IEEE CYBER SECURITY

IEEE CENTER FOR SECURE DESIGN

Design Flaws and Security Considerations for Telematics and Infotainment Systems

IEEE

IEEE computer society

READ ON!

Public Access Encouraged

Because the authors, contributors, and publisher are eager to engage the broader community in open discussion, analysis, and debate regarding a vital issue of common interest, this document is distributed under a Creative Commons BY-SA license. The full legal language of the BY-SA license is available here: http://creativecommons.org/licenses/by-sa/3.0/legalcode.

Under this license, you are free to both share (copy and redistribute the material in any medium or format) and adapt (remix, transform, and build upon the material for any purpose) the content of this document, as long as you comply with the following terms:

Attribution — You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may use any reasonable citation format, but the attribution may not suggest that the authors or publisher has a relationship with you or endorses you or your use.

"ShareAlike" — If you remix, transform, or build upon the material, you must distribute your contributions under the same BY-SA license as the original. That means you may not add any restrictions beyond those stated in the license, or apply legal terms or technological measures that legally restrict others from doing anything the license permits.

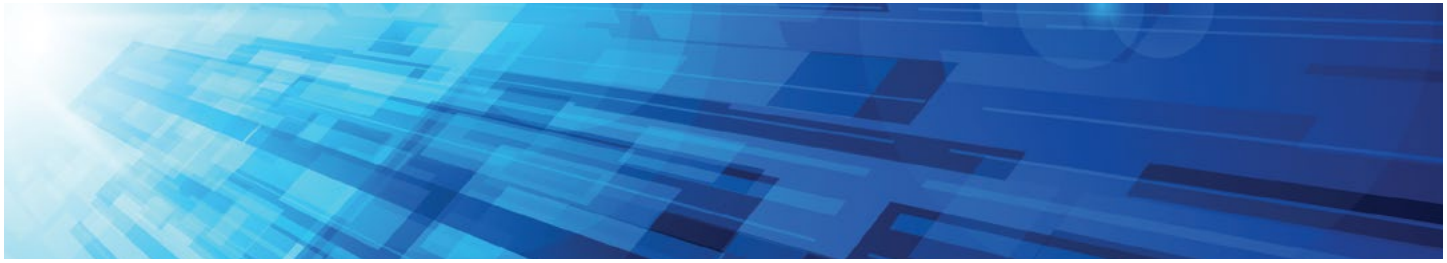
Please note that no warranties are given regarding the content of this document. Derogatory use of the content of this license to portray the authors, contributors, or publisher in a negative light may cancel the license under Section 4(a). This license may not give you all of the permissions necessary for a specific intended use.

Staff

Brian Kirk, Manager, New Initiative Development
Carmen Flores-Garvey, Designer

TABLE OF CONTENTS

Design Flaws and Security Considerations for Telematics and Infotainment Systems



Design Flaws and Security Considerations for Telematics and Infotainment Systems

Matthew Alt
MIT LL

Glenn Atkinson
Geotab

Jack Clark
DOT-Volpe Center

Daniel Chin
DOT-Volpe Center

Jim DelGrosso
Synopsys (formerly Cigital)

Tom Forest
GM

Bob Gruszczynski
Volkswagen

Kevin Harnett
DOT-Volpe Center

Christopher King
Rockwell Automation

Dan Klinedinst
SEI/CERT

Dan Lyon
Synopsys (formerly Cigital)

Anthonios Partheniou
Geotab

Rich Pietravalle
MITRE

Craig Smith
Rapid7

Corey Thuen
IOActive

Graham Watson
DOT-Volpe Center

André Weimerskirch
Lear Corporation

Tim Wiesenberger
SAE International

Michael Westra
Ford Motor Company

BACK TO TABLE OF CONTENTS

DESIGN FLAWS AND SECURITY CONSIDERATIONS FOR TELEMATICS AND INFOTAINMENT SYSTEMS

Telematics incorporates wireless communications, computer system components, and internal automotive bus communication to send and receive data. Telematics/infotainment systems are rapidly becoming standard components within the automotive industry, as these systems perform necessary functions in support of autonomous systems, entertainment systems, fuel tracking, emissions, maintenance, and location data. As with any system that provides external connectivity, a significant consideration in deploying a telematics/infotainment system is the potential security risk the system could have on the vehicle platform.

Telematics/infotainment systems share a few common characteristics:

- They have external connectivity that crosses trust boundaries with internal automotive communication.
- They use short and/or long range wireless technologies (such as WiFi, Bluetooth, or cellular).

