U.S. Department
of Transportation
National Highway
Traffic Safety
Administration

NHTSA

DOT HS 812 807

October 2020

# Cybersecurity of Firmware Updates

## DISCLAIMER

This publication is distributed by the U.S. Department of Transportation, National Highway Traffic Safety Administration, in the interest of information exchange. The opinions, findings, and conclusions expressed in this publication are those of the authors and not necessarily those of the Department of Transportation or the National Highway Traffic Safety Administration. The United States Government assumes no liability for its contents or use thereof. If trade or manufacturers' names are mentioned, it is only because they are considered essential to the object of the publication and should not be construed as an endorsement. The United States Government does not endorse products or manufacturers.

Suggested APA Format Citation:

Bielawski, R., Gaynier, R., Ma, D., Lauzon, S., & Weimerskirch, A. (2020, October). *Cybersecurity of Firmware Updates* (Report No. DOT HS 812 807). National Highway Traffic Safety Administration.

## Technical Report Documentation Page

| 1. Report No. | 2. Government Accession No. | 3. Recipient's Catalog No. |
|---|---|---|
| DOT HS 812 807 | | |

| 4. Title and Subtitle | 5. Report Date |
|---|---|
| Cybersecurity of Firmware Updates | October 2020 |
| | 6. Performing Organization Code |

| 7. Authors | 8. Performing Organization Report No. |
|---|---|
| Russ Bielawski, Ron Gaynier, Dr. Di Ma, Sam Lauzon, and Dr. André Weimerskirch | |

| 9. Performing Organization Name and Address | 10. Work Unit No. (TRAIS) |
|---|---|
| Transportation Research Institute<br>University of Michigan (Ann Arbor, MI)<br>University of Michigan-Dearborn<br>Volkswagen Group of America (Herndon, VA) | 11. Contract or Grant No.<br>DTNH22-15-R-00104 Vehicle Electronics Systems Safety IDIQ |

| 12. Sponsoring Agency Name and Address | 13. Type of Report and Period Covered |
|---|---|
| National Highway Traffic Safety Administration<br>1200 New Jersey Avenue, SE<br>Washington, DC 20590 | Final Report |
| | 14. Sponsoring Agency Code |

**15. Supplementary Notes**

**16. Abstract**

Over-the-Air (OTA) software and firmware updates are widely considered essential for networked devices. In the automotive industry, OTA firmware updates are anticipated to increase the efficiency and decrease the time in updating the critical firmware in vehicles' electronic control units (ECUs). This project had these objectives: understand the scope and relevant attributes of firmware updates, understand their vulnerabilities and update solutions, understand mitigation methods for those vulnerabilities, and learn from adjacent industries.

The report first presents a literature and technology review of the state-of-the-art of software updates in industries related to automotive, including the commercial aviation, medical, and consumer electronics industries. Next it identifies and assesses software update functionality risks in current and near-term future automobiles. Finally, it reviews mitigation methods to address those risks. In addition, this report describes the SAE AS5553A voluntary standard for the detection of and protection against counterfeit electronic parts in the aerospace industry and how it relates to the automotive industry.

| 17. Key Words | 18. Distribution Statement |
|---|---|
| Over-the-Air, Updates, Firmware, Software, ECU (Electronic Control Unit), Automotive, Network, Theft, Counterfeit, SAE AS5553A | This document is available to the public through the National Technical Information Service, www.ntis.gov. |

| 19. Security Classif. (of this report) | 20. Security Classif. (of this page) | 21. No. of Pages | 22. Price |
|---|---|---|---|
| Unclassified | Unclassified | 103 | |

**Form DOT F 1700.7 (8-72)** **Reproduction of completed page authorized**

# Executive Summary

Over-the-Air (OTA) software and firmware updates are widely considered essential for networked devices. In the automotive industry, OTA firmware updates are anticipated to increase the efficiency and decrease the time in updating the critical firmware in vehicles' electronic control units (ECUs). There is a demand to better understand firmware and software updates, particularly for embedded systems, and how to implement them securely.

This work had the following objectives.

- Understand the scope and relevant attributes of firmware updates
- Understand the vulnerabilities of firmware update solutions
- Understand the mitigation methods for those vulnerabilities
- Learn from adjacent industries

The report first presents a literature and technology review of the-state-of-the-art of software updates in industries related to automotive, including the commercial aviation, medical, and consumer electronics industries. Next it identifies and assesses the risks presented by software update functionality in current and near-term future automobiles. Finally, it gives a review of the mitigation methods to address those risks. In addition, this report describes the SAE AS5553A voluntary standard for the detection of and protection against counterfeit electronic parts in the aerospace industry and how it relates to the automotive industry.

**Summary of Lessons Learned in Adjacent Industry:**

Common existing defense mechanisms (e.g., signing, fortification, and intrusion detection) and vulnerabilities are noted in the body of the report as are potential defenses for secure vehicle firmware updates.

**Risk Assessment Conclusions:**

In identifying risks at both the vehicle-level and the technological design and implementation level, the researchers have identified the biggest risk with software update mechanisms as malware installation.

**Mitigation Methods Conclusions:**

In-field software updates are a necessity in the automotive industry to fix flaws without replacing hardware that is already deployed in the field. The current generation of automobiles primarily uses OTA software updates for telematics and infotainment ECUs only.

While software updates are a boon for security, the mechanism, particularly the remote mechanism, creates a new avenue for attackers to exploit.

A matrix of specific mitigations versus risks appears in the report (see Table 17).

**Intellectual Property Theft Risks and Mitigations Conclusions:**

Intellectual property theft, particularly software theft, can be enabled and made easier with software update mechanisms, particularly OTA mechanisms. In discussions with the original equipment manufacturer (OEM) and tier-1 supplier employees, the majority opinion is that protecting the software binaries is not a priority. The prevailing opinion in the industry is that

there are too many other ways for an adversary to obtain a software binary to justify the cost of adding encryption to the software update process.

**Counterfeit and Fraudulent Electronic Parts and Products Conclusions:**

Fraudulent and counterfeit parts can pose a safety and monetary liability risk. SAE AS5553A is an aerospace standard for the creation of processes for detection, prevention, mitigation, and disposition of suspect, fraudulent, and counterfeit electronic parts. In general, SAE AS5553A should apply to the automotive industry quite readily. It is designed to be flexible and risk-informed. The requirements themselves should be applicable to the automotive industry; however, a more tailored collection of best practices might be reasonable to develop for the automotive sector specifically (not developed within this project).

**Final Conclusions:**

Secure in-field software updates are nearly universally considered to be essential for any networked computer system. However, software update functionality creates a new attack surface for attackers to potentially exploit. The installation of malware is one of the biggest risks for software updates.

There is no singular, perfect reference model for securing software updates. Every system has different requirements and user experience targets that shape the design enough to require security to be at a minimum analyzed and usually designed with an application-specific approach.

While software updates have a large surface from which vulnerabilities can potentially spring, many of the mitigations are known. Software update functionality can be attacked at many different places in the distribution process. And, while technical risks exist, many of the risks are social (such as lost passwords, etc.) in nature. The benefit of reliable, prompt software updates for in-field electronics is significant.

# Table of Contents


Executive Summary ..... ii

Summary of Lessons Learned in Adjacent Industry: ..... ii

1. Introduction ..... 1

2. Background, Definitions, and Literature Review ..... 2

2.1 Background ..... 2

2.2 Software Update: Overview and Definitions ..... 2

2.2.1 Current automotive software update mechanism and best practices. ..... 3

2.3 Step-By-Step: The OTA Software Update Process ..... 5

2.3.1 Packaging. ..... 6
2.3.2 Transport. ..... 6
2.3.3 Reception. ..... 6
2.3.4 Installation. ..... 6
2.3.5 Verification and maintenance (optional). ..... 6

2.4 Software Update Packaging ..... 7

2.4.1 Complete image overwrite. ..... 7
2.4.2 Delta update. ..... 8
2.4.3 Dynamic update. ..... 8
2.4.4 Distributed update. ..... 9
2.4.5 Aircraft avionics. ..... 9

2.5 Update Package Transport ..... 9

2.5.1 Medical devices. ..... 9
2.5.2 Aircraft avionics. ..... 10

2.6 Update Package Authentication, Verification, and Unpacking ..... 10

2.6.1 Aircraft avionics. ..... 10

2.7 Software Update Installation ..... 10

2.7.1 Medical devices. ..... 11
2.7.2 Aircraft avionics. ..... 11

2.8 Verification and Maintenance (Optional) ..... 11

2.8.1 Aircraft avionics. ..... 12

2.9 Related Security Issues ..... 12

2.9.1 Secure boot features in relation to secure software update. ..... 12

2.9.1.1 The chain of trust in software. ..... 12

2.10 Lessons Learned From Adjacent Industry for security software updates ..... 12

2.10.1 Industry/device attributes. ..... 12

2.10.2 PC BIOS updates. ..... 14
2.10.3 PC operating system and standalone application software update. ..... 16
2.10.4 Printer firmware update ..... 17
2.10.5 Smart battery firmware update. ..... 18
2.10.6 Medical devices. ..... 18
2.10.7 Commercial aviation. ..... 19
2.10.8 Summary of lessons learned in adjacent industry. ..... 20
2.11 Conclusion ..... 20
3. Risk Assessment ..... 21
3.1 Reference Vehicle Architecture Model ..... 21
3.2 Baseline Threat Model and Methodology ..... 22
3.2.1 Vehicle-level risks. ..... 22
3.2.2 Threat actors. ..... 23
3.2.3 Baseline threat model. ..... 23
3.3 Attack Scenarios ..... 27
3.3.1 Malicious control of vehicle. ..... 27
3.3.1.1 Remote malicious control of many vehicles. ..... 27
3.3.1.2 Remote malicious control of a small number of vehicles. ..... 27
3.3.1.3 Near-range malicious control of a small number of vehicles. ..... 28
3.3.2 Denial-of-service of vehicle. ..... 28
3.3.2.1 Targeted, coordinated denial-of-service. ..... 29
3.3.2.2 Ransomware. ..... 29
3.3.3 Vehicle or contents theft. ..... 29
3.3.3.1 Pairing an unauthorized key to a vehicle. ..... 30
3.3.4 Intellectual property theft / private information exfiltration. ..... 30
3.3.4.1 Eavesdropping. ..... 30
3.3.4.2 Activity logger software installation. ..... 31
3.3.5 Performance tuning or unauthorized feature activation. ..... 31
3.3.5.1 Performance tuning. ..... 31
3.3.5.2 Unauthorized feature or content activation. ..... 32
3.3.6 Summary. ..... 33
3.4 Technical Risks ..... 33
3.4.1 Rogue software (malware) installation. ..... 35
3.4.1.1 Real authority signs unauthorized software. ..... 35
3.4.1.2 Supplier compromise. ..... 35
3.4.1.3 Signing credentials are stolen. ..... 36
3.4.1.4 Attacker forges signature on inauthentic software. ..... 36
3.4.1.5 Attacker remotely exploits a software flaw. ..... 37

3.4.1.6 Attacker uses undocumented bypass functionality. 37
3.4.1.7 Attacker sneaks hidden functionality into app store. 37
3.4.1.8 Attacker with physical access installs malware. 38
3.4.1.9 Attacker installs malware with hacked OBD dongle. 38
3.4.1.10 Physical media tampering in transit. 38
3.4.1.11 Software installation tools are compromised. 38
3.4.1.12 Forged software bypasses system verification routines. 39
3.4.2 Denial-of-service. 39
3.4.2.1 Attacker masquerades as a legitimate server in distribution network. 39
3.4.2.2 Attacker spoofs a legitimate ECU and tampers with software updates. 40
3.4.2.3 Attacker spoofs legitimate wireless interface access point. 40
3.4.2.4 Jamming of wireless interfaces. 40
3.4.3 Unauthorized download of information. 41
3.4.3.1 Attacker masquerades as legitimate ECU to download data. 41
3.4.3.2 Attacker abuses data gathering functionality. 41
3.4.3.3 Man-in-the-middle / man-on-the-side. 42
3.4.3.4 Digital rights management circumvention. 42
3.5 Risk Assessment Discussion 42
3.5.1 Code signing. 42
3.5.2 Automatic updates. 42
3.5.3 Robustness against denial-of-service attacks. 42
3.5.4 Full software updates vs configuration tweaks. 43
3.6 Conclusion 43
**4. Mitigation Methods 44**
4.1 Definitions 44
4.2 Mitigations 45
4.2.1 Malware installation. 46
4.2.1.1 Update authentication. 49
4.2.1.2 Secure channel (authentication and encryption) 50
4.2.1.3 Secure in-vehicle networks. 51
4.2.1.4 Entity authentication. 51
4.2.1.5 User authentication and authorization. 55
4.2.1.6 Use a root-of-trust-for-update. 55
4.2.1.7 Protect keys and security-relevant data stored in ECUs. 56
4.2.1.8 Prevent bypassing of authentication mechanisms. 57
4.2.1.9 Prevent forgery or unauthorized generation of digital signatures. 57
4.2.1.10 Separation of duties. 58
4.2.1.11 Code reviews before code deployment. 58
4.2.1.12 Ethical hacking and penetration testing. 59
4.2.1.13 Quickly fix security bugs for in-house and third-party software. 59
4.2.1.14 Traditional IT best practices. 59

4.2.1.15 App store security. ..... 60
4.2.1.16 Physical security. ..... 60
4.2.1.17 Secure vehicle architecture. ..... 60
4.2.2 Denial-of-service. ..... 61
4.2.3 Unauthorized download of information. ..... 62
4.2.3.1 Update encryption. ..... 63
4.2.3.2 Data gathering encryption. ..... 64
4.2.3.3 Access policies. ..... 64
4.3 Conclusion ..... 64
**5. Intellectual Property Theft Risks and Mitigations ..... 66**
5.1 Abuse Cases and Risks ..... 66
5.2 Mitigations ..... 67
5.2.1 Physical tampering protection. ..... 68
5.3 DRM Circumvention ..... 68
5.4 Conclusion ..... 70
**6. Counterfeit and Fraudulent Electronic Parts and Products ..... 71**
6.1 SAE AS5553A ..... 71
6.1.1 Personnel training. ..... 73
6.1.2 Parts availability. ..... 73
6.1.2.1 Planning and obsolescence management. ..... 73
6.1.3 Purchasing process. ..... 74
6.1.3.1 Risk assessment. ..... 74
6.1.3.2 Supplier selection. ..... 74
6.1.3.3 Supplier auditing. ..... 74
6.1.3.4 Applicability to automotive. ..... 74
6.1.4 Purchasing information. ..... 75
6.1.4.1 Supply chain traceability. ..... 75
6.1.5 Verification of purchased/returned parts. ..... 75
6.1.6 In-process investigation. ..... 76
6.1.7 Failure analysis. ..... 76
6.1.8 Material control. ..... 76
6.1.8.1 Control of suspect, fraudulent, or counterfeit parts. ..... 77
6.1.8.2 Applicability to automotive. ..... 77
6.1.9 Reporting. ..... 77
6.1.10 Postdelivery support ..... 77
6.2 Applicability to Software Products and Software Components ..... 78
6.3 Conclusion ..... 78

**7. Conclusion ..... 80**
**8. Glossary ..... 82**
**References ..... 89**
**Document History ..... 92**

# 1. Introduction

Electronic control units (ECUs), and the ability to update the software they contain, have been in use for decades in vehicle control applications. Traditionally software updates have been the domain of auto dealerships, service centers, and home mechanics with aftermarket programming tools, and little or no authentication was required. With the introduction of wireless communication within vehicles has come the potential capability to distribute software remotely without attaching a programming tool to the vehicle controller area network (CAN) bus. The advantages of remote software updates to vehicle manufacturers are reduced warranty costs, improved customer satisfaction, and the ability to offer customers improved features and content.

The importance of software in computer system architecture makes it an attractive target for attackers. Software modification attacks on various embedded systems have been demonstrated repeatedly at hacking conferences and in academic publications. The capability of OTA updates for vehicle software only widens the attack vector, making it possible for hackers to distribute malware to millions of vehicles simultaneously.

While the threats with respect to an OTA update procedure with cybersecurity vulnerabilities are daunting, there is a need to understand software update techniques, the potential threats, as well as potential countermeasures. This project studies the cybersecurity of automotive software updates. The objectives of this project are to define terms commonly used in this domain and identify interesting attributes; survey available firmware update mechanisms used in the automotive industry and across other industries; perform a literature review that also covers all industries; assess cybersecurity threats due to software update methods and practices; and study, and propose mitigation mechanisms.

# 2. Background, Definitions, and Literature Review

## 2.1 Background

Modern cars are controlled by complex distributed systems comprising millions of lines of code, executing on tens of heterogeneous digital components known as ECUs. ECUs are interconnected by serial data networks such as CAN, Ethernet, or FlexRay buses. They oversee a broad range of functions including engine, transmission, brakes, steering, windows, locks, lighting, and entertainment systems. Operations not mediated by computer control in a modern vehicle are shrinking, increasingly so with the rise of automated driving technology. This shift toward computer-controlled cars has offered significant benefits to efficiency, safety, and cost. The ability to update firmware running on heterogeneous ECUs is an important part of routine maintenance to fix software bugs, support new features, and improve performance of the vehicle.

Traditionally, software is installed during the manufacturing process, and updates are performed in auto dealerships or service centers by trained mechanics using special programming tools that are physically connected to cars. Today, while most vehicle manufacturers are distributing software and data updates for infotainment features directly to vehicles OTA, only one, Tesla, is performing OTA updates of ECUs that support safety-critical vehicle functionality [Bri12]. OTA updates are made possible as modern cars become increasingly equipped with various wireless communication interfaces, for example, Bluetooth, Wi-Fi, radio frequency identification (RFID) near-field communications (NFC), dedicated short-range communications (DSRC), and cellular, some of which provide internet access. The authors refer to wireless interfaces providing internet access, generally, as wireless wide area network (WWAN) interfaces. Although OTA update offer benefits such as reduced warranty costs and improved customer satisfaction, it also makes remote attack through OTA update mechanisms possible and widens the possibility of undesired operation. Moreover, the transition to OTA update will shift some of the responsibility for successful completion of the update to end users. This means the OTA update process must be implemented to be both secure and robust.

## 2.2 Software Update: Overview and Definitions

Software is a general term that can be defined as including one or more of the following:

- **Bootloader:** A small block of software facilitating the startup of an operating system or firmware, and for performing updates.
- **Operating System:** The main software which facilitates the execution of one or many applications.
- **Application:** The software which implements a feature/function on a computer or ECU.
- **Data:** Information that is necessary for proper execution of an application. In ECUs, this is typically constant data that configures the application software for the specific vehicle feature's content.
- **Firmware:** Software designed to perform a set of functions that is updated infrequently.
- **Over-The-Air:** A software update distribution method which uses wireless transmission.

The software update process can be broken down into four individual process steps (described further in Section 3):

1. **Packing:** Preparation of the software update for distribution.
2. **Transport:** Mechanism by which the update is transferred from the source to the target.
3. **Reception:** Receipt and initial processing of the software update data which includes unpacking, authentication and consistency checks.
4. **Installation:** Process of updating the memory with the new software including validation (i.e., what was received was installed properly, not to be confused with the authentication which occurs during reception).
5. **Verification and Maintenance (Optional):** Ongoing security and robustness checks throughout the life of the product.

These four mandatory steps are displayed in Figure 1. For a secure OTA update, an optional fifth step of verification and maintenance could be useful to allow for periodic checks of consistency, troubleshooting, and versioning information in a post update setting. While not immediately falling under the scope of secure OTA updates, this fifth step can be used to determine the conditions for follow-up software updates and alert the software distributor in the event of a security problem because of unexpected changes in software versions.

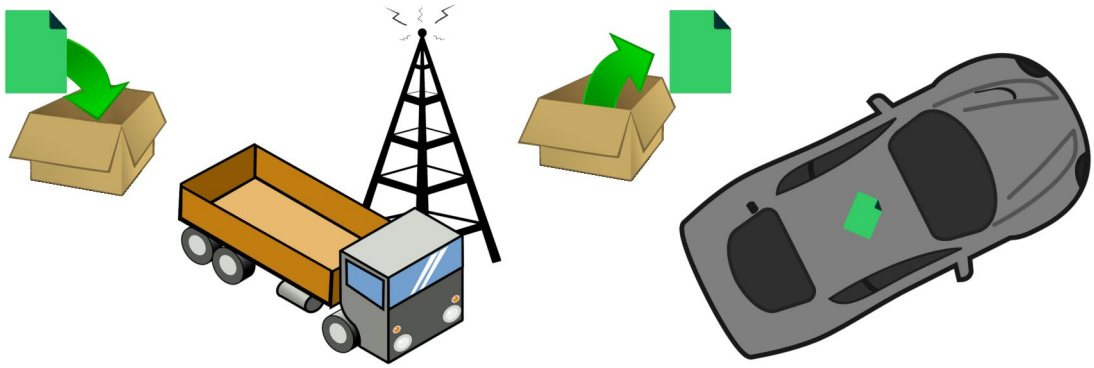

**Figure 1.** Packaging, Transport, Reception, and Installation: The four mandatory software update steps.

### 2.2.1 Current automotive software update mechanism and best practices.

Firmware is a type of software that provides low-level control of computer systems, including embedded systems, computers, computer peripherals, mobile phones, and so on. Firmware derives its name from the fact that it is usually held in non-volatile memory that in normal operation is considered read-only memory (ROM). Firmware is a subset of the term *software*, and in this document, the terms are used interchangeably, although the emphasis remains on firmware.

ROM technology has evolved from mask programmed devices that are truly read-only to devices that can be erased and reprogrammed. The first of these were the erasable programmable ROM (EPROM) that could be programmed electrically by applying special programming signals and erased by ultraviolet light applied through a fused quartz window in the top of the package. This was followed by electrically erasable programmable ROM (EEPROM) that can be both programmed and erased via special programming signals.

Flash memory is a form of EEPROM. Uniquely, flash memory is erased in blocks whose sizes are determined by the device implementation, whereas traditional EEPROM can be erased at the byte level. The minimum erase block size also sets the minimum amount of memory that must be written during any write event, since to write to memory, it must first be erased. Flash memory may be grouped into partitions with each partition containing one or more erase blocks. In most implementations, a flash ROM partition cannot be read while an erase or write operation is being performed on any block within that partition. Flash memory is cheaper than traditional EEPROM, and thus comes in larger capacities. Flash memory is the near universal ROM technology used in automotive embedded control applications.

Error correcting codes (a.k.a. error correcting circuitry or error checking & correction, but always abbreviated ECC) are now offered as a feature of flash memory for safety critical embedded control applications and provide single-bit correction and multi-bit detection. This requires storage of the ECC checksum during write of flash (performed automatically by dedicated circuitry). For example, each ECC code might apply to each 64 bits aligned on a 0 modulo 8-byte address. This sets the minimum amount of memory that must be written to 8 bytes to ensure proper ECC code calculation.

Most ECUs are field updateable; that is, they contain a bootloader, which can receive new software from the network (such as CAN or a wireless link directly in some cases) and programming the ROM memory. Programming the memory is sometimes called “flashing” due to the prevalence of flash memory. The bootloader resides in flash ROM also, but, on most hardware, the code used to erase and write the flash ROM must run out of a different partition or memory device, such as RAM. This bootloader differs from the bootstrap mechanism that is part of the debug port of the microprocessor that is used during development. The bootloader is software dedicated to providing the serial communications handler (typically CAN, supporting the Unified Diagnostic Services (UDS) protocol), flash erase/write routines, plus any security, code validation, and recovery features. The so-called secondary bootloader, including the flash erase and write routines, has, historically, been downloaded over the network interface prior to the installation step when the programming happens, but that design is not universal.

While the bootloader is designed to be robust and allow for recovery from failed programming sessions, updating ECU firmware for critical components has been limited to the assembly plant or dealerships where trained individuals could facilitate restarting/recovery of a failed update process up to and including replacement of the ECU if necessary. Tesla is the exception to this rule [Bri12]. However, the implementation details of Tesla’s OTA update mechanism are not known and some owners have reported having issues during updates. OTA firmware updates to a vehicle in the possession of the owner will necessitate increased robustness, in addition to increased cyber security features, to ensure the vehicle is always left in a drivable state for all but the rarest cases (where damage is extensive that the system cannot be brought back to a safe operational state). Having even a modest percentage of customers adversely affected by a failed firmware update is likely to be unacceptable to manufacturers and the public.

Updating firmware in ECUs relies on first erasing the existing firmware prior to loading in new firmware into the same memory location to keep Flash memory size requirements to a minimum.

In the case of critical ECUs, this renders the vehicle unavailable from the time that the erase begins until the new firmware is completely loaded. Requiring the ECUs to have double the memory space, to provide for a complete firmware update prior to erasing the existing firmware, is one proposed solution to this problem.

A stable power source is necessary during the firmware update to ensure success. Most OEMs recommend that their dealership service technicians place the vehicle on a battery charger if firmware updates are performed. Owners of electric or plug-in hybrids will be accustomed to plugging in their vehicles, but it is unrealistic to expect owners of conventional gasoline and diesel vehicles to own battery chargers (power outages are still a possibility).

