Secure Development

Revision

Version 8
9/8/23 4:18 PM

SME

Charles Wilson

Abstract

This document describes the process used to establish that security-related requirements associated with a functional requirement have been met.

Group / Owner

Development / Software Developer

Motivation

This document is motivated by the need to have formal processes in place to securely implement the functional requirements in the development of safety-critical, cyber-physical systems for certification of compliance to standards such as **ISO/SAE 21434** and **ISO 26262**.

License

Overview

Creating secure code involves the application of multiple techniques. These include:

- Secure coding best practices (SEI CERT Coding Standards [9])
- Language-specific best practices (C++ Core Guidelines [11])
- Industry-specific security rules (MISRA [10] / AUTOSAR [15])
- Tool-specific security checkers

The AVCDL identifies various software development activities for more specific attention. These include:

- Use of appropriate tool security-relevant settings (**Secure Settings Document**)
- Tracking deprecated function use (**Currently Used Deprecated Functions**)
- Use of security checkers during static analysis (**Static Analysis Report**)
- Use of fuzz testing (**Fuzz Testing Report**)
- Use of security-specific dynamic analysis (**Dynamic Analysis Report**)
- Use of security-specific code reviews (**Secure Code Review Summary**)

Note: This document focuses on those practices specific to the programming process and not the verification of those practices as listed above.

Practices

The process of writing secure code mirrors that of secure design regardless of the programming language used. Both the code and data structure upon which it operates need to be taken into consideration. These practices may be categorized as follows:

- General
- Data Handling
- Prohibitions
- Mandates

Note: These practices can be considered the implementation analog to the design principles presented in **Secure Design Principles** [18].

Note: This is not an exhaustive set.

General Practices

Always-on Security

In order to ensure that security-related behavior is properly integrated into functional behavior, security should always be in use. Encryption should not be disabled during development, but rather should make use of null encryption (such as `TLS_NULL_WITH_NULL_NULL` in TLS [19]). This ensures that the security-related code paths are exercised.

Note: When transitioning from development to production builds, the null encryption capability of the encryption systems should be disabled.

Data Handling Practices

Data Lifetime

Does the data being manipulated have a well-defined lifetime? This is usually considered a performance aspect of function, rather than a security consideration. The longer a security-related piece of data is lively, the greater the opportunity for its misuse (leaking / manipulation).

Data Disposal

Once a piece of security-related data is no longer needed, is it disposed of securely? There are well-documented cases of memory scraping being used to access security-related information. When disposing of security-related data it is critical to use mechanisms which will securely erase it. Normal memory overwriting functions may be optimized out as the variable's lifetime would show that no use was made of the data. Use only functions guaranteed to write random bits to the data. Writing zeros to the data makes it easier to determine where important data may live when raw memory is examined.

Data Structure Size

From a programming standpoint, it is simpler to have one huge structure and pass that between various parts of the code than to have smaller, targeted ones. There are multiple problems posed by overly scoped data structures. Firstly, these structures have a disproportional lifetime. As such, they expose any security-related data they contain for longer than necessary. Secondly, they include the likelihood that security-related data will not be properly disposed. Thirdly, when transmitted between systems, there is a greater chance that security-related data will not be properly secured. The recipient may not even be aware that the data structure contains security-related data. Fourthly, steps may not be taken to secure the data structure during transmission because of the overhead related to the size of the structure itself. Whenever possible, security-related data should be handled by itself and not included in other structures.

Data Initialization

Security-related data is very sensitive to initialization state. It is critical to always initialize security-related data to a known invalid value when not initializing it via a function during declaration. This eliminates the possibility of forced values via stack attacks and incorrect behavior due to random valid (but unintended) stack values. Zero values can also be problematic.

Data Visibility

When security-related data is managed by a class, it is important to ensure that the data is private to the class. Accessor and mutator functions should be provided as necessary.

Data Freshness

Given the complexity of modern software, it is no longer appropriate to assume that a resource's state will remain static for any individual accessor. There exists an entire category of attack which take advantage of code assuming that a resource's state is relatively static. It is necessary to check the state of a resource immediately prior to attempting an operation on that resource if exclusive access by the current thread of execution is not insured.

Note: Exclusive access extends beyond contention within a single program. I it important to keep in mind that resources may be manipulated via the operating environment, such as by another user with access to the resource in question.

Prohibitions

No Embedded Credentials

Security-related credentials should never be embedded in the source or data files. There have been numerous documented cases of credential scraping from improperly secured online code repositories and extracted from binaries.

No Fixed Credentials

Credentials should not be unchangeable. This is especially the case for privileged access. The use of a fixed `admin` account greatly reduces the security of the system.

No Predictable Credentials

Credentials should not be constructed from observable system identifiers (such as MAC address or serial number).

No Custom Cryptography

Only use cryptographic systems from trusted vendors.

Mandates

Use Secure Enclaves

When they are available, secure enclaves should be used for storage of credentials.

Use Approved Cryptographic Mechanisms / Key Lengths

Only use cryptographic mechanisms and key lengths as specified by NIST SP 800-78-4 (**Cryptographic Algorithms and Key Sizes for Personal Identify Verification**) [13] and NIST SP 800-131A (**Transitioning the Use of Cryptographic Algorithms and Key Lengths**) [14]. It's important to use the appropriate key size (not too small, not too big).

Use Canonical Representations

Whether file location, timestamp, URL, or any other data type; use the appropriate canonical representation in order to reduce errors and attacks made possible by non-canonical representations.

References

1. **Design Showing Security Considerations** (AVCDL secondary document)
2. **Product-level Security Requirements** (AVCDL secondary document)
3. **Secure Settings Document** (AVCDL secondary document)
4. **Currently Used Deprecated Functions** (AVCDL secondary document)
5. **Static Analysis Report** (AVCDL secondary document)
6. **Fuzz Testing Report** (AVCDL secondary document)
7. **Dynamic Analysis Report** (AVCDL secondary document)
8. **Secure Code Review Summary** (AVCDL secondary document)
9. **SEI CERT Coding Standards** (AVCDL secondary document)
   https://wiki.sei.cmu.edu/confluence/display/seccode/SEI+CERT+Coding+Standards
10. **MISRA**
    https://www.misra.org.uk
11. **C++ Core Guidelines**
    https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines
12. **Secure Coding Guidelines for Developers**
    https://docs.oracle.com/cd/E26502_01/html/E29016/scode-1.html
13. **NIST SP 800-78-4 Cryptographic Algorithms and Key Sizes for Personal Identify Verification**
    https://csrc.nist.gov/publications/detail/sp/800-78/4/final
14. **NIST SP 800-131A Transitioning the Use of Cryptographic Algorithms and Key Lengths**
    https://csrc.nist.gov/publications/detail/sp/800-131a/rev-2/final
15. **AUTOSAR Guidelines for the use of the C++14 language in critical and safety-related systems**
    https://www.autosar.org/fileadmin/user_upload/standards/adaptive/18-10/AUTOSAR_RS_CPP14Guidelines.pdf
16. **Writing Secure Code, 2ed. Howard and LeBlanc**
    https://www.amazon.com/dp/0735617228
17. **NIST SP 800-160v1 Systems Security Engineering**
    https://csrc.nist.gov/publications/detail/sp/800-160/vol-1/final
18. **Secure Design Principles** (AVCDL secondary document)
19. **RFC 5246**
    https://www.rfc-editor.org/rfc/pdfrfc/rfc5246.txt.pdf