They also possess other characteristics, which might not be present on every system. For instance, they

- render media content;
- have a user interface;
- leverage a non-real-time operating system; and
- typically leverage traditional IT technologies.

The IEEE Center for Secure Design (CSD) developed this document as a direct result of a discussion from experts on recurring security flaws and vulnerabilities identified within vehicle telematics/infotainment systems. What emerged from these discussions is an awareness that most if not all of these security flaws and vulnerabilities are avoidable by adhering to existing standards and best practices. Examples of such practices include incorporating security into the System Development Lifecycle (SDLC) and properly using authentication and cryptography, secure software development, and secure software update practices. In most cases, the group found that the previous publication from IEEE CSD, “Avoiding the Top 10 Software Security Design Flaws” applies to telematics/infotainment systems as well as traditional software.

The intended audience for this document is any entity involved in the design, build, implementation, and deployment of telematics/infotainment systems, including Original Equipment Manufacturers (OEM), telematics/infotainment device suppliers, telematics/infotainment service vendors, and aftermarket device vendors.

BACK TO TABLE OF CONTENTS

Each section discusses a practice to follow to avoid common security design flaws found in current telematics/infotainment systems, and includes references for further reading. The issues identified here aren't a comprehensive list of all of the security best practices required to build secure telematics/infotainment systems—rather, they represent areas of improvement based on common design flaws.

Understand Your Risk

The development process is key to producing a secure system. Several publications provide detailed guidance on a secure development process, including the *Cyber Security Guidebook for Cyber-Physical Vehicle Systems* (SAE J3061) and *Security Considerations in the System Development Lifecycle* (NIST SP 800-64).

Some aspects of a secure development process can prevent common design flaws. The following activities aren't a complete list, but are essential in building and maintaining a secure product:

- Threat analysis and risk assessment is necessary not only at the initial stages, but also throughout the development lifecycle. Performing this activity helps you make informed decisions about the system's security and provides a rationale for why implementing security measures is important. The behaviors or functionality necessary to reduce risks should be specified through security requirements.
- To analyze the entire system's security, you need a security architecture. This architecture helps you understand the important assumptions made when defining characteristics, such as key interfaces and data flows.
- When writing software, follow best practices for creating secure software, and apply secure coding standards, such as those provided by CERT. Designers must perform security-focused code reviews and use security-specific static analysis to help identify known vulnerabilities (for example, buffer overflows in a time-of-check to time-of-use race conditions).
- As part of the development process, include activities that identify and address known vulnerabilities in third-party libraries.

Design Considerations and Common Mistakes to Avoid

Attackers routinely find new ways to exploit vulnerabilities. But being aware of common design mistakes and understanding what *not* to do helps minimize your risk.

BACK TO TABLE OF CONTENTS

Don't Allow Additional Privileges

The Principle of Least Privilege states that you only give an entity the privileges needed to complete its task (see NIST SP 800-53, Revision 4, control AC-6). This principle is a fundamental building block to system security. Because of the unique architectural constructs of a telematics/infotainment device, there are two primary categories of least privilege for evaluation.

1. The first consideration is the communication between the telematics/infotainment unit and the vehicle. The telematics/infotainment unit should limit its transmit capability to a predefined list of transmit and receive vehicle network messages required for expected operation. Some mechanism should exist to prevent a compromised telematics unit from sending arbitrary messages over vehicle networks (see the next section). Further, consider the Principle of Least Privilege from a privacy perspective (as a specific example, a corollary property of "least information"—omitting detailed GPS information when all it needs is speed information).
2. The second consideration lies within the telematics/infotainment device itself. Although a telematics/infotainment device's design and complexity could vary wildly, consider the following examples of applying the principle.

- Many embedded operating systems utilize the concept of a *user mode* versus a *kernel mode*, wherein the system kernel controls particular permissions to execute code, read memory, and so on (for example, "running as root" in the nomenclature). Ensure that an application, or user-mode code, can access only that which it has permissions to access. A common pitfall is to execute all code for a telematics unit in kernel mode (or even as a root user). The end result is that any vulnerability in the code running on a device results in complete control of the device. If the device has a separation of privilege and runs code in user mode, an attacker exploiting a vulnerability may need to circumvent the system's privilege restrictions (successfully performing a privilege escalation attack).
- Best practice recommendations for design and implementation within the device vary depending on the underlying OS or system architecture, such as for filesystem permissions, access to personal information, and sandboxing of applications. For example, a telematics/infotainment system built on Android will focus system privilege restrictions by using the app permissions

BACK TO TABLE OF CONTENTS

system in the Android manifest for an app, and systems built on Linux will focus on limiting root processes and system access. QNX and other systems will have similar—and sometimes unique—additional capabilities. We discuss this in more detail in the "OS protections" section.

