2022 NDIA MICHIGAN CHAPTER
GROUND VEHICLE SYSTEMS ENGINEERING
AND TECHNOLOGY SYMPOSIUM
CYBERSECURITY OF GROUND SYSTEMS TECHNICAL SESSION
AUGUST 16-18, 2022 - NOVI, MICHIGAN

Secure Software Updates for Robotic and Autonomous Systems

Cameron Mott[1], Dariusz Mikulski[2], Sabrina Pereira[1]

[1]Intelligent Systems Division, Southwest Research Institute, San Antonio, TX
[2]US Army DEVCOM Ground Vehicle Systems Center, Warren, MI

ABSTRACT

Software updates provide critical new functionality and security improvements to commercial and military vehicles. Organizations across the Department of Defense (DoD) are recognizing that the hardened cybersecurity in robotic and autonomous system (RAS) is essential. A secure software update capability will be added to RAS, providing a peer reviewed security by design solution for securing software updates.

Citation: C. Mott, D. Mikulski, S. Pereira, "Secure Software Updates for Robotic and Autonomous Systems," In *Proceedings of the Ground Vehicle Systems Engineering and Technology Symposium* (GVSETS), NDIA, Novi, MI, Aug. 16-18, 2022.

1. INTRODUCTION

Organizations across the Department of Defense (DoD) are recognizing that the hardened cybersecurity in robotic and autonomous systems (RAS) is essential. After all, without cybersecurity, we cannot create a technical capability – but rather, we create a technical liability. And this is unacceptable for the world's premier fighting force.

As such, the Cybersecurity for Robotic & Autonomous Systems Hardening (CRASH) Joint Capabilities Technology Demonstration (JCTD) was created to be a joint effort to develop a comprehensive cybersecurity software solution tailored for RAS that can be used throughout the RAS lifecycle. The program goal is to make RAS more resilient to existing and emerging threats and to provide deep cyber defenses at various RAS touch points. And as part of this three-year effort, Southwest Research Institute (SwRI) is working with Ground Vehicle Systems Center (GVSC) to develop a secure software update capability that is secure by design, platform-agnostic and based on the Uptane standard.

2. Background

Software updates provide critical new functionality and security improvements to commercial and military vehicles. These updates are typically performed by one or more people that are physically near the vehicle and log into the computers as a

system administrator with all of the capabilities (and responsibilities) to modify anything on the system. An update may include new code from other developers, updated supporting libraries, or system configuration changes. For some scenarios, the entire system is rebuilt and deployed. The security concerns for a manual update process like this include single authentication (only one deployer), single factor authentication (something you know [a password] is the only requirement), lack of transparency (no one else is aware of what changed), and escalated privileges (deployer is an administrator and can change anything).

The update process has been leveraged as an attack vector for adversaries, who are exfiltrating PII, financial data, digitally breaking into the vehicles and even planting ransomware. The loss of a U.S. military asset is a threat to the lives of our soldiers, provides advanced technology to our enemies, as well as having the potential of being used to impersonate a trusted ally. The downtime and maintenance costs are additional non-monetary concerns for an insecure update process. A secure software update process both enables remote update capabilities as well as protecting them from external manipulation.

Securing software updates is achieved through the peer-reviewed security practices standardized in the update system called Uptane. Over the past six years, security professionals from the embedded and automotive domain have been dedicating their expertise to creating an open and peer-reviewed cybersecurity framework. This framework protects software updates that are delivered to the computers running in today's commercial automobiles. The framework can withstand attacks from malicious actors who attempt to compromise in-vehicle computers, servers, and networks used to sign and deliver updates. The security enhancements are achieved through the utilization of robust key management, layers of security protections, and inter-locking security features.

3. Securing Updates

RAS frequently need to receive software updates to support the ever-changing demands of various missions. Current update mechanisms replace the existing software with an updated version through a direct connection to the vehicle. This introduces a threat vector since the update itself or the vehicle's computers could be manipulated into accepting a malicious payload. RAS are frequently operating in hostile environments, with the threat of a pre-existing adversary. The Uptane security framework was designed to protect software updates even in the presence of a persistent attacker with a man-in-the-middle (MITM) exploit and is perfectly suited to addressing the risks and constraints of updating RAS while domestic or deployed.

