THE TEN COMMANDMENTS OF EMBEDDED SOFTWARE SECURITY

TELEGRID

www.telegrid.com

BACKGROUND

The market for embedded systems is growing at a blistering pace. While consumer electronics, automobiles and industrial equipment are a major source of growth, the recent expansion is mainly due to the growth of the Internet of Things (IoT). Research site Business Insider estimates that the number of IoT devices will quadruple in the next 5 years. More importantly, consulting firm IC Insights expects IoT devices to account for 85% of all Internet connections worldwide by 2020. It is this connectivity that makes IoT so exciting but also so dangerous. IoT has created a path for hackers to gain access through end user devices into the wider network. It has exposed vulnerabilities in embedded systems and increased network attack vectors.

Embedded Software Attacks Increasing

2010 - Stuxnet

2012 - 4.5mm DSL Routers hacked in Brazil

2014 - 750k malicious emails sent from 100k consumer gadgets including routers, televisions and at least one refrigerator

2015 - German Steel Mill Furnace Damaged

2015 - TrackingPoint Self-aiming Rifles

2015 - Jeep Grand Cherokee

This white paper describes the ten commandments of integrating software security into embedded systems. These simple rules should be followed by any programmer creating embedded software and any program manager concerned about the security of their embedded software. This is by no means a complete list of security techniques. It does not cover concepts like good requirements documentation, peer reviews and test planning. Rather it focuses on the code and how to build a foundation for any secure product.

#1 PERFORM THREAT ANALYSIS

The first step in developing a secure embedded system is the performance of a threat analysis. This analysis looks at the program's dataflow and makes assumptions of potential security vulnerabilities. It provides a guide for the tools and methods that must be implemented in order to protect the system.

In addition to understanding your own system, a threat analysis requires an understanding of widely used vulnerabilities attacks. One idea is to build a threat library of common attacks which can be used across the organization. This will reduce the time to market and remove duplicate efforts. This library can be built from existing resources including the CWE (Common Weakness Enumeration) and OWASP (Open Web Application Security Project) Top Ten.

Once you have your list of vulnerabilities, group them into easily recognizable areas. One such method is STRIDE developed by Microsoft. STRIDE stands for Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service and Elevation of Privilege.

#2 MINIMIZE USER PRIVILEGES

It is important to consider how qualified users will interact with the system. After making a list of users and functionality, ensure that you provide users with the least amount of privileges they need to perform their work. By limiting access to special privileges, like root access, you vastly reduce the attack surface of your code. Additionally this ensures that there is no

Quick Tip: Test with Limited Privileges

After you have assigned limited privileges be sure to test your code with the limited privilege user roles. Developers tend to test their code with their root access privileges thereby missing bugs that occur with limited privileges.

crossover between domains through functional areas. Unforeseen backdoors are always a target for attackers.

#3 PICK UNCOMMON PASSWORDS

When selecting factory default passwords for the embedded system pick words other than “admin”, “password” or “guest”. While this won’t prevent hackers from breaking in, it will definitely slow them down. Additionally prompt the user to change the password at first log-in. Finally, make sure the password is encrypted and never kept in the clear.

#4 SEPARATE FUNCTIONALITY

Restrict access between different functions in the code. This will ensure that there will not be a crossover attack into a sensitive area if a non-sensitive area is compromised. There are many ways to create this separation including installing embedded hypervisors, limiting the use of shared memory or ensuring that communications between processes use standard protocols like socket connections, message queues, etc. TELEGRID has even developed systems with multiple microprocessors to create a physical separation boundary. If there are communications between processes it is also important to check the input/ output to make sure it is within expected bounds.

#5 CLEAR MEMORY

After you have released a memory resource make sure to delete the items in that resource. To increase processing speed, developers tend to release a data register but do not have their code clear the contents of that register. The assumption is that another trusted source will write over the contents so there is no security concern. This, however, provides a fertile ground for phishing attacks. In order to avoid having information fall into the wrong hands be sure to erase memory space after it has been used.

#6 VALIDATE INPUT AND OUTPUT DATA

Attackers sometimes feed random data into embedded system in order to cause buffer overflows or other unexpected malfunctions. One method to test for the existence of random data is known as “fuzzing”. This, however, is a test tool rather than a development tool. A cleaner approach is to have your developer create a white list of validated values or alternatively a black list of non-validated values. Black listing gives more freedom but requires a robust knowledge of security vulnerabilities, which is difficult in today’s world of zero day attacks.