Overall, failure to heed Least Privilege in the system offers an attacker a means to leverage vulnerabilities to gain system control.

Don't Permit CAN Bus Access—or at Least Limit CAN Bus Access

A major concern for securing telematics/infotainment operations lies in the boundary between a telematics/infotainment system and the vehicle network. The network connectivity characteristic of all telematics/infotainment systems creates a bridge between the vehicle network and other networks at large. This bridge is a potential security weakness that telematics/infotainment system developers need to understand.

Much research exists on the insecurity of vehicle networks—and the Controller Area Network (CAN) bus in particular. Taking the correct steps, telematics/infotainment devices can minimize the increased risk that accompanies an increase in attack surface. Evaluate the following considerations, and document the risk as either accepted or mitigated by system design.

Telematics/infotainment systems can increase the risk of compromise by allowing an attacker to send arbitrary CAN messages that affect safety-critical systems within the vehicle. Mitigate this risk by separating the telematics/infotainment actions from the CAN bus communication functions. For example, a designer might decide on a hypervisor-based separation of these functions or add a second, separate processor dedicated to transmitting and receiving CAN bus messages. Separating the telematics/infotainment main processor actions and the CAN processor actions allows for designing and implementing an API that would restrict the types of CAN messages sent by the telematics/infotainment system as a whole. If this CAN processor API is further hardened against malicious action, an attacker compromising the primary telematics/infotainment system would then need to discover and exploit a vulnerability in the CAN processor's APIs to send any unauthorized CAN messages.

Failure to mitigate this risk exposes an increased attack surface and provides a potential entry point for a remote attacker.

BACK TO TABLE OF CONTENTS

Don't Have Extraneous Functionalities

Each functional block of the system should perform only the functions needed to accomplish its defined tasks. As such, integrate least functionality reviews into the design, development, and deployment process (see NIST 800-53, Revision 4, CM-7 Least Functionality). Part of that review includes being aware of, monitoring, and removing or locking down the following items, both in the in-house software as well as third-party software:

Remove backdoors. Review the system for entry or execution paths that aren't part of the design—mechanisms that could be undocumented additions by developers as shortcuts during testing or for exercising functions and remove them from functioning. Scan for open ports on the embedded modem itself—because sometimes, for example, developers enable Secure Shell (SSH) or other ports during development.

Debug tools, functions, diagnostics, utilities, or settings. Sometimes these items are included for convenience during development or testing, but they could serve as handles for attackers to gain entry into the production product. Disable these functions.

Eliminate support for legacy subsystems or protocols with known vulnerabilities. Look for system mechanisms that use such items, perhaps during components' protocol negotiation, session setup, or similar situations. Remove access to the legacy protocols and subsystems.

Eradicate unused features or APIs. Often functionality deriving from one customer requirement is included in another customer's deployment, and in consideration of maintaining a common source, it might be left in for all models, versions, or implementations. Such features or extraneous APIs, preserved in and existing outside of anything other than the intended context, serve as compromise points for attackers. Review unused features in third-party libraries where unused functions add to the attack surface without adding functional benefit.

Remove or block unused execution paths or code branches. Examine by inspection and execution, focusing on monitoring the paths of execution to spot unused sections of code that might yield footholds for attackers to execute, but aren't needed in the developed product. Likewise, for OS features and functions not used, look for ways to remove them, block their

BACK TO TABLE OF CONTENTS

activation, or intercept usage depending on the facilities of that host OS environment.

Monitor network traffic. Review how network traffic is inspected during system execution; ensure that it filters the traffic down to only what's required, possibly logging traffic that's outside the norm. Certain traffic of known type and impact severity might warrant triggering an alarm or other action.

Isolate and contain the impact. Observe and design the system to limit the impact of a compromised interface or component. Isolate or segment it, especially if it's a more complex or robust system element.

