Securing Software Updates for Automotives Using Uptane

TRISHANK KARTHIK KUPPUSAMY, LOIS ANNE DELONG, AND JUSTIN CAPPOS

Trishank Karthik Kuppusamy is a fifth-year PhD student at the NYU Tandon School of Engineering, where he works with Professor Justin Cappos on the design and deployment of software update security systems. He led the specification for Uptane, a new technology to secure software updates for automotives. trishank@nyu.edu

Lois Anne DeLong is a Research Associate and Technical Writer for the Secure Systems Lab at NYU Tandon School of Engineering. She has served as a writer and editor for technical journals, and has also taught technical writing and basic composition courses. lad278@nyu.edu

Justin Cappos is an Assistant Professor at NYU in the Tandon School of Engineering. Justin's research interests focus on improving the security of real-world systems in a variety of practical applications. His more recent work on software updaters has been standardized by Python and deployed by Docker. jcappos@nyu.edu

Does secrecy improve security or impede securing software updates? The automotive industry has traditionally relied upon proprietary strategies developed behind closed doors. However, experience in the software security community suggests that open development processes can find flaws before they can be exploited. We introduce Uptane, a secure system for updating software on automobiles that follows the open door strategy. It was jointly developed with the University of Michigan Transportation Research Institute (UMTRI), and the Southwest Research Institute (SWRI), with input from the automotive industry as well as government regulators. We are now looking for academics and security researchers to break our system before black-hat hackers do it in the real world—with possibly fatal consequences.

Security Should Not Be a Competitive Advantage

Imagine that you get into your car and turn on the ignition, but the engine does not start. You turn the key again, but the only sound you hear is the automatic door locks closing. After a few more futile attempts to start the car—and to open the doors—you notice a message on the screen of your infotainment system: "$500 in Bitcoin if you want to get out of your car." A hacker has just exploited a security flaw in the system used to deliver software updates to one of your car's on-board computing units, and the result is this simple but effective cyber-attack. We need your help in preventing this scenario from happening in the real world.

Presently, vehicle manufacturers purchase proprietary software update systems from third-party suppliers. This helps to keep costs competitive, because a manufacturer need not worry about developing its own system. These systems are proprietary in nature, and, thus, their security guarantees are unclear. A manufacturer may not even have access to the source code used in parts created by one of their suppliers. What is known is that these systems have been hacked repeatedly [1–3]. At a time when computing units continue to proliferate on vehicles, and where the cost of security flaws in code can be measured in human lives, many manufacturers still follow the design principle of security by obscurity, which has resulted in a substantial number of successful attacks.

We strongly believe that the security of your car should not be based upon which supplier can market their solution best to the car companies. It would not be a desirable outcome for a manufacturer or supplier to advertise that compromises of their software update system only harmed hundreds of people, while their competitors' compromises harmed thousands. Open security reviews have been used time and time again in the design of critically important systems, such as cryptographic algorithms, anti-censorship software, and secure software update systems. Making the design of software systems in a more open manner can benefit manufacturers, suppliers, and the public simultaneously.

Uptane, a new, secure software update system, is a direct product of such an open process. Uptane was designed in collaboration with major vehicle manufacturers and suppliers responsible for 78% of vehicles on US roads, as well as government regulators. We have shared technical documents and a reference implementation to aid manufacturers and suppliers to build, customize, and deploy their own variants of this system. A supplier has begun selling a product that includes Uptane, and a few others are integrating it as we speak. As adoption grows, we are looking to the open source community to give our code a test drive. We welcome white-hat hackers to try to break Uptane and to give us feedback before you, and millions of others, are betting your life on its security.

A Quick Primer on Computers in Vehicles

While most people think of a car as a collection of mechanical parts such as the engine, door locks, and brakes, a modern vehicle is actually a sophisticated container for a collection of microcomputers called *electronic control units* (ECUs). Like any other computer, these ECUs are responsible for executing specific functions, from tightening a seat belt during an accident to adjusting a passenger side mirror. Where ECUs differ from traditional computers is in how heterogeneous their computational speed, memory, and network capabilities are. For example, some ECUs, such as the telematics or infotainment units, have general-purpose CPUs with high speed, large memory, and a wireless connection to the outside world, whereas other ECUs, such as the seat belt pretensioner ECU, use specialized CPUs with low speed, small memory, and no external network connection.