3.1. Security Features

The structure of Uptane provides hierarchical key management security, layers of security protection, and inter-locking security features. A hierarchical key management structure delegates responsibilities and capabilities between multiple keys, all tied back to a root chain of trust (see Figure 1). All keys are managed

Figure 1: Key hierarchy with a root chain of trust

throughout their entire lifecycle, with the ability to revoke and redistribute keys while under threat of a compromise. Computers within the vehicle coordinate to verify any update based on their capabilities. Multiple servers with different roles provide redundancies that allow for security verifications to provide the ability to recognize not just the existence of an active adversary, but where they infiltrated the system.

The update security is focused on the servers and their communications with the vehicle's electronic control units (ECUs). There are three servers involved in the layered security features. An image server contains the latest images for install, a director server determines which images correspond to each vehicle ECU, and a time server can be used to provide the ECUs with a source of secure, reliable time. Each ECU is an Uptane client that must independently download and verify the data from each server before the update is installed. A visual representation of the security features is provided in Figure 2.

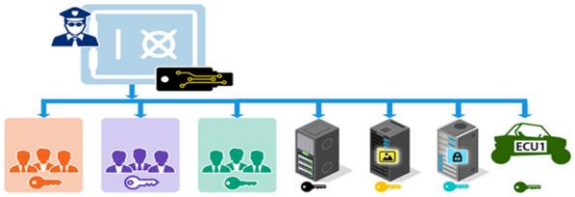

Figure 2: Key hierarchy and layers of security protecting the connections from servers to a vehicle

3.2. System Design

The system architecture was developed in coordination with the program participants and draws from the capabilities offered by each performer. A high-level system design is presented in Figure 3.

Each server functions within a public key infrastructure and additionally stores, generates, and sends metadata about the updates that enables each Uptane client to perform update validation and identify possible attacks. The Image server contains signed metadata about the general images, while the Director server signs metadata for the images on a per-vehicle basis. When an update is performed, the vehicle's main ECU downloads, decrypts and verifies both sets of metadata against each other, and finally verifies that the target image's hash matches what is specified in the metadata. If all checks pass, the image sent to its target ECU, where a similar series of checks are performed on the metadata and image before install. If any verification check fails, a code indicating the detected security attack is logged. The secure update architecture to support these capabilities is provided in Figure 4.

