Analysis of a Secure Software Upload Technique in Advanced Vehicles using Wireless Links

Irina Hossain, Student Member, IEEE, Syed Masud Mahmud, Member, IEEE

Abstract—Software modules of an advanced vehicle can be updated using Remote Software Upload (RSU) techniques. The RSU employs infrastructure-based wireless communication technique where the software supplier sends the software to the targeted vehicle via a roadside Base Station (BS). However, security is critically important in RSU to avoid any disasters due to malfunctions of the vehicle or to protect the proprietary algorithms from hackers, competitors or people with malicious intent. In this paper, we present a mechanism of secure software upload in an advanced vehicle. In order to increase the security level, we propose the vehicle to receive two copies of the software along with the Message Digest (MD) in each copy. The vehicle will install the new software only when it receives two identical copies of the software. To validate our proposition we find analytical expressions of average number of packet transmissions for successful software update. We investigate different cases depending on the vehicle’s buffer size and verification methods. Our analytical and simulation results show that it is sufficient to send two copies of software to the vehicle to thwart any security attack while uploading the software.

Key words—advanced vehicle, authentication, security, software upload and wireless communication.

I. INTRODUCTION

With the use of information and computer-based technologies, advanced electronic systems, sensing and intelligent algorithms, an advanced vehicle incorporates various advanced features, such as drive-by-wire, telematics, pre-crash warning, remote diagnostics, highway guidance, traffic alert etc. Introduction of new features, improvement of existing features, updating navigation information etc. will require software update in vehicle’s electronic modules from time to time. On the other hand, evolution of wireless technologies has directly benefited the nation’s transportation system. The automotive industry and Intelligent Transportation System (ITS) use different wireless technologies for different applications including road safety, traffic management and driver assistance. In the same way, software update in vehicle’s electronic modules could be benefited from using the wireless technology. Remote software upload, using wireless communication links, will aid the update process by saving both consumers’ and auto manufacturers’ time and money [1].

When a particular vehicle experiences some problems with its functionality, then the software provider can establish a point-to-point communication link with the vehicle via a roadside base station (BS) under which the vehicle resides and sends the necessary software to the non-functioning module. However, transmitting software packets over radio channels makes eavesdropping, data altering, theft of service, and denial of service (DoS) attacks easier for adversaries. Hence, additional security mechanisms are needed to protect the communication over wireless network. This paper presents a secure architecture for remote software upload in a vehicle. The proposed technique authenticates software provider as well as the vehicle to which software needs to be uploaded, and provides integrity of the software being transmitted from vendor to vehicles. In order to increase the security level of the proposed mechanism we analyze different scenarios with regard to vehicle’s buffer size and software packet verification methods using analytical and simulation models.

The paper is organized as follows. Section II presents background information about the technology used in this work and a brief review of past researches done, Section III describes the Remote Software Upload (RSU) architecture we proposed and its analytical modeling, Section IV summarizes the results of analysis and simulation, and Section V presents the conclusion.

II. BACKGROUND AND RELATED WORKS

Since wireless communication technologies provide various advantages, such as portability, flexibility, lower installation cost and increased efficiency, they are becoming the communication infrastructure of choice in our everyday lives. Advances in wireless communication system have potential value for nation’s transportation system. Using Global Positioning System (GPS) and voice-activated cellular system, OnStar Corporation has successfully deployed Advanced Automatic Crash Notification (AACN) system in modern vehicles to report accidents in reliable and timely manner [2]. Auto manufacturers are exploring Bluetooth technology for in-vehicle Wireless Personal Area Network (WPAN) which will connect various on-board devices such as cell phones, PDAs, laptops and GPS transceiver [3].

Although wireless technologies offer the users with additional conveniences over the wired technologies, they also introduce unique security challenges. Various threats
and vulnerabilities associated with wireless network and handheld devices are listed in [4]. Hence, additional mechanisms are needed to protect the security, i.e., integrity, authenticity and confidentiality of communication over wireless networks.