Don't Trust External Input

A common problem in insecure systems is that those systems, or individual modules, assume certain features of external input, or trust the validity of external input. Good examples of that are a software module's trust in navigation maps' legitimacy, an MP3 or media file's integrity, an SMS message's authenticity, or text's legitimacy. Such trust leads to a variety of attacks, including SQL injection, command injection, and buffer overflows.

The underlying problem is that software module designers don't take into account the tool's or module's security claims, or make unwarranted assumptions about those security claims. This leads to external inputs that aren't validated, or they're validated only by the interface driver (for example, cellular communication modules) but not by all software modules that use the received data (for example, an MP3 codec). The following are typical errors that make such attacks possible:

- input is validated improperly, with no sanity check executed;
- input validation is executed in the wrong place (see the original top 10 list); and
- failure to check that sizes and boundaries are honored.

The adversary's objective is usually to gain control over the vehicle or telematics/infotainment unit by injecting remote code to control functionality, or to extract data. The best policy is for each software module to consider any received data as untrustworthy and potentially malicious. Use the following steps to mitigate the threat of external input. Note that no individual step will suffice as a standalone solution. All are recommended.

- Ensure that the input complies with the underlying protocol—for example, that an MP3 file is a valid MP3 file. In many cases,

BACK TO TABLE OF CONTENTS

input validation designs do not check for additional data and many protocols don't consider security at all.
- Validate all inputs at all times when they're used.
- Design and execute functional test cases with non-standard input.
- Perform static analysis.
- Perform fuzz testing, ideally supported by fuzzing test tools. Such tools are typically available for common protocols.
- Heed the SAE J3061 standard guidelines section on software testing.
- Use parsers properly, such as an Abstract Syntax Notation One (ASN.1) parser. Improper use might lead to vulnerabilities. Also, be aware that parsers might come with vulnerabilities as well.
- Consider the use of formally verified or thoroughly reviewed and tested source code for critical interfaces—for example, the US National Highway Traffic Safety Administration works on a formally verified vehicle-to-everything dedicated short-range communications (V2X DSRC) parser of basic safety messages (SAE J2735-compliant messages). Limit the damage when something does get through (see the previous section about not running these activities with administrative access, and additional OS protections).

Don't Mishandle Credentials

Common flaws in credential management include password reuse across devices (allowing a single compromise of a credential to compromise every device using the same credential), use of weak password generation algorithms, and easily crackable passwords. All passwords should be unique and have high entropy.

Proper use of credentials within a telematics/infotainment system is critical to protect the system, data, and vehicle operation. This is especially critical for any credential used for privileged access. In fact, privileged access should require specially signed software to enable that interface.

Don't Use Cryptography Incorrectly

Most systems need to use strong cryptography in cases where control information or personally identifiable information (PII) are transferred across a trust boundary. Many telematics/infotainment devices mistakenly fail or decline to use cryptography.

Implement cryptography correctly (refer to top 10 flaws document). When choosing a cryptographic algorithm, use a tried and tested cryptographic algorithm as defined by NIST or other standard bodies. Whenever possible, use

BACK TO TABLE OF CONTENTS

standard libraries for encryption rather than coming up with your own (don't attempt your own cryptographic implementation without a comprehensive understanding of the increase in risk). Consult with cryptography subject matter experts, who can help verify the cryptographic implementation as well as the key management process that will preserve confidentiality and integrity over time.

Cryptography can entail several common issues. One issue is not creating separate keys for distinct functions, and hence not reducing the attack surface and attack impact. For example, an encryption key used for communication functions should differ from the encryption key used for an unrelated purpose, such as control functions. Assign a unique key to each telematics/infotainment device to ensure that if a single device key is reverse-engineered, other devices in the ecosystem won't be affected. Another issue is lack of sufficient randomness, to remedy use of cryptographically strong random value generators for encryption algorithms, where strong random values are necessary.

Don't Overlook Authentication for Messages Used in Critical Functions

Cautionary real-world incidents abound of not including proper authentication on telematics/infotainment messaging. In one instance, the system erroneously used the vehicle identification number (VIN) included in the request as authentication. In others, the system used common global keys to authenticate key features or assumed authentication because the transport channel was encrypted.

The key design guidance here is that the underlying messages themselves should be authenticated—for example, digitally signed or use message authentication code (MAC)—and optionally encrypted. This especially bears true where command and control allow direct control over cyber-physical systems, such as an automobile in this case. Even when the transport has some form of underlying security (possibly WiFi, Bluetooth, or cellular), don't assume that this encrypted communication provides authentication.

