
Wassim Itani  Ayman Kayssi  Ali Chehab
Department of Electrical and Computer Engineering
American University of Beirut
Beirut 1107 2020, Lebanon
{wgi01, ayman, chehab}@aub.edu.lb

ABSTRACT
In this paper we propose PETRA; an energy-efficient and secure software update protocol for severely-constrained network devices. PETRA ensures the authenticity and end-to-end integrity of software update components delivered from trusted content distribution networks. The protocol operates by employing a set of energy-efficient data structures and cryptographic constructs to efficiently detect any form of man-in-the-middle modification attacks on the update packets. This methodology contributes to a sizeable decrease in network traffic and as a result huge energy savings. This makes PETRA a very suitable security protocol for limited-resource battery-operated devices such as low-end mobile phones, wireless sensors, and even Radio Frequency Identification Devices (RFIDs) tags. Moreover, PETRA realizes an incremental security verification mechanism that allows the dynamic eager loading of received software components. This mechanism prevents any form of service disruption or operation downtime during the code upgrade process. A prototype PETRA implementation is tested on a grid of simulated micaz sensor nodes running the TinyOS operating system. A platform-independent performance analysis and an experimental simulation show that PETRA can achieve up to 30% average reduction in network-wide energy consumption.

Categories and Subject Descriptors

General Terms

Keywords
WSN security, secure software updates, energy-efficient code updates.

1. INTRODUCTION
Software update management is increasingly becoming a major pillar in the software development life cycle. Software updates can serve many essential functional requirements that are crucial for the proper operation, configuration, management, and security of already installed applications and operating environments. In today’s Internet age, most of the software update mechanisms are achieved automatically and remotely by leveraging the capabilities of wired and wireless communication networks. In such open environments, it is exceptionally important to protect the code distribution process as a whole and particularly the integrity and authenticity of the update components marshalling over the network links. Failing to satisfy these security requirements would leave the client end systems under the mercy of weak man-in-the-middle attacks. Such attacks would result in serious and unpredictable consequences if the attacker were able to viciously modify the disseminated update package or inject new maliciously crafted code components.

Recently, the rapid proliferation of embedded technologies made the Internet accept a new set of clients represented in mobile phones, PDAs, wireless sensors of various types, and even advanced RFID tags. Today, these devices constitute a major building block in the modern Internet infrastructure. The networking capabilities (mainly wireless), supported in these devices, allows the creation of highly organized functional networks which connect to the Internet via specialized gateways and workstations. Most of today’s networked embedded devices run an assortment of sophisticated software applications and operating systems. This fact necessitates the existence of specialized secure software update protocols specifically targeting limited embedded environments. Embedded devices impose a set of unique and pressing performance requirements which renders the secure update process a highly complex and challenging task.

Generally, a successful software update protocol should take into account the following requirements and constraints:

1. Embedded devices, particularly, sensors and RFID tags have severely limited computing power and memory resources; hence the protocol should be simple and computationally efficient.
2. Many embedded environments, such as WSNs, are deployed on a large scale in relatively inaccessible and even hostile environments. This necessitates the execution of the update process remotely using the wireless communication network.
3. Embedded applications must have a long life span and therefore should be designed to operate for protracted periods of time without changing the batteries. Thus the protocol operating the software update process should be power and energy efficient.
4. To achieve significant energy savings, the secure update protocol must be designed to minimize the network traffic required in the update process. This requirement becomes more
pressing in embedded applications that require frequent software updates. For example, WSNs are usually deployed in unfamiliar environments with rapidly changing conditions and user requirements (forests, undersea, enemy battle fields, etc.). This necessitates frequent code updates to fix environment-related software bugs or to reload more efficient and effective application functionalities. In such scenarios, the code update is designed and developed based on the feedback collected in the initial deployment phase. Moreover, due to the memory constraints in WSNs, it is infeasible to load all application functionalities combined, hence a sensor node needs to be frequently reimaged with different application functionalities as needed. Applications requiring frequent updates are also common on mobile phones. An example is the updates required by virus scan applications which may be as frequent as every two hours in sophisticated virus scanners.