Events that can lead to corruption, such as power failure while writing software to the flash memory, can cause an ECU application to be unusable until a successful update can be performed. In addition, the bootloader must be designed in such a way that it provides a means of recovery in the event of an update failure. Otherwise, even worse than an ECU unusable application, an ECU might become "bricked" where the hardware cannot be field repaired and must be replaced. To minimize the probability of an unusable or bricked ECU and make the update process fail-safe, some best practices include:

1. **Separate program spaces**. Firmware can be divided into multiple program spaces including a bootloader, the firmware OS, Application and Data, and/or a backup program with different read/write policies. In many cases, it is a good idea to have a well-validated, read-only bootloader. However, not all hardware, due to cost constraints, provides a mechanism to guarantee that.
2. **Backup.**
   - In case the bootloader itself needs to be upgradable, a backup bootloader is essential to prevent bricking.
   - If memory allocation is not a problem, the new firmware image could be written into memory in a separate memory space while retaining the current firmware image. This allows for immediate recovery to full operation if the update fails.
3. **Integrity/Checksum.** It is a good practice that the bootloader always does an integrity check before running the firmware application code (at start up, or after a download is completed). Unsuccessful verification indicates corrupted updates. The bootloader can signal to the user to re-start a new download or it can reset to default/previous firmware stored on the device if it is still available in the memory.

## 2.3 Step-By-Step: The OTA Software Update Process

For the purposes of investigation and comparison to adjacent industries, UMTRI has broken down the software update process to the basic components. This allows the research team to then identify the benefits, drawbacks, and security implications during each process step of the secure OTA update. This section provides a detailed description of each step and they are discussed individually in the following sections.

Commercial aviation and medical devices are adjacent industries that seem to offer the most to the auto industry since they are both safety critical applications and have some similarities in

platform and device architectures. The results of what was learned about them to date are contained within the following sections. Discussions of PC software updates and the consumer electronics industry are reserved for Section 2.10. Table 1 provides an overview of the update process steps in the automotive and adjacent industry domains.

### 2.3.1 Packaging.

Packaging is how the data required to update an ECU is prepared for transportation to the destination, (e.g., a vehicle service center, dealership, vehicle owner, website for further distribution) which generally also includes the necessary documentation and a means by which the update data is protected from both corruption and nefarious tampering by outside sources during transport.

### 2.3.2 Transport.

Transport is how the package containing the necessary data makes its way to the target location to be applied. In some cases, this may be by courier on physical media[1] to a service technician (wired/dealership), or, via secure HTTPS connection through wireless cellular modem directly to an ECU. If not properly secured, the transport phase can be used as an attack vector to swap authorized packages with unauthorized or modified versions, or to misdirect authorized packages to an unintended destination leading to potential device compromise.

### 2.3.3 Reception.

Reception is when the intended ECU receives the package from transport, authenticates any signatures contained in the package and validates the consistency.

### 2.3.4 Installation.

Installation is the means by which the data contained in the package is applied to a target ECU in a target vehicle. As ECUs are networked together, the ability for one ECU to program another emerges as a viable solution where only one ECU is exposed outside the vehicle via wireless connection or a physical service port. Thus, these exposed areas must remain secure to ensure only the appropriately authorized update data is applied.

### 2.3.5 Verification and maintenance (optional).

Manufacturers may query the status, versions, consistency, and ECU authentication to find issues that may have arisen after an update occurs. This can be particularly useful in the event of an ECU failing to update, thus reverting to its prior version, or to identify end-user modification to the ECU's software. Conversely, it may be desirable to upload the version of software currently being run if it is not identifiable by the manufacturer and thought to have been modified by a third party.

[1] CD, thumb drive, etc.

| | Automotive | Aviation | Medical | Consumer Electronics |
|---|---|---|---|---|
| **Packaging** | Variable[2] | Standard process, Physical Media or "EDS Crate" | Variable | Variable |
| **Transport** | Wired/Wireless/Dealership | Physical Media/ Secure Network to Airline database. Local wireless to aircraft. | Physical Media | Wireless |
| **Reception** | Single Module | In-Plane | Manual download/ extraction | Automatic |
| **Installation** | Single Module, Manual application | Single Module, Manual application | Single Module, Manual application | Device, Automatic application |
| **Verification & Maintenance** | Post-Application | Post-Application Continual- Local wireless to aircraft. | Post-Application Inspection | Continual |

**Table 1.** Overview of Update Process Steps of Adjacent Industries.

## 2.4 Software Update Packaging

There are many possible ways to package a software update for application in the target system depending on the intended application. Encryption may be required, but generally encryption is used to ensure the privacy of the information the package contains and does not ensure the authenticity of the package. Encryption methods are not discussed here.

### 2.4.1 Complete image overwrite.

The simplest way to implement software updates is to overwrite the entire firmware image. This is by far the most widely used firmware update method in the auto industry today. However, it introduces unnecessary wear and tear (erase/write cycles), prolongs the update time, especially when the actual change is small and increases both the storage space and bandwidth needed to store and transfer the image.

[2] Packaging of software updates for the automotive, medical, and consumer electronics industries can be done in many different ways.

### 2.4.2 Delta update.

Delta update or differential update is an update mechanism where only the difference between different versions is updated. Major automatic OS update implementations, such as Windows', use this method to reduce unnecessary communication cost. Delta updates can add security through obscurity, as patch files may be more difficult to dissect than full updates in the event they are intercepted by a malicious user. Although delta updates are effective in reducing communication cost in theory, they may bring additional issues due to the characteristics of flash memory and the increased complexity of the update process.

First, as firmware is usually stored in flash memory, the characteristics of flash memory needs to be considered. Flash memory requires explicit erasure before data can be written. One cannot write over existing data directly. Moreover, erasure in flash memory is done in units of blocks or pages. Unchanged data may have to be shifted to accommodate the new written data of different sizes. To solve the problem of shifting code, fragment layout has been proposed by inserting gaps between partitions (i.e., erasure units). However, this approach leads to fragmentation. It may no longer be effective after a series of firmware updates.

Control-flow dependency is another problem of shifting code. Once a called function is updated, all the caller functions need to be updated with the new address, and these caller functions may reside in different partitions. Indirect call through a jump table is a common solution to solve the control-flow dependency. Whenever there is a called function update, only the jump table needs to be updated. However, it incurs runtime overhead for each function call. The commercial software industry offers several solutions for performing delta updates, for use in different system architectures, while supporting many image formats. Some solutions claim to have automotive editions that are Motor Industry Software Reliability Association (MISRA) compliant to promote bug-free operation through strict source code auditing. Free software solutions for creating this type of data also exist, such as xdelta and bsdiff, which allow for customized solutions without commercial overhead costs. In the update of software images that contain file systems used by operating systems, a differential update can be implemented to modify only the individual files on the filesystem, rather than update the entire image. As the files being updated may be the same files used at runtime, this patching method may have to exit normal operation modes and enter a programming or recovery mode to accomplish this operation. Having a redundant recovery mode may also be desired by a system implementation to direct a user in the event of a failed programming operation or general system failure.

### 2.4.3 Dynamic update.

Traditionally, firmware update involves overwriting existing firmware in a flash memory, and restarting the computing system to execute the new firmware so that the new firmware can take control of hardware resources. Dynamic update is an update mechanism that is used to update software systems while they are still running, and there is no need to stop and restart the system. If any running program is presented as a tuple ($S$, $P$), where $S$ is the current program state and $P$ is the current program code, dynamic update can be thought of as a transformation from a running program ($S$, $P$) to a new version ($S'$, $P'$).

Dynamic update is not commonly used in industry although a wide variety of dynamic systems has been proposed by the research community and tested on real-world systems. Some distributions of Linux systems such as Oracle Linux and Red Hat Linux use dynamic update techniques for live kernel patching [Vau15]. Unisys' patent for "Dynamic Firmware Updating System for use in Translated Computing Environments" provides a mechanism for dynamic firmware updates in a running computing system [U12].

There are no known example companies within the auto industry adopting dynamic updates yet. Still, dynamic update might be interesting as it does not require the system to be stopped first. For example, dynamic update can be used to deploy a security fix on a running and vulnerable vehicle system. Imagine an attack on vehicles in a major metropolitan area during rush hour traffic; rapid restoration of traffic flow would require a dynamic update process.

### 2.4.4 Distributed update.

Another approach is described extensively in [SKHR15]. An IEEE 802.11s mesh network is used for distributing software update files. An 802.11s wireless device is plugged into the vehicle's on-board diagnostics (OBD) port, which connects to both the vehicle's CAN bus and a mesh network in the local area. A technician with an 802.11s-enabled device (laptop, tablet, etc.) could join the network, transmit update instructions and data to the device connected to the vehicle, and diagnose any issues that arise.

### 2.4.5 Aircraft avionics.

Prior to networked means of distributing Loadable Software Aircraft Parts (LSAP), hard media (floppies) were sent to the airline maintenance centers. They were carefully labeled and sent by bonded courier. With the advent of secure network protocols, the industry developed the Electronic Distribution of Software (EDS) Crate, as described in the ARINC 827 standard [AR827], "This standard describes the format for electronic distribution of aircraft software parts and other contents between aerospace business partners using a digital container referred to as an EDS crate." This standard established the format and authentication requirements for LSAP transferred electronically. The LSAP is digitally signed, any required paperwork is added, and the entire package is signed again. Complete image overwriting is done since each new LSAP is a separate part from any that it replaces and a maintenance technician is required to validate and test each new part installation.

## 2.5 Update Package Transport

Transport is described in Section 2.3.2. The package transport is likely one of the most important aspects of the secure software OTA system. Any vulnerability in the transport mechanism may lead to the inability to reliably communicate with a vehicle in the field, loss of intellectual property via unauthorized interception, or possibly even leaving vehicles susceptible to third-party reprogramming and control.

### 2.5.1 Medical devices.

The initial investigations have shown that the distribution and transport of updates for medical devices is largely at the discretion of the device manufacturer. Generally, packages are

distributed over the internet and accessed by the end professional via the service website. In one instance, surveys showed that device updates for a basic heart monitor are generally available without authentication, on a website designed specifically for download by the medical IT staff. While in the case of larger imaging devices, medical IT staff said that the websites require identification of the machine to be updated (via serial number) prior to making a package available for download.

### 2.5.2 Aircraft avionics.

As with the packaging, there also are standards for the transport of LSAP electronically. ARINC Report 666 [AR666] states: "This document describes a secure internet facility for sending all types of aviation software using the World Wide Web (www). Software suppliers may use this document as a starting point for the construction of a secure web server utility. It provides sufficient flexibility and is compatible with numerous software distribution models. The file management structure is consistent with other ARINC standards for loadable software, and references ARINC 665 (*Loadable Software Standard*) [AR665] and ARINC 667 (*Guidance for the Management of Field Loadable Software*) [AR667]. While some flexibility exists in implementing the transport step, the standards do create an environment of careful management and security for LSAP distribution on the web. Also, like the medical devices, airlines are notified by the manufacturer when an update is available and it is the responsibility of the airline to retrieve and apply the update.

## 2.6 Update Package Authentication, Verification, and Unpacking

Once received by the destination vehicle, the package must be checked for consistency and its authenticity verified to remain secure. Optionally, a package may be decrypted if encrypted at the time of packaging. Encryption protects the privacy of the package content but does not prove its authenticity. Generally, the operations in this step will entirely depend on the operations used at the time of packaging. Ideally, this data should not be decrypted or unpackaged in a place that is readable or accessible by the end user.

### 2.6.1 Aircraft avionics.

ARINC 835 (*Guidance for Security of Loadable Software Parts Using Digital Signatures*) [AR835] establishes the standard for use of digital signatures for authentication. As noted in section 5.2.2, the LSAP within the EDS Crate is signed, and the assembly of LSAP and documents within the crate is signed separately. As stated in Section 2.5.2 it is the responsibility of the airlines to retrieve the crate from the manufacturer when it has been made available.

## 2.7 Software Update Installation

The installation of the unpackaged data will largely depend on the architecture of the target ECU (single or multiprocessor, etc.), the type of data being updated (application software, configuration data, calibration data, etc.), the state of the vehicle, and any input that may be required by the user. Care must be taken to ensure the vehicle is in a prepared state to perform an update operation; the use of ignition or transmission locks may be necessary to ensure safety, the

state of charge of the battery should be taken into consideration to ensure robustness, and proper user instruction is required.

### 2.7.1 Medical devices.

The current findings indicate that installation of software on these devices is done strictly by a medical IT professional. These professionals have a general understanding and/or specific training on the device being updated, and in some cases, specific login credentials to the device as well as credentials to retrieve the update from the manufacturer's website.

### 2.7.2 Aircraft avionics.

Airlines work in cooperation with the manufacturers to establish proper electronic inventories for their software parts. These are the repositories where the LSAP are loaded when retrieved from the manufacturer. The LSAP are transferred to the aircraft via a local secure wireless connection provided by the airline at some airports, and each aircraft has keys for validating the connection. The LSAP is then transferred to the aircraft and staged for installation. The LSAP remains staged until a maintenance professional can assess the aircraft for the final installation, verification, and testing of the LSAP.

## 2.8 Verification and Maintenance (Optional)

While not entirely scoped to the secure OTA update process, the manufacturer's desire to query the status, versions, consistency, and ECU authentication of the vehicles they are responsible to update in the field is roughly estimated to increase in a similar pace as the desire to deploy OTA update functionality. The useful information that can be attained in a wireless and unattended manner can also give manufacturers further insight into how often these vehicles are serviced by third-party technicians, how often equipment fails, how much distance the vehicle has driven, and possibly even which radio station is listened to most often or other personal operator statistical data.

In the scope of secure OTA update, the ability to query the status, version and current state of the module allows a manufacturer to structure a differential update specifically for that ECU. Generally, to apply a binary patch (such as in a differential update) to a specific piece of software, both the current and desired versions must be entirely known. A difference is taken at the development facility between the desired and current versions, and those differences are stored in what is known as a *patch* or a *diff*. This patch can then be applied to the current version deployed in the field to create the desired version on the target dynamically. If the current version in the field differs from the current version in the development facility, the patching operation will fail.

The transmission of this differential data is desired from a security aspect as it allows for some protection against reverse engineering. In the event the transport mechanism is ever compromised, the attacker would end up with only a partial record of the data in the ECU contained in a differential update. Compared to a compromise in a full update transmission, an attacker will only have bits and pieces of the data. This makes it much more difficult for the attacker to understand the inner workings of the ECU.

### 2.8.1 Aircraft avionics.

Commercial aircraft are subjected to constant maintenance that is mandated by the Federal Aviation Administration (FAA). The loadable software parts are tracked and logged just as any other aircraft component. Commercial aircraft carry a means of maintaining a spare copy of all LSAP on board if a hardware component fails, is replaced, and needs the LSAP installed. This means that once the LSAP is loaded on the aircraft, it can be reinstalled in the target device locally, provided a maintenance professional is available.

## 2.9 Related Security Issues

### 2.9.1 Secure boot features in relation to secure software update.

Secure boot is a feature being offered by many manufacturers of system-on-chip (SoC) devices, such as Texas Instruments and Freescale Semiconductor. The secure boot feature generally includes the ability for the SoC device to cryptographically authenticate and verify the initial bootloader prior to execution in the normal boot process. Once the initial bootloader has been loaded, verified, and executed, the secure boot hardware feature has no further responsibility for the security and functionality of the system for the remainder of the boot cycle. The bootloader must check the rest of the software components for authenticity prior to execution, forming a chain of trust down to the hardware boot.

#### *2.9.1.1 The chain of trust in software.*

*Chain of trust* is a term used in a secure boot environment, where all software executed must be authenticated by previously authenticated software. In the simplest example, a secure boot device will load and authenticate the bootloader. The bootloader would then load and authenticate a firmware image. The bootloader would only execute the image if authentication was successful.

The above example can be extended to an ECU with a general-purpose operating system such as GNU/Linux, Android, or QNX by simply replacing the firmware image with the initial program loader of the desired system. At that point, the entire basic operating system environment will have been authenticated and can be deemed secure. It would then be the responsibility of the running system to verify applications, libraries, and data being used at runtime.

This chain of trust must be carefully observed during software development, as any component that can load and execute software must also have the capability of verifying that software prior to execution.

## 2.10 Lessons Learned From Adjacent Industry for security software updates

In the previous sections, details were included from both the commercial aviation and medical industries. In this section, the authors summarize the lessons learned while reviewing the state of the art in similar industries to automotive, including consumer electronics.

### 2.10.1 Industry/device attributes.

During investigations into related industries, the researchers found it helpful to not only consider the update process steps but also consider aspects of the industry/device attributes. Table 2

provides a comparison of attributes among the adjacent industries, both current and future, and a description of each attribute follows.

- **Update Methodology**: Refers to the type of firmware-loading mechanism that is used. A low-level methodology uses the microprocessor debug port (e.g., IEEE-ISTO 5001-2003, a.k.a., Nexus). A high-level methodology uses a bootloader that contains additional features (usually for in-service reprogramming) such as serial port driver (e.g., CAN driver), checksum validation, and/or authentication.
- **Update Target**: Data, applications, OS, or all levels of software and data within the target.
- **Communications Channel:** Wired (OBD port, USB), OTA, mixture (OTA to data logger and then physical connection to target), and hard media.
- **User and Access Level**: End user, specialist (doctor or nurse), elevated (IT professional, aircraft mechanic), minimal or none (automatic).
- **Interaction**: Automatic (e.g., PC updates), minimal user interaction (user request or acknowledgement of update), professionally trained technician (aircraft mechanic, IT professional, auto service technician).
- **System Topology**: Many networked devices each of which can be updated (e.g., automobile, aircraft), single networked devices (e.g., smart phone, PC, medical device).

Multicore devices with shared memory or connected via an on-board bus, such as a serial peripheral interface (SPI) or an inter-integrated circuit (I2C) bus are considered single devices.

The commercial aviation industry is like the auto industry in many ways: both automobiles and aircraft have many hardware devices containing loadable firmware components, both industries have a need/desire to track installation of those updates across their fleets, and security of the update process is considered critical to safety. However, in the future, the auto industry would like the User and Access Level and Interaction attributes to move towards those of consumer electronics in that the OEMs want to require no expertise and little interaction from the vehicle owner beyond notifications that are necessary; commercial aviation will still rely on the airlines and their trained maintenance staff.

The important distinction between “Interaction” and “User and Access Level” is that “Interaction” refers to the level of knowledge or skill level required to perform the update whereas “User and Access Level” refers to who is permitted access. For example, updating a device may be as simple as hitting “enter” when prompted, but the same device may first prompt for a password that is only known by the IT professional (Minimal interaction but with Elevated access).

| Attributes | Adjacent Industries | | | | |
|---|---|---|---|---|---|
| | **Current/Future** | **Automotive** | **Commercial Aviation** | **Medical Devices** | **Consumer Electronics** |
| | **Update Methodology** | High | High | High | High |
| | **Update Target** | All | All | All | All |
| | **Communication Channel** | Wired/ **Wireless** | Hard media / **Wired (Electronic Crate) Mixture** | Wired/ **Mixture** | Wireless |
| | **User and Access Level** | Specialist / **End-User, Minimal** | Elevated | Elevated | End-user/None |
| | **Interaction** | Professional/ **Minimal** | Professional – FAA cert. | IT Professional | Automatic / Minimal |
| | **System Topology** | Distributed system | Distributed system | Single micro-controller | Single micro-controller |

**Table 2**. Comparison of Attributes Across Adjacent Industries.

Will it be possible to provide the ease of update for consumer electronics while retaining the security of commercial aviation and medical devices? Tampering with an aircraft or aircraft components is a Federal offense, and the risk of terrorism means aircraft systems are not physically accessible by the public. Medical devices are not readily accessible to the public, and there is little to be gained monetarily from hacking them. In the automotive industry, the Library of Congress has decided that altering computer programs in cars for modification and repair is exempt from the Digital Millennium Copyright Act's (DMCA) provisions on technology circumvention [LOC14]. Aftermarket products for modifying automobile performance through changes to hardware and software within the ECUs are a big business and will continue to be. Traditionally, the auto industry has looked the other way regarding aftermarket software changes, and it has even been said anecdotally that it is a marketing feature of the high-performance cars (auto enthusiasts may purchase the vehicle knowing it can be later modified for greater performance). So, the auto industry faces the challenge of trying to provide owners with a secure and robust software update process without destroying the aftermarket possibilities where possible.

### 2.10.2 PC BIOS updates.

System firmware on modern computers is also known as the system basic input/output system (BIOS). The primary function of the system BIOS is to facilitate the hardware initialization and testing process and to load the operating system. It is initially distributed to the end users by computer hardware and may later be updated by computer manufacturers to fix bugs, patch vulnerabilities, and support new hardware.

Older desktop and laptop computers (pre ~2012) use conventional BIOS firmware often executing in the 16-bit real mode (no memory protection, multitasking, or code privilege levels). Newer computer systems use firmware based on the Unified Extensible Firmware Interface (UEFI) specification running in 32- or 64-bit protected mode (protected virtual addressing and safe multitasking) on the CPU. UEFI firmware is designed to replace conventional BIOS. Compared with conventional BIOS, UEFI firmware has better security, faster startup times and resuming from hibernation, supports larger drives, and additional benefits. Industry computer systems are now shipping with UEFI BIOS.

In both conventional BIOS and UEFI BIOS, the first task after the firmware is running is to execute the core root of trust of the system. This is done through a small core block of firmware which executes first and can verify the integrity of the rest of firmware. This small core block of firmware is usually logically separated from the rest of the BIOS. It is traditionally known as the BIOS Boot Block. After the core root of trust is established, the system BIOS initializes and tests key hardware on the computer system, loads and executes additional firmware modules, selects the Boot Device where the operating system resides, and finally loads and hands the control to the operation system.

Due to the BIOS's unique and privileged position in the PC architecture, unauthorized modification of BIOS firmware constitutes a significant security threat. Malicious code running at the BIOS level can compromise any components that are loaded later in the boot process. Because the BIOS is the first piece of software that runs after the system is powered on, malware at the BIOS level is very difficult to detect as anti-malware products running in the OS have no opportunity to authoritatively scan the BIOS. Malware written into the BIOS can be used to re-infect the computer system even after new operating systems are installed and hard drives are replaced because the BIOS is stored in persistent non-volatile memory. An attacker can also corrupt the BIOS to cause permanent denial-of-service.

The National Institute for Standards and Technology (NIST) provides BIOS protection guidelines in its Special Publication 800-147 [NIST11]. Guidelines in NIST 800-147 are not designed to cover all threats during the system's lifetime. Instead they focus on preventing potential threats due to vulnerabilities in BIOS security controls, BIOS itself, and network-based system management tools, assuming the system arrives with the manufacturer's intended system BIOS installed. The objective is to maintain the integrity of the BIOS by securing the BIOS firmware update mechanism. System BIOS updates should be performed either though an authenticated mechanism or through a secure local update without using the authenticated update mechanism.

***Authenticated Update.*** In the authenticated BIOS update mechanism, each BIOS update image should be digitally signed before being delivered to a computer system. Update only proceeds after the image is successfully verified. The computer system contains a public key for verification and a signature verification algorithm. The public key and signature verification algorithm are part of the Root-of-Trust-for-Update (RTU). NIST recommends that the RTU, including the verification algorithms and keys, be stored in a protected fashion in the computer system, as any unauthorized change to the RTU will compromise security of the system.

***Secure Local Update***. Alternatively, system BIOS updates can be performed through a secure local update without using the authenticated update mechanism. However, the secure local update mechanism should only be used in special conditions such as loading the first BIOS image or recovering a corrupted BIOS that cannot be fixed through the authenticated update mechanism. Additional protections (e.g., requirements of user presence, administrator password, unlocking of a physical lock) may be implemented before the system BIOS is permitted to be updated.

***Integrity Protection***. Both the secure BIOS update mechanisms rely on the integrity protection of the RTU and the system BIOS. There should be a protection mechanism to protect the RTU and the system BIOS from unintended or malicious modifications. The protection mechanism itself should also be protected from unauthorized modification.

***Non-Bypassability***. The design of the system and accompanying system components shall ensure secure BIOS update mechanisms (authenticated update mechanism and secure local update mechanism) cannot be bypassed by any system processor or system component. Even when some system components may have read access to the flash memory, they shall not be able to modify the system BIOS except through the authenticated update mechanism or through the secure local update mechanism with user intervention.

The NIST document also recommends best practices for managing system BIOS, focusing on key activities around provisioning, deploying, managing, and decommissioning of the system BIOS in an enterprise environment.

### 2.10.3 PC operating system and standalone application software update.

In 2006, Bellissimo, Burgess, and Fu published an analysis on the security of automatic update mechanisms for major operating systems and standalone application software [BBF06]. The authors analyze the mechanisms' resistance to man-in-the-middle attacks mainly through answering the following two questions:

a. Is the update distributed through a secure channel?
b. Is the update digitally signed?