An *original equipment manufacturer* (OEM), such as Ford or General Motors, chooses the ECUs that will reside on a vehicle model. However, these units are usually produced by third-party suppliers, such as Bosch or Lear. The software for an ECU is maintained by its supplier and delivered to the OEM to be distributed to vehicles.

To distribute software updates, the OEM maintains a software *repository*, which hosts and distributes images and metadata. An *image* is a self-contained archive of code and/or data required for an ECU to function. *Metadata* is information about images or other metadata files. Typically, this metadata lists the cryptographic hashes and file sizes of images.

This metadata should be signed, using well-protected keys, so that attackers cannot tamper with images without being detected. Some manufacturers and suppliers do not provide signed metadata about images. Instead, ECUs can be reflashed over the network if attackers know the fixed challenge-response algorithm used to unlock ECUs. Although these fixed algorithms are supposed to be secret, they are known by the car tuning community [1, 2]. To take another example, Tesla did not, to the best of our knowledge, sign its images at all until security researchers used a wireless connection to rewrite software on its ECUs and exert physical control over its vehicles [3]. Although it is important to sign metadata, the security of ECUs depends on precisely how it is signed.

Existing Software Update Systems Do Not Fit the Automotive Industry

Existing software update systems force an unacceptable tradeoff upon OEMs. To achieve maximum security, they often have to sacrifice the customizability that allows them to offer different images to different vehicles. On the other hand, other systems offer customizability but no security when attackers have compromised the repository itself.

Some security systems use *online keys*, or signing keys that are accessible from the repository, to sign metadata, protecting ECUs from man-in-the-middle attacks. For example, these systems may use the SSL/TLS or CUP transport protocol sign images and metadata in transit. The upside of using an online key is that it allows *on-demand customization of vehicles*, an attribute that was considered very important by our industry collaborators for various legal and technical reasons.

Unfortunately, the downside of using an online key to sign all metadata is that attackers who compromise the repository can also immediately abuse this key to sign and distribute malware. This is true even if the online key is protected behind a Hardware Security Module (HSM).

To solve this problem, some security systems use *offline keys*, or signing keys that are not accessible from the repository, to sign all metadata. These systems may use, for example, the PGP/GPG or RSA cryptographic schemes for this purpose. The upside of using offline keys is that it provides *compromise-resilience*: attackers who compromise the repository are unable to tamper with images without being detected. In practice, however, it is typically a precarious form of compromise-resilience, because often a single offline key is used to sign all metadata.

Unfortunately, the downside of using only offline keys to sign all metadata is that we have lost on-demand customization of vehicles. This is because the repository cannot dynamically respond to fresh information that indicates what is currently installed on a vehicle and decide what should be installed next.

Besides the on-demand customization of vehicles, there are other critical constraints in designing a secure software update system for automotives. Above all else, the system must be simple for manufacturers and suppliers to implement, customize, and deploy. Another important constraint is that ECUs are often limited by speed, memory, or network connection. Many

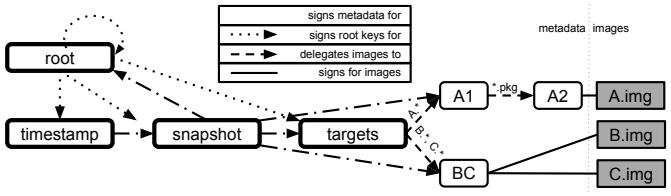

Figure 1: Separation of duties between roles on a compromise-resilient repository

ECUs are highly optimized for a specific function in order to keep costs low. Thus, many ECUs may not have enough storage space to maintain a large amount of metadata, may not have a direct network connection to the repository, and may not be able to compute or verify a signature in a reasonable amount of time.