In previous, secure software update techniques in mobile devices over IP network have been proposed [5-6]. Since security protocols in WLAN and WWAN have security flaws, they proposed both symmetric and asymmetric/public-key cryptographic techniques and the combination of two (symmetric and asymmetric). In order to increase the security level, we propose the vehicle to upload two copies of the software and the message digest (MD) in each copy. Since the vehicle will not accept the software unless the packets in two copies match, there is no chance that the vehicle will upload the software that is changed by the hacker.

III. THE SECURE SOFTWARE UPLOAD ARCHITECTURE

The detail description of Remote Software Upload (RSU) technique in a vehicle using wireless communication link was described in [1]. In our architecture, we assume that the Auto Company (AC) might have its own software distribution center or it has agreement with a third party Software Vendor (SV) to provide the required software. Each vehicle has its wireless unit installed in it to communicate with each other as well as with the BSs. The AC, the SV and BSs are connected through high-speed wired/wireless networks, whereas the vehicles that travel between cells, can communicate with underlying network via BSs using long-range wireless communication links, e.g., cellular or Wi-Fi links. The BS, under which the targeted vehicle resides, receives software packets from the SV using secure communication technique such as SSL/TLS and transmits the packets to the targeted vehicle through secure wireless link (Fig. 1).

A. Authentication and Key Agreement Process

While a wireless device is installed in a vehicle $V_m$, a set of authentication keys $K_m = \{k_{V_m}^1, \ldots, k_{V_m}^n\}$ is provided to it.

Each key is used to authenticate $V_m$ at each software distribution session. A copy of these keys will also be kept in a secure Central Server (CS) which is maintained by the AC or any trusted party. The key management process for vehicle’s authentication key was described in [1]. The AC or any other Certification Authority (CA) issues Digital Certificates to the SV and all BSs which contain their authentic public-keys. We assume that all the vehicles and the BSs have a copy of the SV’s authentic public key and the BSs have each others public key.

When the AC decides to upload software to a vehicle $V_m$, it sends an unused authentication key $k_{V_m}^j$ and the module number to which software needs to be uploaded to the SV using a secure link such as SSL/TLS. Upon receiving the message, the SV creates a $SW_{update\_join\_request}$ message that consists of a message ID, a Vehicle’s ID (VID, could be a part of its VIN number), a module ID to which the software needs to be updated, the version number of the software and a session key $k$. The SV digitally signs it, encrypts the message and the signature using $k_{V_m}^j$ and sends it to the BS under which $V_m$ is currently located. The BS honestly relays the message to $V_m$. After receiving the $SW_{update\_join\_request}$ message, $V_m$ decrypts the message using $k_{V_m}^j$, verifies the signature and version number of the software and sends a $join\_acceptance$ message. If authentication fails, the vehicle $V_m$ ignores the message.

B. Sending the Software Packets

After successful authentication of both the vehicle and the SV, the SV starts sending the software packets encrypted with the session key $k$. The SV can use this key to create a MAC (Message Authentication Code) value of each software packet and send it along with the packet. The vehicle verifies the authenticity of the packet by checking the MAC, and the integrity of the packet by comparing the hash value of the received packet and the one contained in the MAC. Since both parties share the same secret key, anyone who has the key could generate the MAC, thus it does not guarantee non-repudiation in case of dispute between the SV and the vehicle. Moreover, if an intruder could successfully change both the packet and the MAC value then there is no way that the vehicle could verify the software. A better solution for software upload was proposed in [1] where it was suggested that the vehicle receives two copies of the software along with the Message Digest (MD) in each copy. If some packets of the first copy do not match with the corresponding packets in the second copy, the vehicle requests to send the unmatched packets. After receiving both the copies along with the MDs, the vehicle calculates an MD based on the received software and compares it with the received MD. The vehicle accepts the software only when the calculated MD and received MD match. Fig. 2 shows the flow diagram of the technique. In the next section we present several ways how the vehicle
Fig. 2. Two-copy Software Upload Technique receives two copies of the software and find analytical expressions for average number of packet transmissions ($N$) for successful software reception in each case. In order to do the comparison, we also present the expression of $N$ for single-copy software upload technique.