The answers the authors found to those two questions for several popular PC software projects, including Windows and MacOS general-purpose operating systems, are show in Table 3 [BBF06].

| Software | Authenticated Connection? | Authenticated Binaries? |
|---|---|---|
| Apple Software Update | No | Yes |
| Windows Update | Partially | Yes |
| Adobe Acrobat | No | Yes |
| Microsoft Office | No | Yes |
| Mozilla Firefox | Partially | No |
| Fugu | No | No |
| McAfee VirusScan | No | No |
| McAfee VirusScan Enterprise | Unknown | Yes |
| McAfee Virex | No | No * |
| Debian | No | Yes |

**Table 3.** Comparison of Major OS and Common Applications [BBF06]. This table is based on the data published in Bellissimo, Burgess, and Fu.

This study shows that operating systems (e.g., Windows, iOS) tend to have better designed update methods than standalone application software (e.g., Mozilla, Fugu) in that

- Updates are usually distributed through trusted content distribution networks/servers;
- Software is digitally signed under well-known public keys.

The paper also points out that firmware update for embedded devices face additional challenges by comparing with PC OS and application software updates. Embedded devices usually have only sporadic network connectivity and limited local resources. Also, there is a lack of trusted infrastructure for embedded devices.

### 2.10.4 Printer firmware update.

[CCS13] presents techniques for exploiting firmware update vulnerabilities in HP LaserJet printers. The authors implement a proof-of-concept printer malware (capable of network reconnaissance, data exfiltration, and propagation to general computers) and conduct a case study with HP-RFU (remote firmware update) LaserJet printer by mounting a firmware modification attack. The attack is based on a fundamental design flaw in the HP-RFU procedure. In HP-RFU, an RFU file is ***printed*** to the target device via the raw-print protocol over standard channels. HP- RFU works as follows:

- When a print job is received, the printer uses a proprietary mechanism to determine the presence of a valid firmware update package;
- If a valid RFU package is present, the integrity of the RFU payload is verified and decompressed; and
- Then the payload's unpacked payload is written to persistent storage.

As can be seen from the RFU process, firmware modification is coupled with the printing subsystem, which **must accept** incoming printing requests **in an unauthenticated manner** as per general specification. This means anyone can issue an update request in the form of a print request containing their own update/attack. This attack could be potentially prevented through firmware update signing such that only RFU files that are digitally signed by the manufacturer can be written to the persistent storage.

In their study, the authors also find that firmware can contain known vulnerabilities found in third-party libraries. Firmware update signing is NOT the panacea of embedded defense because mandatory firmware signature only allows known vulnerable code to be signed and verified. It does not help in removing the actual vulnerabilities, detecting, or mitigating the exploitation of the actual vulnerability.

The authors discuss two defense techniques to prevent or mitigate firmware modification attacks. One is a fortification technique that tries to output a security hardened, functionally equivalent variant of the original. The other is to inject intrusion detection functionality into the binary firmware of existing embedded devices.

### 2.10.5 Smart battery firmware update.

Reference [M11] demonstrates how to hack MacBook battery firmware updates (of a specific battery model). This attack is possible due in part to Apple's use of default passwords for both unsealing the battery and opening full access mode to it.

The authors can also disable the checksum so that they can make changes in updates.

### 2.10.6 Medical devices.

Reference [HRMP11] uses a case study to show that medical devices have firmware update features that are not sufficiently protected by proper user authentication. Vulnerabilities found in an automated external defibrillator (AED) include

- Software module with integer/buffer overflow vulnerability
- Weak password authentication
    - Password stored locally and
    - Password obscured using XOR
- Credentials stored in plaintext
- Improper use of weak CRC as **digital signature**

The paper does not give details about the firmware update process. The Food and Drug Administration (FDA) leaves it up to the device manufacturers to determine how to distribute and install firmware updates.[3]

[3] www.fda.gov/downloads/MedicalDevices/DigitalHealth/UCM544684.pdf

In addition to meeting the FDA's quality system regulations, section 164.306 of the Health Insurance Portability and Accountability Act of 1996 (HIPAA), titled "Security Standards: General Rules," contains requirements for the security of electronic protected health information (PHI) by covered entities and their business associates. The requirements listed in this section are quite flexible, thus permitting a covered entity and business associates to use any security measures that "reasonably and appropriately implement the standards and implementation specifications as specified in this subpart." [45 CFR § 164.306(b)(1).] by design and calls for a covered entity or business associate to ensure the confidentiality, integrity, and availability of all electronic protected health information against reasonable threat. In practice, this permits device manufacturers to decide what works best for their equipment.

Two individuals involved in medical device firmware updates were interviewed and described the medical device firmware update process as detailed below. The devices discussed to date are interesting in that they are representative of the extremes in terms of the types of devices that are considered medical devices, and the very different level of security required.

Simple devices include monitoring devices that do not record or log patient information. In the specific case of the device discussed, it is updated by anyone with the appropriate update contained on a USB stick. The update files for this device are distributed by a website and are freely available to anyone with knowledge of the appropriate link, which is not published and difficult to guess.

Complex devices include sophisticated imaging devices that capture and retain data directly from a patient. These hold private patient information and require authentication from the skilled technician to simply operate. Further, a separate authentication is required to enter a service menu to select a firmware update procedure. The maintenance professional first logs into the manufacturer's website from a PC to download the firmware update package. The serial number of the device to be updated must also be entered before an update package can be downloaded. The update package is then transferred from the PC to hard media for installation in the imaging device. Since the update package is specific to the device by serial number, it cannot be used on any other device even if the password required to enter the service menu of the device is known. This device-specific process provides a measure of security but it also guarantees that the software update has been validated on the hardware specific to the device, in the case where hardware differences may exist between devices that otherwise appear to be identical.

More participants are needed who can provide information about additional medical devices, such as surgical robots and laboratory test equipment. It has been learned through literature reviews that great (perhaps more) effort is also placed on protecting patient records in addition to protecting the integrity of the firmware for devices that hold patient information.

### 2.10.7 Commercial aviation.

Perhaps the most salient comment made during the interviews with commercial aviation professionals was: "Nothing changes without someone touching the aircraft." The team concludes from the interviews that it is not in the foreseeable future that commercial aircraft will receive software/firmware updates directly from the manufacturer via OTA transmission without intervention from an aircraft maintenance professional. Additionally, it continues to be the

responsibility of the airline to ensure their fleet is maintained, including performing software updates as necessary.

### 2.10.8 Summary of lessons learned in adjacent industry.

Common existing defense mechanisms:

- Trusted content distribution network
- Digitally signed software update

Common existing vulnerabilities:

- Software vulnerabilities
    - Software bugs, (e.g., buffer overflow)
    - Known vulnerabilities in underlying third-party libraries
- Weak (user) authentication
    - Weak password
    - Default password
    - Password stored locally (and can be extracted through reverse engineering)
- Weak/improper software authentication
    - E.g., CRC is used as digital signature for HP LaserJet printers.
- System design vulnerability
    - E.g., update is coupled with raw printing service -> (should be secure) update function is requested through low-security printing service.

*Potential defenses for secure vehicle firmware update:*

- Secure software engineering
    - Secure software design
    - Static and dynamic analysis
- Secure cryptographic primitives
- Secure update protocol design
- Secure software attestation

## 2.11 Conclusion

There is no single, perfect reference model for securing software updates. Every application has different requirements, and user experience shapes the design enough to require security to be at a minimum analyzed and usually designed with the application in mind. Even though there is no one-size-fits-all software update architecture, there are several mature industries and products from which the automotive world can learn the commercial aviation field is very like automotive. In Section 2, the authors presented different approaches to software updates, particularly securing them, across similar industries and electronics products in general.

# 3. Risk Assessment

This section provides a risk analysis of software update mechanisms that are currently being used or considered in automobiles and attempts to lay the groundwork for further identifying, analyzing, and mitigating risks identified now and as the technological path unfolds. In this section, different risks are identified and assessed. In the following section, mitigations to address these risks are detailed.

## 3.1 Reference Vehicle Architecture Model

The research team assessed the risks posed by software update mechanisms in current and near-term future automobiles. For that purpose, they defined some reference models of the software update capabilities and vehicle electrical architecture to support those capabilities. In this risk analysis, a high-end, high-connectivity (current model year) vehicle was targeted, which is most likely to contain more software update mechanisms and paths than a lower-end or lower-connectivity (older) vehicle.

Today, Tesla already performs remote updates over an internet connection while the vehicle is in the customer's possession for both fixing issues and licensing newly purchased content [Tesla16] [Mas15]. For that reason and because of the high level of automation in Tesla vehicles as compared to competitors, the Tesla Model S makes a good choice for a reference vehicle. However, there are many data paths for software updates, and different automakers use different combinations of connectivity interfaces. For that reason, the researchers consider the features of the Tesla (as a level 2 automated passenger vehicle). However, for identifying technical risks with software update mechanisms, we consider the whole swath of current or near-term potential transmission mechanisms for software updates such as OBD port, Bluetooth, Wi-Fi, or cellular data connection.

The risk assessment will cover the traditional, OBD-attached software update mechanism, particularly as we relate those methods to newer ways of delivering software and firmware updates to vehicles. Traditional, OBD-attached software updates can have a variety of potential differences from vehicle to vehicle, such as travelling over a different data link layer (e.g., CAN [ISO11898], J1850 [J1850]) or what diagnostic protocol is used (e.g., UDS [ISO14229], KWP2000 [ISO14230]). In addition, emissions related ECUs are mandated to support the SAE J2534 standard PC software tool application programming interface standard [J2534]. Tools implementing J2534 are easy to use and readily available to the public. The authors consider the following software update (transmission) mechanisms:

1. [Wireless internet] Updates over a cellular connection with internet access.
2. [Wireless internet] Updates over a Wi-Fi connection with internet access.
3. [Wireless internet] Updates over an OBD-attached dongle with internet access.
4. [Wireless long-range] Updates over a satellite link.

5. [Short-range wireless internet] Updates through a Bluetooth-attached mobile device.
6. [Short-range wireless] Updates over short-range RF link (standard or proprietary).
7. [Short-range wireless] Updates over an OBD-attached dongle with short-range RF link.
8. [Wired] USB thumb drive updates.
9. [Wired] Traditional updates with an OBD port-attached device.

## 3.2 Baseline Threat Model and Methodology

The purpose of the baseline threat model is to identify the vehicle-level impacts, severity, threat actors, and attack scenarios in modern and future automobiles. The baseline risks and threats can be viewed as the end goals of a would-be attacker (a threat actor). These vehicle-level risks have some motivating appeal to some subset of unauthorized parties, which we attempt to coarsely quantify. In addition, baseline risks have some potential damage to stakeholders, which can also be quantified. Detailed threat modeling at different levels is a whole activity itself. In this risk analysis of software updates, the baseline threat model applies to software updates specifically and is not a comprehensive threat model of the full vehicle, which is beyond the scope of this risk analysis of software update mechanisms. The baseline threat model is presented here to motivate and frame the more technical software updates risk analysis that follows in this section.

In this section, some potential malicious parties (i.e., threat actors) are listed. Threat actors are the groups or individuals with motives to carry out attacks (specifically, cyber-attacks) on automobiles. A threat model identifies the highest-order cybersecurity risks, specifically those that can be described naturally in terms of a realistic potential attacker with some motivation and ability to succeed.

Throughout the remainder this document, we will present the highest-order risks in a few different ways, including discussion of the high-level attack scenarios. We then identify the technical risks, which might lead to a realized attack for any one of the higher-level risks.

### 3.2.1 Vehicle-level risks.

The following risks apply, at a high-level, to the whole vehicle. The items represent bad outcomes that an attacker might want to cause for some gain. Each vehicle-level risk is considered for a single vehicle only. With long-range attack vectors, such as a cellular module with full or filtered internet access, it becomes possible for an adversary to produce one or more of these attacks on several vehicles at the same time and in coordination. The vehicle-level risks for an automobile are

- Intentional vehicular crash.
- Disruption of operation (for example, loss of some or all controls while driving).
- Disruption of service (for example, the inability to use a parked vehicle).
- Coordinated attack (involving one or more of the prior attacks in coordination).
- Vehicle theft.
- Vehicle parts or contents theft.

- Intellectual property theft.
- Private information theft.
- Unauthorized activation of upgrade features (e.g. software piracy).
- Aftermarket performance tuning.

### 3.2.2 Threat actors.

Table 4 lists the threat actors that were identified. These are the entities with some motivation to carry out a cyber-attack on a vehicle. Threat actors are the identified theoretical adversaries and their associated resource access and motivations, which influence their likelihood to carry out a cyber-attack on an automobile. The threat actors and their associated resource access and motivations are listed in Table 4.

| Threat Actor | Resources | Motivation |
|---|---|---|
| Nation states | Well- to very well-funded<br>Backed by military force | Self-defense<br>Control<br>Ideological |
| Terrorist groups | Moderately to well-funded<br>Backed by militia | Control<br>Ideological |
| Organized crime | Moderately to well-funded<br>Backed by violence | Financial<br>Control |
| Activists/ideologues/terrorists or small groups | Minimally funded | Ideological<br>Attention |
| For-profit BlackHat hackers or small groups | Minimally to well-funded | Financial<br>Attention |
| Thieves or small groups | Minimally to moderately funded | Financial |
| Aftermarket tuners (owners or third party) | Minimally to moderately funded | Financial<br>Sport |
| Owners | Minimally funded | Financial<br>Sport |

**Table 4.** Threat actors.

### 3.2.3 Baseline threat model.

Table 5 provides a baseline threat model for a vehicle. Because the threats are not tied to functionality, this threat model is not software update functionality-specific. However, this threat model gives a good reference of the types of threats that apply to a modern or near-term future automobile.

The threat model serves to provide the highest-order risks to the vehicle and the associated threat actors, motivations and severities. Probability of success will be a function of the probabilities of the next level risks that the team will analyze in the following sections. The baseline threat model for the vehicle establishes the potential attacks and high-level risks that are affected by software updates.

Table 5 shows several examples of the possible bad outcomes of an attack on software update mechanisms at the vehicle-level. The sheer power of malware installation means that essentially anything is possible. Almost any software-only attack can be launched via the installation of malware. For that reason, we show the threats and high-level risks to motivate the risk analysis of the malware installation risk.

| Attack Scenarios | | | | | |
|---|---|---|---|---|---|
| **Type of attack** | **Malicious control of vehicle** | **Denial-of-service of vehicle** | **Vehicle or contents theft** | **Intellectual property theft / private information exfiltration** | **Performance tuning or unauthorized feature activation** |
| **Attack description** | The attacker can control the vehicle, either partially or in full. | The attacker prevents the use of the vehicle. This may be performed on many vehicles in a coordinated manner. | The attacker or attackers steal the vehicle or its contents. | The attacker can remotely track the vehicle, its operators and their behavior and other private information. | The attacker is an owner or third party who changes the control firmware to get different performance characteristics. |
| **Threat Actors** | | | | | |
| **Threat Actors** | Nation states<br>Terrorists<br>Organized crime<br>Activists | Nation states<br>Terrorists<br>Organized crime<br>Blackhats<br>Activists | Car thieves<br>Terrorists | Nation states<br>Organized crime<br>Activists<br>Blackhats<br>Owners | Tuners<br>Owners |
| **Resultant Motivation** | Self-defense<br>Control of adversaries<br>Financial<br>Ideological | Self-defense<br>Control of adversaries<br>Financial<br>Ideological | Financial | Control of adversaries<br>Financial | Financial<br>Sport |
| **Attack Potential** | | | | | |
| **Time elapsed** | Months-Years | Months | Weeks-Months | Months | Days-Years |
| **Finances** | High<br>Medium | High<br>Medium | Medium<br>Low | High<br>Medium | Medium |

| | | | | | |
|---|---|---|---|---|---|
| **Expertise** | High | High | Medium<br>Low | Medium | Medium |
| **Knowledge of system** | Private<br>Public | Private<br>Public | Public | Private<br>Public | Private<br>Public |
| **Window of opportunity** | Unlimited | Unlimited | Moderate | Unlimited | Unlimited |
| **Equipment** | Custom | Custom | Custom | Custom | Custom |
| **Motivation of Attacker** | | | | | |
| **Financial gain** | Medium<br>Low | Medium<br>Low | **High** | Low | Medium |
| **Ideology** | **High** | **High** | None | Low | None |
| **Passion** | Medium | Medium | Low | **High**<br>Low | **High** |
| **Likelihood** | Medium | Medium | **High** | Medium | Medium |
| **Loss to Stakeholders** | | | | | |
| **Financial** | Moderate | Moderate | **High** | None | Medium |
| **Privacy** | Low | Low | None | **High** | None |
| **Safety violation** | **Very high**<br>**High** | **Very high**<br>**High** | None | None | Medium |

**Table 5.** Baseline Threat Model Matrix. This table presents the baseline threat model as a matrix with motivation, cost of attack, and other relevant attributes. This is a non-comprehensive threat model of a modern automobile (with potential attacks enabled by software update functionality). Insecure software updates can provide a powerful threat vector for these scenarios: the installation of non-authentic software (i.e., malware).

## 3.3 Attack Scenarios

In this section, the authors present several vehicle attack scenarios and analyze them. The main headings in this section correspond to the five columns in Table 5, the Baseline Threat Model Matrix. The risks themselves are at the vehicle-level, presented with their associated real-world threats and severities (potential impacts). Threat actors who are not motivated to attack are omitted from the attribute tables below. For each scenario, software update mechanisms create new attack vectors for malicious entities. Each scenario is tied to the software update risks that affect them.

### 3.3.1 Malicious control of vehicle.

The most serious safety risk of all the scenarios listed in this report are those that allow an adversary to actively control the vehicle in some way (for example, change the steering wheel angle, engage the brakes, or cause an engine stall). Within this class of threats, the most serious scenario is when an adversary can remotely control many similar vehicles from an arbitrary location. However, there are other, less severe scenarios.

Malicious loss of vehicle operation at speed is extremely like malicious control, because causing a loss of operator control is a form of control. For that reason, this section also applies to attacks that can cause some loss of vehicle controls while the vehicle is in operation. Loss of vehicle operation while not in operation is covered under the next section, Denial-of-service.

#### *3.3.1.1 Remote malicious control of many vehicles.*

The most serious form of malicious vehicle control is control of multiple vehicles from a remote location. Remote malicious control of many vehicles has the potential for widespread impact in the event of a coordinate attack. Unknown and remotely exploitable software bugs (i.e., zero-day vulnerabilities) could allow an attacker to gain a foothold into many vehicles and control critical vehicle functions. Table 6 presents risk attributes for remote malicious control of many vehicles.

| **Attack** | **Remote Malicious Control of Many Vehicles** | | | |
|---|---|---|---|---|
| **Threat Actor** | Nation state | Terrorist | Organized crime | Ideologue/activist |
| **Probability of Success** | High | Medium | Medium | Low |
| **Motivation** | High | High | Medium | Low |
| **Stakeholder Impact** | Very High | | | |

**Table 6.** Remote Malicious Control of Many Vehicles—Risk Attributes. Remote malicious control of many vehicles has the potential for widespread impact in the event of a coordinated attack.

#### *3.3.1.2 Remote malicious control of a small number of vehicles.*

There is a possibility that an attacker can control a single vehicle or a small number of vehicles in an instance. For example, an attacker might be able to influence the owner of a vehicle to perform some action or give up some vital information using social engineering such as a spear-phishing campaign. In this case, due to the dependency on naïve owner action influenced by a malicious actor, the likelihood of a coordinated or widespread attack is low.

Another possible way an attacker might gain a foothold into the vehicle is physical access at a prior time. Due to the necessity of one-time access to the vehicle, such an attack cannot be executed on an arbitrary number of similar vehicles. However, even though such an attack requires physical access to a vehicle, it only requires this access once; afterwards, the attack can be launched remotely (arbitrarily far away). Table 7 lists risk attributes for remote malicious control of small number of vehicles.

| **Attack** | **Remote Malicious Control of Small Number of Vehicles** | | | |
|---|---|---|---|---|
| **Threat Actor** | Nation state | Terrorist | Organized crime | Ideologue/activist |
| **Probability of Success** | High | High | Medium | Low |
| **Motivation** | High | Low | Medium | Medium |
| **Stakeholder Impact** | High | | | |

**Table 7.** Remote Malicious Control of Small Number of Vehicles—Risk Attributes. While the risk is remote control of vehicles as in Table 6, the impact is lower. The motivations for such an attack are lower for terrorists, and this attack is easier than controlling many vehicles in coordination.

#### *3.3.1.3 Near-range malicious control of a small number of vehicles.*

Attacks that enable control of vehicles that might be launched when in range of the vehicle in some way (e.g., within range of a dedicated short-range communication such as DSRC or Bluetooth). This attack may be translated to a remote malicious control of a limited number of vehicles by installing a rogue device with a long-range transmitter with one-time physical access to a vehicle. An attacker might execute a near-range attack without gaining physical access to the vehicle by, for example, exploiting a bug in the Bluetooth communications with an infotainment system or the DSRC interface used for V2X communications. Table 8 shows risk attributes for near-range malicious control of vehicles.

| **Attack** | **Near-Range Malicious Control of a Small Number of Vehicles** | | | |
|---|---|---|---|---|
| **Threat Actor** | Nation state | Terrorist | Organized crime | Ideologue/activist |
| **Probability of Success** | High | Medium | Medium | Low |
| **Motivation** | High | Low | Medium | Medium |
| **Stakeholder Impact** | High | | | |

**Table 8.** Near-Range Malicious Control of Vehicles—Risk Attributes. Near-range control of vehicles is the risk that an attacker will be able to partially or fully control vehicles within some range of an attack point such as a rogue cellular base station. The stakeholder impact is lower than full remote-control due to the limited area of attack.

### 3.3.2 Denial-of-service of vehicle.

Malicious loss of vehicle operation at rest is a form of denial-of-service attack. From the vehicle perspective, this is a DoS of the vehicle itself rather than some feature (such as air conditioning or software update functionality). One growing class of a DoS attack on consumer products is so-called ransomware. CryptoLocker [CERT14] was a very visible example of ransomware, which plagued average PC users, and recently, a hacking group exacted a ransom of 40 bitcoins (worth

over $16,000) from the Hollywood Presbyterian Medical Center, a hospital in Los Angeles, to restore operation of its computer network [Dal16]. While not a safety issue, ransomware can spread like a computer virus, leading to large loss.

#### *3.3.2.1 Targeted, coordinated denial-of-service.*

A powerful attacker might wish to perform a targeted, coordinated DoS on, for example, a fleet or on all vehicles sharing some property such as physical location. A military at war would have great success if it could shut down enemy vehicles. A terrorist might see this as an attractive way to create mayhem. If all emergency vehicles in a large region were disabled, that could lead to a severe situation. Table 9 shows targeted, coordinated DoS.

| **Attack** | **Targeted, Coordinated Denial-of-service** | | | |
|---|---|---|---|---|
| **Threat Actor** | Nation state | Terrorist | Organized crime | Ideologue/activist |
| **Probability of Success** | High | Medium | Medium | Low |
| **Motivation** | High | High | Medium | Low |
| **Stakeholder Impact** | Very High | | | |

**Table 9.** Targeted, Coordinated Denial-of-service. While the DoS on a single, arbitrary vehicle is not a high-impact attack, a coordinated attack disabling many vehicles is. For example, if an attacker were to disable all emergency vehicles in a relatively large or dense area, the impact could be large.

#### *3.3.2.2 Ransomware.*

Ransomware is a growing class of attacks whereby the target is denied use of their property by a hacker or hacking group unless and until a ransom is paid. By nature, ransomware attacks are almost always remotely executed (from an unknown location).

Software update mechanisms are an attractive path to ransomware and other broadly targeted malware installations. While app stores for mobile phones have safeguards, malware is still difficult to completely prevent. As vehicles start to include application store-like functionality, this opens the door further (that is, increases the attack surface) to malware installation like ransomware. The authors assume that ransomware does not normally cause permanent damage and has a low stakeholder impact. Table 10 shows ransomware attack levels.

| **Attack** | **Ransomware** | |
|---|---|---|
| **Threat Actor** | Organized crime | For-profit black hats |
| **Probability of Success** | High | High |
| **Motivation** | High | High |
| **Stakeholder Impact** | Low | |

**Table 10.** Ransomware.

### 3.3.3 Vehicle or contents theft.

Vehicle theft remains a goal for criminals the world over. Although cybersecurity and the connected vehicle do not change that fact, they add additional attack vectors by which a thief or would-be thief can steal a vehicle or its parts or contents. Today, many security features are

software controlled. For that reason, software updates can be a path by which a would-be thief may weaken or defeat the anti-theft features of a modern automobile.

#### *3.3.3.1 Pairing an unauthorized key to a vehicle.*

In modern (and near-term future) vehicles, mating new keys with existing vehicles in the field is generally done almost purely with software. For this reason, software update mechanisms can be an attractive attack path for pairing unauthorized keys with a vehicle targeted for theft (or content theft). Table 11 shows risk attribute paring of unauthorized key to a vehicle.

| **Attack** | **Pairing an Unauthorized Key to a Vehicle** | |
|---|---|---|
| **Threat Actor** | Organized crime | Thief |
| **Probability of Success** | Medium | Medium |
| **Motivation** | Medium | Extreme |
| **Stakeholder Impact** | Medium | |

**Table 11.** Paring an Unauthorized Key to a Vehicle—Risk Attributes. Modern vehicles are equipped with immobilizers to prevent the vehicle from driving away without an authorized key present. However, pairing of new keys and vehicles is largely done in software, and software update mechanisms can provide a path to pairing a new, unauthorized key for vehicle or contents theft.