5. The secure update mechanism should only induce a limited or negligible service disruption to the embedded application operation.

In this paper we propose PETRA; an energy-efficient and secure software update protocol for severely-constrained network devices. PETRA ensures the authenticity and end-to-end integrity of software update components delivered from trusted content distribution networks. The protocol operates by employing a set of energy-efficient data structures and cryptographic constructs to efficiently detect any form of man-in-the-middle modification attacks on the update packets. This methodology contributes to a sizeable decrease in network traffic and as a result huge energy savings. This makes PETRA a very suitable security protocol for limited-resource battery-operated devices such as low-end mobile phones, wireless sensors, and even RFID tags. Moreover, PETRA realizes an incremental security verification mechanism that allows the dynamic eager loading of received software components. This mechanism prevents any form of service disruption or operation downtime during the code upgrade process. A prototype PETRA implementation is tested on a grid of simulated micaz sensor nodes running the TinyOS [14] operating system. A platform-independent performance analysis and an experimental simulation show that PETRA can achieve up to 30% average reduction in network-wide energy consumption.

The rest of this paper is organized as follows: Section 2 presents the threat model assumed in this work. Section 3 provides a brief background overview of the cryptographic mechanisms and data structures used in the protocol design. The system design model is presented in Section 4 followed by an analytical performance analysis in Section 5. Section 6 presents the simulation performance results obtained when testing PETRA on the TinyOS platform. Section 7 provides a literature survey of the main protocols related to the work proposed. Conclusions are presented in Section 8.

2. THREAT MODEL

The threat model we assume in this work is described as follows: we have a limited wireless device (update client) that is capable of performing wireless network interactions with one or more trusted content distribution servers. The network data communicated between the update client and the content distribution server may traverse several hops of wireless and wired links. We assume that the limited network device is trusted and that its software components and security mechanisms are correctly configured, installed, and operational. The attacker in PETRA’s threat model can jeopardize the integrity and authenticity of the software update components by executing weak man-in-the-middle attacks on network traffic. In other words, the attacker is assumed to have the necessary expertise and tools to conduct any type of modification and fabrication attack on the network data without being able of physically compromising the communicating end systems, namely the content distribution server and the update client.

3. BACKGROUND

The integrity-enforcement mechanism employed in PETRA mainly relies on Message Authentication Codes (MACs) and a probabilistic data structure known as a Bloom filter [15]. MAC functions are one-way hash functions that require a private key to verify the hash of the message. MACs are used to verify the authenticity and integrity of messages between parties sharing a symmetric shared key. A Bloom filter is a space-efficient data structure that is used to perform set membership verification using low-cost hash functions in constant time. The main components of a Bloom filter are:

- A bit array of size $m$ bits initially set to zero
- A set of $k$ different hash functions that maps an input element to $k$ different addresses in the $m$-bit array.

An element is added to the Bloom filter by generating a set of $k$ addresses using the $k$ hash functions. These addresses are mapped to $k$ positions in the $m$-bit array. The bit value of the mapped addresses is set to 1 in the bit array. To test for set membership, the element is input to each of the $k$ hash functions to get $k$ array addresses. The bit values of the $k$ addresses are fed into an AND logical function. If the result of the AND operation is 0 (any of the bits at these positions is 0), then the element is not in the set. Otherwise, (all bit values are 1), then the element is in the set with a very high probability.

Set membership false positives may rarely occur if for instance the $k$ bits have been set to 1 during the addition of other elements to the bloom filter. The false positives probability increases with the number of elements added to the Bloom filter and decreases with the number of bits $m$ in the Bloom filter bit array. A formula representing the false positives probability of a Bloom filter is presented in Section 5.