1) Notation
The symbols and notations that will be used throughout the paper are presented in Table I.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Significance</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M$</td>
<td>Total number of software packets without MD</td>
</tr>
<tr>
<td>$m$</td>
<td>Number of packets in a segment</td>
</tr>
<tr>
<td>$S$</td>
<td>Number of segments = $M/m$</td>
</tr>
<tr>
<td>$p$</td>
<td>Packet error probability due to hacking</td>
</tr>
<tr>
<td>$P_{\text{pair}}$</td>
<td>Probability that a packet-pair do not match due to hacking</td>
</tr>
<tr>
<td>$P_{\text{soft}}$</td>
<td>Probability that the received software is in error due to hacking</td>
</tr>
<tr>
<td>$T$</td>
<td>Average number of trial to send one packet or one segment or total software successfully</td>
</tr>
<tr>
<td>$P_i$</td>
<td>Probability of success in $i^{th}$ trial</td>
</tr>
<tr>
<td>$N_p$</td>
<td>Average number of packet transmission to receive one good packet</td>
</tr>
<tr>
<td>$N$</td>
<td>Average number of packet transmission for successful software upload</td>
</tr>
</tbody>
</table>

2) Definitions
Fig. 3 shows different software upload techniques that we consider in our analysis.

a) Single-copy Software Upload
If there is only one buffer in vehicle’s software module to accept the new software and one copy of the software packets is sent appended with the MD then it is called Single-copy Software Upload.

b) Segmented Single-copy Software Upload
If the software packets are divided into segments of certain number of packets and each segment is sent with the MD then it is called Segmented Single-copy Software Upload.

c) Multiple-copy Software Upload
If there are more than one buffer and multiple copies of the software packets are sent with the MD in each copy unless there is a match found then it is called Multiple-copy Software Upload.

d) Infinite Buffer Case
If there are infinite number of buffers to accept multiple copies of a packet to compare a new copy of the packet with the packets already received until a match is found then it is called the Infinite Buffer Case. This is the ideal case and not practical, which requires minimum number of packet transmissions for a successful software upload.
If there are two buffers to accept two copies of a packet and one or both of the packets are replaced by the new packets transmitted until the vehicle receives a good packet then it is called the Finite buffer Case.

If a packet-pair do not match then the vehicle could delete both packets and request to send another pair until a matched pair is found. This case is defined as the Finite Buffer with Pair Transmission.

If a packet-pair do not match then the vehicle could delete one randomly chosen packet and request to send another packet until a matched pair is found. This case is defined as the Finite Buffer with Random Packet Delete.

If a packet-pair do not match then the vehicle always deletes the older packet and requests to send another packet until a matched pair is found. This case is defined as the Finite Buffer with Two Consecutive Good Packets.

3) Single-copy Software Upload

After receiving all the encrypted software packets and the MD, the receiving vehicle decrypts the packets, calculates an MD and compares it with the received MD. If both the MDs match, then the vehicle accepts the software. Otherwise, it requests the supplier to retransmit the entire software. In this method, if a hacker changes at least one software packet, then the calculated MD will differ from the received MD. Since the vehicle or the supplier does not know which packet has been changed, the supplier needs to retransmit the entire software including the MD which requires more network bandwidth. Moreover, if a hacker can successfully change a packet from every transmission, it is not possible at all to upload the software successfully.

For packet error probability $p$ due to hacking, the probability that the software is in error is:

$$p_{soft} = 1 - (1 - p)^{M+1}$$

(1)

The average number of trials required to send the software successfully is

$$T = \sum_{i=1}^{M} i p_{soft}^{i-1} = \frac{1}{1 - p_{soft}} = 1 - p_{soft}$$

(2)

The average number of packets transmission for successful software upload is

$$N = (M + 1)T = \frac{M + 1}{(1 - p)^{M+1}}$$

(3)

4) Segmented Single-copy Software Upload

In case of Single-copy Transmission, if the number of software packets $M$ increases, the average number of packet transmission for successful software upload increases exponentially (1). An alternative approach could be to divide $M$ software packets into $S$ segments with $m$ packets in each segment. Then the average number of trials required sending one segment successfully is

$$T = \frac{1}{(1 - p)^{m+1}}$$

(4)

