TECHNOLOGY WHITEPAPER

Application Development, Testing, and Analysis for Optimal Security

Trusted Computing: The COTS Perspective Series

CURTISS-WRIGHT

DEFENSE SOLUTIONS

Read About

- Securing data transfer between applications
- Performing thorough testing and analysis
- Leveraging existing processor features and security libraries

Introduction

Application software in military systems enables warfighters to carry out their missions, so it's essential that this code is trusted and secure. Other hardware and software is designed to start the application software securely; once it starts, application software relies on the system's fundamental security building blocks, but requires special attention to ensure it functions as intended.

However, this is often easier said than done because only a small, finite number of system developers typically have access to the application code. Most application software is custom-built to execute a specific mission or run a particular algorithm, so far fewer software engineers will access or review it than would see open-source, or even most commercial, software.

Figure 1: Application code must be thoroughly tested before deployment in order to ensure it is secure.

This can result in undiscovered vulnerabilities, which can be made worse because opportunities to review and update deployed application code typically are few and far between. Application code in military systems is developed to a particular specification; once tested, the system often is deployed with less opportunity for re-testing than a general-purpose system would.

Complicating matters is the narrow technology refresh window of deployed systems. Limitations on time, budgets, and mission requirements can make it nearly impossible to update application software once it's in the field. Even if users discover code issues or security vulnerabilities, the costs to bring a deployed system back for an update is excessive.

On the other hand, it takes far less time and cost to find, fix, and test software problems prior to deployment. For this reason, it's imperative for system developers to make the right decisions about application code from the very beginning.

Organizing Systems to Reduce Vulnerabilities

Application software can fall into several categories: libraries, middleware, and custom-built code to perform specific functions. Custom-built middleware requires special scrutiny because it often is widely reused. Middleware provides the glue that holds libraries and applications together, and any middleware vulnerability could make the entire system susceptible to malicious cyber-attacks.

It's preferable for software engineers to organize systems so that system functions and inputs naturally fall into related buckets. It's important to allocate applications correctly to the user and kernel space. Designers should keep the amount of code running in kernel space to a minimum to reduce the impact of security vulnerabilities, since kernel space executes at a high privilege level, with broad access to system resources. It befits the developer to consolidate related functions closely to each other. That way they don't have to reach out to other parts of the application.

Securing Data Transfer between Applications

Moving large amounts of data also can create security concerns. Large data sets tend to transmitted unencrypted because it's inefficient to encrypt a large amount of data moving at high speeds. Some systems, moreover, must share large pieces of data, especially when separate executable portions of an algorithm are located in multiple places; it's hard to verify and define this data flow in a secure manner.

A better solution is to divide the software such that an application using the algorithm treats it like a black box, with no knowledge of its internal workings. Applications can instruct these black boxes to run specific portions of the algorithm, and receive a concise response in return. This provides a much better-defined information flow between applications. This data flow is easy to verify because it validates the information the system sends and receives it. This approach also helps encapsulate information better to enable encryption and authentication.

Systems designers should define the information flow logically in terms of security boundaries. It's best to minimize the number and complexity of logical interconnections because each connection is a possible point of infiltration. Designers must examine, lock down, test, and verify each connection, as well as consider a secure messaging mechanism that uses authentication to make sure the data comes from a trusted source.

Utilizing Secure Coding Practices

Secure coding practices are an important part of designing-in application security from the beginning. Using standard secure coding practices can minimize security vulnerabilities from programmer errors. The best-known example of such an industry standard is CERT. Others include MISRA, DO-178C, IEC61508, and ISO 26262.

Using secure coding practices can eliminate the likelihood of unintentional errors. A smart approach is to train all application programmers to know and follow the rules and submit to peer reviews. Automation tools also are available to scan code and verify adherence to the rules. Being consistent with coding rules also can enable programmers to move between projects without introducing unnecessary security issues.

Prioritizing Security Testing and Analysis

Funding constraints can lead to program cutbacks, and when this happens, program managers might be tempted to eliminate security testing. Doing so, however, can introduce critical gaps and vulnerabilities. It's important to budget for testing at the program's front end.

Static and Dynamic Code Analysis

One approach for analyzing how application components work together is static code analysis, which uses tools to check the code for any potential issues. Software engineers can use the same plug-in tools they use to verify secure coding standards compliance to perform static code analysis. Examples of these tools include Synopsys® Coverity, cppcheck®, Klocwork, lint, Parasoft, and Understand. Some of these tools work in several languages, some are language-specific, some are commercial, and some are available as free open-source software.

Using dynamic code analysis tools can help software developers analyze how the application will run under test conditions. Dynamic code analysis hooks into the running software to analyze what the system is doing, and works in the background to enable systems developers to validate their applications with normal inputs and outputs to make sure that everything works properly.

Static and dynamic code analysis tools each look at different things, and produce different results. Static code analysis typically reports more false positive errors because it flags every possible error no matter how unlikely. Dynamic code analysis, on the other hand, verifies that the same error never occurs, under test conditions. Tools that provide dynamic code analysis include BoundsChecker, dmalloc, Parasoft Insure++, Purify, and Valgrind. Dynamic code analysis also can provide additional benefits, such as thread coherence validation and code coverage analysis.