4. SYSTEM DESIGN

This section presents an overview of PETRA’s system design and operation. Figure 1 shows the network model assumed and the protocol execution steps on the sender and receiver ends. The update protocol is activated and executed whenever a new code update is available. The update components are deployed and packaged on one or more content distribution servers. PETRA assumes that the content distribution servers are trusted entities and that the update package components are legitimately designed and developed. The embedded client devices can either explicitly check and request the download of any available updates (as in the case of mobile devices directly operated and controlled by human entities) or the content distribution server may push the updates to be disseminated to a particular network of embedded devices (as in the case of WSN and RFID updates). It should be noted here that the dissemination protocol is implementation
dependent and the protocol implementer may choose any of the dissemination protocols discussed in Section 7.

PETRA provides an end-to-end integrity-enforced channel between the source distribution server and the embedded client device. In the rest of this paper, we refer to the source distribution server as the update server and the embedded client device as the update client. The secure protocol on the update server is responsible of creating the necessary cryptographic constructs and data structures to enforce the integrity of the update components against any form of tampering or modification. As stated in Section 3, PETRA utilizes MACs and a Bloom filter data structure to enforce the integrity of the update package. We assume that the update server and client share a common secret key $SK$. In a network of several update clients, each client may share a separate secret key with the server or the key may be the same on all the individual nodes. The operation of the server protocol is described as follows: the update package components $C_i , \ldots , C_K$ are individually added to a Bloom filter data structure $BF$. The whole contents of the update package are then “MACed” using $SK$ to get $MAC(SK, C_i||C_2||\ldots||C_K)$. In the same way, a MAC for $BF$ is generated using $SK$ to get $MAC(SK, BF)$. After the MAC generation step, the update server transmits $BF$, the MAC of $BF$, and the MAC of the complete update package to the update client.

$$BF||MAC(SK, BF)||MAC(SK, C_i||C_2||\ldots||C_K)$$

Afterwards, the update server starts the transmission of the update components one after the other. On the update client side, and after receiving $BF||MAC(SK, BF)||MAC(SK, C_i||C_2||\ldots||C_K)$ the protocol can check the integrity of received components incrementally by testing them against $BF$. If $BF$ returns true for a given component $C_i$, then the protocol accepts $C_i$ with a very high confidence level. This confidence level is the complement of the false positives probability of the Bloom filter. As will be shown in Section 5, the false positives probability is kept very low by tuning the parameters represented by the Bloom filter size and the number of update components $K$ fed into $BF$. On the other hand, if $BF$ returns false, then the protocol rejects $C_i$ and requests for a server retransmission. For a definitive integrity check, the secure update protocol verifies the MAC of the whole package content after receiving the last update component. The detailed protocol steps are presented in Figure 1.

Using a Bloom filter serves two main advantages: firstly it eliminates the need of authenticating each individual packet using a separate MAC value. This aids in a dramatic reduction in network traffic and hence an overall energy savings compared to traditional security protocols that authenticate each individual packet with a separate MAC. Secondly, using a Bloom filter allows the client to incrementally verify the integrity of individual update packets, even if they arrive out of order, with high confidence level. The confidence level can be flexibly configured based on the application’s security and performance requirements. The incremental verification property is very crucial in preventing any form of service disruption or application downtime.

5. MATHEMATICAL ANALYSIS AND PERFORMANCE

This section presents a formal mathematical analysis of PETRA’s performance. The equations presented in this section are conducted in a platform-neutral manner without depending on any device infrastructure or operating environment. Throughout this analysis, we assume that the network nodes are organized in an
Figure 2. a. % Energy Savings Network-wide vs. Number of Update Packets for P = 38 bytes, N=9, PO=8, PL=30, ETB=10EPB, ERB=5EPB, SHA_In_MAC=2, SHA_In_Bloom=3.
b. % Energy Savings Network-wide vs. ETB/EPB for ERB=0.5ETB, P = 38 bytes, N=9, PO=8, PL=30, K=30, B=38, SHA_In_MAC=2, SHA_In_Bloom=3.
c. % Energy Savings Network-wide vs. Network Size for P = 38 bytes, B=38, K=30, PO=8, PL=30, ETB=10EPB, ERB=5EPB, SHA_In_MAC=2, SHA_In_Bloom=3.
d. % False positives Probability vs. Number of Update Packets

