

SIG EVALUATION CRITERIA SECURITY: GUIDANCE FOR PRODUCERS

Version 3.01

Authors

Rob van der Veer
+31 62043 7187
r.vanderveer@softwareimprovementgroup.com

Version 3.01 – June, 2021

TABLE OF CONTENTS

1.	Introduction	3
2.	Results of SIG’s security model	4
2.1	Background of SIG’s security model	4
2.2	Results and process of SIG’s system security evaluation	4
3.	Measurements in SIG’s security model	5
3.1	Quality characteristics in ISO 25010	5
3.2	Mapping of ISO 25010 security characteristics to system properties	5
3.2.1	Confidentiality and Integrity.....	6
3.2.2	Non-repudiation and Accountability	6
3.2.3	Authenticity.....	6
3.3	Explanation of security properties.....	6
3.3.1	Secure communication.....	6
3.3.2	Authentication strength	7
3.3.3	Session management strength.....	7
3.3.4	Authorized access.....	7
3.3.5	Secure user management.....	8
3.3.6	Input and output verification	8
3.3.7	Secure data storage.....	9
3.3.8	Security logging	9
3.3.9	Dependency strength	10
4.	References	11

1. INTRODUCTION

This document describes the SIG evaluation criteria for security of software systems. These criteria are intended for the standardized evaluation of the security of a software system. The purpose of such evaluation is to provide an instrument:

- To developers for guiding improvement of the products they create and enhance.
- To acquirers for comparing, selecting, and accepting pre-developed software.

This guidance document provides explanation to software producers about the measurement method of SIG applied for evaluation.

This document is not intended to be a complete guide to developing secure software.

2. RESULTS OF SIG'S SECURITY MODEL

2.1 BACKGROUND OF SIG'S SECURITY MODEL

SIG evaluates system security with SIG's security model. The SIG security model is based on the ISO 25010 standard, which defines security characteristics for software product quality in a technology-independent manner. For explanation of ISO 25010, see Chapter 3.

2.2 RESULTS AND PROCESS OF SIG'S SYSTEM SECURITY EVALUATION

SIG's system security evaluation results in findings and risks and measures the degree to which security good practices (controls) are implemented in the software. This is expressed in a rating from 1 to 5 stars to reflect the quality of the implementation. The security evaluation is based on methodical assessment by SIG security experts using various tools. This document reflects the structure and process of that security evaluation.

Inputs for the evaluation are: source code, the implemented design and details on deployment & operation, and functionality. This means that even though the SIG security model focuses on software implementation, omission of e.g. deployment details will result in a lower confidence in the evaluation result.

Generally, the star rating is interpreted as follows:

- ★☆☆☆☆ – extremely low degree of security controls
- ★★☆☆☆ – very low degree of security controls
- ★★★☆☆ – low degree of security controls
- ★★★★☆ – moderate degree of security controls
- ★★★★★ – high degree of security controls

3. MEASUREMENTS IN SIG'S SECURITY MODEL

3.1 QUALITY CHARACTERISTICS IN ISO 25010

ISO 25010 defines eight quality characteristics in its software product quality model: functional suitability, performance efficiency, compatibility, usability, reliability, security, maintainability and portability. In ISO 25010, the security characteristic is defined as:

“The degree to which a product or system protects information and data so that persons or other products or systems have the degree of data access appropriate to their types and levels of authorization”.

In ISO 25010, security is composed of five characteristics (simplified definitions):

- **Confidentiality:** ensures that data are accessible only to those authorized.
- **Integrity:** prevents unauthorized access or modifications.
- **Non-repudiation:** actions or events can be proven to have taken place.
- **Accountability:** actions of an entity can be traced uniquely to the entity.
- **Authenticity:** the identity of a subject or resource can be proved to be the one claimed.

Note that in ISO 25010, availability is not part of security (contrasted with the classic Confidentiality-Integrity-Availability security characteristics) – it is part of the ISO 25010 quality characteristic “reliability”.

3.2 MAPPING OF ISO 25010 SECURITY CHARACTERISTICS TO SYSTEM PROPERTIES

Based on the five ISO 25010 security characteristics, SIG defines nine system security properties that can be measured. Their mapping to ISO 25010 is presented in Figure 1. The rows represent the five security characteristics defined by ISO 25010, the columns represent the nine system properties defined by SIG. The model combines the characteristics *Confidentiality* and *Integrity*, *Non-repudiation* and *Accountability* since they share many attributes in their definitions in ISO 25010.