For message-level communications, unique device credentials should be used rather than using hardcoded credentials embedded across a range of devices. This causes numerous security issues, because an attacker would only need to extract keys from one device to exploit a larger number of devices.

An additional pitfall is relying on seemingly unique or non-random values for authentication. This confuses two security terms: identification and authentication. Identification will use a unique, often common identifier protected by authentication to determine which device is

BACK TO TABLE OF CONTENTS

communicating. Using a unique per-device serial number is common but not secure. In other instances, values such as the VIN, phone number, or email address have been used. Take care when selecting what identifier to use, because several identifiers in the previous list are personally identifiable and have legal implications with their wide use within systems. Even seemingly innocent identifiers like a VIN reveal considerable data about the vehicle's make, model, year, and features; in many countries, this is considered PII. Authenticate using proof of knowledge, such as a shared secret that isn't public (unlike the identity used for identification). You can implement this as a combination of a symmetric key, digital certificate, or cryptographic token—just make sure to protect this secret within the device (we discuss this more in the "Best Practices to Employ" section).

Check authentication at the appropriate level and between components. Numerous security issues have arisen because a client locally performed authentication and the party it communicated with didn't perform independent authentication. This allowed the attacker to bypass the client's local authentication and make unauthenticated requests directly. Authentication can be end-to-end between a user, or through the back end (including the vehicle), but more commonly it occurs explicitly between components.

The final point is that encryption isn't the same as authentication. Encrypted data can be manipulated and cybersecurity experts have found predictable ways to manipulate data within encrypted streams or replay portions of known ciphertext. Confirm authentication and integrity through digital signing or message authentication codes. Both encryption and integrity controls are necessary.

Don't Assume Underlying Communication Channels Are Secure

Telematics/infotainment units operate on a variety of different communication networks. These may or may not be under the control of the device's vendor or consumer. These include cellular (data) networks, short message service (SMS) and voice, wireless (IEEE 802.11), Bluetooth, CAN, and others. The telematics/infotainment unit should assume the underlying communication channels are compromisable and that someone is watching the communications. Therefore, the unit's design must protect against common network attacks such as man-in-the-middle (MITM) and spoofing, without relying on protections offered by the underlying network. For example, cellular protocols such as Global System for Mobile communication (GSM) and Long-Term Evolution (LTE) have known security limitations, such as the ability to force fallback

BACK TO TABLE OF CONTENTS

to older protocols that don't use encryption, and a variety of methods for location tracking. (Note that some embedded LTE modems don't have hardware capability for fallback support; but don't assume that because an LTE modem doesn't have fallback support that it isn't vulnerable.) Bluetooth also has known vulnerabilities in both its encryption and authentication methods.

Device makers can mitigate these risks by treating the underlying networks as potentially hostile and implementing compensating mitigations. For example, end-to-end encryption at the device or application level can mitigate problems with any underlying weak encryption methods or the lack of encryption at all. Data received from these networks should also be treated as untrusted.

Best Practices to Employ

Now that we've highlighted common mistakes to avoid, here we outline key helpful practices.

Do Heed Vulnerabilities in Integrated Software Components

Many well-known system exploits come from outside components integrated into the larger system. Examples include achieving control over a telematics system through a weak media parsing library when presented with a malicious media file through exposures, such as Heartbleed, or through vulnerabilities in cellular or communications libraries.

Often, companies ship systems with outdated and vulnerable versions of a component, even though a newer version exists that remediates the known vulnerabilities. The existence of these known vulnerabilities in the components can be particularly consequential, because they're often easy for an attacker to identify, and often a proof-of-concept of how to exploit the vulnerability is publicly available. Software needs to ship with the most up-to-date version that can be incorporated at the time the system is produced. Hence, it's important to know all the components (for example, libraries, external code samples, and frameworks) that exist within the system. This should be done to support licensing requirements and security measures. Use this list when researching and examine the entries for known common vulnerabilities and exposures (CVEs). Consider the number and severity of known vulnerabilities—as well as the ongoing support of the team that creates each component—when deciding whether to use an external component. Look, for example, at the team's responsiveness in remediating identified vulnerabilities, and contemplate how support is likely to evolve in the future, along with other operational factors such as licensing and cost.

BACK TO TABLE OF CONTENTS

Tools can assist with creating an inventory of integrated software components or bill-of-materials. Tools exist to scan source code and unscramble library binaries. Ideally, the third-party source will attest to what's included in the component since a scan may not enumerate all the content of protected binary code. Overall, you need an accurate inventory of components for proper security protection of the device produced.