Uptane: A New, Secure Software Update System

Uptane is a new, secure software update system that is specifically designed to solve problems in the automotive domain [4]. The key idea is to use two repositories, one to provide compromise-resilience and the other to provide on-demand customization of vehicles.

Uptane uses four design principles that help to achieve compromise-resilience [5, 6]. First, different types of metadata are signed using different keys, so that the impact of a key compromise is minimized and does not necessarily affect the security of the whole system. As illustrated in Figure 1, and summarized in Table 1, there are four top-level roles on a repository: the root, timestamp, snapshot, and targets roles. Second, a threshold number of signatures may be required to sign a metadata file, so that a single key compromise is insufficient to publish malicious images. Third, there must be a way to revoke keys when they are compromised. Keys can be revoked explicitly by publishing new keys to replace old ones, or they can be revoked implicitly by setting expiration timestamps in metadata files. Finally, use of offline keys can minimize the risk of a key compromise for high-value roles whose compromise can lead to malicious images.

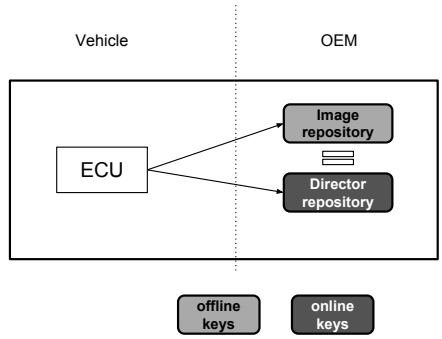

Figure 2: Using two repositories to provide both compromise-resilience and on-demand customization of vehicles.

On the *image repository*, offline keys are used to sign all metadata about all images for all ECUs on all vehicles manufactured by the OEM. Metadata for the top-level roles are signed by the OEM's administrators. The OEM may delegate the signing of images to their respective suppliers, or it may sign them itself. This repository provides compromise-resilience but not on-demand customization of vehicles.

The *director repository* instructs vehicles on what should be installed next, given information about what they have currently installed. This repository uses online keys to sign fresh timestamp, snapshot, and targets metadata for each vehicle that indicates which images from the image repository should be installed next.

As depicted in Figure 2, vehicles install images only if both repositories agree on their contents. That is, the contents of images chosen for installation by the director repository must match the contents of the same images available on the image repository. Since the director repository has more complicated functionality, it is more likely to contain vulnerabilities that can be remotely exploited, and thus compromised. By separating both repositories, we are able to prevent attackers who compromise one repository from being able to distribute malicious images.

Role	Responsibilities
Root	The **root** role is the locus of trust. It indicates which keys are authorized for the **targets, snapshot,** and **timestamp** roles. It also lists the keys for the **root** role itself.
Targets	The **targets** role provides crucial metadata about images, such as their hashes and lengths. This role may delegate the signing of images to their respective suppliers.
Snapshot	The **snapshot** role indicates the latest versions of all metadata on the repository. This prevents an ECU from installing outdated images.
Timestamp	The **timestamp** role is responsible for indicating if images or metadata have changed.

Table 1: A summary of responsibilities of the top-level roles on a repository

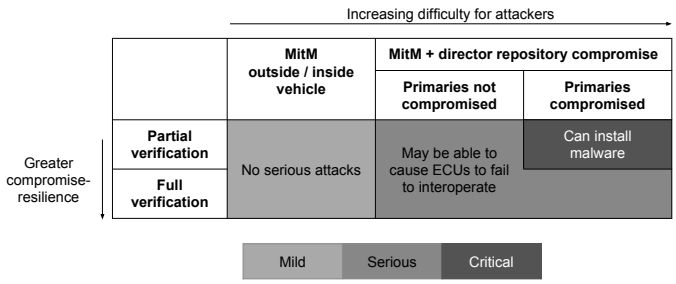

Figure 3: A brief security analysis of ECUs using Uptane, depending on which repositories and ECUs attackers have compromised

There are two types of ECUs. A *primary* downloads, verifies, and distributes images and metadata to secondaries. A *secondary* receives them from a primary, and installs a new image only if it has been successfully verified against the signed metadata.