### 3.3.4 Intellectual property theft / private information exfiltration.

Eavesdropping can allow any sort of information or data theft, including intellectual property or private information exfiltration. While the installation of malware can allow data theft, OTA software update mechanisms can create a larger attack surface for theft of a customer's personal information beyond just malware-based attacks. Because metadata is often an integral part of software updates, not only does the addition of OTA software update functionality broaden the attack surface, it can be a driver for adding streaming analytics data to a vehicle. If not carefully designed, streaming analytics and metadata intended to support software updates might inadvertently reveal private information about the vehicle operators and occupants. In addition, if this information is not properly protected, an unauthorized party may exfiltrate metadata intended to support software updates (or any other analytics data).

#### *3.3.4.1 Eavesdropping.*

In an eavesdropping (e.g., man-in-the-middle or man-on-the-side) attack, an interested unauthorized party can capture firmware images, which are sent to a legitimate vehicle ECU as part of the software update. If those images are not encrypted, that party or another downstream party might be able to steal the intellectual property either through reverse engineering or simple counterfeiting of parts. intellectual property theft can hurt legitimate companies' bottom lines. For this reason, the potential impact is assessed to be high due to the widespread stakeholder loss possible. Table 12 shows ease dropping (intellectual property theft) risk attributes.

| Attack | Eavesdropping (intellectual property Theft) | | |
|---|---|---|---|
| **Threat Actor** | Organized crime | For-profit black hats | Aftermarket tuners |
| **Probability of Success** | High | High | Medium |
| **Motivation** | Low | Medium | High |
| **Stakeholder Impact** | Low | | |

**Table 12.** Eavesdropping—Risk Attributes. The purpose of passive listening (eavesdropping) is for stealing secrets (either private owner/occupant information like location information or credit card information or intellectual property). Although the stakeholder impact is low, the loss due to IP theft can be quite large even for a single incident.

#### *3.3.4.2 Activity logger software installation.*

Surreptitious tracking falls into two categories: targeted (unlikely to affect a large group) or untargeted (advertising, for example). Installation of key loggers is a common form of surreptitious tracking in the PC realm. An activity logger is similar to a key logger in traditional PC security. Vehicles arguably convey even more personal information about the user and, at least, more information about the user's whereabouts. Nation states, organized crime, and even individuals are motivated to secretly track others. Software update mechanisms can be exploited to install tracking malware onto ECUs. Even relatively benign ECUs (such as button readers or chime triggers) can listen in on the CAN bus for tracking relevant data. Table 13 shows activity logger software installation risk attributes.

| Attack | Activity Logger Software Installation | | |
|---|---|---|---|
| **Threat Actor** | Nation state | Organized crime | Ideologue/activist |
| **Probability of Success** | High | Medium | Low |
| **Motivation** | High | High | Medium |
| **Stakeholder Impact** | Low | | |

**Table 13.** Activity Logger Software Installation—Risk Attributes. An activity logger is like a key logger in traditional PC security. Nation states, organized crime, and even individuals are motivated to secretly track others.

### 3.3.5 Performance tuning or unauthorized feature activation.

Unauthorized performance or feature modifications can range from powertrain performance modifications, a.k.a., tuning, to software modifications to customize alerts (e.g., to disable a seatbelt chime) to software modifications to unlock digitally licensed content without proper authorization. The impact of unauthorized performance tuning is moderate due to the possibility of expensive warranty claims because of improper or overly aggressive tuning of parts.

#### *3.3.5.1 Performance tuning.*

Third-party aftermarket tuners (i.e., tuning shops) as well as individuals who perform the tuning modifications on their own vehicles, purchase off-the-shelf tuning performance tuning software for modifying the performance envelope of vehicles and their components. Tuning software is

often community supported and may even be produced by individual or mass tuners themselves. Automakers might incur loss from warranty claims due to improper or out-of-specification performance tuning.

Because better performing cars are good for brand building, automakers sometimes turn a blind eye to performance tuning. In the past, improperly secured software update functionality is one of the ways that aftermarket tuners and owners have could install custom software on ECUs and/or perform custom calibrations. When software updates are properly secured, however, automakers will either need to bless performance tuners with ways to update and customize vehicles and their ECUs or lock them out entirely.

| **Attack** | **Performance Tuning** | |
|---|---|---|
| **Threat Actor** | Aftermarket tuners | Owners |
| **Probability of Success** | Medium | Medium |
| **Motivation** | Very High | Low |
| **Stakeholder Impact** | Low | |

**Table 14.** Performance Tuning—Risk Attributes. Performance tuning is the act of creating unauthorized modifications (often in software) to increase the performance of stock parts (such as engines). The property loss in aggregate is still rather small, but some automakers seek to limit aftermarket performance tuning, both to sell more performance upgrades and prevent warranty claims from out-of-spec tuning.

#### *3.3.5.2 Unauthorized feature or content activation.*

Today, the performance profile of a vehicle or a system within the vehicle, such as the engine, may be enabled by nothing more than a different software build or configuration. Economically, automakers can save money in this case by only designing a single hardware variant and changing its behavior with software alone. By configuring the performance of the vehicle with software, OEMs gain the manufacturing economy of scale and engineering efficiency of a single hardware variant while still retaining an enforcement mechanism so that customers can be charged for purchased upgrades. Some owners will want to enable these software-configured upgrades without paying for them. Just like performance tuning, there are vibrant communities on the internet dedicated to unlocking features or content that are software controlled.

With the rise of digital feature distribution, even performance upgrades can be downloaded to a vehicle. Tesla, for example, uses OTA software updates to distribute digital authorization for their “Ludicrous Mode,” a performance upgrade that adds increased acceleration to the Model S [Mat15]. In the future, digital distribution of features, and even digital content such as music, will grow much larger. Automakers will likely want to secure the activation mechanism to protect their revenue stream.

| Attack | Unauthorized Feature or Content Activation | |
|---|---|---|
| **Threat Actor** | Aftermarket tuners | Owners |
| **Probability of Success** | Medium | Medium |
| **Motivation** | Low | High |
| **Stakeholder Impact** | Low | |

**Table 15.** Unauthorized Feature or Content Activation. With the rise of the internet as the universal distribution platform, even automobiles will be able to receive upgrades to performance and convenience functionality through OTA software updates. Owners will attempt to unlock new features or content without paying.

### 3.3.6 Summary.

This section details different attack scenarios, matching them up with probable threat actors' and loose values' probability of success, motivation, and stakeholder impact. In the following section, we describe the technical risks that can lead to these attack scenarios. The installation of malware on the right ECU can allow for all the attack scenarios described in this section. For that reason, much of the following section is devoted to risks that can lead to malware installation.

## 3.4 Technical Risks

In this section, the authors look at the possible technical risks that are created or broadened with the addition of software update functionality. Whereas Section 3.3 focuses on the possible attacks from the vehicle perspective, this section focuses on the technical risks that manifest at a lower level of the design, risks that might lead to those outcomes described in the previous section. Table 16 shows the risks identified, and the following subsections provide further detail. In the table, each risk is listed with an assessment of whether it allows for viral malware distribution. The Packaging stage risks provide the most opportunities for an attacker to create malware which is distributed and installed in a fast moving, viral fashion.

| | Risk | Allows Viral Malware Distribution? |
|---|---|---|
| **Packaging** | Real Authority Signs Unauthorized Software | Yes |
| | Supplier Compromise | Yes |
| | Signing Credentials Are Stolen | Yes |
| | Attacker Forges Signature on Inauthentic Software | Yes |
| | Attacker Remotely Exploits a Software Flaw | Yes |
| **Transport** | Physical Media Tampering in Transit | No |
| | Software Installation Tools Are Compromised | No |
| | Attacker Sneaks Hidden Functionality into Application Store | No |
| | Attacker Masquerades as Legitimate Wireless Endpoint | No |
| | Attacker Masquerades as Legitimate Server in the Distribution Network | Yes |
| **Reception** | Attacker with Physical Access Installs Software | No |
| | Attacker Installs Software with Hacked OBD Dongle | Yes |
| | User Overrides Security Feature | No |
| **Install** | Attacker Uses Undocumented Bypass Functionality | Yes |
| **Verification & Maintenance** | Attacker Spoofs Legitimate ECU and Reports False Installation Information | No |
| | Forged Software Bypasses System Verification Routines | No |

**Table 16.** Risk. This table shows the risks we have identified and maps them onto the 5 software update stages identified in Section 2.3.

### 3.4.1 Rogue software (malware) installation.

The execution of rogue software—that is, malware—is one of the biggest risks to all software systems, and the connection to software update mechanisms is obvious. While attackers certainly find ways to gain full control of processors, they must work with what vulnerabilities are available. Attackers will abuse software already found in a vulnerable device when possible to make system compromise easier. For example, if an attacker can convince the software update mechanism installed on an ECU to install software of the attacker's choosing, this provides a convenient path to abuse (and potentially to remotely abuse with OTA updates). If an attacker can get malware installed on a customer's vehicle, almost any feature that is software controlled can be potentially compromised, obviously depending on where the software is installed. The bulk of this section is devoted to risks that can lead to malware installation.

The following subsections list common ways that an attacker might install malware (a.k.a., rogue software) into ECUs. The researchers make very few assumptions about the security controls in a potential software update system.

#### *3.4.1.1 Real authority signs unauthorized software.*

There are several ways that a real authority might sign malware. If an attacker can find a way to get rogue software signed by the true signing authority, the most basic security control, the software signature, is defeated. In addition, the distribution authority might distribute the software as well if the software is not otherwise detected. The flaws that can lead to this attack are mainly in the IT infrastructure, for which industry standard security controls are more mature as compared to the electronics systems themselves.

Ways that the real authority might sign malware include the following:

- A flaw in the OEM database upload interface allows an unauthorized person to upload software by circumventing the credentials check.
- A flaw in the OEM database allows an attacker to modify or replace authentic software once it has been accepted into the OEM database, but which will not be detected afterwards.
- Any other software flaw known to allow compromise on traditional computers and servers. Research on traditional computer security is quite mature, and detailing all possible paths at the low-level is beyond the reasonable scope of this risk analysis.
- Supplier compromise (see below).

#### *3.4.1.2 Supplier compromise.*

We split supplier compromise into its own section due to its size, but in these scenarios, the real authority signs the malware just as in section 3.4.1.1. Compromise is possible at all levels in the supply chain, not merely the tier-1 supplier. We focus on suppliers generically, but the risks tend to cluster around the interfaces between different parties. Would-be attackers may target developers, especially those with credentials to submit software releases into the official stream of software updates. The following are ways an attacker might compromise a supplier and get non-authentic software signed:

- A rogue software supplier employee uploads bad software to the OEM database via valid credentials.
- An unauthorized person uploads software to the OEM database via valid, stolen credentials.
- An attacker launches a social engineering attack and convinces a legitimate software release developer to upload malware to the software repository.
- A computer virus or PC malware makes changes to the actual software, which gets committed to the software supplier's software repository.

#### *3.4.1.3 Signing credentials are stolen.*

The keys used to authorize software can be stolen. Keys are often hierarchical, with parent keys having the ability to revoke their children. Signing keys, at least at the highest levels, are often protected via a hardware security module (HSM). An HSM is a high-security datacenter device designed to hold very sensitive information, especially cryptographic information like keys. The term *hardware security module* is used in both server and embedded contexts to mean conceptually similar but practically very different devices. In the context of storing signing credentials, we are referring to the traditional, datacenter type of HSM. Flat key architectures, where no key is parent to or child of any other key, are also possible, but more difficult to quickly recover from a key breach. In the worst case, the compromise of keys could lead to a recall to fix.

#### *3.4.1.4 Attacker forges signature on inauthentic software.*

Credential (e.g., certificate, key) forgery, like theft, is particularly relevant for software updates, as digital signatures for authentic software are one of the strongest and most typical security controls for ensuring only authentic software is uploaded to the target hardware. If a digital signature check is part of the software update design, then defeating that check is a necessary building block in most possible attacks.

Signature credential forgery or theft relates specifically to the credentials used to authenticate the software image to the target. In many designs, this is a certificate, but in embedded systems, might simply be an RSA public key / private key pair. If an adversary can sign inauthentic software (i.e., malware) using forged or stolen credentials, the adversary will not need to defeat the digital signature check on the software update image using another mechanism. Credentials could be stolen from an automaker's data center, for example. Credentials can be forged due to a flaw in the way the credentials were created. An attacker's possession of credentials, which allow them to create a digital signature on arbitrary files, which will look authentic to the update target, without any modification of that target, is the common denominator between credential forgery and theft.

The use of digital signatures is a mitigation against basic naïve spoofing and impersonation. Because using end-to-end digital signatures on software update files is near universally considered necessary today, this baseline system architecture assumes that any reasonable software update architecture will include the use of end-to-end digital signatures. Therefore, the authors do not consider naïve spoofing and impersonation to be a reasonable risk. Rather, the risks are that vulnerabilities will exist allowing an attacker to achieve the same goal, albeit with a

more difficult path to finding key vulnerabilities. Credential forgery or theft is one way an attacker could achieve the goal of malware installation without circumventing the digital signature check on the target itself.

#### *3.4.1.5 Attacker remotely exploits a software flaw.*

Software bugs are not fully avoidable, and some software flaws are security vulnerabilities. Some software flaws that allow or assist an attacker in installing malware include the following:

- The signature check is performed, but the memory can be changed between checking the signature and checking that it matches the software download (or vice versa). The attacker changes the signature in between.
- The attacker can replace the public key (or certificate) on the ECU to an arbitrary public key (or certificate). Afterwards, an attacker can sign a firmware update, which, if downloaded, will look authentic to the ECU.
- The attacker can change memory to make the software think that it has performed all security validations on software before it has.
- The attacker can change the memory to make the software think that security validations have passed when they failed.

#### *3.4.1.6 Attacker uses undocumented bypass functionality.*

Backdoors are undocumented functionality that can bypass software update security checks. Backdoors can be left in or added by suppliers (either authorized or unauthorized) or governments or powerful criminal agents. Backdoors could be added intentionally (by the original supplier) or unintentionally (by a rogue agent either inside the original supplier or by breaching the security of the original supplier). Backdoors could also be added after the software has been release from the software supplier. However, that is an instance of exploiting a flaw in or infiltrating the software distribution network itself, rather than an instance of using undocumented bypass capability. Examples of undocumented bypass functionality are the following:

- Supplier has left a backdoor for software updates.
- Government or powerful criminal group has infiltrated the supply chain and added a backdoor to the software.

#### *3.4.1.7 Attacker sneaks hidden functionality into app store.*

Mobile phones have had application (app) stores for several years now, and the technology heart of an app store is a software distribution and update framework. As vehicles move to OTA, go-anywhere software updates, app store models might develop for automotive computing (as well as software for mobile phones which pair with vehicle services). App stores have safeguards, but they aren't perfect. The app store or aftermarket features delivery network represents a new attack vector for malware. Research into this type of app store security is relatively mature but still active.

#### *3.4.1.8 Attacker with physical access installs malware.*

Although physical access is a high barrier for installing malware and preventing local installation methods from being part of viral, coordinated, or widespread attacks, there are plenty of targeted attack scenarios that motivate installation of malware on just one or a small number of vehicles. Traditional physical software updates using OBD port-attached programming tools can be exploited to install malware with local access.

Current physical software update mechanisms (i.e., traditional mechanisms using OBD port-attached devices) do not necessarily include any strong security controls (such as a digital signature). Seed/key algorithms are used to provide rudimentary security before diagnostics and reprogramming routines are allowed. Seed/key algorithms might be cryptographically strong, but current solutions are weak because secrets are widely shared (e.g., in service tools accessible by almost anyone without too much effort). There are several ways in which an attacker with physical access may reprogram software in modules:

- The attacker programs malware into ECUs using official or otherwise standard diagnostic tools for vehicle service and/or manufacturing.
- The attacker steals the secret information needed to perform authentic seed/key exchanges and then reprograms ECUs with a standard reprogramming protocol (e.g., the UDS programming flow) using commodity bus interface hardware.

#### *3.4.1.9 Attacker installs malware with hacked OBD dongle.*

OBD dongles are sold targeting non-technical vehicle owners adding health and usage tracking features in a simple installation (plug in to the OBD port). OBD dongles are not necessarily validated to the same standard as automotive ECUs, and because traditional ECU programming procedures run over the OBD port, any device with OBD access can use those procedures. Therefore, an attacker may hack into OBD dongles to facilitate installation of malware on the vehicle. Dongles with internet access may be compromised from anywhere in the world. OBD-attached devices with large sales may be attacked at the same time leading to a coordinated attack.

#### *3.4.1.10 Physical media tampering in transit.*

An attacker can capture and tamper with physical media for software updates distributed in that way. In traditional software updates over the OBD port, media is distributed to some manufacturing and service facilities on physical media. If an attacker can switch out the software stored on the physical media, that route can allow for malware distribution.

#### *3.4.1.11 Software installation tools are compromised.*

In the traditional software update model, a trained technician installs software updates on vehicles using a discrete or PC-based software tool. The software update tool itself is also a target for attackers. If the software installation tools are compromised, this may allow an attacker to install malware onto ECUs.

#### *3.4.1.12 Forged software bypasses system verification routines.*

In the model of the software update process described here, the fifth and optional stage is Verification and Maintenance. In this stage, the systems in the vehicle and network perform various logging and ongoing verification checks. In some ECUs, the software installed might be checked periodically or at each boot. While this isn't technically part of the software update functionality itself, it is closely related. If a system performs ongoing verification, an attacker might create malware that circumvents those checks.

### 3.4.2 Denial-of-service.

Denial-of-service attacks are a broad class of attacks whereby an adversary prevents the intended operation of a system or subsystem. DoS attacks can even prevent the vehicle from operating. This section only contains risks beyond malware installation. Malware can be used to carry out a DoS attack (ransomware is one such example). However, since the malware installation risks have already been covered, the risks in this section do not require malware to be installed on any part of the vehicle's electrical system.

The scenarios listed below all assume that the DoS purpose is to prevent software updates from working. An attacker might prevent an ECU with a known security flaw from performing a software update to fix it. Or, an owner might prevent software updates on his or her own vehicle. If software update mechanisms are not designed properly, an attacker might be able to brick ECUs (bricking happens when recovery from a failed software update is impossible, and the hardware must be scrapped and replaced). A bricked (non-field-repairable) ECU will cause degraded or even full loss of vehicle functionality until serviced at a dealership or auto repair shop.

The most important scenario to mention is a DoS of the vehicle itself. If OTA software updates are not designed correctly, an attacker could prevent operation of the vehicle for an indefinite amount of time. Vehicle DoS is a useful targeted attack (e.g., an organized criminal group that attempts to prevent an enemy from using his or her vehicle), but if a single vehicle can be rendered inoperable with a denial-of-service attack, it should also be possible to render a large swath of vehicles inoperable in coordination. In general, if ECUs are not bricked or duped into installing malware, a DoS attack creates a temporary condition.

#### *3.4.2.1 Attacker masquerades as a legitimate server in distribution network.*

With any OTA update mechanism, new software binaries must be downloaded onto the vehicle ECUs from a remote device. With internet-based OTA updates (as are the most common), the remote device is a whole software distribution network (e.g., a collection of load-balanced edge servers for the software distribution network). An attacker with the ability to inject packets into the network used to deliver software (or spoof that network with reasonable range) can attempt to masquerade as a legitimate server in the delivery network, causing protocol confusion. Recovering from protocol confusion (for software updates or otherwise) usually entails starting over at some agreed upon good point in the sequence (like the beginning).

An attacker who can arbitrarily (or with limited constraints) inject spoofed messages into the communication between the vehicle and delivery network can trigger repeated protocol failures and restarts, a DoS of the software update mechanism. If the vehicle or some of its ECUs are in an intermediate and temporarily inoperable state when the software update protocol is disrupted

(e.g., if their memory is erased but not programmed), the DoS can be extended to the vehicle itself. A good software update design should include mitigations against this scenario. If the end-to-end validation of software updates is not sufficiently robust, an attacker could cause ECUs to become bricked creating a permanent DoS on said ECUs and potentially the vehicle itself. The vehicle must be serviced at a dealership or repair shop to recover from a persistent inoperable state, such as when an ECU is bricked.

An attacker might spoof an OTA software distribution network such that:

- The attacker injects a bad message in the middle of a legitimate server/vehicle communication, preventing successful software update even after an update has started.
- The attacker interrupts communication during the OTA software update process causing the download to not finish.
- The attacker provides incorrect metadata to the vehicle and/or its ECUs so that the ECU is tricked into performing an erroneous software update action. Without robust validation, manipulated metadata can cause ECUs to perform illogical or improper software update activities.

#### *3.4.2.2 Attacker spoofs a legitimate ECU and tampers with software updates.*

To engineers, it often seems obvious that a server requires credentials to authenticate to its clients. For most websites, the server presents a credential but the user does not, providing one directional verification. However, software updating is a two-way communication between the clients (ECUs) and servers, with vital metadata going in both directions. Even if an attacker cannot spoof the delivery network servers, he or she may disrupt software updates by spoofing traffic coming from the target ECUs themselves.

- An attacker with access to the software distribution network may spoof legitimate ECUs, reporting incorrect metadata to tamper with and/or confound the software update mechanism.
- An attacker with access to a vehicle's internal networks (e.g., CAN networks) may spoof a legitimate ECU in that vehicle to the same end.
- An attacker with access to a vehicle's internal networks may spoof legitimate ECUs in other vehicles, and if the software distribution network servers are not performing robust validation, they may regard this communication as authentic.

#### *3.4.2.3 Attacker spoofs legitimate wireless interface access point.*

An attacker with near range access to a vehicle can spoof a legitimate wireless network access point, such as a Wi-Fi access point or cellular tower. If other controls are not in place, an attacker, spoofing a legitimate network that the vehicle's wireless interfaces expect, can masquerade as the delivery network servers and upload rogue software.

#### *3.4.2.4 Jamming of wireless interfaces.*

Radio frequency (RF) jamming is inelegant, but for that reason, it is difficult to prevent. Jamming is not usually a high-value attack, both because it lacks precision and because it

requires physical locality. Nonetheless, RF jamming will deny service to OTA software update functionality.

### 3.4.3 Unauthorized download of information.

Because data from the vehicle and its ECUs is likely to be gathered in support of software update functionality, it is possible that some data is private information, that is, sensitive or private details about the operator or occupant. Such data ought to be protected against theft or leak to unauthorized parties, and, in some countries, regulations may mandate certain steps be taken. In addition to theft of occupants' information, the software binaries themselves might be deemed sensitive (e.g., to protect against reverse engineering by a competitor or counterfeiting).

While suppliers might view software binaries as containing descriptions of trade secrets and proprietary algorithms, attackers might also use software reverse engineering to look for exploitable flaws. This compromise is not nearly as damaging, but preventing software reverse engineering can increase the difficulty of a successful attack. Further analysis for the unauthorized download of information risks is found in Section 5.

#### *3.4.3.1 Attacker masquerades as legitimate ECU to download data.*

An attacker might masquerade as a legitimate ECU, downloading authentic software from the distribution server/network to steal intellectual property for reverse engineering or counterfeiting. This reference design assumes the ECUs have a way to verify authentic software signed by an authentic server, but does not assume that the ECU has any strong way to identify itself. An attacker can potentially masquerade as a legitimate ECU to download firmware (for reverse engineering or counterfeiting). This can be done remotely if the software update distribution network is not protected against injection by outside parties. Because intellectual property theft can be a one-time thing and still have a big impact, there is risk that an attacker connects to a legitimate vehicle's CAN networks and masquerades as a legitimate ECU for downloading software binaries. Suppliers might design mitigations on top of those used by automakers to protect their intellectual property.

#### *3.4.3.2 Attacker abuses data gathering functionality.*

Software updates rely on accurate metadata. However, metadata is necessarily secured to the same standard as the actual software binaries. If metadata is gathered from the vehicle in support of software updates (or any other reason), it's possible that the information is private to the occupants and should be protected from unauthorized access. If the metadata containing private information from vehicles is not properly protected, an attacker can access that data. Here are some ways this might happen:

- The attacker can query vehicle metadata without providing any sort of credentials.
- The attacker can query vehicle metadata with credentials, and the attacker bypasses the check or forges the credentials.
- The attacker infiltrates the delivery network and only minimal checking is performed (e.g., IP address) on the delivery network by the ECU that is providing metadata.

#### *3.4.3.3 Man-in-the-middle / man-on-the-side.*

In a man-in-the-middle or man-on-the-side attack, the attacker acts as a go-between between some backend entity and the vehicle and its ECUs or merely finds a place in the network topology to passively spy on the exchange. This is usually the result of a compromise of the delivery network, but data might flow through untrusted paths as well.