$N \times N$ grid ($N$ is odd) and that their wireless transmission range is $\sqrt{2}$ grid units. The data dissemination protocol followed is fairly simple and is described schematically in Figure 3 for a $7 \times 7$ grid network. Higher order networks can be analyzed analogously. For simplifying the analysis, the base station is assumed to be situated in the center of the network grid. The following notation will be used in the rest of this section:

- $P$: is the size of a single update packet in bytes.
- $PL$: is the size of the packet payload in bytes.
- $PO$: is the size of the packet overhead in bytes.
- $K$: is the number of packets comprising the software update package.
- $M$: is the size of a MAC in bytes. Typically $M$ is 20 bytes if using a MAC algorithm based on the SHA-1 hash function such as HMAC.
- $B$: is the size of the Bloom filter in bytes.
- $NT$: is the number of nodes transmitting update packets in the $N \times N$ network grid.
- $NR$: is the number of nodes receiving update packets in the $N \times N$ network grid (typically this number is $N^2 - 1$ since all the network nodes should receive the update image)
- $NP$: is the number of nodes processing update packets in the $N \times N$ network grid (typically this number is $N^2 - 1$ since all the network nodes should process the update image)
- $NU$: is the number of nodes receiving update packets not destined to them due to the broadcast nature of the wireless medium. We assume that these packets are dropped by the node after receiving 6 bytes of the packet header.
- $ETB$: is the energy consumed when transmitting 1 byte in $\mu J$.
- $ERB$: is the energy consumed when receiving 1 byte in $\mu J$.
- $EPB$: is the energy consumed when processing 1 byte in the SHA-1 hash function in $\mu J$. Note that $EPB$ is expressed in terms of SHA-1 energy cost due to the fact that SHA-1 is the major building block of the MAC algorithm and the Bloom filter data structure.
- $SHA\_In\_MAC$: is the number of SHA-1 mechanisms required to produce a MAC. This value is 2 for the HMAC algorithm.
SHA_In_Bloom: is the number of SHA-1 mechanisms required to generate a Bloom filter. PETRA uses 3 SHA-1 hash functions to generate the Bloom filter representation.

$ET$: is the network-wide energy consumption due to data transmission in $\mu$J.

$ER$: is the network-wide energy consumption due to data processing in $\mu$J.

$EP$: is the network-wide energy consumption due to data processing in $\mu$J.

$E_{Total}$: is the network-wide energy consumption due to transmission, receiving, and processing in $\mu$J.

$\Phi$: is the false positive probability experienced by the bloom filter. $\Phi$ mainly depends on the size of the bloom filter $B$ and the number of packets $K$ in the update package.

$S_r$: is the percent saving in network energy consumption achieved due to applying the PETRA security protocol.

5.1 Network-Wide Energy Consumption Using the Traditional Security Approach

As mentioned previously, a conventional security approach for ensuring the authenticity of update packets is to append the MAC of each individual packet and the MAC of the whole update image to the package data. The packets, their MACs, and the MAC of the complete software image, are transmitted over the network to the designated clients. Using this model:

$E_{Total} = ET + ER + EP$

$ET$ is obtained by calculating the total number of bytes transmitted multiplied by the energy to transmit 1 byte $⇒$

$ET = NT \times (K \times P + (K + 1) \times M + PO \times \left( \frac{(K + 1) \times M}{PL} \right)) \times ETB$

where

$NT = \sum_{i=1}^{N} 8 \times i$

$ER$ is obtained by calculating the total number of bytes received multiplied by the energy to receive 1 byte $⇒$