Quick Tip: Signed Integers

Avoid the use of signed integers for values which cannot be negative including memory allocation, array indexes, buffer sizes, etc.

#7 PREVENT BUFFER OVERFLOWS

Buffer Overflows constitute some of the most common embedded software attacks. Protecting against buffer overflows requires a multipronged strategy. Firstly, it is important to validate the input to a buffer to make sure it is within bounds of an expected range. A corollary is not letting a user write to the last element of an array. Secondly, avoid or minimize the use of unsafe buffer functions including strcpy(), scanf(), gets(), memcpy(), etc.

Finally, ensure your code is optimized for your embedded platform. For instance, attackers know that embedded systems have limited memory and therefore will try to maximize use of these valuable resources. If the memory allocator runs out of memory and returns a value your software is not prepared for (e.g., 0), your software might write to that address and overwrite existing code.

#8 USE LOG FILES FOR FORENSIC ANALYSIS

It is good practice to keep audit logs of attack occurrences in order to determine, after the fact, the nature of the security vulnerability and the best way to prevent it in the future. These logs should, ideally, be kept separately from monitoring logs which are mainly used for troubleshooting.

#9 PLAN AHEAD FOR PATCHES

It is inevitable that a bug fix or unforeseen security vulnerability will arise in the future so you must have a plan for patching embedded software. If, for example, there is no network connection to the device and the plan requires physical access, how will you be able access it? Is there a USB port on the back panel or will it require the user to physically open the device? What is the plan for IAVA (Information Assurance Vulnerability Alert) patching? Is someone in your team tasked with tracking IAVA releases? Who has that responsibility?

Quick Tip: Bounded Loops

On New Year's Eve 2008 thousands of Microsoft Zune music players failed. The reason was that on the last day of a leap year, the value of days became 366 which triggered a loop that would not terminate. This failure could have been avoided with bounded loops.

#10 USE THE RIGHT CRYPTO AND PROTOCOLS

Make sure you are using the right cryptographic engine and that your protocols use the most up to date and secure versions. At a minimum ensure you are using FIPS 140-2 Level 1 certified encryption for all cryptographic functions. Additionally make sure you employ secure storage, management and access to encryption keys. Only store keys in secure file storage and only allow access to keys by secure applications. Securely delete keys when they are no longer needed.

For network communications ensure you use validated standards like TLS and IPSEC. For network management ensure you use secure protocols like SSH and SNMPv3. For Authentication and Authorization employ PKI/ PKE with two factor authentication (e.g., CAC/PIV), RADIUS and/or LDAPS. For Network Access Control employ 802.1x.

CONCLUSION

While resource consumption continues to be the primary focus of embedded system design, the Internet of Things is forcing developers to add security to the list. Network connectivity has given attackers the ability to take down networks and damage expensive machines from simple end devices. The number and size of these attacks has increased in recent years and there is no sign of decline. Following this list of best practices will help developers recognize the attack vectors of embedded systems and potential solutions. Test tools and compilers exist to validate the implementation of many of the coding concepts detailed in this white paper. Additionally firms unfamiliar with security could hire a consultant to create a threat analysis and mitigation strategy.

Proper coding and planning will limit the access to important functionality and data, however, the best way to prevent an attack is to stop an intruder accessing your device. This requires the right crypto and using the right protocols. While there exist both hardware and software-based embedded security modules, these products focus almost exclusively on encryption. TELEGRID's Embedded Security Framework views security as a combination of encryption, authentication and management. Each is an important facet of embedded security and their mutual application defines the overall security of a device. TELEGRID's Embedded Security Framework follows DISA's Security Technical Implementation Guides (STIGs) and gives developers a platform to build secure products. For more information email sales@telegrid.com.

THE TEN COMMANDMENTS OF EMBEDDED SOFTWARE SECURITY

#1 PERFORM THREAT ANALYSIS

#2 MINIMIZE USER PRIVILEGES

#3 PICK UNCOMMON PASSWORDS

#4 SEPARATE FUNCTIONALITY

#5 CLEAR MEMORY

#6 VALIDATE INPUT AND OUTPUT DATA

#7 PREVENT BUFFER OVERFLOWS

#8 USE LOG FILES FOR FORENSIC ANALYSIS

#9 PLAN AHEAD FOR PATCHES

#10 USE THE RIGHT CRYPTO AND PROTOCOLS

TELEGRID

www.telegrid.com

973-994-4440

sales@telegrid.com