There are two types of metadata verification designed to accommodate ECUs with different security and cost requirements. *Full verification* requires checking that the images chosen for installation by the director repository match the same images on the image repository. Primaries always perform full verification in order to protect secondaries from security attacks. *Partial verification* requires checking only that the signatures from the director repository are valid.

A brief security analysis is illustrated in Figure 3. The difference between ECUs that perform full and partial verification is in how resilient they are against a repository compromise. When there are only man-in-the-middle attacks but no key compromise, attackers do not pose a serious threat. When attackers have compromised the director repository, there are two cases: primaries that have been compromised and primaries that have not.

If attackers have not compromised primaries, then they may be able to cause both types of ECUs to fail to interoperate. This is because attackers can control which images are installed on which ECUs. However, it is possible to limit the attackers' choices by including metadata that prevent ECUs from installing incompatible or conflicting images. Nevertheless, they cannot install malicious updates, because primaries always perform full verification on behalf of secondaries.

However, attackers that have compromised primaries can install malicious updates, but only on partial verification ECUs. Attackers cannot install malicious updates on full verification ECUs, even if they have also compromised the image repository, because they must also compromise offline keys.

In summary, Uptane offers basic security guarantees for all ECUs and greater compromise-resilience for ECUs that can afford additional computation. In addition, by separating concerns over multiple repositories, Uptane also provides on-demand customization of vehicles.

A Call to Action

We believe that Uptane provides the strongest solution to a real-world problem, without sacrificing usability and flexibility. However, we do not know of a better way to guarantee the security of any system than subjecting it to a critical, rigorous, and open review. We want you to scrutinize Uptane and find any design flaws before the black-hat hackers use them against us. You can drop us comments on our Google Docs or report issues and send pull requests on our GitHub projects. To do so, please visit our Web site at https://uptane.github.io.

Acknowledgments and Disclaimers

Our co-authors include Akan Brown (NYU), Sebastien Awwad (NYU), Damon McCoy (NYU), Russ Bielawski (UMTRI), Sam Weber (NYU), John Liming (SWRI), Cameron Mott (SWRI), Sam Lauzon (UMTRI), and André Weimerskirch (UMTRI/Lear Corporation).

Uptane is supported by US Department of Homeland Security grants D15PC00239 and D15PC00302. The views and conclusions contained herein are the authors' and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the US Department of Homeland Security (DHS) or the US government.

References

[1] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, and S. Savage, "Experimental Security Analysis of a Modern Automobile," 2010 IEEE Symposium on Security and Privacy: http://www.autosec.org/pubs/cars-oakland2010.pdf.

[2] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, and T. Kohno, "Comprehensive Experimental Analyses of Automotive Attack Surfaces," in *Proceedings of the 20th USENIX Security Symposium*, 2011: http://www.autosec.org/pubs/cars-usenixsec2011.pdf

[3] A. Greenberg, "Tesla Responds to Chinese Hack with a Major Security Upgrade," *Wired*, September 27, 2016: https://www.wired.com/2016/09/tesla-responds-chinese-hack-major-security-upgrade/.

[4] T. K. Kuppusamy, A. Brown, S. Awwad, D. McCoy, R. Bielawski, C. Mott, S. Lauzon, A. Weimerskirch, J. Cappos, "Uptane: Securing Software Updates for Automobiles," 14th ESCAR Europe 2016: https://ssl.engineering.nyu.edu/papers/kuppusamy_escar_16.pdf.

[5] J. Samuel, N. Mathewson, J. Cappos, R. Dingledine, "Survivable Key Compromise in Software Update Systems," in *Proceedings of the 17th ACM Conference on Computer and Communications Security (CCS '10)*, pp. 61–72: https://ssl.engineering.nyu.edu/papers/samuel_tuf_ccs_2010.pdf.

[6] T. K. Kuppusamy, S. Torres-Arias, V. Diaz, and J. Cappos, "Diplomat: Using Delegations to Protect Community Repositories," in *Proceedings of the 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI '16)*, pp. 567–581: https://www.usenix.org/conference/nsdi16/technical-sessions/presentation/kuppusamy.