$ER = NR \times (K \times P + (K + 1) \times M + PO \times \left( \frac{(K + 1) \times M}{PL} \right)) \times ERB + NU \times 6 \times \left( K + \left( \frac{(K + 1) \times M}{PL} \right) \right) \times ERB$

where

$NU = 40 + \sum_{i=2}^{N} (8 \times i - 4) \times 7 + 20$ for $N \geq 7$

$EP$ is obtained by calculating the total number of bytes processed by all the SHA-1 constructs multiplied by the energy cost to process one SHA-1 byte $⇒$

$EP = 2 \times NP \times K \times PL \times SHA\_In\_MAC \times EPB$

5.2 Network-Wide Energy Consumption Using the PETRA Security Approach

Similar to the analysis presented in Section 5.1,

$E_{Total} = ET + ER + EP$

Where

$ET = NT \times (K \times P + (B + 2 \times M) + PO \times \left( \frac{(B + 2 \times M)}{PL} \right)) \times ETB$

$ER = NR \times (K \times P + (B + 2 \times M) + PO \times \left( \frac{(B + 2 \times M)}{PL} \right)) \times ERB$

$EP = NP \times K \times PL \times SHA\_In\_Bloom \times EPB$

The network-wide energy saving $S_r$ is given as follows:

$S_r = \frac{E_{Total}(traditional) - E_{Total}(PETRA)}{E_{Total}(traditional)} \times 100$

5.3 Network-Wide Energy Saving

The network-wide energy saving $S_r$ is calculated as follows:

The false positive probability $\Phi$ of a bloom filter is related to $K$ and $B$ as follows:

$\Phi \approx 0.6185 \frac{K}{B} \quad [16]$

The graph presented in Figure 2. a, shows that the percent savings in network energy consumption increases with the number of update packets. The percent savings can achieve a 28.16% gain for $B=38$ and $K=30$. This results in a false positive probability of 0.76% as shown in Figure 2. d. The graph presented in Figure 2. b plots the percent savings in network energy consumption vs. the ratio of $ETB$ to $EPB$, with $ERB=0.5ETB$. This graph shows that the savings in network energy consumption dramatically increases with the increase in the $ETB/EPB$ ratio. In Figure 2. c, the graph shows the effect of network size on the savings in network consumption. As the network size increases, $S_r$ slightly increases. For a 17x17 grid network, $S_r$ reaches 29.3%.

6. SIMULATION MODEL

This section presents the simulation model we considered to experimentally test the secure update protocol performance. The update clients are micaz wireless sensor nodes organized in an $N \times N$ grid sensor network and running the TinyOS v2 operating system. TinyOS is an open source operating system for limited-resource WSNs. It is considered the reference standard for developing embedded WSN applications and protocols. TinyOS provides a component-based model that aids in the development of rapid and maintainable sensor applications. We simulated the sensor network using the TOSSIM simulator. TOSSIM is a discrete event simulator that ships with the TinyOS library. It allows the simulation of full-fledged TinyOS networked applications. As stated in Section 5, the data dissemination protocol followed in the simulation is described schematically in Figure 3. Note that the WSN base station is situated in the middle of the grid. The “meyer-heavy” noise model was used in the simulation. Figure 4 presents the percent savings in network-wide energy consumption for different network sizes. The simulation
configuration employed the following attribute values: \( P = 38 \), \( B = 38 \), \( K = 30 \), \( PO = 8 \), \( PL = 30 \), \( ETB = 10 \text{EPB} \), \( ERB = 5 \text{EPB} \), \( SHA \text{ In MAC} = 2 \), \( SHA \text{ In Bloom} = 3 \). The energy savings increases with the network size to reach nearly 32% for a 17x17 grid network. Note that the simulation results obtained in Figure 4 surpasses the theoretical results in Figure 2 c. This is mainly due to dynamic network conditions and packet retransmissions which gives PETRA an extra advantage in reducing network traffic compared to the traditional security approach. Figure 5 compares the update dissemination time using the PETRA and conventional security approaches for different network sizes. The same simulation configuration presented in Figure 5 is used. This graph demonstrates the advantages PETRA achieves in reducing the update dissemination time which supports the scalability of the proposed solution. The simulated results recorded are obtained by implementing special monitors within each network node to calculate the inbound network traffic entering the node, the outbound network traffic leaving the node, and the amount of data processed by the cryptographic constructs within each node. Moreover, the monitors provide power measurements by automatically mapping the data transmission, receiving, and processing operations to their corresponding energy consumption units.