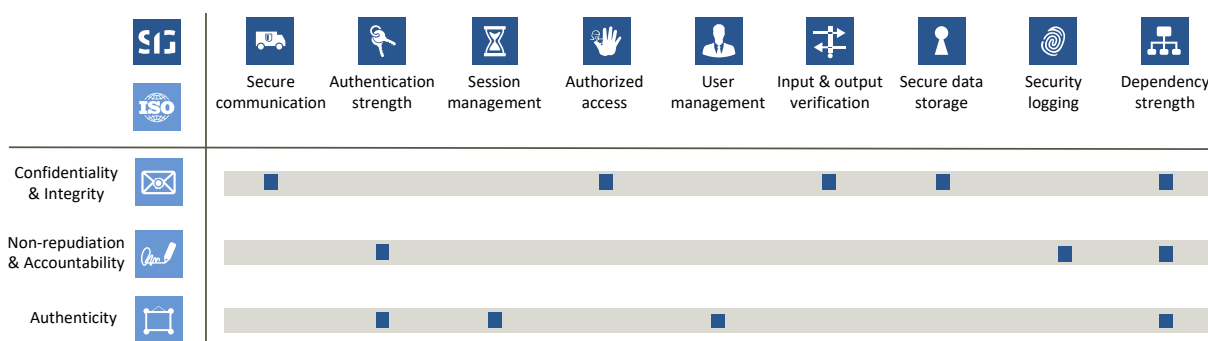


Figure 1: The mapping of system properties to the ISO 25010 security characteristics.

The following paragraphs describe what the ISO 25010 security characteristics mean and what they attempt to accomplish for security.

3.2.1 Confidentiality and Integrity

Confidentiality and integrity are achieved by ensuring a system has the following system properties:

- Secure communication – Data is securely transported into and out of the system, as well as between parts of the system.
- Authorized access – For all system actions and data access, the system verifies the user is properly authorized.
- Input and output verification – The system does neither accept insecure data nor sends out unintended data.
- Secure data storage – Data stored or accessed by the system is protected against unauthorized reads or modifications.
- Dependency strength – The system only depends on secure and up-to-date libraries and frameworks.

3.2.2 Non-repudiation and Accountability

Non-repudiation and accountability are achieved by ensuring a system has the following system properties:

- Security logging – The system collects evidence to prevent repudiation of user actions in the system.
- Authentication strength – All users are properly authenticated when they start using the system.
- Dependency strength – The system only depends on secure and up to date libraries and frameworks.

3.2.3 Authenticity

Authenticity is achieved by ensuring a system has the following system properties:

- Authentication strength – All users are properly authenticated when they start using the system.
- Session management strength – All users remain properly authenticated throughout their use of the system.
- Secure user management – Users can securely be added to, removed from and updated in the system.
- Dependency strength – The system only depends on secure and up to date libraries and frameworks.

Note that dependency management maps to all three characteristics. This is intentional, as it can be placed under either of the characteristics depending on the type of vulnerability/dependency.

3.3 EXPLANATION OF SECURITY PROPERTIES

The following paragraphs describe shortly the meaning of the nine SIG system security properties and how they are assessed. The list of security properties that is assessed is not exhaustive.

3.3.1 Secure communication

For transported data to be secure, it needs to be protected with a sufficiently strong protection method. In addition, the system should try to minimize caching of sensitive data outside of the system.

The rating for this property is determined by the degree to which, at least the following is implemented:

- Transport protection is used in a consistent and correct manner:
 - Transport protection is *used* for all relevant content.
 - Transport protection is *forced* for all relevant content.
 - Transport protection is used for all *parts* of the content.
 - A proper *method* is used for transport protection, i.e. a method that withstands currently known viable attacks. This includes the appropriate type of encryption (e.g. symmetric/asymmetric) that are not compromised and recommended by the National Institute of Standards and Technology (NIST). An example is AES data-encryption with at least 128-bit keys. Other generally accepted security standards are TLS 1.2 or newer for web traffic encryption and Elliptic-curve based TLS key-exchange.
 - The chosen method is *implemented* correctly, i.e. the system should execute the chosen security methods in a manner that leaves no omissions where a user can circumvent the security measures.

3.3.2 Authentication strength

Authentication should be enforced for all system functions and for all use of system functions. The authentication method should be intrinsically strong and the implementation of the method should be sufficiently strong. In case of failed authentication, the system should not perform any functions or expose information.

