-CODE-2 - Client Side - Signing and Integrity
- TASVS-CODE-3 - Client Side - Static Code Analysis
- TASVS-CODE-4 - Client Side - Validation, Sanitization and Encoding
- TASVS-CODE-5 - Client Side - Business Logic
- TASVS-CODE-6 - Client Side - Fuzzing
- TASVS-CODE-7 - Client Side - Privilege and Rule of two
- TASVS-CONF-1 - General Configuration Checks
- TASVS-CRYPTO-1 - Communication
- TASVS-CRYPTO-2 - Storage
- TASVS-CRYPTO-3 - General
- TASVS-NETWORK-1 - Data leakage
- TASVS-NETWORK-2 - Licensing & Authentication Servers
- TASVS-NETWORK-3 - Piracy Detection
- TASVS-NETWORK-4 - Connected Services
- TASVS-STORAGE-1 - Sensitive Information Review
- TASVS-STORAGE-2 - DLL Hijacking

Application Security Verification Levels

We folow the same levelling methodology as the Web Application Security Verification Standard. It defines three security verification levels, with each level increasing in depth.

- L1 - TASVS Level 1 is for low assurance levels and is completely verifiable through penetration testing.
- L2 - TASVS Level 2 is for applications that contain sensitive data, which requires protection and is the recommended level for most apps.
- L3 - TASVS Level 3 is intended for the most critical applications, such as those handling high-value transactions, containing sensitive medical data, or any application demanding the highest level of trust.

TASVS-ARCH: Architecture & Threat Modelling

Control Objective

Architecture and threat modeling are inextricably linked. Threat modeling informs architectural decisions, while architecture provides the context for identifying and addressing threats systematically. This symbiotic relationship is essential for delivering secure software and systems that meet their intended design goals.

Testing Checklist

TASVS-ID	Description	L1	L2	L3
TASVS-ARCH-1	Threat Modeling			
TASVS-ARCH-1.1	Completed a low fidelity threat model for thick client.	X	X	X
TASVS-ARCH-1.2	Completed a high fidelity threat model for thick client which is in currently in production.		X	X
TASVS-ARCH-1.3	Threat model includes server-side components and dependencies (cloud APIs, OIDC provider, file storage, etc.).		X	X
TASVS-ARCH-1.4	Threat modeling process included all phases (system modeling, auto-threat identification, manual threat identification, threat mitigation).		X	X
TASVS-ARCH-1.5	Threat model checked-in to source code repository.	X	X	X
TASVS-ARCH-1.6	Threat model updated regularly as part of a documented process within development team's SSDLC.		X	X

TASVS-CODE: Code Quality and Exploit Mitigation

Control Objective

To ensure that the application's source code is developed and maintained in a manner that minimizes the introduction of security vulnerabilities.

Testing Checklist

TASVS-ID	Description	L1	L2	L3
TASVS-CODE-1	Server Side			
TASVS-CODE-1.1	If the thick client relies on server side API's or services, defer that testing to the appropriate application security verification standard (ASVS). If testing has begun using that guide, mark this item as reviewed.	X	X	X
TASVS-CODE-2	Client Side - Signing and Integrity			
TASVS-CODE-2.1	Making Sure that the thick client binary is properly signed	X	X	X
TASVS-CODE-2.2	Testing File Integrity Checks	X	X	X
TASVS-CODE-2.3	Testing Runtime Integrity Checks	X	X	X
TASVS-CODE-2.4	The client has been built in release mode, with settings appropriate for a release build.	X	X	X
TASVS-CODE-2.5	Enable framework applicable security features like byte-code minification and stack protection.	X	X	X
TASVS-CODE-3	Client Side - Static Code Analysis			
TASVS-CODE-3.1	All third party components used by the thick client, such as libraries and frameworks, are identified, and checked for known vulnerabilities and are up to date, they should not be unsupported, deprecated or legacy.	X	X	X
TASVS-CODE-3.2	Search the source code for cases where exceptions are thrown and not properly handled. E.g for C# use 'findstr /N /s /c:"throw;" *.cs'. Also be on the lookout to see if the exception allows a bypass of authentication or some other critical operation.	X	X	X
TASVS-CODE-3.3	Perform binary static analysis. (verify that the binaries are compiled with the latest compiler, examine compilation settings and validates binary signing)	X	X	X
TASVS-CODE-3.4	Depending on the language(s) in use, choose appropriate static application security testing (SAST) tooling to analyze source code to identify vulnerabilities.	X	X	X
TASVS-CODE-3.5	If applicable, ensure any internal tooling, policies and test cases are being implemented and evaluated correctly.	X	X	X
TASVS-CODE-3.6	Identify and clear out any unused code. Remember, it stays in the source code repository history if needed later. Use README/changelog files for preserving high value historical context or deprecated details. Do not keep obsolete project repositories, consider archiving the repository.	X	X	X
TASVS-CODE-4	Client Side - Validation, Sanitization and Encoding			
TASVS-CODE-4.1	Untrusted data through features such as macros or templating is protected from code & command injection attacks. Where there is no alternative, any user input being included must be sanitized or sandboxed before being executed.	X	X	X
TASVS-CODE-4.2	Verify that the application protects against OS command injection.	X	X	X
TASVS-CODE-4.3	Verify that unstructured data is sanitized to enforce safety measures such as allowed characters and length.	X	X	X