Regression and Continuous Testing

Another important decision for application code developers is whether to use regression testing or continuous testing. Regression testing might not uncover failures until the application goes through acceptance testing, or worse, somewhere further down the line.

The continuous method runs tests at on a continuous basis to help find problems as early as possible. It should include a security-specific testing apparatus to identify software changes quickly, that break some of the application's security constraints. While continuous testing may add cost at the front-end, it can reduce cost at the back-end.

Employing Processor-Specific Security Features

Some processors, such as the Intel® Software Guard Extensions (SGX), have built-in security features. Intel SGX allows software developers to create enclaves in the application to securely partition data. If the programmer needs a key to encrypt some data, he or she can place the variable into a separate enclave. Using enclaves to separate the encryption key from other system functions prevents a cyber-attacker from gaining access to system security.

Another processor-specific security feature is Arm® TrustZone, which enables an Arm processor to separate memory, operating system, and application into trusted and non-trusted areas. TrustZone ensures that only the trusted area can access trusted data.

Leveraging Existing Security Libraries

One of the major tenets of cryptography is "don't try to build it on your own", as it's complex and easy to get wrong. It's much safer and wiser to use something that already exists, is readily available, and is kept up to date continually from the feedback of many users. Take advantage of the security libraries already

available within your operating system to implement common security concepts, such as authentication, secure communication, encryption, and key derivation. Examples of robust security libraries include OpenSSL and IPsec.

Mandatory Access Controls (MACs), which most software operating systems support, enable developers to create configuration files that define how different people can use the application and its resources. Operating systems that support MACs include Security-Enhanced Linux® and Windows® Integrity Levels.

Conclusion

Testing software and finding and fixing problems prior to deployment can be less costly and time consuming than a reactive approach. While tight timelines and budget constraints may make development shortcuts tempting, it's critical for developers to follow best practices from the very beginning in order to build truly trustworthy code.

Strategically organizing systems to reduce vulnerabilities, ensuring data is securely transferred between applications, and adhering to secure coding practices are key steps to follow in the beginning of the development cycle. Then, performing thorough security testing and analysis – which much be properly budgeted for – is essential to ensure applications are functioning as intended and that any bugs are caught before deployment. Developers can benefit from leveraging processor-specific security features, such as Intel SGX and Arm TrustZone, as well as tried and trusted security libraries, such as OpenSSL and IPsec.

Not All Solutions Are Created Equal

When defense organizations, aerospace companies, and system integrators are evaluating embedded computing solutions, it is extremely important to understand exactly what vendors mean when they say their solutions provide Trusted Computing. It is even more important to understand the difference between solutions that provide Trusted Computing and those that offer a TrustedCOTS™ level of protection.

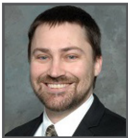
Curtiss-Wright TrustedCOTS solutions extend Trusted Computing best practices to every part of the development process, from design and testing to supply chain and manufacturing. The highest possible levels of protection are built into every aspect of solution development to increase the overall value that COTS solutions can provide in a secure system. This process includes careful analysis of the relationships, intersections, and dependencies among all of the various protection domains, and investigation into potential faults and failures at the lowest levels to identify the associated security vulnerabilities.

Curtiss-Wright takes a holistic view of Trusted Computing, going above and beyond the efforts of other vendors to apply the advanced protection capabilities needed to develop truly secure COTS solutions. It's one of the main reasons the company has been a trusted, proven leader in the global defense and aerospace industries for decades.

CURTISS-WRIGHT

Author(s)

Steve Edwards
Director, Secure Embedded Solutions & Technical Fellow
Curtiss-Wright Defense Solutions



David Sheets
Security Architect
Curtiss-Wright Defense Solutions

Learn More

Curtiss-Wright Products

- Data at Rest Protection
- Security Enabled Curtiss-Wright SBC and DSP Products

Curtiss-Wright Case Study

- Building a Truly Trusted Computing Solution with COTS Hardware and Intel Security Capabilities

Curtiss-Wright White Papers

- The Many Faces of Trusted Computing: What You Need to Know to Protect Critical Platforms and Data
- Getting Secure, Intel-Based Solutions to Market Faster - Why the Hardware Vendor's Boot Security Implementation Is So Important
- Beyond Trusted Computing: Extending Protection Capabilities to Deliver TrustedCOTS Solutions
- Trusted Computing: The COTS Perspective Series
  1. Introduction to COTS-based Trusted Computing
  2. Trusted Boot
  3. Hardware Features for Maintaining Security During Operation
  4. Considering the Role of Hardware in Securing OS and Hypervisor Operation
  5. Application Development, Testing, and Analysis for Optimal Security
  6. Developing a Secure COTS-Based Trusted Computing System
  7. The Impact of Protecting I/O Interfaces on System Performance
  8. Decomposing System Security Requirements
  9. Establishing a Trusted Supply Chain
  10. Certification Authorities for Trusted Computing in Military and Avionics Products

© 2020 Curtiss-Wright. All rights reserved. Specifications are subject to change without notice. All trademarks are property of their respective owners. I W164.0620