After product delivery, it's important to continue to update the inventory with new software versions. Maintain and compare all historical lists with new CVEs as they're discovered. This can be challenging, as multiple versions of components might exist even in a single instance of a system. Then it might be necessary to understand where and how the component is used to determine the exact implication for a given CVE.

When a CVE is discovered for a component, the first considered course of action should be updating the system to a component's newer version. In some instances, this might not be feasible. This could be because the component is part of another component that's no longer supported or is incompatible with the updated component. In this case, understanding the risk for the CVE in the component as used is important to manage the risk based on the response. For instance, if you must use a software with known vulnerabilities, then it's essential to implement compensating controls, such as strictly limiting the available API or running the vulnerable component in a sandbox. Furthermore, you might monitor the component continuously in the system to detect abusive action, and, when noticed, immediately stop such action.

Do Harness Existing Protections

There are a wide variety of hardware, operating systems, and compilers used in telematics/infotainment units, from 8-bit microcontrollers to robust platforms with capabilities similar to a desktop operating system. The security features in these also vary widely. It's impossible to enumerate all or even most of the relevant security controls. The following examples illustrate some of the common mitigations available to a telematics/infotainment unit's architect.

Many of these controls traditionally are seen as exploit mitigation. That is, if a vulnerability is compromised, the platform makes it more difficult for the attacker to compromise the device's Confidentiality, Integrity, or Availability (CIA; see the definition on p. 56 of J3061), along with its connection to other devices. The following discusses examples of how to make use of platform-based security features for hardware, OS, and compiler controls.

BACK TO TABLE OF CONTENTS

Security hardware protections. Use hardware protections (or hardware with security protections) to protect access to your code. This generally involves checking the signature on code before executing it. Methods vary depending on the processor. See the datasheet of your processor.

If your hardware supports a secure enclave or Trusted Platform Module (TPM), use the functions of that coprocessor to securely generate cryptographic keys, execute security-sensitive code, or perform standard cryptographic operations. (Usually, writing such code is difficult; only undertake this responsibility if you have technical staff members with suitable experience.)

Because these devices are easily accessible to consumers, they're likely to be physically disassembled and analyzed. Before shipping a device to customers, close off any hardware debugging paths via hardware. For example, you might use fuses to disable the JTAG interface or cut the "transmit" wire on a serial console. Disabling the JTAG interface via fuse is a secure option if the fuses are internal to the process, but if they're external, and especially in the case of serial consoles or JTAG where the wires aren't brought simply to the connector, this can give you a dangerously false sense of security. To be effective, you must be disable them inside the device. Locking JTAG with a device-unique key or password is a possibility if you don't want to disable it entirely. Where that isn't desired, locking JTAG with a device-unique key/password is less ideal, but still an acceptable option.

OS protections. More full-featured OSs have inherent security features available, including exploit mitigation (and if the OS doesn't have this capability, consider using an OS that supports it instead). Many of these security features involve making exploitation payload execution efforts difficult. Preventing the execution of code in user-writable data space (heap or stack) is a common feature, often referred to as *Data Execution Prevention* (DEP) and set via the No eXecute (NX) bit. Another is address space layout randomization (ASLR), which makes it more difficult for an attacker to determine where the code they want to execute is located in memory.

The OS also commonly offers the ability to run applications in a less-privileged execution environment (for example, among users, apps, or

"Before shipping a device to customers, close off any hardware debugging paths via hardware. For example, you might use fuses to disable the JTAG interface."

BACK TO TABLE OF CONTENTS

sandboxes). This makes a subset of the device's memory accessible to the application. It also enforces standardized access to devices, via device drivers, and other applications via various forms of inter-process communication. Be aware of any open ports (with applications listening) on the network interfaces of the device or modem, if applicable (for example, SSH port 22).

Compiler protections. Many compilers have built-in functions that either warn the developer of potential security issues or attempt to mitigate future attacks. One such protection is the use of a stack canary, which is known data that's checked before executing certain code. An attack would overwrite the canary, causing this check to fail. An example of warning the developer comes in the form of the /sdl flag in Visual Studio or -Wall flag to GCC (these are just two examples). Another technique is the use of inline reference guards—control flow integrity and software fault isolation (CFI/XFI)—which are protections inserted into target binaries.