The rating for this property is determined by the degree to which, at least the following is implemented:

- It is encouraged to have a dedicated service for authentication, where controls are configured for all applications that use its services.
- The *combination* of authentication method and factor is strong:
 - The authentication method is intrinsically strong and best practices for the authentication method are followed. An intrinsically strong method could be e.g. biometrics and/or multiple factor authentication.
 - Authentication credentials are set to the appropriate strength, e.g. a minimum set of requirements for passwords, such as 8+ alphanumeric characters for passwords.
- The system enforces authentication repeatedly and may deny access after failed attempts:
 - Authentication is enforced for all system functions for which authorization or non-repudiation is required.
 - Authentication is enforced for every use of the system.
 - Access is blocked after repeated authentication failures.
 - In case of authentication failure, neither action should be performed nor information disclosed.

Note that authentication is often performed with the aid of a 3rd party authentication solution (LDAP, AD, etc.) This is encouraged. In this case, the analysis is limited to:

- Those parts under control of technical staff, and
- Specific integration of 3rd party code with the system.

3.3.3 Session management strength

After authentication, a session manager should monitor the actor's actions in a secure manner. As sessions avoid continuous re-authentication, they are a proxy for the actor's identity. So usually sessions are managed with the help of a secure session token. A session token must be created and expired in a secure manner, and must not be spoofable or contain sensitive information. When session management is insecure in web applications, vulnerabilities may occur such as cross-site request forgery (CSRF) and clickjacking.

The rating for this property is determined by the degree to which, at least the following is implemented:

- The system monitors the actor's action within a session in a consistent and secure way:
 - The programming technology's or tools' default session manager is used.
 - The session manager checks that requests are intentionally performed by the actors.
 - The session manager accepts and uses session tokens via one channel.
 - The session token is sufficiently long and sufficiently random.
 - The session token has a high level of entropy.
 - The session token does not disclose any user or private information.
- Sessions have specified endings and can be stopped actively:
 - Users can explicitly expire their session.
 - There is a soft session expiration limit.
 - There is a hard session expiration limit.
 - Sessions expire with privilege changes.

3.3.4 Authorized access

Secure authorization takes place within the system so that the user cannot circumvent it. Authorization should take place for every system function and at every attempt to access a system function. If authorization fails, the system should record this event and inform the user only that authorization failed. Users receive least privilege for authorization.

The rating for this property is determined by the degree to which, at least the following is implemented:

- Authorization consistently and repeatedly takes place within the system:
 - Authorization takes place within the system (e.g. not at the user).
 - Authorization takes place based on information from the system, not from the user.
 - Authorization is checked for all system functions.
 - Authorization is checked for each (repeated) system function access.
 - Authorization is checked for each data item access.
 - Authorization takes place based on system functions rather than user roles.
- Systems and users have the minimum amount of privileges needed to perform their duties, and not more (least privilege, separation of duties).
- When authorization fails, the system records details but discloses to the user only that authorization fails.

3.3.5 Secure user management

A system should employ secure user management by implementing secure user sign-up, secure user blocking and removal and secure user credentials management.

The rating for this property is determined by the degree to which, at least the following is implemented:

- It is encouraged to have a dedicated service for user credential management, where controls are configured for all applications that use its services.
- The system implements secure user sign-up:
 - A side-channel for signup user contact is established and verified.
 - The sign-up procedure is identical for all implementations.
 - The system should not provide any default users.
- The system implements secure user blocking:
 - Users can be removed unilaterally.
 - User removal is immediate and pervasive.
 - Users accounts become blocked after a specified period of long inactivity.
 - The system manages a list of inactive and blocked users.
- The system implements secure user credentials management:
 - Users are able to change their credentials, and they are forced to do so in case of loss of credentials.
 - The credential change procedure is only conducted when user-initiated and conducted using a side-channel.
 - The credential change procedure is identical for all implementations.
 - The credential change procedure encourages the user to choose a secure password.
- The system uses unique users:
 - User accounts are not shared between physical users.
 - For system accounts, credentials are not shared between different systems, and whenever possible the end user identification is shared with the system.

Note that user management is often performed with the aid of a 3rd party solution (AD, etc.) This is encouraged. In this case the analysis is limited to those parts that can be configured with/by the system, and to the integration of the 3rd party software with the system.

3.3.6 Input and output verification

Invalid system input should be rejected by the system and the rejection should not disclose information. Input validation should take place within the system and should be validated against a whitelist. Protection mechanisms should be applied and unsafe functions should not be used in order to prevent injection and overflow vulnerabilities.

All output should be escaped to match the output format. This should be done in a single place with a safe API or an escaping library. Systems should never unnecessarily expose details about their implementation.