#### *3.4.3.4 Digital rights management circumvention.*

In a naïve implementation of software updates used to enable new content and features, it might be possible to circumvent digital rights management (DRM) for add-on and purchased content by capturing a legitimate authorization. This is a replay attack, as a legitimate buyer of an add-one feature or content captures the traffic used to authorize his vehicle. The key messages sent from the network (backend) would be authentic and signed, and without additional controls, would enable the same features or content for other vehicles. This user could then give out or sell the content at a lower price to others.

## 3.5 Risk Assessment Discussion

### 3.5.1 Code signing.

Code signing is the act of appending digital signature metadata to software binaries as one of the most basic and fundamental end-to-end security controls. Signatures on the software binary (a.k.a., code signing) are widely considered an essential security mechanism, especially for OTA methods.

### 3.5.2 Automatic updates.

Automatic updates can be preferable to updates that require explicit user confirmation because security updates can be pushed seamlessly without nagging the user. However, with automatic updates, if malware were to make its way into the official software updates channel, it could spread very quickly. Coordinated attacks have higher severities than similar attacks on just a few vehicles or a single vehicle. Automatic updates, if exploited to allow viral distribution, could make it easier for an attacker to launch a coordinated attack on many vehicles. Requiring user interaction before software updates are applied would mitigate this problem. If malware makes its way into the official software updates channel, a significant number of vehicles will be infected. In any case, rigorous engineering checks should be performed before any binaries are distributed through the production software updates distribution network.

### 3.5.3 Robustness against denial-of-service attacks.

DoS attacks are, as a class, very difficult to prevent completely. In OTA software updates, for example, RF jamming is unlikely to be defeated as the physical phenomenon that is used for data transmission (an RF band) is heavily degraded. However, both DoS of vehicles and bricking should be prevented at all costs, even if prevention of all DoS attacks on the software update mechanisms is not possible. Single-vehicle DoS is only a minor stakeholder loss in most cases, but coordinated vehicle DoS could be quite severe.

### 3.5.4 Full software updates vs configuration tweaks.

Software updates in vehicles can range from simple configuration tweaks of just a few bits or a full software update of all memory in the ECU. Larger software updates have additional logistical difficulties. Increased download and installation time for full software updates increase the time the vehicle is out of operation. Today, vehicles in service are frequently connected to a charging device during software updates to prevent the battery from dying in the middle. Today, we do not know how these issues will be handled if full software updates on critical control systems are to be performed over the air while the vehicle is in the customer's possession. We do not know of any gasoline-powered vehicles which receive full software updates over the air.

## 3.6 Conclusion

In identifying risks at both the vehicle level and technological design and implementation level, the researchers have identified the biggest risk with software update mechanisms as malware installation, which makes sense. The team teased apart the different paths to compromise and analyzed the potential attacks (risks) at the vehicle level in Sections 3.2 and 3.3 and the technical-level in Section 3.4 While risks from a relatively high level were identified, software updates come with a large variety of risks from security threats. Yet, software updates are essential functionality for automobile electronics, like any other pervasively networked computer system. In the following section, the authors discuss technical (and non-technical) mitigations for the risks identified.

# 4. Mitigation Methods

ECUs, and the ability to update the software they contain, have been in use for decades in vehicle-control applications. Traditionally, motor vehicle software updates have been the domain of auto dealerships, service centers, and home mechanics. With aftermarket programming tools, with minimal or no authentication required, the introduction of wireless communication within vehicles brings the potential to distribute software directly through the internet without attaching a programming tool to the vehicle CAN bus. The advantages to vehicle manufacturers are reduced warranty costs, improved customer satisfaction, and the ability to offer customers improved features and content.

The importance of software in computer system architecture makes it an attractive target for attackers. At hacking conferences and in academic publications, software modification attacks have been demonstrated repeatedly on various embedded systems including automotive systems. [Che11] [MV15] [RM15] [Fos15] The capability of OTA updates for vehicle software only widens the attack vector, making it possible for hackers to distribute malware to millions of vehicles simultaneously.

While the threats with respect to an OTA update procedure with cybersecurity vulnerabilities are daunting, there is a need to understand software update techniques, the potential threats, as well as potential countermeasures. This project studies the cybersecurity of automotive software updates. The objectives of this project are to define terms commonly used in this domain and identify interesting attributes, survey available firmware update mechanisms used in the automotive industry and across other industries, perform a literature review that also covers all industries, assess cybersecurity threats due to software update methods and practices, and study and propose mitigation mechanisms.

In the previous sections, the authors presented different approaches to software updates across similar industries and electronics in general and a risk identification and analysis of software update mechanisms that are currently being used or planned in automobiles, considering the features of a Tesla Model S for a near-term future design reference. In this section, the team will present mitigations, either required or optional for securing software update mechanisms, both traditional and OTA, for the risks identified in the previous section.

## 4.1 Definitions

In this section, terms are defined that are used throughout this section, are context specific, and may or may not have a meaning outside this document. Although this project considers security of firmware updates, the writers prefer the more generic term software. The risks and mitigations apply the same to software in general as they do to firmware particularly. Where possible, the writers attempt to use these terms in a way that does not conflict with other uses outside this document or cause confusion with similar terms, but it is possible that they do so inadvertently. To put forth a meaning for these terms and to resolve confusion or conflict with existing

terminology in use, the following terms are defined. A full glossary is found at the end of this document.

- The **target ECU** is the electronic control unit (ECU) that is to receive a software update.
- A **software update package** is a file or bundle of files representing the data required to perform a software update on the target ECU. It is a binary representation of a version of software, possibly a diff update created by programmatically comparing (or diffing) the new software version and the version installed in the target ECU for communications efficiency. A distinction between diff and full software update packages is not made.
- A **secondary bootloader** is a file or bundle of files, which is usually downloaded to the target ECU prior to performing the software update process. The secondary bootloader might contain arbitrary code for the target ECU to run. The secondary bootloader often contains routines to interact with (i.e., read, write and erase) the flash memory on the target ECU. The authors do not make a distinction between the secondary bootloader and the software update package in this document. The same mitigations must be applied to the secondary bootloader and to software update packages.
- The **software distribution network** is the network used to support software updates, including OTA and traditional local, OBD port-attached updates. For OTA updates, there is network infrastructure, which is to be built out to support connecting to the vehicle and performing the remote update procedure. For local updates, for completeness, the authors consider the software distribution network to include the full path from the software repository to the vehicle installation, including the portal to request and receive software update packages electronically, media transportation and the mechanics, PC tools, and vehicle-interface hardware. For most mitigations, we do not deeply consider the architecture of the software distribution network, considering it to be a heterogeneous collection of servers, some possibly hosted and/or managed by a trusted third party. Other network functions supporting software updates might or might not be part of the software distribution network. In this document, when two other network functions are referred to, the authors assume an OEM's IT infrastructure will include
    - The **software repository** is the IT infrastructure, which stores, receives, and transports the software update packages themselves.
    - The **bookkeeping database** is the IT infrastructure, which stores the vast configuration and status data for vehicles in the field. It may be part of the software distribution network or separate in practice.

## 4.2 Mitigations

In this section, the researchers reiterate the risks identified previously and map them to the mitigations identified. Mitigations that are mandatory or optional are not specifically called out. An OEM, when implementing software updates, can use a risk-driven approach based on their design, including functional and security goals, to determine which mitigations are not necessary, if any.

Much of this section is devoted to the high-level risk of malware installation. In addition to the high-level risk of malware installation, the section finishes with the two lower-level technical

risks for software updates: DoS of the software update mechanism and unauthorized download (either from the vehicle or the network servers). Further detail on unauthorized download of information is found in Section 5, including intellectual property theft and private information exfiltration.

The authors draw mitigations from internal expertise in systems security and automotive electrical architecture as well as both academic, industrial, and government publications. We examine the failings and vulnerabilities found in the attack publications of Checkoway, et al., [Che11], Foster, et. Al. [Fos15], Miller and Valasek [MV15] and Rogers and Mahaffey [RM15], and consider their recommendations for fixes and mitigations. In addition, the authors draw upon successes and lessons learned from existing industries, particularly mobile and aviation. Finally, many of the mitigations follow existing guidelines, particularly from the United States National Institute of Standards and Technology (NIST). Where NIST guidelines do not exactly match the needs of automotive software but are similar (for example, the NIST BIOS Protection Guidelines [NIST11]), we merely follow their lead and take cues for the mitigations proposed.

### 4.2.1 Malware installation.

Malware installation is the risk with the highest impact, especially in modern vehicles, which have not been internally well partitioned for security. [MV15] If an attacker can get rogue software installed on a customer's vehicle, almost any feature that is software controlled can be potentially compromised, obviously depending on where the software is installed. However, the researchers have recently seen that, in today's vehicles, malware on just the telematics ECU is powerful enough to control vital vehicle control functions [MV15].

Table 17 shows the mapping of the proposed mitigations to the risks identified by the software update stages. Risks are listed on the right as rows and mitigations at the top as columns. Where an X is found, the mitigation corresponding to the column applies to the risk corresponding to the row. Following the table, the mitigation methods are detailed.

| | Mitigations | Update Authentication | Secure Channel (Authenticated and Encrypted) | Entity Authentication | User Authentication and Authorization | Use a Root-of-Trust-for-Update | Protect Keys and Security-Relevant Data Stored in ECUs | Prevent Bypassing of Authentication Mechanisms | Prevent Forgery or Unauthorized Generation of Digital Signatures | Separation of Duties | Code Reviews before Code Deployment | Ethical Hacking and Penetration Testing | Quickly Fix Security Bugs for In-House and 3rd Party Software | Traditional IT Best Practices | App Store Security | Physical Security | Secure Architecture |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Packaging** | Real Authority Signs Unauthorized Software | | X | | | | | | X | | | X | X | X | | | X |
| | Supplier Compromise | | X | | | | | | X | X | X | X | | X | | | X |
| | Signing Credentials Are Stolen | | | | | | | | X | | | X | | X | | X | X |
| | Attacker Forges Signature on Inauthentic Software | | X | | | | X | | X | | | X | | | | | X |
| | Attacker Remotely Exploits a Software Flaw | | X | | | | | | | X | X | X | X | | | | X |
| **Transport** | Physical Media Tampering in Transit | X | | | | X | X | | | | | | | | | X | X |
| | Software Installation Tools Are Compromised | X | | | | X | X | | | X | X | X | | X | | X | X |

| | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Transport, contd.** | Attacker Sneaks Hidden Functionality into App Store | | | | | X | | | | | | X | | | X | | X |
| | Attacker Masquerades as Legitimate Wireless Endpoint | | X | X | | | | | | X | X | X | | | | | X |
| | Attacker Masquerades as Legitimate Server in the Distribution Network | | X | X | | | | | | X | X | X | | | | X | X |
| **Reception** | Attacker with Physical Access Installs Malware | X | | X | | X | X | | | | | X | | | | X | X |
| | Attacker Installs Malware with Hacked OBD Dongle | X | | X | | X | X | | | | | X | | | | | X |
| | User Overrides Security Feature | | | | X | | | | | | | | | | | | X |
| **Install** | Attacker Used Undocumented Bypass Functionality | | | | | | | X | | | | | | | | | X |
| **Verification & Maintenance** | Attacker Spoofs Legitimate ECU and Reports False Information | | X | X | | | | | | | | | | X | | X | X |
| | Attacker Bypasses System Verification Routines | | X | X | X | X | X | X | X | | | | | | | | X |

**Table 17.** Mapping Malware Installation Risks to Mitigations. In this table, we list the risks (which might result in malware installation) from Section 3 as rows and mitigations as columns. Cells with an X indicate that the mitigation applies to the risk in that column.

#### *4.2.1.1 Update authentication.*

Message authentication on the software update package itself, or simply update authentication, is the primary (and most vital) mitigation against the risk of malware installation. Conceptually, update authentication is authentication of the software update package which must be performed before the package is installed. The authors assume that installation must be completed before the new application can have a functional effect on the target ECU. When working correctly, update authentication ensures that only authentic software can be installed on the ECU, providing a guarantee of authenticity and integrity of the received data.

Update authentication is almost always achieved with a digital signature. When using update authentication, each software update package is digitally signed before being delivered into the software distribution network or any vehicle. A digital signature is created by the publishing party or parties (for example, the supplier, OEM, and/or a trusted third-party provider) who create and append it to the software update package using their private key and an asymmetric cryptosystem such as RSA or Elliptic Curve Cryptography (ECC). NIST, in the Federal Information Processing Standards Publication 186-4, FIPS 186-4 [FIPS186], recommends the use of one of three standards for digital signature usage: The Digital Signature Algorithm [FIPIS186], RSA Security's PKCS #1 [PKCS1], or the Elliptic Curve Digital Signature Algorithm. Implementation of a digital-signature check based on one of the standards recommended in FIPS 186-4 before installation of a software update is widely considered a vital security control for software updates.

Asymmetric cryptography is the uncontroversial recommendation for update authentication, unlike generic message authentication for commands and requests from the network, where a symmetric-only solution might make sense. Authentication beyond update authentication is covered in Section 4.2.1.4. The authors believe that all or nearly all microcontrollers in a modern vehicle can perform a digital signature verification in a reasonable amount of time (processor time on the order of seconds or better).

It is currently believed that even the most powerful adversaries in the world cannot defeat digital signature verification, which employs modern high-strength cryptography and key lengths, although the design of the full system may include other, exploitable vulnerabilities allowing for bypassing the authentication provided by a digital signature. To protect against the installation of non-authorized software, the software update data must be verified according to the update authentication mechanism before it can be installed. Prior to installation, the contents of a software update package must not functionally affect the operation of the ECU.

There are best practices to protect the integrity of the design and ensure that the strong authentication methods are not bypassed. Providing a root-of-trust including secure storage of the verification key can prevent a keen adversary from finding a way to compromise the secrecy of the key itself, rather than breaking the algorithm, which is likely impossible with modern computing hardware (although the rise of quantum computing will break that assumption [Sch15]). Discussion of using a root-of-trust-for-update is given in Section 4.2.1.6 and secure storage of keys in Section 4.2.1.7.

If the target ECU cannot verify the authenticity of a software update package prior to self-installation, another ECU in the vehicle can perform this verification on a cached copy of the software update package before sending it to the target ECU for installation. Regardless, in an end-to-end update authentication design, the target ECU verifies the authenticity of the software update prior to allowing it to run for the first time. This check is in the reception stage from the previously identified software update stages. This check is performed in the bootloader, which, along with its cryptographic keys forms the root-of-trust-for-update. If the target ECU is convinced to start installation of a non-authentic software update package, the update verification check before running will prevent the non-authentic software from running, but the application will be disabled until a working application is reinstalled. This path can allow an attacker to cause a DoS of an application by sending it malicious or just junk data. Entity authentication from the server to the target ECU prior to the target ECU starting installation can protect against arbitrary DoS attacks using the software update mechanism on the target ECUs. Entity authentication beyond update authentication is discussed in Section 4.2.1.4.

#### *4.2.1.2 Secure channel (authentication and encryption).*

A secure transport channel is an important front-line defense against intrusion and unauthorized monitoring and data gathering. On the Internet, for example, hypertext transfer protocol (HTTP) over transport layer security (TLS) (HTTPS) is commonly employed to provide server authentication and encryption. Passwords or two-factor authentication provide the mechanism for the user to authenticate to the servers. For secure automotive software updates, particularly those delivered OTA, a secure channel between the vehicle and the software distribution network can be used. An end-to-end secure channel between the target ECU and the software distribution network servers is the most secure choice, but there are practical difficulties. Automakers today use a secure channel between the company's servers and a single, connectivity master ECU (generally, this is a telematics ECU).

While an end-to-end secure transport mechanism between the target ECU and the distribution network servers is the most secure solution, the authors do not expect most automakers to choose that design, at least not in the near term. The primary challenge is that it requires online operation for most of the duration of a software update. During the installation stage of the software update procedure, the target ECU being updated enters a state when the application is broken temporarily which varies in duration based on the design. Duration and external dependencies are minimized during the vital installation step, as during this period, the target ECU does not perform its functionality and this will affect the functionality of the whole vehicle. Local caching of software update packages within the vehicle prior to performing the software update installation procedure on the target ECUs removes dependency on an external network connection when performing the installation, particularly for OTA updates.

In their DEF CON talk from 2015, Marc Rogers and Kevin Mahaffey identified that Tesla is using OpenVPN to provide a bidirectionally authenticated and encrypted link between the vehicle's central information display (CID), the main connectivity and telematics ECU, and the company's servers [RM15]. While Tesla's Model S did not stand up to attackers with physical access, in their analysis of Tesla's architecture and design, Rogers and Mahaffey applauded Tesla's use and specific implementation of OpenVPN for transport authentication and encryption between the vehicle and the network, making this a reasonable reference design going forward.

The researchers believe that all modern vehicles with telematics ECUs provide at least some encrypted channel between the company's servers and some connectivity master ECUs ((e.g., telematic ESC) or even a dedicated software update master ECU, usually the telematics ECU. Most likely, implementations of transport-layer security (TLS) are used in most of today's vehicle telematics designs, for example over HTTP or, in Tesla's case, within OpenVPN.

Tesla's use of OpenVPN serves as a good example. When used correctly, TLS and OpenVPN is believed to be secure against even the most powerful adversaries. The connectivity master ECU will have both a public and private key for a bi-directional secure channel between the vehicle and network. Key management is non-trivial. Several mitigations that follow involve secure key creation, storage, and handling.

#### *4.2.1.3 Secure in-vehicle networks.*

In the future, vehicle networks and ECUs are likely to include secure communications channels within the vehicle as well. With the move towards Ethernet and/or CAN-FD as replacements for CAN as the internal backbone of vehicle networks, authentication and possibly encryption can be used to create a secure channel within the vehicle itself. However, in the near-term, CAN technology itself makes adding cryptographic authentication mechanisms difficult due to its relatively small frame payload size. In addition, there are still technological and operational challenges to meet before vehicles can use secure communication for much of the traffic on their internal networks, including re-visiting the vehicle communication architectures, provisioning the bandwidth and handling the complexity of keys.

Without an end-to-end secure channel between the target ECU and software distribution network, purpose-specific authentication can be used to add application-layer, end-to-end guarantees of authenticity to select communications. The next section discusses this mitigation.

#### *4.2.1.4 Entity authentication.*

Entity authentication for specific communications beyond the update package itself can be incorporated to better secure traditional, local software update mechanisms. For OTA updates, entity authentication is even more important. Two things motivate for authentication above and beyond the two mitigations previously described:

- An end-to-end secure channel between the target ECU and software distribution network is impractical today.
- The connectivity master ECU, which contains the trust basis for the secure channel between the vehicle and the network, is today usually the telematics ECU, a high-complexity ECU class on which successful, remote malware installation attacks have been demonstrated [Che11] [MV15] [Fos15].

Entity authentication beyond the update authentication itself is an important mitigation against compromised telematics units (or other ECUs for that matter). Today's vehicles are using a "coconut" design with strong mitigations used for communications between the vehicle and the outside world (as discussed in the previous section) and weak or no security controls used on

internal communications. End-to-end entity authentication for certain communications used for software updates is an important mitigation as well.

Digital signatures can be used for all entity authentication. Alternatively, symmetric cryptography like the Advanced Encryption Standard (AES) may be used to create message authentication codes (MAC), which can serve the same purpose as a digital signature in practice, providing guarantees of authenticity and integrity. (A MAC cannot provide what is called non-repudiation, meaning that any key holder can generate a valid MAC, unlike a digital signature, which can only be created by the unique holder of the private key.) Mitigations later in this document cover some aspects of key creation, storage and handling.

Entity authentication exists for two purposes:

- Server authentication
- ECU authentication

### *Server authentication*.

For server authentication, the software distribution network or related servers in the OEM's network architecture have private keys, either asymmetric or shared private keys. Previously, the authors assumed that the target ECUs in vehicles will have the computing power to verify an asymmetric digital signature for update authentication, and that holds for entity authentication. Using digital signatures allows for a secure multiple-verifier communication pattern, meaning that the server can sign a single message, and many different parties (such as the target ECU and another ECU like the telematics ECU) can verify that signature without compromising the secret key.

Server authentication can allow the ECU to authenticate messages coming from the network at large (for software updates, this means the software distribution network servers). The target ECU, with a public key for the server/network, can verify digital signatures or message authentication codes (MACs) appended to certain requests by the server/network with the ability to create those signatures or MACs with the private key. It is important that the design take precautions in creating, storing, and handling keys. Exactly what messages, requests, commands are to include message authentication can vary according to design. With entity authentication properly implemented, an ECU other than the target ECU that is compromised, such as the telematics ECU, cannot easily cause the software update mechanism to do something that either prevents correct, regular functionality of another ECU or do something to weaken the security controls/mitigations. As discussed in Section 4.2.1.1, one example of server authentication is using a command from the server with authentication prior to a target ECU beginning a software installation used if the target ECU cannot verify the authenticity of the software update package prior to self-installation.

### *Software version rollback protection*.

Software version rollback can be used as a building block for an attacker. Here, we do not mean rollback after a failed update procedure (i.e., re-installing the current version of software after a failed software update procedure used as a robustness mechanism). Rather, we mean rollback

from one working version of software to an older, but still legitimate, software version. One of the primary purposes of OTA software updates is to fix known security vulnerabilities. Often those vulnerabilities are relatively well known and possibly even published. An attacker with the ability to arbitrarily install old software versions can use this as a building block for getting malware onto vehicle systems by installing a software version with a known vulnerability and then exploiting that vulnerability.

One use for authentication is to authorize software versions for installation with a command sent from the software distribution network to the target ECU directly. Apple uses this design where devices running iOS must request authorization from the Apple installation authorization server before installing a version of the operating system [App15]. The design uses a digital signature and a combination of device-specific personalization and a random nonce from the mobile device to prevent replay of the authorization message later or for a different device. A nonce, derived from "number used once," is just that: a number that is probably only ever used once in the lifetime of the system (or a very long space of time). "Probably" is the preferred term because nonces are often random numbers, and the guarantee that a nonce is not used again in that case is a probabilistic guarantee, not absolute.

### *Replay protection.*

Entity authentication is usually paired with a scheme preventing replay attacks where an attacker replays valid messages again later. Replay protection protects against replay attacks. Adding freshness also adds a time limit to messages even if they are not consumed by the receiving party, so valid commands are not valid forever. Notice that in the case of update authentication replay protection is not recommended. Generally, software update packages are applicable, without modification, on many devices (a "class" of devices). Therefore, it is not important to add replay protection onto the software update package itself, and is more difficult logistically.

A nonce is the typical way to guarantee replay protection (and, optionally, freshness, if used in a design that guarantees freshness). Generally, the target device generates a good pseudorandom number, the nonce, and sends it to the server. The server then includes that nonce in their authorization message. Because, in a replay attack, the attacker masquerades as the server, but does not control the ECU, the ECU choosing the nonce randomly makes it practically impossible for the attacker to generate the correct authentication message. When used correctly, nonces are resilient against attackers who can spend significant time building a "codebook," a database of nonces/commands. The size of the nonce determines its strength and, in practice, so does the quality of randomness.

Apple uses device personalization to further add to the difficulty of gathering nonce/command pairs. By adding device personalization (that is, including some unique device identifier in the message over which the signature or MAC is created by the server), an attacker who can spend time gathering nonce/command pairs can only apply that codebook to a single device (the same one that was being monitored).

In automotive ECUs, however, random nonces are problematic because many ECUs will not be able to generate good random numbers. However, nonces do not need to be unpredictable; they merely need to be non-repeated (or have a very long repeat cycle). Using a nonce and device

personalization to messages, commands, and/or requests with entity authentication is an important piece to the authentication scheme to prevent replay attacks and, optionally, guarantee freshness.

### *ECU authentication*.

For ECU authentication, the target ECU itself has private keys, either asymmetric private keys or shared private keys. Those keys are stored securely with the root-of-trust-for-update, which is detailed in Section 4.2.1.6. Secure storage and handling of keys in the ECUs themselves is detailed in Section 4.2.1.7. ECU authentication may be achieved with either digital signatures or MACs. Today, however, hardware-assisted, asymmetric cryptography solutions are not widely available and may not be for some time. Asymmetric cryptography can be logistically much simpler due to the single signer/multiple verifier pattern. However, that pattern fails for ECU authentication and asymmetric cryptography is not significantly less complex than a symmetric scheme.

### *Data and status reporting.*

Accurate reporting of information from vehicles and their electronics and ECUs in the field is an important part of software updates, for both those delivered locally and remotely. In traditional OBD-attached software updates, data is generally gathered offline using traditional diagnostics routines for gathering data from the vehicle (e.g., those in the Unified Diagnostics Services (UDS) standard [ISO14229]). Software update packages and any related data in traditional programming are obtained either through a web portal or media received through the mail. For OTA software updates (and traditional, local updates with a network-attached or augmented design), data and status reporting can be transmitted by the vehicle ECUs themselves. In many modern designs, a telematics ECU assists or facilitates this communication between the bookkeeping database, software distribution network, and ECUs in the vehicle.

By giving the ECU an authentication mechanism (that is cryptographic algorithms, keys, and an authentication scheme), the ECU can securely transmit communications to the software distribution network (and bookkeeping database, directly or indirectly). Key management is a large hurdle to providing ECU authentication in practice.