TASVS-ID	Description	L1	L2	L3
TASVS-CODE-4.4	Verify that the application correctly restricts XML parsers to only use the most restrictive configuration possible and to ensure that unsafe features such as resolving external entities are disabled to prevent XML eXternal Entity (XXE) attacks.	X	X	X
TASVS-CODE-4.5	Verify that the application uses memory-safe string, safer memory copy and pointer arithmetic to detect or prevent stack, buffer, or heap overflows.	X	X	X
TASVS-CODE-4.6	Verify that format strings do not take potentially hostile input, and are constant.	X	X	X
TASVS-CODE-4.7	Verify that sign, range, and input validation techniques are used to prevent integer overflows.	X	X	X
TASVS-CODE-4.8	Verify that serialized objects use integrity checks or are encrypted to prevent hostile object creation or data tampering.	X	X	X
TASVS-CODE-4.9	Verify that deserialization of untrusted data is avoided or is protected in both custom code and third-party libraries (such as JSON, XML and YAML parsers).	X	X	X
TASVS-CODE-4.10	Is the thick client's handling of spawning processes done securely. (Validating and sanitizing process arguments)	X	X	X
TASVS-CODE-4.11	Verify that user-submitted filename metadata is not used directly by system or framework filesystems to protect against path traversal. One example is "ZipSlip" style attacks.	X	X	X
TASVS-CODE-4.12	In unmanaged code, memory is allocated, freed and used securely.	X	X	X
TASVS-CODE-5	Client Side - Business Logic			
TASVS-CODE-5.1	No sensitive data, such as passwords or pins, is exposed through the user interface.	X	X	X
TASVS-CODE-5.2	Check for design practices that trick or manipulate users into making choices they would not otherwise have made and that may cause harm. AKA "deceptive patterns". See https://www.deceptive.design/types for examples.	X	X	X
TASVS-CODE-5.3	Is the thick client only using workflows that do not violate common security advice?	X	X	X
TASVS-CODE-5.4	Verify that the attack surface is reduced by sandboxing or encapsulating third party libraries to expose only the required behaviour into the application.	X	X	X
TASVS-CODE-5.5	Check that import files cannot be abused in	X	X	X
TASVS-CODE-5.6	If the thick client registers a URL handler / protocol handler, verify that it can't trigger dangerous action or introduces common vulnerabilities (memory corruption, command and argument injection, etc.)	X	X	X
TASVS-CODE-6	Client Side - Fuzzing			
TASVS-CODE-6.1	Perform "dumb fuzzing" of the application with randomised input to try to cause a crash.	X	X	X
TASVS-CODE-6.2	Perform "smart fuzzing". Intelligently generate test cases that maximize code coverage and explore complex program states to increasing the likelihood of finding vulnerabilities over "dumb fuzzing".			X
TASVS-CODE-7	Client Side - Privilege and Rule of two			
TASVS-CODE-7.1	Ensure that the software follows the principle of least privileges and runs with the lowest level of privileges for it to work as expected. If several levels of privileges are required, their IPC interfaces are well-defined and do not expose more features than required.	X	X	X
TASVS-CODE-7.2	The thick client follows the "Rule of 2", where it cannot have more than 2 of: works with untrustworthy inputs, is written in memory unsafe language, runs with high privileges / without a sandbox.	X	X	X

TASVS-CONF: Configuration and Building

Control Objective

Testing Checklist

TASVS-ID	Description	L1	L2	L3
TASVS-CONF-1	General Configuration Checks			
TASVS-CONF-1.1	Verify that the application build and deployment processes are performed in a secure and repeatable way, such as CI/CD automation, automated configuration management, and automated deployment scripts.	X	X	X
TASVS-CONF-1.2	Verify that compiler flags are configured to enable all available buffer overflow protections and warnings, including stack randomization, data execution prevention, and to break the build if an unsafe pointer, memory, format string, integer, or string operations are found.	X	X	X
TASVS-CONF-1.3	Verify that the application, configuration, and all dependencies can be re-deployed using automated deployment scripts, built from a documented and tested runbook in a reasonable time, or restored from backups in a timely fashion.		X	X
TASVS-CONF-1.4	Verify that all unneeded features, documentation, sample applications and configurations are removed.	X	X	X
TASVS-CONF-1.5	Verify that third party components come from pre-defined, trusted and continually maintained repositories.	X	X	X
TASVS-CONF-1.6	Verify that a Software Bill of Materials (SBOM) is maintained of all third party libraries in use.	X	X	X
TASVS-CONF-1.7	Ensure that all software components, libraries, frameworks, and runtimes used in the application are up-to-date and not end-of-life or obsolete. Outdated or obsolete components can introduce security vulnerabilities, performance issues, and compatibility problems.	X	X	X