The average number of packet transmission needed for successful upload of $S$ segments is

$$N = (m + 1)S + \frac{(m + 1)M}{(1 - p)^{m+1}}$$

(5)

5) Multiple-copy Software Upload – Infinite Buffer Case

For each software packet, the vehicle first receives two copies of the packet. If the packets do not match, it requests to send another copy of the packet. The third copy is compared with the previous two. If no match is found it requests for another copy. Since there is infinite number of buffers, after receiving the $i^{th}$ packet it compares the packet with previous $i - 1$ packets. The process continues until a matched-pair is found.

The probability that a packet is received successfully in the $i^{th}$ trial is

$$p_i = ip_{pair}^{-1}(1 - p)^2, i = 1, 2, 3, \ldots, \infty$$

(6)

The average number of packet transmission for successful upload of one packet is

$$N_p = \sum_{i=1}^{\infty} (i + 1)p_i = \frac{2}{1 - p}$$

(7)

The average number of packet transmission for successful software upload is

$$N = (M + 1)N_p = \frac{2(M + 1)}{1 - p}$$

(8)

6) Finite Buffer with Pair Transmission

In this case, if both the copies of a packet do not match, the supplier will send another pair of packets.

The probability that a pair does not match is

$$p_{pair} = 1 - (1 - p)^2$$

(9)

The average number of trials to send one packet successfully is

$$T = \sum_{i=1}^{\infty} i(1 - p)^2(1 - (1 - p)^{i-1}) = \sum_{i=1}^{\infty} i(1 - p)_{pair}^{-1}p_{pair}^{i-1}$$

$$\Rightarrow T = \frac{1}{1 - p_{pair}} = \frac{1}{(1 - p)^2}$$

(10)

The average number of packet transmissions for successful software upload is

$$N = 2(M + 1)T = \frac{2(M + 1)}{(1 - p)^2}$$

(11)

7) Finite Buffer with Two Consecutive Good Packets

When the two received copies of a packet do not match, the vehicle replaces the first copy in buffer 1 with the second copy in buffer 2, requests to send another copy and places in buffer 2. The average number of packet transmissions for
successful upload of one packet is The average number of packet transmissions for successful upload of one packet is

\[ N_p = \sum_{i=1}^{\infty} (i+1)p = \frac{2-p}{(1-p)^2} \]  \hspace{1cm} (12)

Then the average number of packet transmissions for successful software upload is

\[ N = (M+1)N_p = \frac{(M+1)(2-p)}{(1-p)^2} \]  \hspace{1cm} (13)

IV. SIMULATION RESULTS

We now present simulation results to validate the analytical expressions developed for different software upload Techniques. For a particular packet error probability \( p \) due to hacking, we generated a uniformly distributed random number using drand48() function in C++ with gcc compiler. If the random number is less than \( p \) then the packet was considered as a bad packet and vise versa.

Fig. 4 and Table II show the resemblance between the analytical and simulation results for the average number of packet transmissions for the Single-copy and Multiple-copy software upload techniques, respectively. For the Single-copy transmission, at higher \( p \) the average numbers of packet transmissions \( (N) \) for successful software upload increases exponentially as the software size increases. However, if the software is sent in segmented form, it reduces \( N \) considerably. Fig. 5 exemplifies the effect of segmentation for the software size with 1024 packets and different number of segments. The more the number of segments, the lesser is the number of packet transmissions necessary for successful software upload. Conversely, as the number of segments increases, it might take more time to encrypt, decrypt and transmit all the segments. Hence, there should be a trade-off between number of segments and processing time.

The Two-copy software upload is always superior to the Single-copy software upload as long as security is concerned. Since the second copy will be transmitted after a random time interval in a random packet order, it is very unlikely that an intruder would know whether a second copy will be transmitted or not. Moreover, even if an intruder changes one packet of the first copy, it would be difficult for him to change the same packet in the second copy due to the randomness of packet transmission.