The authors believe that some OEMs will use ECU authentication in their OTA updates, but possibly not all initially due to the logistical and technical challenges that remain.

### *Secure remote attestation.*

Remote attestation (RA) is a technique that allows a trusted server to verify the integrity of the software running on a remote untrusted and possibly compromised computer system, such as an ECU in this context. [Cok11] After an update is installed, remote attestation can be used later to check whether the installed software has unauthorized changes or not. If the system cannot pass the attestation process, it means the installed software has been modified in some way and become potentially unsecure and unsafe. Thus, the server can issue a warning and alert the driver to install a healthy version of the software.

RA has been considered a promising technique to fortify embedded systems security. A successful attestation allows the establishment of the root of trust of the remote entity. Various RA techniques have been proposed and can be classified into three categories based on whether secure hardware is used to assist in the attestation.

*Software-based remote attestation* techniques usually ask the remote party (e.g., an ECU) to compute a checksum of its memory using a specially crafted function, which can result in observable side effects such as too much delay if there is any cheating attempt to emulate the function. *Hardware-based remote attestation* techniques usually use secure hardware to assist attestation. Commercial and standardized hardware-based RA techniques include ARM TrustZone and Trusted Platform Module (TPM) based high-end microprocessors. *Hybrid remote attestation* techniques explore the design space between the two extremes of software and hardware based RA and use a hardware-software co-design to allow remote attestation.

The electrical system, including the ECUs, of a modern vehicle is a complicated distributed cyber physical system with multiple tens of ECUs of various kinds. Many challenges exist to applying RA techniques to a complex automotive system. High-end ECUs may have secure hardware such as ARM TrustZone to allow hardware-based attestation. Medium-end ECUs may be modified to support hybrid remote attestation methods. Low-end ECUs, lacking hardware to support RA, may use software-only RA. Alternatively, high-end and medium-end ECUs can be used to help remote attestation of low-end ECUs.

#### *4.2.1.5 User authentication and authorization.*

The user (such as a technician in a dealership or an end-user driver) or the OBD debugging tool that performs the update needs to be authenticated first before the update process starts. That is, an update process starts only after an authenticated user allows the update process to proceed. Along with update authentication (Section 2.1.1), user authentication adds another layer of security check to avoid potential unsafe operations. User authentication can take many forms including password verification, hardware authentication tokens, biometric-based authentication methods, or cryptographic challenge-response authentication protocols. If the presence of a human can be determined securely, this indicates possession of the vehicle (something you have), one factor of two-factor security. The second factor can be a password, passphrase, or pin (something you know).

User authentication can be particularly useful for securing traditional OBD port-attached software updates. It is important that remote attacks are not able to use the "local" update functionality. By securely communicating to the target ECU that a valid user (either the owner or a mechanic) has authorized a local update, the target ECU can be quite certain that the local update is allowed. In NIST's "BIOS Protection Guidelines," local update routines can be secured via some physical mechanism, like a hardware jumper [NIST11]. User authentication is a logical extension of that concept to automotive.

#### *4.2.1.6 Use a root-of-trust-for-update.*

In the NIST-published "BIOS Protection Guidelines," recommendations for securing BIOS firmware, the authors recommend the use of a root-of-trust-for-update[NIST11]. The root-of-

trust-for-update, in the NIST recommendations, includes the verification algorithms and keys for the verification. For the automotive space, the root-of-trust-for-update is essentially the bootloader along with its verification routines and verification keys. In the past, automotive ECU bootloaders did not perform cryptographic verification, instead relying on a CRC for integrity guarantees before starting a new application. However, the risk of malware installation either using OTA or traditional OBD software update mechanisms means that cryptographic authentication is now widely recommended for software updates (as described in the previous sections). Because keys are somewhat different from software routines, the writers treat the keys separately in the next section. This section is devoted to securing the bootloader and verification routines.

### *Prevent easy modification of the bootloader.*

The NIST "BIOS Protection Guidelines" recommend that any update to the root-of-trust-for-update must include at least the same mitigations as an update to the software [NIST11]. This is the bare minimum. If possible, the root-of-trust-for-update can be stored in a protected region of ROM. However, this might not be possible, in which case the bootloader must include software mechanisms to prevent allowing itself to be overwritten.

Preventing bootloader-type malware could include making so-called dark regions, or areas of the storage that are rarely modified and are not part of the file system, un-writeable. By restricting firmware bootloader writes to physical access, this type of malware could be defeated. Any attempt to write to these regions can instantly raise as a red flag.

### *Use a trusted region that cannot be updated remotely.*

One mitigation against bootloader malware (and invalidation of the root-of-trust-for-update) is to completely disallow updating of the root-of-trust-for-update from being updated remotely. When working properly, a local OBD port-attached software update would be required to update any part of the root-of-trust-for-update, perhaps except for keys. The keys in the root-of-trust-for-update are discussed in the next section.

#### *4.2.1.7 Protect keys and security-relevant data stored in ECUs.*

To protect bypass of correct verification, each ECU can store its verification keys in such a way that an outside party without authorization cannot modify them. If those keys are modifiable by an authorized party, the cryptographic mechanisms and design to authorize key change can be, at minimum, as strong as the mechanisms used for software updates themselves.

### *Hardware-assisted secure storage.*

Secure storage is a solution that is particularly resilient against software attacks and is currently being investigated by the SAE Vehicle Electrical System Security Committee [J3101]. Silicon vendors are beginning to offer microcontroller system-on-a-chip (SoC) solutions with embedded hardware security modules (HSMs) for secure key storage and accelerated cryptographic operations. Embedded HSMs have their roots in the German auto industry, specifically the Hersteller-Initiative Software's (HIS) Secure Hardware Extension [HIS09] and the E-safety vehicle intrusion protected applications (EVITA) group's fuller HSM design [EVITA10].

In the near term, the researchers believe that HSMs offered with microcontrollers will remain limited to symmetric keys and algorithms, although generic secure storage might be an optional use case in some designs. Additionally, today's automotive computing hardware does not usually include secure storage, and many ECUs will likely continue to use computing hardware targeted for low-cost, eschewing non-essential peripherals. Therefore, even ECUs which do not have hardware-assisted secure storage for some or all keys must secure their internal storage and RAM, especially sensitive areas where cryptographic material is stored using modern best practices for secure product and software development.

#### *4.2.1.8 Prevent bypassing of authentication mechanisms.*

The NIST document about BIOS updates recommends "preventing bypass of message authentication" [NIST11]. This seems obvious, but proves to be difficult in practice. Backdoors left in production systems plague computers of every kind, from powerful servers to puny single-purpose ECUs. Many otherwise robust security designs are defeated by backdoors or other engineering- and test-only interfaces, either willfully or accidentally left in production systems. Government-mandated backdoors are also at risk of compromise or discovery by unauthorized parties.

Backdoors are generally left in production systems through carelessness or the belief that they will not be discovered in practice. However, hackers and penetration testers look for backdoors as a matter of course. Examples of backdoors in production embedded systems are serial ports, network services such as telnet, JTAG ports, and secret network diagnostics routines. At the Black Hat security conference two years ago, Charlie Miller and Chris Valasek exploited a powerful interface called D-Bus to gain malicious code execution on a Jeep's Infotainment ECU [MV15]. That interface likely should have been disabled in production systems. It is not unheard of for suppliers to leave secret powerful diagnostics routines enabled on production systems for configuration or calibration in the field.

All mechanisms that can allow for code installation and configuration on production ECUs can be enumerated as front doors and protected with proper authentication mechanisms as described previously in this document. Engineering and test interfaces that are not essential can be disabled in production ECUs, ideally irrevocably.

#### *4.2.1.9 Prevent forgery or unauthorized generation of digital signatures.*

Forgery and unauthorized signature generation are the other main attack paths for getting around a digital signature, which cannot be defeated directly using modern computers. Forgery may exploit weaknesses in the key selection where the algorithm itself is otherwise probabilistically impossible to defeat. Unauthorized generation of signatures can be because of key theft or insufficient security controls on the production signing process and system.

Proper key selection and handling is a topic in its own right. Key management for traditional IT systems is well understood, and best practices exist and can be followed. In recent years, weaknesses in key generation procedures, usually due to poor entropy, have been a hot topic [Hen12]. The problem seems to be more prevalent in networked, embedded systems where good entropy is scarce. Charlie Miller and Chris Valasek showed that the Uconnect infotainment

system creates Wi-Fi Protected Access (WPA) keys with little entropy, resulting in a very few possible selections, which can be brute forced based on an estimation of a vehicle's assembly date [MV15]. Best practices for properly creating cryptographic keys can be followed as well.

Because software update package signing is a semi-automated procedure involving some coordination between employees within at least one supplier, the OEM and possibly a trusted third party, this provides a significant source of human interaction in the signing process. Humans lose passwords and can be coerced or convinced to betray their loyalties. In the production software-acceptance/signing process, best practices can be followed. Proper authentication mechanisms and management can be used. Separation of duties can prevent a single point of human failure. Code reviews can prevent backdoors or malicious code from being snuck into software at the source level. These mitigations are discussed further in the next sections.

#### *4.2.1.10 Separation of duties.*

Maintaining a separation of duties is important for secure software development. Separation of duties ensures no one could potentially, while undetected, siphon information or maliciously modify code or system infrastructure. For instance, when using digital signature for update authentication, the signature would be generated by one party (e.g., the security department) and added to the update by another party (e.g., the development department). In this way, no single entity can abuse the use of digital signatures.

Like many security mitigations and controls, separation of duties creates layers, which an attacker must penetrate to form a successful attack. This, again, like many security mitigations, creates a less efficient process for greater security. That said, most modern software companies practice separation of duties to a relatively large degree.

Human beings are one of weakest links in a security chain. Unlike computer systems, humans are not deterministic. While machines can be subverted, the mechanism is purely deterministic and, when discovered, can be fixed or mitigated. With humans, anything is possible. Humans can be coerced or enticed to work against the interests of their job role, for example, through blackmail or bribery. Moles within large organizations are not unheard of either. For this reason, it is important that secure software update procedures are designed with a healthy separation of duties so that a single or small number of compromised (or even lazy, incompetent, or ill-trained) employees cannot be a single point of security failure. For example, while a software supplier may be able to submit malware to the software repository, a healthy separation of duties would mean that, at minimum, checks among the OEM and other employees at the supplier and even possibly trusted third parties before that software update package is signed or delivered to vehicles is important. Separation of duties is important for a secure software update process.

#### *4.2.1.11 Code reviews before code deployment.*

Code reviews are an important technique that must be used to ensure that no malicious code or software vulnerabilities exist in the codebase. Thorough peer review, code change tracking and integration testing can be done so that the code is free from backdoors or other malicious implanted behaviors. The next generation of vehicles are very complex and often use third-party

supplier software or open source software. Constant attention to security bulletins and regression testing for security flaws is important. Open-source flaws can be detected many years after a piece of software exists in production systems, and applying these updates can be swiftly done.

#### *4.2.1.12 Ethical hacking and penetration testing.*

Penetration testing is the specific engagement of using a set of white hat hackers to attempt to exploit a production or would-be production system just as an attacker would. Some penetration testing is done with no inside information (so-called "black box") or some inside information (so-called "gray box"). Ethical hacking is a broader category, including security research community engagement and even bug bounties.

Penetration testing is an important part of a security-conscious product development life cycle, particularly for electronics. It is a form of measurement, which cannot be achieved with traditional testing mechanisms. This measurement is important because, without it, system designers and engineers generally continue to believe that their security controls cannot and will not be defeated. Even the most experienced and talented engineers can fall into that trap.

Therefore, it is a powerful mitigation to use outside ethical hackers to help find any system vulnerability unknown to the developers. Recently, Tesla had security researchers from firms Lookout and Cloudflare attempt to find and exploit security vulnerabilities [RM15]. Until recently, the automotive industry has been a security black box and has downplayed cybersecurity vulnerabilities with emphasis on protecting trade secrets and mitigating legal risks. Automakers can move into a more modern software-driven culture and engage with white hat hackers and the security community at large.

#### *4.2.1.13 Quickly fix security bugs for in-house and third-party software.*

Using outdated and often insecure software has plagued networked embedded systems. The researches have known about this very problem within automotive software for a few years now, as well, particularly in infotainment and telematics ECUs [Che11] [Fos15] [RM15]. For any electronic system, it is important that known security vulnerabilities be patched in a timely manner. Suppliers and OEMs can embrace the reality that networked software must be up-to-date to be secure. Zero-day vulnerabilities (i.e., vulnerabilities that are not known prior to real-world exploitation) are a problem, but the exploitation of well-known security vulnerabilities in the field is particularly important to avoid.

#### *4.2.1.14 Traditional IT best practices.*

There are a host of traditional IT best practices that can be followed to secure the network servers used for software updates (including the software distribution network, software repository, and bookkeeping database). Traditional IT security is a mature field and more than can be covered in this report. The following items, however, can be addressed:

- *Protection of private keys used for software update package signing and authentication.* Key management is non-trivial, but there are many well-known best practices and mitigations.

- *Strong authentication used for submission of software update packages to the software repository*. Strong, bi-directional authentication can be achieved with two-factor authentication.
- *Logging and auditing*. The use of logging and auditing is very important and serves to both debrief after an incident as well as detect potential incidents in progress. The benefit of good logging is that auditing can be improved and automated over time.
- *Quick response to cyber incidents*. Companies must have procedures in place to handle cybersecurity incidents prior to one occurring. Creating or recommending best practices for designing and implementing a cybersecurity incident response process is well beyond the scope in this document and will vary from company to company.

#### *4.2.1.15 App store security.*

Experts have suggested that the emerging automotive infotainment ecosystem is ripe for app stores, which are integral in the mobile phone ecosystem. While this is not technology of today, soon software add-ons to automobiles will be distributed via app stores. App stores are difficult from a security perspective because the software is only somewhat controlled by the OEM. Fortunately, there is a wealth of knowledge and lessons learned in the mobile industry, including thorough systematic vetting of developers and their apps, API access control, and swift responses to malware when detected. If app stores are rolled out by automakers, these mitigations can be adapted from the mobile space.

#### *4.2.1.16 Physical security.*

Physical security is the set of mitigations used to keep physical locations and things secure. For software updates that are delivered on physical media, physical security mitigations can be used to preserve the integrity of the software update packages. Physical security mechanisms include controlled access to secure locations, break-in deterrents such as cameras or barbed wire, and tamper-detection packaging of media. The exact physical security mechanisms applied to storage, transportation, and creation of physical media or tools for software updates can be tailored to the specific technical design and logistical infrastructure of the suppliers, OEM, and possibly, trusted third parties.

#### *4.2.1.17 Secure vehicle architecture.*

Modern vehicles are highly electrified. With the rise in partially autonomous and active driver assistance features, almost all functions of vehicles are under electrical control, and the ECUs in control are, for the most part, networked together using serial bus technology like Ethernet, FlexRay, Universal Asynchronous Receiver/Transmitter (UART)/Local Interconnect Network (LIN), or most commonly, CAN.

The traditional assumption that only persons with physical access (or proximity) can affect vehicles' electrical systems and internal networks has driven cost-first optimization of the vehicle's data and control architecture and partitioning without accounting for security. However, with the rise of connected vehicles equipped with cellular data modems, Wi-Fi, or an attached mobile phone providing pass-through access to the Internet, that assumption no longer holds.

Up to this point in the document, the mitigations described have been augmentations to the vehicle's electrical architecture to improve the security posture, at minimum to the current known best practices. However, the difficulty in creating drastic security and even safety attacks can be increased by rethinking the entire electrical architecture. As a motivating example, it is not uncommon today for a connectivity device like an infotainment unit or a telematics device to share a CAN network with a dozen or so controllers with disparate functionality. This architecture, combined with non-intentional gateway logic between network buses, has shown to be particularly vulnerable to remote control attacks [MV15].

Intentionally revisiting the electrical architecture and partitioning of future automobiles allows for drastically increasing the probability of success for high-impact outcomes like remote vehicle control. In the perfect case, connectivity modules like OnStar or Ford SYNC or U-Connect would be completely partitioned (that is, disconnected) from the remainder of the vehicle and its electronic controls. While this design removes much of the expanding utility of connectivity to the vehicle and is, therefore, unlikely to be adopted in practice, it also removes the possibility of a remote-control attack by malware installed on an "air-gapped" infotainment system.

While air gapping so-called connectivity devices is unlikely to be implemented in practice, a more intentional secure architecture and functional partitioning can come a lot closer to the perfect, air-gap solution than the naïve, cost-driven architectures of many of today's automobiles. By segmenting the bus functionally, different control systems can be better isolated into unique segments, with less data needing to be copied from one bus to another by a gateway. To maintain the sanctity of the segmentation, gateways—ECUs that move data between different network buses (for example, between a powertrain controls CAN bus to a body electronics CAN bus)—will need to be seen as firewalls. Firewalls can be chosen intentionally for their security properties. In some cases, firewalls can include new logic to block and/or monitor suspicious traffic. Secure architecture may include the use of intrusion detection systems (IDSs), and security-specific applications (either dedicated or built into another ECU) specifically tasked with monitoring for and flagging malicious, suspicious, and/or anomalous traffic on the vehicle networks.

### 4.2.2 Denial-of-service.

A denial-of-service attack happens when an attacker with some amount of control over a system causes it to fail to perform some of its duties. For example, an attacker might spawn many clients to overwhelm an HTTP server (or servers). When many attackers or an attacker with control of a botnet, a collection of computers or devices running malware and at the command of an attacker, performs a DoS attack in coordination, it is called a distributed denial-of-service (DDoS). The design of the software distribution network and/or datacenter must protect against DDoS attacks, for example, by using authentication in communications with clients and using load balancing. This topic is mature and outside the scope of this work.

These traditional DoS and DDoS attacks on IT infrastructure are certainly risks for any server network/datacenter, but they are not the focus in this work. Rather, the focus here is to specifically consider DoS attacks against the software update mechanism itself, which might allow an attacker, possibly the owner, to bypass software updates to retain an older version of software, potentially with known security vulnerabilities.

DoS attacks are extremely difficult to prevent in practice. In fact, one of the best deterrents against DoS attacks is the potential legal consequences (either criminal or civil). The team does not propose any specific mitigation against DoS attacks that are not also relevant to malware installation. Therefore, any mitigations that apply to DoS have already been described. Table 18 reiterates the DoS risks from Section 3.4.2 and lists the mechanisms to mitigate those risks. Mitigations have been described in Section 4.2.1.

| Risk | Mitigation | Technique |
|---|---|---|
| **Attacker spoofs delivery network servers** | Server authentication | Entity authentication. |
| **Attacker spoofs legitimate ECU to tamper with software updates** | ECU authentication | Entity authentication. |
| **Attacker spoofs legitimate wireless interface access point** | Network authentication | Digital signature, message authentication, proper key and password management. |
| **Jamming of wireless interface** | Anti-jamming techniques | No specific anti-jamming techniques recommended, as this is well outside the scope of software update mechanisms. |

**Table 18.** DoS—Risks and Mitigations. This table reiterates the DoS risks from Section 3 and lists the mechanisms to mitigate those risks. Mitigations have been described in Section 4.2.1.

### 4.2.3 Unauthorized download of information.

In Section 3, the team defined the high-level risk of unauthorized download of information as the unauthorized access of either intellectual property or private information related to drivers or owners. This section reiterates the risks from Section 3 and maps them against the mitigations proposed. Table 19 shows the risks and mitigations.

This report focuses on the mitigations for the malicious software installation (and to a lesser extent, DoS) misuse cases. In this section, some mitigations related to unauthorized data access/download are briefly mentioned. Many of the mitigations presented in the malware installation scenario also apply to unauthorized download of information, including authentication mechanisms prior to download (ECU-to-server authentication) and using an encrypted channel between the vehicle and the back end.

| Risk | Mitigation | Technique |
|---|---|---|
| **Attacker masquerades as legitimate ECU to download data** | ECU authentication<br>End-to-end encryption | Entity authentication. Update encryption. |
| **Attacker spoofs legitimate ECU to tamper with software updates** | ECU authentication | Entity authentication. |
| **Attacker abuses data gathering functionality** | Server authentication<br>Secure channel<br>End-to-end encryption | Entity authentication. TLS secure channel. Data gathering encryption. |
| **Man-in-the-middle** | Secure channel<br>Physical security<br>End-to-end encryption | TLS secure channel. Update encryption. Data gathering encryption. |
| **DRM circumvention** | Entity authentication<br>User authentication<br>Secure channel<br>Access control | Personalized message authentication for feature authorization. User authentication to tie content to a person instead of a vehicle or its hardware. TLS secure channel. Access control to authorization servers. |

**Table 19.** Unauthorized Download—Risks and Mitigations. This table reiterates the unauthorized download of information risks from Section 3 and lists the mechanisms to mitigate those risks. Most mitigations have been described in Section 4.2.1.

#### *4.2.3.1 Update encryption.*

Because loss of intellectual property is a one-and-done attack (that is, once the information has been stolen, it is unlikely that an attacker has more to do, if intellectual property theft is the attacker's purpose), a stronger privacy mechanism is justified. Particularly, end-to-end update encryption can be used to secure the intellectual property contained in a software update package all the way to the target ECU. Update authentication can be augmented with two signatures: one on the application binary and one on the encrypted software update package itself, so that parties without access to the decryption key can still verify the authenticity of an encrypted software update package. When update encryption is properly designed and implemented, only the target ECUs and software distribution network (and related) servers have access to the private key used for decryption.

Symmetric cryptography is used for encryption, AES being the de facto standard in use today. Using AES to achieve a secure, private data transmission is well understood and best practices can be followed. It is possible to perform a key exchange between the ECU itself and the back end using asymmetric cryptography to authenticate the ECU to the server, but this is a less likely design for the same reasons it is unlikely to use asymmetric cryptography for ECU authentication. Private keys, whether asymmetric or symmetric, can be managed securely. Mitigations related to creation, storage, and transportation of keys are discussed in Section 4.2.1.

It is possible to use the same private key for encryption and decryption across a broad class of ECUs so that the software update package does not need to be encrypted differently for each individual ECU. This is analogous to the non-personal nature of the update authentication mechanism.

Update encryption is technically a weak mitigation for the malware installation risk. By encrypting updates and only providing plaintext (that is, unencrypted data) to authorized parties, it becomes more difficult for an adversary to gain access to the software binary for reverse engineering. Reverse engineering is very useful in building a successful attack; the alternative is trial-and-error. However, update encryption may not be very strong against reverse engineering because a dedicated adversary will still likely find a way to get firmware from a device in the field.

#### *4.2.3.2 Data gathering encryption.*

The data gathering functionality to support software updates might possibly contain private information on the drivers and/or owners of vehicles. Alternatively, the metadata to support software updates might be mixed with analytics of all kinds, including private information. To secure this data against unauthorized access, the metadata can be encrypted from the target ECU to the software distribution network/bookkeeping database. As with update encryption, when properly designed and implemented, only the target ECUs and software distribution network (and related) servers have access to the private key used for decryption. And, again, symmetric key cryptography is the most likely way to achieve encryption with a private key shared between the ECU and the back end. In the case of data gathering encryption, the ECU should have a unique private key.

#### *4.2.3.3 Access policies.*

Finally, because unauthorized download can happen from anywhere by anyone, access policies can be created and enforced on the software distribution network and/or feature authorization servers. This is to prevent unauthorized individuals from accessing sensitive information, either to prevent theft of intellectual property or private information. Separation of duties and traditional IT best practices can also be followed and are discussed in Section 4.2.1.10 and 4.2.1.14, respectively.

## 4.3 Conclusion

In-field software updates are a necessity in the automotive industry to fix flaws without replacing hardware. With the rise of the connected vehicle, OTA software updates have also become necessary to keep software patched when security vulnerabilities are found. The current generation of automobiles primarily uses OTA software updates for telematics and infotainment ECUs only. However, Tesla, for example, has a mechanism to update any ECU remotely OTA, and automakers that do not are moving towards OTA software updates for other ECUs quickly.

While software updates are a boon for security, the mechanism, particularly the remote mechanism, creates a new avenue for attackers to exploit. In addition to adding to the threat surface of an automobile's electrical system and ECUs, software update mechanisms include the

ability to download binary software from a network and install it. If the update functionality itself can be compromised, this is very attractive to attackers for installing malware.

In this report, the team created a list of mitigations that apply to the risks identified previously. Next, the risks are mapped onto the update stages, and a matrix is created showing which mitigations apply to which risks, shown in Table 17. OEMs will choose what mitigations to implement as they design and roll out OTA software updates and improve upon the security of traditional OBD port-attached software updates.

# 5. Intellectual Property Theft Risks and Mitigations

Intellectual property theft can take many forms. For example, a competitor may want to steal intellectual property such as software in a current or future product design. Or, maybe a counterfeiter may seek to download a software or firmware binary to place into a counterfeit electronics part without modification. Even rich information about the vehicle, owner, and/or operators can be viewed as OEM intellectual property beyond the privacy concern, which necessitates care. Finally, digital rights management (DRM) is used to license purchased content in the field, or after-sale add-on content. Circumvention of DRM features is attractive to vehicle owners to enable additional content without paying, such as new vehicle features like adaptive cruise control or even just media files.