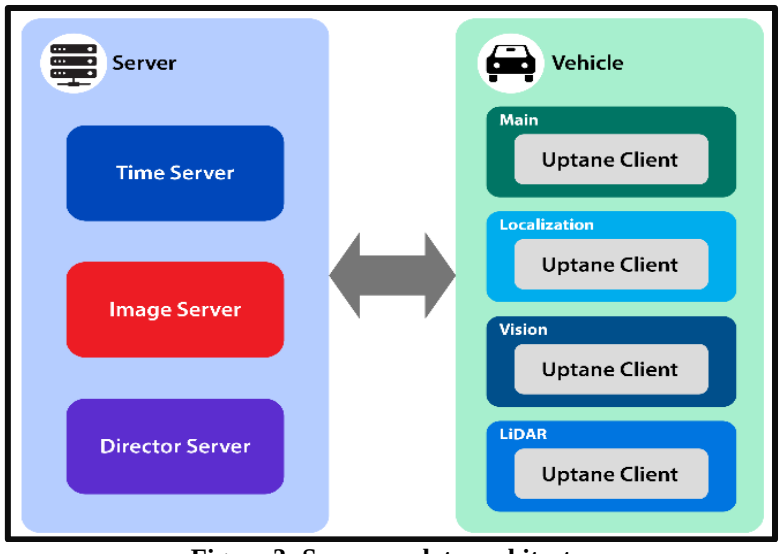

Figure 3: Secure update architecture

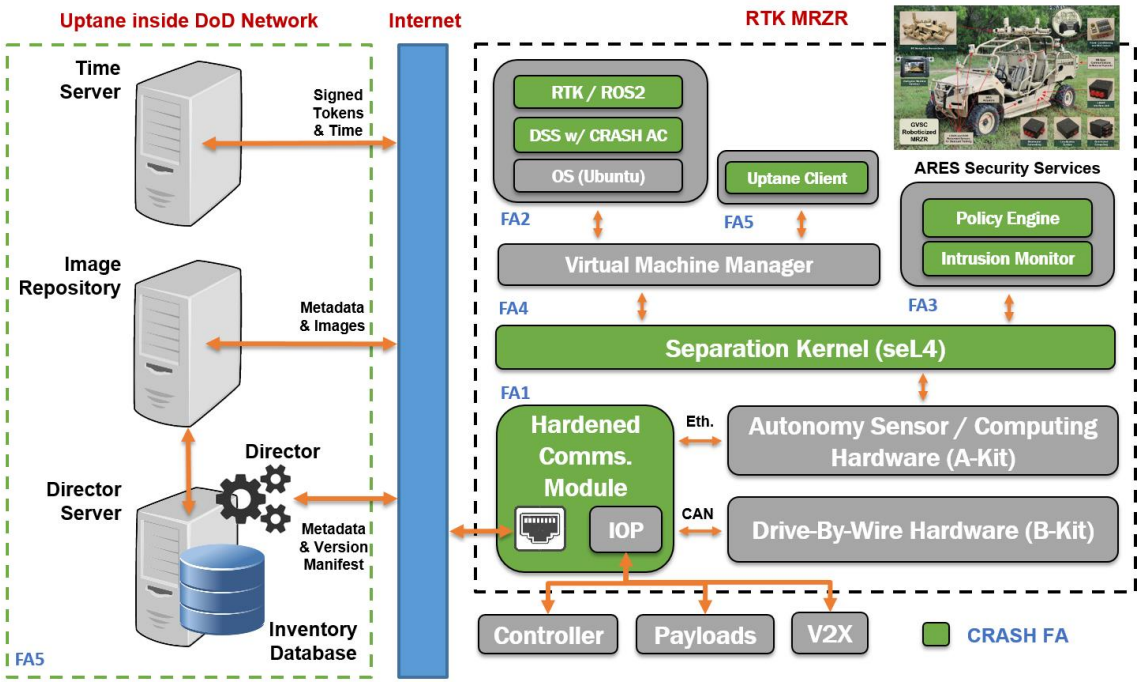

Figure 4. RAS system structure

3.3. Threat Model

The threat model was created based on a cyber tabletop (CTT) exercise that incorporated all of the program participants. Based on this CTT, the vulnerabilities that were related to the update functionality were vetted. Attacks against the update functionality are presented in Table 1, along with the indication of whether the attack is demonstrated during year 1.

Table 1: Update Attacks

Attack Name	Year 1 Demo
Eavesdropping	✓
Denial of Service	
Partial Installation	
Infinite Data	✓
Incorrect Installation	✓
Rollback	✓
Mixed Bundle	✓
Arbitrary Edit	✓
Injection	✓
Insider Attack	

3.3.1 Eavesdropping Attack

This attack is successful if an attacker can extract information from updates sent from the repository to an ECU. Protections against this attack include encrypting the images, encrypting the communication link between the client and the server, and revoking and replacing compromised encryption keys. In order for this attack to be successfully performed, either an inside actor that is able to prevent image encryption or a vulnerability with the ECU encryption keys must be present along with a MITM.

3.3.2 Denial of Service Attack

This attack is successful if an attacker can block network traffic and prevent an ECU from receiving updates. Protections against this attack include active recognition of network traffic conditions and refusal of any updates until a determined period after recovery of denial-of-service conditions. For this attack to be attempted, there must be a MITM interfering with the communication.

3.3.3 Partial Installation Attack

This attack is successful if an attacker can allow only part of an update to install by dropping selected traffic. Protections for this type of attack include active recognition of network traffic conditions, analysis of metadata for an update, and refusal of any updates until a determined period after recovery from malicious network activity. For this attack to be attempted, there must be a MITM with the ability to interfere with transmissions.

3.3.4 Infinite Data Attack

This attack is successful if an attacker can force an ECU to store a large enough amount of data that the ECU runs out of storage. Protections against this type of attack include limiting the amount of data that is downloaded in an update based on the metadata and monitoring the available storage space. For this attack to be attempted, there must be a MITM with transmit and spoofing capabilities.