TASVS-CRYPTO: Cryptography

Control Objective

Testing Checklist

TASVS-ID	Description	L1	L2	L3
TASVS-CRYPTO-1	Communication			
TASVS-CRYPTO-1.1	The TLS settings are in line with current best practices,	X	X	X
TASVS-CRYPTO-2	Storage			
TASVS-CRYPTO-2.1	The thick client doesn't re-use the same cryptographic key for multiple purposes.	X	X	X
TASVS-CRYPTO-2.2	All random values are generated using a sufficiently secure random number generator.	X	X	X
TASVS-CRYPTO-2.3	The thick client does not use cryptographic protocols or algorithms that are widely considered deprecated for security purposes.	X	X	X
TASVS-CRYPTO-2.4	The thick client does not rely on symmetric cryptography with hardcoded keys as a sole method of encryption.	X	X	X
TASVS-CRYPTO-3	General			
TASVS-CRYPTO-3.1	Verify that all cryptographic modules fail securely, and errors are handled in a way that does not enable Padding Oracle attacks.	X	X	X
TASVS-CRYPTO-3.2	Verify that industry proven or government approved cryptographic algorithms, modes, and libraries are used, instead of custom coded cryptography.	X	X	X

TASVS-NETWORK: Communication and Privacy

Control Objective

Testing Checklist

TASVS-ID	Description	L1	L2	L3
TASVS-NETWORK-1	Data leakage			
TASVS-NETWORK-1.1	Tokens and keys sent in plain text or otherwise easily decodable/decryptable by MITM attack	X	X	X
TASVS-NETWORK-1.2	Data is encrypted on the network using TLS. The secure channel is used consistently throughout the app.	X	X	X
TASVS-NETWORK-2	Licensing & Authentication Servers (if present)			
TASVS-NETWORK-2.1	Verify that session tokens possess at least 64 bits of entropy.	X	X	X
TASVS-NETWORK-2.2	Verify the application generates a new session token on user authentication.	X	X	X
TASVS-NETWORK-2.3	Verify that session tokens are generated using approved cryptographic algorithms.	X	X	X
TASVS-NETWORK-2.4	If authenticators permit users to remain logged in, verify that re-authentication occurs periodically both when actively used or after an idle period.	X	X	X
TASVS-NETWORK-2.5	Verify shared or default accounts are not present (e.g. “root” or “admin”).	X	X	X
TASVS-NETWORK-2.6	Verify the application uses session tokens rather than static API secrets and keys, except with legacy implementations.	X	X	X
TASVS-NETWORK-2.7	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.	X	X	X
TASVS-NETWORK-3	Piracy Detection			
TASVS-NETWORK-3.1	Memory monitoring in place	X	X	X
TASVS-NETWORK-3.2	Telemetery capturing data when binary tampering detected, the software’s behavior is unusual or when the internet connection is lost or the license is invalid.	X	X	X
TASVS-NETWORK-4	Connected Services			
TASVS-NETWORK-4.1	Verify that the application sanitizes user input before passing to mail systems to protect against SMTP or IMAP injection.	X	X	X
TASVS-NETWORK-4.2	Verify that the application sanitizes user input before passing to AD systems to protect against LDAP injection.	X	X	X
TASVS-NETWORK-4.3	Verify that data selection or database queries (e.g. SQL, HQL, ORM, NoSQL) use parameterized queries, ORMs, entity frameworks, or are otherwise protected from SQL injection attacks.	X	X	X
TASVS-NETWORK-4.4	Verify that the thick client doesn’t expose services on the network like debugging features, even if bound to the local host.	X	X	X

TASVS-STORAGE: Data Storage

Control Objective

Testing Checklist

TASVS-ID	Description	L1	L2	L3
TASVS-STORAGE-1	Sensitive Information Review			
TASVS-STORAGE-1.1	Binaries or config files contain usernames, password, connection strings or API keys etc	X	X	X
TASVS-STORAGE-1.2	Registry entires contain usernames, password, connection strings or API keys etc	X	X	X
TASVS-STORAGE-1.3	Make sure that logs are not capturing or saving sensitive data such as PII or any types of credentials used for connecting to external resources or even other resources on the machine where the application is running.	X	X	X
TASVS-STORAGE-1.4	The thick client does not hold sensitive data in memory longer than necessary, and memory is cleared explicitly after use.	X	X	X
TASVS-STORAGE-1.5	Trivial static analysis does not reveal important code or data.	X	X	X
TASVS-STORAGE-1.6	Verify that regulated private, health or financial data is stored encrypted while at rest, such as Personally Identifiable Information (PII), sensitive personal information, or data assessed likely to be subject to EU's GDPR.	X	X	X
TASVS-STORAGE-1.7	Authentication or session tokens cannot be easily obtained.	X	X	X
TASVS-STORAGE-2	DLL Hijacking (Trusted application manipulation into loading a malicious DLL)			
TASVS-STORAGE-2.1	DLL Replacement: Swapping a genuine DLL with a malicious one, optionally using DLL Proxying to preserve the original DLL's functionality.	X	X	X
TASVS-STORAGE-2.2	DLL Search Order Hijacking: Placing the malicious DLL in a search path ahead of the legitimate one, exploiting the application's search pattern.	X	X	X