Software intellectual property often represents an enormous investment in its development. One of the difficulties with intellectual property theft is that, in some scenarios, such as software binary theft, even a single incident of improper access is enough for an attacker to achieve their goal and the information to be leaked. For that reason, it is extremely difficult to completely protect against intellectual property theft. It is very reasonable to assume that the adversaries can spend countless hours with physical access to the hardware attempting to get the software binary or other intellectual property out. Software binaries are essentially machine code, which can be used to counterfeit the electronics without recreating the software or for reverse engineering by competitors or hackers. Vehicle owners are also interested in circumventing DRM protections for licensing add-on content. Even private information about vehicles and their owners and operators can be viewed as OEM intellectual property.

## 5.1 Abuse Cases and Risks

The following abuse cases are slightly different than the risks identified in Section 3.4.3. Abuse cases are effectively use cases from the perspective of a motivated adversary. Each abuse case begins with the adversary named as the subject of the abuse case.

1. Competitor steals intellectual property in the form of software or firmware.
2. Counterfeiter steals intellectual property in the form of software or firmware to produce counterfeit parts that run the stolen application software.
3. Customer uses flaws in the DRM of after-sale purchased content to gain access to content that was not actually purchased.
4. Unauthorized entity gathers data from many vehicles in the field.

The technical risks that can lead to the first two abuse cases are the same, although the actor is different. The specific risks map quite closely to the four abuse cases. These risks are the same as were identified in Section 3.

1. Attacker masquerades as legitimate ECU to download software. (Abuse cases #1 and #2)
2. Attacker circumvents DRM to install add-on content that has not been paid for or licitly acquired. (abuse case #3)

3. Attacker abuses data-gathering functionality and masquerades as legitimate server to siphon information from vehicles in the field. (Abuse case #4)
4. Attacker creates a man-in-the-middle attack and siphons off the intellectual property data desired. (Abuse cases #1-4)

There are other intellectual property theft risks; however, they are not related to software updates. One such example is the risk that an attacker attaches directly to the JTAG port on an ECU and downloads the software. That attack works just as well to download ECU software, but it is not related to in-field software update functionality in any way. This analysis is limited in scope to only risks relevant to in-field software updates.

In Section 4, a table was created mapping malware installation risks to mitigations. Many of those mitigations are applicable to intellectual property theft risks as well. Table 20 is the risk/mitigation mapping for intellectual property theft risks. Risks and mitigations that were retained from the malware installation table are non-bold, and intellectual property theft specific risks and mitigations are bold.

For dedicated, moderately funded threats, the motivation and severity attributes are mostly fixed. The stakeholder impact is monetary only. The motivation for attackers is high. The mitigations proposed here, then, seek to reduce a would-be attacker's probability of success.

Interestingly, most people that the research team speaks with regularly from the automotive industry, both automakers and tier-1 suppliers, do not express a strong intention to do anything about intellectual property protection for software binaries. The prevailing opinion seems to be that there are so many ways for attackers to get a hold of software binaries that it simply isn't worth adding additional protections like update encryption for the software update process (discussed in Section 4 and the next section). However, according to the automakers, protection of owners' and operators' private information, which might be considered OEM intellectual property, is very important. The team believes that integrity of DRM in licensing after-sale, add-on content will be important to OEMs in tier-1 suppliers as that opportunity grows in the future.

## 5.2 Mitigations

The primary mitigation applied to software updates to reduce the risk of intellectual property theft of the software is update encryption. Just like update authentication using a digital signature is the primary, essential mitigation against the installation of malware, update encryption is the primary, essential mitigation against unauthorized download of information with regards to software distribution. End-to-end encryption from the software distribution network to the target ECU can prevent an adversary from being able to access the software binary while in transit or storage on a server ready for download, even if an adversary can perform a man-in-the-middle attack between the vehicle and parts or all the software distribution network.

It is important that the encryption cannot be bypassed (in this case, it is the distribution network which must ensure this) and that cryptographic keys are created, stored, transported, and maintained properly, both on the ECUs and in the software distribution network. In addition, ECUs can be locked down on local interfaces to protect software binaries. This includes using, at

minimum, password control on ECUs to access debug features which can read from memory. It also includes not permitting any sort of legitimate interface, such as UDS diagnostic routines, to allow the binary to be downloaded *from* the ECU in production, even with physical access to the vehicle or ECU itself.

Encryption is difficult to use securely in practice. The main difficulty is the logistics of the keys and of the encrypted software images. If a single symmetric key or small set of keys is shared among many ECUs (such as an entire vehicle line or model year), the security of the encryption is reduced because there are many places where keys might be compromised. The usual solution is to avoid cryptographic key reuse and give each unique ECU its own key. However, using unique encryption keys for each ECU is logistically difficult and confounds caching and distribution (because each different ECU with a different key will need to download a completely different binary which has been encrypted for that one unique key specific to that ECU).

A sensible middle- ground design might be to share keys among only homogeneous ECUs for software update encryption. If each ECU in the homogeneous "class" protects the keys in the same way and software is not otherwise significantly personalized to each ECU, this makes perfect sense. Once a symmetric encryption key is compromised for a homogeneous class of ECUs, the firmware for that class can be decrypted. The attacker cannot do much more with the encryption key in that class of ECU. This assumes that proper key provisioning best practices are used, and the encryption/decryption keys are not used for authentication; rather, dedicated keys are used for that purpose (generally asymmetric as well).

### 5.2.1 Physical tampering protection.

Physical tampering protection is a mitigation against intellectual property theft as it can prevent an attacker with local, physical access from extracting the software or other intellectual property from a device in his or her physical presence. Tamper-proofing hardware can be created in such a way that attempting to remove the tamper proofing destroys the software and hardware preventing access to the information. This mitigation is not discussed further here because it is not related to software distribution.

## 5.3 DRM Circumvention

The astute reader will observe that no mitigations specific to the DRM circumvention risk are listed. From the perspective of software updates, the secure way to provision add-on content is to use the existing secure software update mechanism. Secure software distribution can be safely assumed to be in place before or concurrently with large amounts of after-sale, add-on content becomes mainstream. If software updates are implemented correctly, the distribution of authorization commands can use the same utility (these could be called *tokens* or *tickets*). Therefore, the mitigations against DRM circumvention are the same as the mitigations against malware installation. DRM features, just like other intellectual property theft features, must be protected against adversaries with physical access as well. The one primary difference between DRM tokens and software binaries is that replay of DRM tokens is a legitimate concern and, therefore, they can be protected against replay attacks.

| | Mitigations | Secure Channel (Authenticated and Encrypted) | ECU Authentication | Protect Keys and Security-Relevant Data Stored in ECUs | Separation of Duties | Ethical Hacking and Penetration Testing | Quickly Fix Security Bugs for In-House and 3rd Party Software | Traditional IT Best Practices | Physical Security | **Update Encryption** | **Data Gathering Encryption** |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Packaging** | Supplier compromise | | | | X | X | | | X | | |
| **Transport** | Physical media tampering in-transit | | | | X | X | | | X | X | |
| | Software installation tools are compromised | X | | | | X | X | X | | X | |
| | **Attacker masquerades as legitimate ECU to download data** | | X | X | | X | X | X | | X | X |
| | **Man-in-the-middle** | X | | X | | X | X | X | | X | X |
| **Reception** | **DRM circumvention** | X | X | X | | X | X | X | | X | |
| **Verification & Maintenance** | **Attacker abuses data gathering functionality** | X | X | X | | X | X | X | | | X |

**Table 20.** Risks and Mitigations. Intellectual property theft risks are the rows in the table and mitigations are the columns. Cells with an X indicate that the mitigation applies to the risk in that column. Risks and mitigations for intellectual property theft were discussed in Section 4, which included a similar table for malware installation risks. Risks and mitigations specifically added for intellectual property theft are in bold.

## 5.4 Conclusion

Intellectual property theft, particularly software binary theft, can be enabled and made easier with software update mechanisms, particularly OTA mechanisms. In sections 3 and 4, the team identified the intellectual property theft risks and mitigations. In this section, intellectual property theft risks and mitigations were revisited, and then a mapping was created between the two based on the frameworks from the previous sections.

In discussions with the OEM and tier-1 employees, the majority opinion is that protecting the software binaries is not a priority. The prevailing opinion in the industry is that there are too many other ways for an adversary to obtain a software binary to justify the cost of adding encryption to the software update process. Some specific ECUs and/or applications are likely to use software update encryption, however.

# 6. Counterfeit and Fraudulent Electronic Parts and Products

Fraudulent and counterfeit parts can pose a safety and monetary liability risk. Even if out-of-spec electronic parts don't always compromise safety, they can wear out more quickly and perform poorly. In the automotive industry, counterfeit parts not only impact warranty and maintenance costs, but can also lead to brand damage. SAE aerospace standard AS5553A [AS5553A] is a quality standard that organizations can adopt to create a robust process for detecting, preventing, mitigating, and disposing of fraudulent or counterfeit parts.

SAE AS5553A is an existing standard for the prevention, detection, mitigation, and disposition of counterfeit and fraudulent electronic parts in the aerospace industry. Due to the similarity of aerospace and ground vehicle electronics and the existence of an SAE standard for protection against counterfeit and fraudulent electronics in aerospace, the team's analysis of that document guides this section. They attempt to identify unique insights for the automotive industry. That said, the SAE AS5553A standard is general enough to apply to the automotive industry with only minor modification.

SAE AS5553A defines the following types of parts:

- A **suspect part** is "any part for which there is some indication that it might be fraudulent or counterfeit."
- A **fraudulent part** is "any part misrepresented to the customer as meeting the customer's requirements."
- A **counterfeit part** is "a fraudulent part that has been confirmed to be a copy, imitation, or substitute that has been represented, identified, or marked as genuine, and/or altered by a source without legal right with intent to mislead, deceive, or defraud."

## 6.1 SAE AS5553A

This section follows the requirements put forth in SAE AS5553A—Fraudulent/Counterfeit Electronic Parts: Avoidance, Detection, Mitigation, and Disposition. It is recommended that the reader have a copy of SAE AS5553A for reference. No figures were duplicated in this document. SAE AS5553A is a not a regulatory standard.

SAE AS5553A contains both hard requirements and recommended practices; therefore, the requirements themselves are very high level. Meeting those requirements requires a careful analysis of the needs of the implementing organization and its customers as well as the application under consideration. Risk assessments on the part of the implementing organization (and possibly, its customers) drive the detailed mitigations applied to meet the requirements of AS5553A.

Each requirement (of the single root requirement) from AS5553A is a subheading in this section. AS5553A defines the following requirements:

1. **Personnel Training**: The implementing organization must train all relevant personnel.

2. **Parts Availability**: The implementing organization must train all relevant personnel as appropriate to their job function, including management.
3. **Purchasing Process**: The implementing organization's processes must
    a. assess potential parts supply sources, documenting the criteria and maintaining records for assessed suppliers.
    b. specifically prefer procurement from original component manufacturers (OCMs) or authorized suppliers based on criteria set.
    c. assure that approved sources are maintaining effective counterfeit and fraudulent parts processes.
    d. include documented risk assessments and mitigation plans for all non-OCM/non-authorized sources.
4. **Purchasing Information**: The implementing organization's processes must specify contractual requirements to minimize the risk of receiving counterfeit or fraudulent parts and the processes must
    a. include traceability to the original source (either OCM or aftermarket manufacturer), which identifies the name and location of all intermediaries in the supply chain between the original source and procurement by the implementing organization. Require a documented risk assessment if traceability document is unavailable or suspected to be falsified.
    b. specify flow-down of requirements from the implementing organization's processes (compliant to SAE AS5553A) to all contractors and sub-contracts. Require a documented risk assessment for any supply chain intermediaries that do not have an AS5553A-compliant process.
    c. require disclosure at the time of quotation of whether the source is authorized for the parts being quoted and whether a full manufacturer's warranty is provided.
5. **Verification of Purchases/Returned Parts**: The implementing organization's processes must
    a. include detection of suspect, fraudulent, and counterfeit parts prior to formal acceptance.
    b. specify inspection as part of the returns process.
6. **In-Process Investigation**: The implementing organization's processes must address detection and control of suspect, fraudulent, and counterfeit parts post acceptance from supplier and in-service.
7. **Failure Analysis**: When a specific part instance is determined to be the cause of a failure, the implementing organization's process must include provisions to document whether the part is a suspect, fraudulent, or counterfeit part.
8. **Material Control**: The implementing organization's processes must specify methods to
    a. control excess and nonconforming parts.
    b. control and quarantine suspect, fraudulent, and counterfeit parts, including enforcing access rules for controlled areas.

9. **Reporting**: The implementing organization's processes must include provisions to report all occurrences of suspect, fraudulent, and counterfeit parts to customers, government, and industry bodies and authorities having jurisdiction.
10. **Postdelivery Support:** The implementing organization's processes must handle resolving nonconforming products or parts delivered to the organization's customers due to suspect, fraudulent, or counterfeit parts including investigation and reporting processes.

### 6.1.1 Personnel training.

The implementing organization must train all relevant personnel as appropriate to their job function, including management.

### 6.1.2 Parts availability.

The implementing organization's counterfeit and fraudulent parts avoidance, detection, mitigation, and disposition processes must maximize availability of authentic, originally designed, or certified parts. The primary takeaway from this requirement is that the implementing organization's processes must include a risk-driven plan for the lifetime of the product or part. Upstream components may have a shorter lifespan than a part or product that uses them. The part availability requirement comes with guidance, but no mitigations are required.

Automotive electronics parts do not have distinct needs regarding the parts availability requirement or guidelines, and AS5553A can be applied directly.

#### *6.1.2.1 Planning and obsolescence management.*

"[E]lectronic equipment manufacturers should proactively manage the life cycle of their products using an Obsolescence Management Plan or Diminishing Manufacturing Sources and Material Shortages (DMSMS) management plan." [AS5553A] Steps to reduce exposure to fraudulent and/or counterfeit parts are bridge buying, system redesigning, using multiple sources, part substitutions, and planning for adequate procurement lead times.

There are many ways to reduce the risk of supply chain disruption. AS5553A is written for aerospace, where vessels have very long lifetimes, even longer than automobiles in most cases. However, automobiles, and commercial vehicles especially, can still have long lifetimes. Vehicles with 50+-year lifetimes are not unheard of, especially performance and collector cars as well as heavy vehicles. The availability of high-quality parts is still critical for vehicles with long lifetimes.

The primary difference between automotive and aerospace is the volume of parts used. The basic mitigations are the same with small adjustments. This requirement and the optional guidance in AS5553A translates to automotive with minimal adjustment.

### 6.1.3 Purchasing process.

Parts should be purchased directly from OCMs or authorized suppliers whenever possible. Independent distributors are not covered by franchise agreements with the OCM. The standard accepts that as fact. Some distributors will be authorized distributors for some OCMs and not others or even potentially some parts and not others. Franchise agreements with OEMs protect the user and customer by ensuring product integrity and supply chain traceability, and are generally required for warranty from the OCM.

#### *6.1.3.1 Risk assessment.*

A counterfeit/fraudulent parts risk is assigned based on either the source (e.g., OCM, certified manufacturer directly, authorized supplier, independent distributor) or the product or part application (e.g., life dependent, mission critical, non-critical). When either indicates a higher risk, the organization should take additional mitigation steps against counterfeit, fraudulent, or sub-standard parts.

Within the document, there is a reasonable flow chart for what due diligence is required at a minimum. At each point, if the application dictates extra stringency, the path can be taken which requires more additional steps and documentation, even if the supplier is low-risk (e.g., the OCM itself or an authorized supplier).

#### *6.1.3.2 Supplier selection.*

Processes related to detection, prevention, mitigation, and disposition of counterfeit and fraudulent parts should begin prior to execution of a purchase contract with a supplier. When choosing a supplier, selection criteria should include historical experience with that supplier, any previously documented problems with the supplier, how long the supplier has been in business, supplier's demonstrated adherence to quality standards (e.g., AS5553A itself), results of audits, the supplier's documented processes, the use of in-house or external lab testing, the use of certified quality inspectors, and the terms of the warranty, return policy, and product liability.

#### *6.1.3.3 Supplier auditing.*

SAE AS5553A Figure B2, "Supplier Assessment Pyramid," shows different risk-mitigation steps. Lower items in the pyramid are more important (because they have larger area). The base of the pyramid is "customer audited and approved with site visit." Yet, in the automotive industry, auditing is most likely not as common as in aerospace.

Audit frequency and scope should be based on the assessed risk of the supplier.

#### *6.1.3.4 Applicability to automotive.*

The purchasing process guidelines in AS5553A are generic enough to cover automotive with minimal modifications. Automotive applications will be lower risk on average, but the specification and guidelines are flexible enough to handle both circumstances.

### 6.1.4 *Purchasing information.*

Purchasing contracts must have certain clauses and/or requirements to minimize the counterfeit and fraudulent parts risk. The following items should be included in procurement contracts:

- Supply chain traceability provided.
- Tests and inspections performed by the customer to assure authenticity and accept/reject criteria established.
- Quality management system required of the supplier.
- Seller's acceptance of financial responsibility, including remedial costs.
- Length of seller's obligation.
- Required documentation that the seller must keep and finish.
- Penalties associated with fraud.

Suppliers should be required to provide a certificate of conformance, a formal statement that the contract terms have been met.

#### *6.1.4.1 Supply chain traceability.*

The implementing organization must require supply traceability of its authorized suppliers, including names and organizations of all intermediaries between the OCM and the customer. If a supplier cannot meet the documentation requirement, a documented risk assessment must be performed and greater inspection and testing likely performed.

For the automotive industry, these items are probably extreme. Creating a process that is AS5553A compliant, however, is possible, because the requirements themselves are general. The implementing organization, whether in automotive or aerospace, must choose correct processes based on risk assessment.

### 6.1.5 Verification of purchased/returned parts.

SAE AS5553A provides guidance in the Appendices, which do not form actual requirements for conformance to the specification. The inspection and/or verification steps taken for purchased and returned parts, like previous steps, must be informed by the risk. Risk is assessed based on the specific supplier, the supplier's relationship with the manufacturer for that part (e.g., is the supplier an authorized supplier for the part in question), and the product criticality. Higher criticality applications dictate greater due diligence.

The following inspection methods are recommended (applied based on the risk assessment).

- Packaging and documentation inspection
- Visual inspection of parts
- Destructive inspection for remarking or resurfacing
- Solvent test for remarking
- Solvent test for resurfacing
- Scanning electron microscope inspection

- Scanning acoustic microscopy inspection
- Destructive or non-destructive X-Ray inspection
- Destructive or nondestructive lead finish evaluation
- Electrical testing
- Burn-in testing
- Thermal cycle testing
- Hermeticity verification (for hermetic parts)
- Destructive decapsulation physical analysis
- Destructive physical analysis

Again, scope and frequency of the verification and inspection methods are informed by the risk assessment performed by the implementing organization as part of its fraudulent and counterfeit parts protection processes. It is quite clear that there are many robust and increasingly insightful inspection procedures for electronic parts. These guidelines should work for the automotive industry without modification, although the same inspection plans might not be applied in practice between the aerospace and automotive industries.

Because the requirement is general and the guidelines flexible, this requirement does not seem to need any modification for automotive use. Again, for many automotive ECUs, less due diligence may be required as compared to avionics, but the specification and guidelines in AS5553A are general enough to handle automotive application.

### 6.1.6 In-process investigation.

The implementing organization's processes must address detection and control of suspect, fraudulent, and counterfeit parts post-acceptance from supplier and in-service.

### 6.1.7 Failure analysis.

When a specific part instance is determined to be the cause of a failure, the implementing organization's process must include provisions to document whether the part is a suspect, fraudulent, or counterfeit part. Failure analysis in automotive is quite different than in aerospace. Ordinarily, vehicular crashes and incidents are not investigated for parts failure. Therefore, the requirement to perform failure analysis and document/report when a part determined to cause a failure is also suspect, fraudulent, or counterfeit is less applicable to the automotive industry.

### 6.1.8 Material control.

The implementing organization's processes must specify methods to control excess and nonconforming parts, and specifically, control, segregate, and quarantine suspect, fraudulent, and counterfeit parts.

Excess or nonconforming product or part may be one of the following.

- **Scrap product**: scrap product should be physically marked and segregated.

- **Surplus product**: surplus product should only be transferred (e.g., sold) to organizations with a robust suspect, fraudulent, and counterfeit products and parts processes. Ideally, those organizations should have demonstrated adherence to high-level quality standards and AS5553A and/or rigorous business, ethical, and quality standards.
- **Return product**: When products or parts are returned, steps should be taken to permit validation of authenticity. Returns should not be accepted without proper return material paperwork.
- **Suspect, fraudulent, or counterfeit product**.

#### *6.1.8.1 Control of suspect, fraudulent, or counterfeit parts.*

When a part is determined to be suspect, fraudulent, or counterfeit, the following actions should be taken.

- The part should be physically marked and segregated.
- Segregated parts should be quarantined with physical barriers and access control.
- Suspect, fraudulent, and counterfeit parts should not be returned to the supplier if doing so would allow the suspect, fraudulent, or counterfeit parts back into the supply chain and not prevent the supplier from performing an investigation.
- Confirm the authenticity of the part (for example, through further testing or research with the part's presumed OCM).
- When a part is confirmed fraudulent or counterfeit, place a hold on all other potential fraudulent or counterfeit parts in storage or installed.
- Report the incident to the appropriate authorities.

#### *6.1.8.2 Applicability to automotive.*

In automotive, parts are not nearly as strictly controlled as in the aerospace and military domains. This is true in the manufacturing environment due to the large volumes of vehicles built every year comparatively. It's doubly true for vehicles in service. It's not completely clear how applicable the martial control actions recommended for aerospace and military in AS5553A are to the automotive industry.

### 6.1.9 Reporting.

SAE AS5553A requires that the implementing organization's processes include steps to report instances of suspect, fraudulent, and counterfeit parts. While industry reporting is a good best practice, automotive applications will not have the same regulatory requirements as aerospace and military applications. These reporting best practices are still sensible for automotive, but the regulatory aspects will not apply.

### 6.1.10 *Postdelivery support.*

The implementing organization's processes must handle resolving nonconforming products or parts delivered to the organization's customers due to suspect, fraudulent, or counterfeit parts, including investigation and reporting processes. As discussed previously in section 6.1.8, Material control, a nonconforming product or part may be one of the following:

- Scrap product
- Return product
- Suspect, fraudulent, or counterfeit product

## 6.2 Applicability to Software Products and Software Components

Software counterfeiting is not out of the realm of possibility. The SAE AS5553A standard relates solely to electronics parts and products. While those parts/products may contain software, the software itself is not accounted for in the standard. It is not out of the realm of possibility for fraudulent software, that is, software that has been misrepresented to the customer, to enter the supply/development chain.

With software, especially, there is very little auditing performed by the OEMs. Software oversight and secure development processes are an area of active research, and more research is needed in the automotive industry.

## 6.3 Conclusion

Fraudulent and counterfeit parts can pose a safety and monetary liability risk. SAE AS5553A is an SAE aerospace standard for the creation of processes for detection, prevention, mitigation, and disposition of suspect, fraudulent, and counterfeit electronic parts.

In general, SAE AS5553A should apply to the automotive industry quite readily. It is designed to be flexible and risk-informed. Because of that, the meat of the specification is best practices for an organization attempting to create AS5553A-compliant fraudulent parts processes. Two differences were seen between the automotive and aviation industries that make the requirements of AS5553A potentially insufficient or a bad match for automotive:

- Automotive parts are distributed through a much broader distribution network than aviation parts and are produced and sold in much higher volumes. This makes detailed traceability and material control much more difficult in automotive.
- In the automotive industry, customers do not have a direct relationship with OEMs or OCMs. Dealerships and automotive service centers may have a greater duty to the customers than the OEM. This makes collection and inspection of suspect parts difficult.

The requirements themselves should be applicable to automotive; however, a more tailored collection of best practices might be reasonable to develop for automotive specifically. Many of the processes recommended for aerospace and military are potentially too burdensome for the automotive industry or difficult to implement logistically given the much greater volume of products and parts in automotive.

While air vessels have extremely long lives, even longer than automobiles on average, some automobiles will have very long lifetimes, especially commercial vehicles. Performance and collector vehicles may remain in the field for 50+ years. The availability of quality parts for long-lifetime vehicles is just as critical for automotive and heavy vehicles with long lifetimes.

Interestingly, AS5553A places customer audits of suppliers and site visits to be the greatest risk-reducing mitigation when assessing suppliers. The automotive industry OEMs likely do not perform regular audits of that depth. For software, neither AS5553A nor the standard automotive industry practices today provide auditing against suspect, fraudulent, or counterfeit software parts.

In addition, the traceability management and parts accounting used in aerospace might be a bad fit for automotive warranting further research. Automotive products and, therefore, parts are much higher volume than aerospace; therefore, this system seems to be too arduous for the automotive environment. There may be value to further identification of the unique attributes of the automotive industry regarding counterfeit parts detection and prevention.