3.3.5 Incorrect Installation Attack

This attack is successful if a legitimate vehicle can access an update that is not intended for it. Protections for this type of attack include implementing unique security keys for each vehicle or ECU and having unique designated installations that require those keys. For this attack to be successfully performed, there must be a MITM with transmitting capabilities and inside actor/s with access to multiple keys on separate systems.

3.3.6 Rollback Attack

This attack is successful if an attacker can send out a previously deployed update to an ECU when a newer update exists. Protections against this type of attack include implementing monotonically increasing version numbers (with exception of a base-level functionality built into the system) and rejection of any transmissions with duplicate metadata. There are currently two existing implementations of this attack, and both currently assume no key compromise. The first is a replay attack where old Director Timestamp metadata is saved and then replayed after sending an additional update. If successful, a replay of metadata could result in a rollback of installed software. In the second version of the attack, all the responses from the servers over the course of a valid update are recorded, and upon a later update, a tool is used to replay the responses for corresponding new requests. For this attack to be successfully performed, there must be a MITM with transmit and spoofing capabilities, and inside actors with access to the director key and multiple offline keys.

3.3.7 Mixed Bundle Attack

This attack can force an ECU to install incompatible versions of software updates that must not be installed at the same time. Protections for this type of attack include signing the metadata for each version of an update, and peer reviewing updates. For this attack to be successfully performed, there must be a MITM with transmit and spoofing capabilities and access to the director key and multiple online keys.

3.3.8 Arbitrary Edit Attack

This attack is successful if an attacker can edit updates sent from the repository to an ECU. Protections for this type of attack include implementing Uptane protection, detecting external identity spoofing, testing, and verifying updates, peer review code modifications, audits of all libraries, and scheduled revocation and redistribution of keys. There are two implementations of this attack. The first implementation of the attack targets data-at-rest on the Director server by attempting to replace a valid update file with a malicious one. In the second version of this attack, data-in-transit is intercepted and

edited. For either of these attacks to be successfully attempted, there must be a MITM with transmit and spoofing capabilities and access to a developer key and a director key.

3.3.9 Injection Attack

This attack is successful if an attacker can add their own functionality to an update by injecting data into the update package. Protections for this type of attack include implementing Uptane protection, detecting external identity spoofing, testing, and verifying updates, performing code reviews, audits of all libraries, and scheduled revocation and redistribution of keys. For this attack to be successfully performed, there must be a MITM with transmit and spoofing capabilities and inside actor/s with access to the director key as well as multiple offline keys. Additionally, any malicious code changes or library adjustments would need to go undetected by reviewers.

3.3.10 Insider Attack

This attack is successful if an attacker can send a fully verified malicious update. Protections for this type of attack include implementing Uptane protection, testing and verifying updates, peer reviewing code modifications, audits of all external libraries, and scheduled revocation and redistribution of keys. For this attack to be successfully performed, an external library would have to be compromised without being detected, or there must be a MITM with transmit and spoofing capabilities and inside actor/s with access to the director key and multiple offline keys.

4. Future Efforts

This three-year program improves the security of RAS starting with the vehicle (the most forward-deployed asset). In Year 1, the mitigations for the attacks against vehicle update functionality will be demonstrated in a test bench environment.

During Year 2, development of the software will continue, focusing on the server-side with a target of execution on DoD Servers. Attacks that affect server functionality will be included in the threat model. A demonstration of the capabilities running on an MRZR or representative vehicle environment will be provided.

Year 3 will focus on validation and verification of the update system, executing penetration testing and Uptane compliance testing on the MRZR as well as a DoD-approved server. The demonstration during year 3 will include a live mock field exercise, complete with an active red team attempting to leverage a cyber vulnerability against the vehicle.

Additional advancements that are outside of current program expectations include post-quantum cryptography, fleet monitoring, forward deployed security for denied or contested environments, and zeroing-out compromised equipment.

5. Conclusion

Over the course of this three-year program, the cyber resilience of robotic and autonomous systems will be markedly improved. Demonstrations of the capabilities and verifications through both internal and external efforts will be performed. Through this effort, the DoD will have a comprehensive secure software update solution tailored for robotic & autonomous systems.

6. References

[1]Joint Development Foundation Projects, LLC, Uptane Series, "Uptane Standard for Design and Implementation v1.2.0," [Online]. Available: https://uptane.github.io/papers/uptane-standard.1.2.0.html. [Accessed 17 March 2022].