The rating for this property is determined by the degree to which, at least the following is implemented:

- The system validates and possibly rejects all system input based on allowed inputs:
 - Invalid user-input is rejected by the system instead of corrected. Note that adjusting input during input validation is allowed (e.g. canonicalization/normalization/encoding of input) as long as this does not change the integrity of content.
 - In case of input validation rejection, neither action is performed nor information disclosed.
 - Input validation takes place within the system (e.g. not user-side) such that it is not possible to circumvent it.
 - Input validation is done by using whitelists or allowed input patterns (i.e. not blacklists).
- The system validates input at the appropriate time and validation is combined with data access control (to avoid race conditions such as TOCTOU).
- The system makes use of safe API's:
 - Safe API's, that (automatically) separate data from code, are used to avoid injection vulnerabilities such as SQL injection and Cross-Site Scripting (XSS).
 - Safe API's, that do check buffer lengths, are used to avoid overflow vulnerabilities such as Buffer Overflow and Integer Overflow.
- Output escaping is done consistently, correctly and safely:
 - Output escaping is done consistently, preferably in a single place.
 - Output escaping is applied correctly for each technology-specific context.
 - Output escaping is done as safely as possible with the available tooling, e.g. with standard APIs for this purpose.
- The system restricts the amount and type of (technical) information to the user:
 - The communication protocol must not expose unnecessary information.
 - Output data must not expose any unnecessary information.
 - The system does not expose raw error messages.

3.3.7 Secure data storage

Access to a data store should be prevented or restricted and a data store itself should be physically protected. Sensitive data should be protected against unauthorized access by encrypting it. If access to the original data is not required, read access should be made impossible by correctly applying a one-way hash to the data.

The rating for this property is determined by the degree to which, at least the following is implemented:

- Logical access to data is prevented or restricted with appropriate authentication and authorization:
 - Access to the data store interface is restricted.
- Sensitive data is protected (with sufficiently strong measures that withstand currently known attacks):
 - Sensitive items in the data store are protected with e.g. AES-256 or better.
 - If the original sensitive data is not required, but it needs to be compared (e.g. a password), only a derivative of sensitive data is stored, not the sensitive data itself. E.g. passwords are hashed using a salt with PBKDF2 or comparable.
 - Sensitive data is not stored or sent to a location with less security controls (e.g. logging, less secure systems)
- Data caching is minimized in time and location, especially for sensitive data. Caching locations may include application-level caches, temporary file storage, queues, browsers (via HTTP headers).

3.3.8 Security logging

To allow non-repudiation and accountability, the system must be able to prove that an actor actively approved and performed an action. This proof should be stored in a secure way and in a manner that facilitates retrieval and analysis.

The rating for this property is determined by the degree to which, at least the following is implemented:

- Proof of actions is secure:
 - The system can prove that an actor explicitly approved of the action before performing it by explicitly asking permission to the user to perform tasks that influence e.g. sensitive data.
 - It can be made plausible that the proof was not tampered with.
- The proof is appropriate for retrieval and analysis:
 - The proof contains appropriate level of detail for meaningful analysis.
 - The proof is stored for a sufficient long period of time, for a reasonable period of traceability.
 - The proof is retrievable within enough time for appropriate follow-up.
- Logging is of sufficient quality:
 - Actor origin is logged and sufficiently precise.
 - The combination of actor identification and origin is not spoofable.

3.3.9 Dependency strength

A system should actively manage its 3rd party dependencies to ensure security of the system as a whole.

The rating for this property is determined by the degree to which, at least the following is implemented:

- Dependencies are kept up to date (“fresh”).
 - A system is in place to automatically check for freshness.
 - The dependencies are largely up to date.
- Dependencies contain no known vulnerabilities.
 - A system is in place to automatically check for known vulnerabilities.
 - The systems dependencies contain no known vulnerabilities.
- Dependencies are managed.
 - There is a defined process for dependency management.
 - The inventory of dependencies is supported by a tool.

4. REFERENCES

The ISO 25010 standard provides the full definitions on which the security model is based:

- ISO, “ISO/IEC 25010:2011 Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models,” 2011.

The system properties, underlying sub-properties and best practices were based on existing standards, notably Common Criteria and ASVS.

Common Criteria is an extensive construction and verification model:

- CCRA, “Common Criteria - Common Methodology for Information Technology Security Evaluation,” Version 3.1R5, 2017.


The ASVS is a very detailed verification standard, but is explicitly limited to web applications:

- OWASP, “OWASP Application Security Verification Standard 4.0”, 2019.



Fred. Roeskestraat 115
1076 EE Amsterdam
The Netherlands

www.softwareimprovementgroup.com
marketing@softwareimprovementgroup.com

 Getting software right for a healthier digital world