7. RELATED WORK

A considerable amount of research work has dealt with the problem of efficient code updates in embedded environments and particularly in WSNs. Most of this research focused on the more general problem which is efficient data dissemination protocols. This section will discuss the most important protocols available. Of these protocols we can mention Deluge [1] which is a very popular data dissemination protocol for reliable broadcast in WSNs, AdapCode [2] which uses network coding to reduce broadcast network traffic in the process of code updates, and SPIN [3] which uses meta-data negotiations and application specific knowledge of data to avoid the transmission of redundant data packets. It should be noted that using the flooding technique in data dissemination is considered inefficient and may lead to excessive drain in battery resources. Moreover, it is worth mentioning that using the popular data aggregation technique to reduce network traffic in WSNs is infeasible in software update applications. This is due to the fact that in such applications receivers are required to consume an exact copy of the original update package and not an approximated aggregation result. In [4] Reijers and Langendoen proposed an incremental differential update mechanism to reduce the network traffic resulting from transmitting complete software images. The main concept here is to create and transmit an “edit script” that generates the difference between the current code version and the new code version instead of transmitting the complete software image. The above technique suffers from two main drawbacks: firstly, this approach is limited since in most cases the new binary software version is highly dissimilar from the current one (even if the source code is relatively similar) and thus it is more feasible to transmit the whole software image in this case. Note that this fact is supported by the TinyOS default code distribution mechanism which operates by completely replacing the old software version with the new software version. A second drawback of the incremental update protocol is that if a node loses one of the differential updates it become out of sync. This second drawback was targeted by the work in [5], however, the first drawback is still hindering the adoption of the incremental update approach. Some
workarounds for the first drawback of the incremental update approach is to run a small virtual machine or a dynamic linker [6, 7, 8] on the sensor node. In such a case the new binary image represented in high-level virtual machine instructions would be closer to the old binary image. This approach suffers from a high runtime overhead resulting from the additional layer of virtualization on top of the native sensor OS.

Note that all the approaches discussed above do not take into account the security requirements of software update mechanisms. In [9], Bellissimo et al. presented a survey of available secure software update applications and emphasized the challenges facing software updates on embedded devices. They gave examples of some popular software update mechanisms that were vulnerable to weak man-in-the-middle attacks. [10, 11, 12] presented a set of protocols based on authenticating update images using digital signatures, and cryptographic hash functions, and Merkle hash trees. However, the approach proposed in all these protocols is to add different forms of cryptographic hash authenticators to each update packet. This increases the network traffic necessary to execute the data dissemination process and thus increases energy losses when the update image is relatively large. Note here that PETRA’s advantage in reducing network traffic makes it a suitable choice for modular operation in existing WSN security protocols that provide hybrid protection mechanisms. For example [11] provides, in addition to the code update authentication service, a denial of service resistance service. The authentication service in such protocol could be substituted by PETRA’s integrity and authentication services to achieve elevated network-wide energy savings. It should be noted here that public-key approaches (even ECC-based ones [13]) to secure software updates in WSN are still very computationally-intensive and thus infeasible in embedded environments.

8. CONCLUSION

In this paper we presented PETRA, an energy-efficient security protocol for enforcing the integrity of software update components in embedded environments. PETRA employs efficient cryptographic mechanisms and data structures to reduce the network traffic necessary to carry out the software update process. Moreover, PETRA realizes an incremental security verification mechanism that allows the eager loading of received software components, thus preventing service disruption during the code update process. An experimental simulation on the TinyOS platform shows that PETRA can practically achieve an average of up to 30% reduction in network energy consumption.

9. REFERENCES