# 7. Conclusion

Secure in-field software updates are nearly universally considered to be essential for any networked computer system. However, software update functionality creates a new attack surface for attackers to potentially exploit. The installation of malware is one of the biggest risks in computing systems, both because of the vast number of ways that a system can be compromised for installing malware and because malware is very useful, particularly in small embedded systems which frequently lack an operating system running at a higher trust level than the application code.

***Summary of Lessons Learned in Adjacent Industry:***

Common existing defense mechanisms (e.g., Signing, Fortification, and Intrusion Detection) and vulnerabilities are noted in the body of the report as are potential defenses for secure vehicle firmware updates.

**Risk Assessment Conclusions:**

In identifying risks at both the vehicle-level and the technological design and implementation level, the researchers have identified the biggest risk with software update mechanisms as malware installation.

**Mitigation Methods Conclusions:**

In-field software updates are a necessity in the automotive industry to fix flaws without replacing hardware that is already deployed in the field. The current generation of automobiles primarily uses OTA software updates for telematics and infotainment ECUs only.

While software updates are a boon for security, the mechanism, particularly the remote mechanism, creates a new avenue for attackers to exploit.

A matrix of specific mitigations versus risks appears in the report (see Table 17).

**Intellectual Property Theft Risks and Mitigations Conclusions:**

Intellectual property theft, particularly software theft, can be enabled and made easier with software update mechanisms, particularly OTA mechanisms. In discussions with the OEM and tier-1 supplier employees, the majority opinion is that protecting the software binaries is not a priority. The prevailing opinion in the industry is that there are too many other ways for an adversary to obtain a software binary to justify the cost of adding encryption to the software update process.

**Counterfeit and Fraudulent Electronic Parts and Products Conclusions:**

Fraudulent and counterfeit parts can pose a safety and monetary liability risk. SAE AS5553A is an aerospace standard for the creation of processes for detection, prevention, mitigation, and disposition of suspect, fraudulent, and counterfeit electronic parts. In general, SAE AS5553A should apply to the automotive industry quite readily. It is designed to be flexible and risk-informed. The requirements themselves should be applicable to the automotive industry; however, a more tailored collection of best practices might be reasonable to develop for the automotive sector specifically (not developed within this project).

There is no singular, perfect reference model for securing software updates. Every application has different requirements and user experience targets that shape the design enough to require security to be, at minimum, analyzed and usually designed with an application-specific approach. This work presents a literature and technology survey of software update procedures in related industries, a risk assessment of firmware updates in modern and near-term future automobiles, and presents mitigations for those risks.

Because there are not standard meanings (or loose standard meanings) for many of the words used throughout this document, there is the possibility of miscommunication between the readers and authors. In addition, while the authors strive to use the generic but contextually specific words (such as risk, threat, or vulnerability) in the most common, correct, and reasonable way, it's possible that they deviate from the reader's expected meaning. To reduce the risk of miscommunication, definitions to specific terms are presented in this section. When the following words appear in this document, the writers intend the definitions found in this section.

# 8. Glossary

| | |
|---|---|
| Abuse case | An **abuse case** is a use case from the perspective of a would-be attacker. |
| Advanced Encryption Standard (AES) | The **Advanced Encryption Standard (AES)** (originally named Rijndael) is the de facto standard for symmetric key encryption, established by an open competition held by the U.S. National Institute of Standards and Technology (NIST). Symmetric cryptography includes methods based on private keys shared between parties, and symmetric-key algorithms like message authentication codes (MACs) are built with symmetric-key ciphers such as **AES**. |
| Aftermarket manufacturer | An **aftermarket manufacturer** of an electronic part or process is one that is not the original component manufacturer (OCM). |
| Approved supplier | An **approved supplier** is one who has been approved by the SAE AS5553A implementing organization as having acceptable fraudulent and counterfeit parts mitigation processes. |
| Asymmetric cryptography | **Asymmetric cryptography** is the use of cryptographic algorithms where a single party has a private key that is not shared and any number of other parties can have the associated public key without compromising the security of the private kay. Generally, for authentication, the private key is used to sign and public keys to verify; for encryption, public keys can encrypt data which can only be decrypted with the private key. RSA and elliptic curve cryptography (ECC) are examples of asymmetric cryptosystems. |
| Attack | An **attack** is some unintended and usually undesirable control of a system by an unauthorized party, the attacker. |
| Attacker / adversary | An **attacker** or **adversary** is an actor (frequently theoretical) who might perform some manipulation of a system that is not originally intended and could lead to a security compromise. |
| Authorized distributor | An **authorized distributor** of electronic products or parts is a distributor in a franchise contract with the OCM. |
| Backdoor | A **backdoor** is undocumented functionality in a production system. It may or may not include circumvention of the system's security mechanisms. |

| | |
|---|---|
| Baseline reference architecture | The **baseline reference architecture** is the reference design the team assume for how software updates would be implemented in a current or near-term, high connectivity automobile and is used as the basis for the risk assessment and mitigations proposed. The team's reference architecture intends to make the minimal set of assumptions about the design of a software update mechanism to give the most useful risk analysis possible. |
| Bookkeeping database | The **bookkeeping database** is the IT infrastructure that stores the vast configuration and status data for vehicles in the field. It may be part of the software distribution network or separate in practice. |
| Bootloader | The **bootloader** is a trusted piece of software in an ECU (or other electronics device) that performs booting and loading the application software (or operating system in more complex systems). A **bootloader** may perform cryptographic verification prior to booting, and generally should perform a cryptographic authentication verification prior to loading new software. |
| Bricking/bricked | An electronic part or product is **bricked** when it suffers permanent or semi-permanent loss of functionality. Historically, **bricking** of an electronics part referred to putting it in an unrecoverable state, but due to the nuance of different recovery methods, particularly as they change during different lifecycle states, such as in development or in production, a **bricked** part may retain some path to recovery that is prohibitively difficult or costly. This technical jargon term comes from the fact that the part becomes as useful as a **brick**. |
| Bridge buy | A **bridge buy**, also called a lifetime buy, is a purchase of a part all at once for the full lifetime of the product into which it will be integrated. Bridge buying is a mitigation against upstream parts obsolescence. |
| Compromise | A **compromise** is a realized risk, whereby an adversary has breachedbreachedbreachedbreachedbreachedbreachedbreachedbreached a system leading to loss of security control in some way. |
| Counterfeit part | Per SAE AS5553A, a **counterfeit part** is a fraudulent part that has been confirmed to be a counterfeit, that is, a clone knowingly misrepresented and sold as genuine. |
| Credential | A **credential** is a secret that is used to provide authentication of the party who knows or has that secret to another party, who may then verify the authentication. An RSA private key, for example, is a **credential**. |
| Denial-of-service | **Denial-of-service attacks** are a class of attacks where an adversary prevents normal activity, totally or partially, permanently or temporarily. |

| | |
|---|---|
| Diff update | A **diff update** package (or differential update) is a software update package that does not contain all the necessary application data, but rather contains the data required to compose the new software version with the current version on the target ECU and the **diff update** package. A diff update is created by programmatically comparing (or "diffing") the installed software version and the new software version. |
| Digital Rights Management | **Digital rights management (DRM)** is a class of cryptographic schemes for controlling the licensed use of content, generally by a customer. |
| Digital signature | A **digital signature**, or just signature, is a piece of cryptographic metadata providing a guarantee that an authentic party issued or attested the data. **Digital signatures** are one of the most basic controls for secure software updates, but may also be used for exchanging metadata or remote commands securely. A **digital signature** is created by a party which has a secret credential and may be verified by a receiving party. |
| Electronic Control Unit | An **electronic control unit (ECU)** is any electrical controller within a vehicle. An ECU is assumed to have a computer processor (e.g. a microcontroller) at its heart and be largely software-controlled. |
| Elliptic Curve Cryptography | **Elliptic curve cryptography (ECC)** is a class of asymmetric cryptography algorithms that rely on the presumed (but unproven) hardness of finding the discrete logarithm of a random elliptic curve element based on a publicly known base point. It is an alternative to RSA, and the two are presumed to be similarly secure and interchangeable for most functional purposes, although **ECC** achieves comparable strength with shorter key lengths. |
| Exploit | An **exploit** is an exploitation of a vulnerability, part of a successful attack and/or compromise. |
| Firmware | **Firmware** is software that runs from normally immutable memory, such as read-only memory (ROM). Firmware does not need to run from ROM, and in the case of software updates, it is usually necessary for some firmware to run from RAM while ROM is being re-programmed. In this document, the authors use the terms firmware and software interchangeably. |
| Foothold/beachhead | A **foothold** or **beachhead** is an initial (and possibly persistent) malware installation in software systems, through which further exploitation of the compromise may be executed. |
| Fraudulent part | Per SAE AS5553A, a **fraudulent part** is any suspect part misrepresented to the customer as meeting the customer's requirements. |
| Freshness | **Freshness** is the property of a message that uses cryptographic authentication and replay protection that guarantees that the message is recent, in addition to not having been used twice. |

| | |
|---|---|
| Hardware Security Module | A **hardware security module (HSM)** is a peripheral for the highly secure storage of cryptographic keys. **HSMs** are traditionally associated with server-class devices found in secure datacenters. An embedded **HSM** is a microcontroller peripheral/secondary core for performing high-security tasks, especially storing keys. Embedded **HSMs** are resilient against temporary software intrusions for keeping keys private. |
| Independent distributors | Per SAE AS5553A, **independent distributors** are electronics parts distributors that are not in franchise agreements with the OCM. |
| Intrusion Detection System | An **intrusion detection system (IDS)** is an electronics device or application that monitors network communication (or system activities) for malicious or anomalous patterns, which are logged and reported to some authority. An **IDS** may be distinguished from an intrusion detection and protection system (IPS) which actively seeks to prevent malicious activity, although this distinction is not always made. |
| Keylogger | A **key logger** is software (often installed on a system without the users' knowledge or consent) which logs the users' most basic actions, such as key strokes on a keyboard, and saves for or sends to a third party. |
| Man-in-the-middle attack / man-on-the-side attack | A **man-in-the-middle attack** may be successfully launched in a variety of ways and allows the attacker the privilege of masquerading as one side of a communication to the other (in either direction). A **man-on-the-side attack** allows an attacker to somehow listen in on an otherwise private communication between two other parties. Colloquially, both types of attacks may be called **man-in-the-middle attacks**. Throughout this document the writers refer to **man-on-the-side** attacks as **man-in-the-middle attacks** generically. |
| Malware/rogue software | **Malware** is any software created to harm or interfere with the intended operation of a user's computer. Sometimes malware is differentiated from software that is unintentionally harmful (such as well-meaning software which opens security holes, for example), called *badware***.** In this document, the authors use the terms **malware** and **rogue software** interchangeably. |
| Message Authentication Code | A **message authentication code (MAC)** is a cryptographic scheme for guaranteeing the authenticity and integrity of a piece of data using symmetric cryptography, such as the Advanced Encryption Standard (AES). In practice, it can be an alternative authentication mechanism to a digital signature. |
| NIST | The United States **National Institute of Standards and Technology** is a non-regulatory division of the United States Chamber of Commerce. **NIST** publishes standards and best practices for securing electronic systems, among other things. |

| | |
|---|---|
| Nonce | A **nonce**, a portmanteau of "number used once," is a number which is guaranteed not to be repeated for a very long time (possibly a probabilistic guarantee). Combined with a cryptographic authentication mechanism like a digital signature or MAC, it is used to prevent replay attacks and possibly guarantee freshness. |
| On-Board Diagnostics | **On-board diagnostics (OBD)** is used to mean several things: the regulated diagnostics that must be supported for electronic emissions reporting, proprietary diagnostics used by engineers and service technicians accessible via the same port as the regulatory services, and the actual physical port itself, the **OBD** port. |
| Original Component Manufacturer | An **OCM** is an entity that designs or engineers a part, owning the intellectual property rights to that part. |
| Over-the-Air | An **OTA** software update is one that is delivered over the Internet or short-range wireless without a physical connection to the vehicle or device. |
| PKCS | **PKCS**, or the public key cryptography standards, are a suite of usage and best practices standards published by RSA Security, Inc. The **PKCS** standards are de facto standards for many asymmetric cryptography techniques, including digital signature generation and verification |
| Ransomware | **Ransomware** is malware that prevents a rightful owner or user of a computer system from using the system until a ransom is paid. A **ransomware** attack is usually a reversible denial-of-service attack, but in some cases, **ransomware** may threaten or cause permanent damage to the system if the ransom is not paid. |
| Read-Only Memory | **ROM** is memory in a computer system, such as an ECU, which retains its contents even when the memory is not powered. ROM is typically read-only during normal operation, but most ROM used is automotive systems today is flash-based, which can be reprogrammed in-service. The bootloader is the piece of software which reprograms the ROM of an ECU during in-service software updates. |
| Replay attack | A **replay attack** is an attack where an adversary replays a legitimate message later for a purpose not intended by the legitimate sender. |
| Risk | A **risk** is a theoretical bad outcome. The security **risks** in this document are possible compromises or attacks, which could be executed by a threat actor. |
| Root-of-Trust-for-Update | The **root-of-trust-for-update** is a protected section of an electronics system containing the bootloader, cryptographic algorithms and cryptographic keys. It was proposed for BIOS software updates by NIST. [NIST11] |

| | |
|---|---|
| RSA | **RSA**, an acronym for the initials of its creators, Rivest, Shamir, and Adleman, is a class of asymmetric cryptography algorithms that rely on the presumed (but unproven) hardness of factoring the product of two large prime numbers. It is an alternative to ECC, and the two are presumed to be similarly secure and interchangeable for most functional purposes, although ECC achieves comparable strength with shorter key lengths. |
| Secondary bootloader | The **secondary bootloader** is a file or bundle of files, which is usually downloaded to the target ECU prior to performing the software update process. The secondary bootloader might contain arbitrary code for the target ECU to run. The secondary bootloader often contains routines to interact with (i.e., read, write and erase) the flash memory on the target ECU. |
| Secure Hardware Extension | The **Secure Hardware Extension (SHE)** is a peripheral capable of storing symmetric keys and performing symmetric cryptography. It is also known as the HSM Light. [EVITA10] |
| Stakeholder impact | The **stakeholder impact** of an attack is a categorization of the potential for loss to stakeholders, including safety, privacy, or financial. |
| Software distribution network | The **software distribution network** is the interface between the vehicle and all other entities it connects to for software updates. For OTA updates, the vehicle and its ECUs must communicate directly with the **software distribution network**, whereas with traditional OBD-attached updates, mechanics may receive software update packages through an internet portal or even digital media sent through the mail. In this document, it is assumed the **software distribution network** connects to some bookkeeping database for tracking installed versions in the field. |
| Software repository | The **software repository** is the IT infrastructure that stores, receives and transports the software update packages themselves. |
| Software update package | A **software update package**, or just package, is a piece or multiple pieces of data (usually, a file or files) which, when downloaded and applied to an ECU, allow the ECU to update to a different software version. |
| Spoofing/masquerading | **Spoofing** is when a nonauthentic party sends communications claiming to be an authentic party. Spoofing is also called **masquerading** as the non-authentic party is **masquerading** as someone else (the authentic party). Although outside of this document, these terms may have nuanced differences in meaning, **spoofing** and **masquerading** are used interchangeably in this document. |
| Suspect part | Per SAE AS5553A, a **suspect part** is any part for which there is some indication that it might be fraudulent or counterfeit. |

| | |
|---|---|
| Symmetric cryptography | **Symmetric cryptography** is a class of cryptographic algorithms that use symmetric ciphers, such as the AES, which perform encryption and decryption using a single private key shared between all communicating parties. |
| Target ECU | The **target ECU** is the ECU that is to receive a software update. This is not a standard term outside of this document. |
| Threat actor | A **threat actor** is a theoretical party who has motive to carry out an attack (a potential attacker). |
| Transport-Layer Security | **TLS** is a collection of cryptographic protocols that provide the ability to create a secure channel between two parties over which to communicate. **TLS** is the successor of the SSL, and, today, the terms **TLS** and SSL are generally both used interchangeably to mean **TLS**. |
| Update authentication | **Update authentication** is message authentication applied to the software update package itself, used to allow the receiving ECU to authenticate the software update package prior to installation or first run of the new software or both. |
| Vehicle automation level | NHTSA has assigned different **vehicle automation levels** to vehicles with different capabilities [NHTSA13] where level 0 is no autonomy (e.g., most vehicles produced more than 15 years ago) and level 4 is full autonomy (i.e., no driver interaction necessary whatsoever). |
| Vulnerability | A **vulnerability** is a weakness or flaw in design or implementation of a product (in this document, a computer product) which could allow an adversary to carry out an attack. |
| Wireless Wide Area Network | A **WWAN** interface is a wireless network interface, such as 3G cellular data or Wi-Fi, that provides internet access. |
| Zero-day | A **zero-day** vulnerability is a software vulnerability that is known to third parties before it is known to the primary stakeholder. Zero-day vulnerabilities are often discovered in the wild being exploited by an adversary, rather than during official product testing or security evaluation. |

# References


[App15] Apple Inc., *iOS Security*, 2015.

[AR665] Aeronautical Radio Incorporated (Rockwell Collins), *ARINC 665-3: Loadable Software Standard*.

[AR666] Aeronautical Radio Incorporated (Rockwell Collins), *ARINC 666: Electronic Distribution of Software*.

[AR667] Aeronautical Radio Incorporated (Rockwell Collins), *ARINC 667-1: Guidance for the Management of Field Loadable Software.*

[AR827] Aeronautical Radio Incorporated (Rockwell Collins), *ARINC 827: Electronic Distribution of Software by Crate (EDS Crate)*.

[AR835] Aeronautical Radio Incorporated (Rockwell Collins), *ARINC 835-1: Guidance for Security of Loadable Software Parts Using Digital Siguatures*.

[AS5553A] SAE International, *AS5553A: Fraudulent/Counterfeit Electronic Parts; Avoidance, Detection, Mitigation, and Disposition*. SAE International, AS5553A, January 21, 2013.

[BBF06] A. Bellissimo, J. Burgess, & K. Fu, "Secure Software Updates: Disappointments and New Challenges," in *Proceedings of USENIX Hot Topics in Security (HotSec)*. USENIX, July 2006.

[Bri12] A. Brisbourne, *Tesla's Over-the-Air Fix: Best Example Yet of the Internet of Things*???? Wired, February 2014.

[CCS13] A. Cui, M. Costello, and S. J. Stolfo, "When firmware modifications attack: A case study of embedded exploitation," in *Proceedings of 20th Annual Network & Distributed System Security Symposium (NDSS'13)*. NDSS'13.

[CERT 14] US-CERT (2014): *CryptoLocker Ransomware Infections*. www.us-cert.gov/ncas/alerts/TA13-309A.

[Che11] Checkoway S.; McCoy D.; Kantor B. et al., "Comprehensive experimental analyses of automotive attack surfaces," in *Proceedings of 20th USENIX conference on Security (SEC'11)*. USENIX, 2011.

[Cok11] Coker G., Guttman J., Loscocco P., et al, "Principles of Remote Attestation," in *International Journal of Information Security*, 10, 2, 63-81.

[Dal10] Dalton A. (2016): *Hospital paid 17K ransom to hackers of its computer network*. Associated Press. Retrieved from http://bigstory.ap.org/article/d89e63ffea8b46d98583bfe06cf2c5af/hospital-paid-17k-ransom-hackers-its-computer-network.

[DiB16] DiBlasio N. and Weise E. (2016): *Here's why the FBI forcing Apple to break into an iPhone is a big deal*. USA Today. www.usatoday.com/story/tech/2016/02/16/heres-why-fbi-forcing-apple-break-into-iphone-big-deal/80481766/.

[EVITA10] Apvrille L.; El Khayari R.; Henniger O. et al., *Secure Automotive On-Board Electronics Network*. E-safety vehicle intrusion protected applications (EVITA), 2010.

[FIPS186] National Institute of Standards and Technology (2013): *Federal Information Processing Standards Publication 186-4: Digital Signature Standard (DSS)*., FIPS 186-4. NIST, 2013.

[Fos15] Foster I.; Prudhomme A.; Koscher K. et al. (2015): Fast and Vulnerable: A Story of Telematic Failures. In: *9th USENIX Workshop on Offensive Technologies (WOOT '15)*. Washington, D.C.: USENIX, 2015.

[Hen12] Heninger N.; Durumeric Z.; Wustrow E. et al., Mining Your Ps and Qs: Detection ofWidespread Weak Keys in Network Devices. In: *21st USENIX Security Symposium*. USENIX, 2012.

[HIS09] Escherich R.; Ledendecker I.; Schmal C. et al., *SHE – Secure Hardware Extension Functional Specification Version 1.1*. Hersteller-Initiative Software (HIS) AK Security, 2009.

[HRMP11] S. Hanna, R. Rolles, A. Molina-Markham, P. Poosankam, K. Fu, and D. Song, "Take Two Software Updates and See Me in the Morning: The Case for Software Security Evaluations of Medical Devices," in *Proceedings of the 2nd USENIX conference on Health security and privacy*. USENIX Association, 2011.

[ISO11898] International Organization for Standardization, *ISO 11898:2015 Road vehicles – Controller area network (CAN) – Part 1: Data link layer and physical signaling*, ISO, December 14, 2015.

[ISO14229] International Organization for Standardization, *ISO 14229:2013: Road Vehicles – Unified Diagnostic Services (UDS)*. ISO, March 15, 2013.

[ISO14230] International Organization for Standardization, *ISO 14230:2063: Road Vehicles – Diagnostic communication over K-Line (DoK-Line)*. ISO, August 15, 2016.

[J1850] SAE International, *J1850 Class B Data Communications Network Interface.* SAE International, 2015.

[J2534] SAE International, *J2534/1 Recommended Practice for Pass-Thru Vehicle Programming*. SAE International, 2004.

[J3061] SAE International, *J3061 Cybersecurity Guidebook for Cyber-Physical Vehicle Systems*. SAE International, 2016.

[J3101] SAE International, *SAE J3101 [Work in Progress]*. http://standards.sae.org/wip/j3101/. SAE International, 2016.

[LOC14] U.S. Copyright Office, Library of Congress, *Exemption to Prohibition on Circumvention of Copyright Protection Systems for Access Control Technologies*. Library of Congress, October 28, 2015.

[Mas15] Masunaga S., *Researchers hack a Tesla Model S, bring car to stop*. Los Angeles Times, 2015. Retrieved from www.latimes.com/business/la-fi-hy-tesla-hack-20150806-story.html.

[Mat15] Matthews, L., *Tesla P85D software update reduces the car's already insane 0-60 time*. Geek.com, 2015. Retrieved from www.geek.com/apps/tesla-p85d-software-update-reduces-the-cars-already-insane-0-60-time-1614741/.

[M11] C. Miller, "Battery Firmware Hacking," in *Black Hat USA*. UBM, 2011.

[MV15] Miller C. and Valasek C., Remote Exploitation of an Unaltered Passenger Vehicle. In: *Black Hat*. UBM, 2015.

[NHTSA13] National Highway Traffic Safety Administration, *Preliminary Statement of Policy Concerning Automated Vehicles*. (Press Release) Washington, D.C., NHTSA 14-13, 2013.

[NIST11] Cooper D.; Polk W.; Regenscheid A. et al., *BIOS Protection Guidelines*. National Institute of Standards and Technology (NIST), Special Publication 800-147, 2011.

[PKCS1] RSA Laboratories, *Public-Key Cryptography Standard PKCS #1 v2.2: RSA Cryptography Standard*. EMC Corporation, PKCS #1 v2.2. RSA Laboratories, 2012.

[RM15] Rogers M. and Mahaffey K.,: *How to Hack a Tesla Model S*. DEF CON Communications, Inc., 2015.

[Sch15] Schneier B., *NSA Plans for a Post-Quantum World*. Schenier on Security, 2015. Retrieved from www.schneier.com/blog/archives/2015/08/nsa_plans_for_a.html.

[Sch16] Schneier B., *Decrypting an iPhone for the FBI*. Schenier on Security, 2016. Retreived from www.schneier.com/blog/archives/2016/02/decrypting_an_i.html.

[SKHR15] Steger, M.; Karner, M.; Hillebrand, J., et. al., *Applicability of IEEE 802.11s for Automotive Wireless Software Updates*, 2015.

[Tesla16] Tesla Motors: *Support: Software updates*. Retrieved from www.teslamotors.com/en_GB/support/software-updates.

[U12] Unisys Corporation, *Dynamic firmware updating system for use in translated computing environments,* U.S. patent US8972964, filed July 26, 2012.

[Vau15] Vaughan-Nichols, S., *No reboot patching comes to Linux 4.0*. ZDNET, March 3, 2015.

# Document History

| Document Title | | | | Cybersecurity of Firmware Updates | |
|---|---|---|---|---|---|
| Document Status: | | **Report** | Version: | | **1.0** |
| | | | | | |
| Revision Number: | Revision Author: | Date: | Company: | Changes: | Review Author: |
| 0.1 | Russ Bielawski | 9/1/2016 | UMTRI | Initial draft. | |
| 1.0 | Russ Bielawski | 9/28/2016 | UMTRI | Final report. | Di Ma |

DOT HS 812 807
October 2020

U.S. Department
of Transportation

**National Highway
Traffic Safety
Administration**


14455-101420-v4