Do Update Components Securely

All software has flaws, and during a system's lifetime, it's likely that some of these flaws will be identified and possibly represent a system risk. An important characteristic of a system's overall lifecycle security is its ability to be updated to address flaws that are identified while the device is in service. We recommend that telematics/infotainment devices have a secure and efficient way to update software that controls the devices' operations, including independent software that controls critical system subcomponents (for example, the baseband firmware in the cellular modem used in a telematics system).

Although such mechanisms can be used to substantially improve these systems' overall security, by allowing identified flaws to be remediated, update mechanisms can themselves represent a significant security risk to the system if they aren't implemented securely. These mechanisms allow code updates that fundamentally determine device behavior, but history shows many examples of systems in this space using insecure software update methods. Examples include using weak methods (or even no method at all) to verify the authenticity and integrity of software presented to a device for update, failure to prevent rollback to previously insecure versions of the software, and methods that only address the integrity (for example, cyclic redundancy checks over the software that prevent undetected accidental modification)—none of these address security concerns adequately.

Telematics/infotainment systems should ensure that software updates are obtained only

BACK TO TABLE OF CONTENTS

from authorized sources, without modification in the process of delivery to the device. Ideally the protection afforded should be end-to-end, with protection maintained all the way from the legitimate software source to the device (a specific processor, for example, that actually executes the code). Preferably a single mechanism should protect the software's end-to-end delivery. Consider the development needs, and the risks that development capabilities could present to a vehicle fleet, when designing a secure update mechanism.

This is fundamentally an application of authentication, and thus the advice of the section "Don't Overlook Authentication for Messages Used in Critical Functions" is applicable. The mechanisms used to achieve this authentication (as well as confidentiality, if required) will likely rely on cryptography, so the issues discussed in the section "Don't Use Cryptography Incorrectly" deserve careful consideration.

It's worth noting that other projects have considered several of the issues involved with software updates—see, for example, the issues and techniques discussed in the automotive-specific Uptane project (see also https://uptane.github.io/).

Do Understand What Your System Logs and Monitors

Systems log a variety of data for different purposes, such as enabling engineers to run diagnostics procedures, understand usage behavior, performing forensics, or understanding a security compromise. Today no clear consensus exists on whether such logging is useful and needed, what actions and what level of details must be monitored, what actions should be taken after monitoring detects an issue, and whether such monitoring should be executed in the vehicle, in the cloud, or in both. (Note that there's consensus in the PC world to log as much as possible to recreate events, assuming that the data can still be efficiently searched for errors.) Rather than clarifying such points, here we provide guidance about monitoring once the system owner decides on a strategy.

Consider the following as guidance when creating a log file:

1. Don't log key credentials or sensitive information (such as a VIN or social security number).
2. Monitor key decisions and actions. Such key actions could include firmware

BACK TO TABLE OF CONTENTS

updates (both successful and failed tries, including signature verification errors), and authentication failures of any kind.

3. Logging, if improperly implemented, might provide quite a bit of information about the system that could be useful to the attacker. The system state information itself might be useful to understand the system's operation, but even the logging code, and the messages that it generates, if stored in clear text, can provide invaluable information that can help the adversary understand how the system operates, particular portions of the code's functions, the capability of functions, and so on. Consider these risks and remediate or accept them.
4. The options to log locally or to backend systems has multiple implications.

    - Local logs might lack a sufficient baseline to detect anomalies and might be compromised if the system is compromised. They also could fail to alert central authorities of issues.
    - Backend logging might have PII and tracking concerns, as data are centralized and aggregated. Additionally, the data could be quite large and unwieldy, making it either impossible or costly to transport all data from the vehicle to the backend.
    - Often a combination of local and cloud-based logging is deployed.

5. Logging can potentially include, or relate to, anomaly detection. For example, a telematics/infotainment unit can monitor the in-vehicle network bus to detect anomalies.

Conclusion

This document is part of a series of practical artifacts from the Center for Secure Design. If you're interested in keeping up with the Center for Secure Design's activities, follow us on Twitter @ieeecsd or via the website (cybersecurity.ieee.org). If you would like to help with CSD activities, contact us at ieee-csd@ieee.org.

For more information about design considerations and vulnerabilities for vehicles' telematics and infotainment systems, see IOactive's white paper related to this topic. Also, see Geotab's "15 Security Recommendations for Building a Telematics Platform Resilient to Cyber Threats" for more information.

BACK TO TABLE OF CONTENTS