Fig. 6 represents the average number of packet transmissions \( (N_p) \) to upload a single packet successfully in the multiple-copy software upload scenario. Unlike the single-copy software upload, the total number of packet transmissions necessary to upload the entire software is linearly dependent on the software size (eq. (8), (11) and (13)).

From Fig. 6 it is also observed that for low values of \( p \), on average only two packets need to be transmitted for any of the techniques we mentioned above. For a high value of \( p \), *Finite Buffer with Random Packet Delete* provides the least number of packet transmissions with respect to the ideal case where we have infinite number of buffers. In general, the hacking probability is very low. Thus, any of the techniques could be used if there are one or more unmatched packet pairs. In addition, \( N \) does not vary notably between the two buffer case and the infinite buffer case. Addition of more buffers would not increase the performance of software upload remarkably. Consequently, we propose to use not more than two buffers in vehicle’s software modules to upload two copies of software.

At lower \( p \), single-copy software uploads requires fewer number of packet transmissions than the multiple-copy software uploads. However, the later technique offers additional security if the software packets are transmitted in random order and the second copy is transmitted after a random time interval with a very long average value.

Hence, we recommend that initially the supplier should send two copies of the software in the vehicle.
TABLE II. COMPARISON OF ANALYTICAL AND SIMULATION RESULT FOR DOUBLE-COPY TRANSMISSION

<table>
<thead>
<tr>
<th>$p$</th>
<th>Infinite Buffer (Simulation)</th>
<th>$N_p$ (Analytical)</th>
<th>Finite Buffer with Pair Transmission (Simulation)</th>
<th>$N_p$ (Analytical)</th>
<th>Finite Buffer with Two Consecutive Good Packets (Simulation)</th>
<th>$N_p$ (Analytical)</th>
<th>Finite Buffer with Random Packet Delete (Simulation)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>2.2228</td>
<td>2.2222</td>
<td>2.4704</td>
<td>2.4691</td>
<td>2.3443</td>
<td>2.3457</td>
<td>2.2847</td>
</tr>
<tr>
<td>0.01</td>
<td>2.0204</td>
<td>2.0202</td>
<td>2.0406</td>
<td>2.0406</td>
<td>2.0303</td>
<td>2.0304</td>
<td>2.0253</td>
</tr>
<tr>
<td>0.001</td>
<td>2.0020</td>
<td>2.0020</td>
<td>2.0040</td>
<td>2.0040</td>
<td>2.0031</td>
<td>2.0030</td>
<td>2.0024</td>
</tr>
<tr>
<td>0.0001</td>
<td>2.0002</td>
<td>2.0002</td>
<td>2.0004</td>
<td>2.0004</td>
<td>2.0003</td>
<td>2.0003</td>
<td>2.0003</td>
</tr>
<tr>
<td>0.00001</td>
<td>2.0000</td>
<td>2.0000</td>
<td>2.0000</td>
<td>2.0000</td>
<td>2.0000</td>
<td>2.0000</td>
<td>2.0000</td>
</tr>
</tbody>
</table>

V. CONCLUSION

The paper presents detail architecture of RSU in an advanced vehicle’s software modules using an existing wireless communication technology such as Wi-Fi or cellular. In this architecture, the BSs act as proxies to reliably and honestly relaying the software packets from the SV to the vehicle. Since they do not have access to the software packets, it eliminates any security threat that might exist if the BSs locally decrypt and encrypt the packets. The architecture provides mutual authentication of the SV and the vehicle. A vehicle’s authentication keys are shared between the AC and the vehicle, and different authentication keys are used for different software distribution sessions which prevent known-key attack. We suggest the SV to send two copies of the software to the vehicle to increase the level of security. Moreover, digital signature of the SV ensures non-repudiation and the MD of the entire software provides integrity of the software.

This paper focuses on the software upload in a single vehicle. However, if the AC needs to upload software to a large number of vehicles, then wireless multicasting would be a better solution than multiple unicasting to individual vehicles.

The RSU will have huge demand in Auto industry in the near future. If it could be implemented successfully, it will save both AC and consumers’ time and money.

REFERENCES


