Poster Abstract: Secure Dissemination of Code Updates in Sensor Networks*

Patrick E. Lanigan
Information Networking Inst.
Carnegie Mellon University
Pittsburgh, PA, 15213
lanigan@cmu.edu

Rajeev Gandhi
ECE Department
Carnegie Mellon University
Pittsburgh, PA, 15213
rgandhi@ece.cmu.edu

Priya Narasimhan
ECE Department
Carnegie Mellon University
Pittsburgh, PA, 15213
priya@cs.cmu.edu

ABSTRACT

Existing code update protocols target efficiency and assume correct behavior from participating sensor nodes. This work aims for the progressive, resource sensitive verification of code updates in sensor networks to ensure that unauthorized updates from malicious nodes are not propagated, while correct updates continue to be efficiently disseminated.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Sensor Networks; C.2.3 [**Network Operations**]: Network Management

Keywords

security, sensor networks, network programming, code dissemination

General Terms

security, verification, reliability

1. MOTIVATION

Given the long lived nature of wireless sensor networks and an increasing need to be able to debug/upgrade their software, network (re)programming and code dissemination have emerged as important issues. A number of protocols [3, 5, 8] have been developed to facilitate over-the-air software updates. For the most part, these protocols have focused on providing reliable data dissemination with low resource consumption and latency. They ensure that code updates are eventually propagated to all of the sensors in the system, but do not place any bounds on propagation time.

These protocols typically use publish-subscribe mechanisms [8] or three phase advertisement-request-data handshakes [3, 5] to signal the availability of a new update. Protocols such as Deluge [3] and MNP [5] divide a program image into equal sized *fragments* called pages. This permits *pipelining*, i.e., piece-wise dissemination, where sensors need not wait to receive the entire program image but can start to forward image fragments to other sensors. Pipelining allows for the efficient dissemination of new code updates.

*Funded by ARO grant DAAD19-02-1-0389 to the Center for Computer and Communications Security at CMU.

SenSys'05, November 2–4, 2005, San Diego, California, USA.
ACM 1-59593-054-X/05/0011.

From a reliability perspective, the code update protocols tend to assume well behaved or correct sensors, and are primarily tolerant to fail-stop nodes and packet losses. A malicious sensor node can cause serious disruptions to the code update process. Efficient network reprogramming mechanisms such as pipelining are clearly susceptible to abuse since both correct and malicious nodes can equally exploit them. While a correct node can propagate its updates quickly through the network, so too can a malicious node. An adversary who is able to inject packets into the network can hijack the update mechanism by propagating arbitrary code images efficiently. Not only can the adversary thereby accomplish the wide spread, rapid installation of corrupt code, but it can also drain energy and bandwidth, both of which are valuable in resource constrained sensor networks.

2. PROBLEM STATEMENT

The primary focus of our work is to address the following research question: How do we enable the *progressive, resource-sensitive verification of code updates* in sensor networks so that malicious/corrupt updates are not propagated or installed, while correct updates can continue to exploit the efficiency mechanisms?

We require the piece-wise verification of fragments, as they are propagated, instead of waiting for the entire program image to be assembled before we can detect a malicious update. Thus, good program images/fragments should be quickly propagated while malicious ones should be quickly halted. We require that the program update originate from a trusted source, such as a base station. We desire to be efficient for authorized updates so that mechanisms such as pipelining can be used for faster propagation of correct fragments. We desire resource sensitivity, and aim to amortize security costs over an entire program image and to impose modest transmission and processing overheads compared with insecure update protocols.

Candidate Solutions. We first investigated potential solutions based on other relevant security research. Techniques such as SWATT [7] verify the program image and memory contents of embedded devices, and can be useful for the post update verification of an entire program image; we are, however, interested in the progressive verification of fragments of a program image while the update is in progress. Symmetric key protocols like TinySec [4] and SNEP [6] prevent eavesdropping, message tampering, and message injection by outsiders, but do not protect against

compromised nodes. μTESLA [6] uses symmetric primitives with delayed key disclosure to provide authenticated broadcast, but requires loose time synchronization across the network. μTESLA's timing requirements are inappropriate for current dissemination protocols that do not place time bounds on the dissemination process. While asymmetric cryptography has long been thought to be impractical on severely constrained sensor nodes, recent work [9, 2] shows that with careful implementation and judicious use, public key cryptography is quite feasible.

3. OUR APPROACH

We assume that (i) individual sensor nodes can become compromised, giving an adversary complete control of the compromised node's functionality and cryptographic material, (ii) there exists a single, trusted authority that is hardened against compromise, and from which an authorized update will originate, and (iii) each node in the network is preloaded with this trusted authority's public key, or there exists a secure key distribution mechanism.

In keeping with code update protocols that aim for the eventual dissemination of a new image, we make no assumptions regarding time synchronization. As a departure from previous work that did not account for malicious nodes, we do not require each node to eventually install the image that it receives. Instead, we guarantee that every correct node will always be executing *some* correct program image.

Using a digital signature to verify every single program fragment might be too expensive. Using a single digital signature over an entire program image would require each node to receive the entire image before verifying the source, which would preclude the use of pipelining.

We follow an approach based on digital signatures and off-line hash chains [1]. Given n fragments $\{p_0, p_1, \ldots p_n\}$ of a program image, the trusted base station constructs the hash chain as $h_i = \texttt{HASH}(p_i|h_{i+1})$ where `HASH` is a collision resistant one-way function. The value h_i is distributed with fragment p_{i-1}. The base station signs h_0 and appends the signature σ to its initial advertisement. A node receiving the advertisement for p_0 extracts the appended signature, σ, which serves as a commitment to the entire hash chain. To verify the authenticity of an update, a receiving node compares the σ appended to the initial advertisement with the hash of p_0, and the hash of each subsequently received fragment with the hash-value appended to the previous fragment, as shown in Figure 1.

Evaluation and Tradeoffs. To inject a malicious fragment p_i, an adversary would need to find a collision for the hash value h_i contained in p_{i-1}. Signing the initial hash value h_0 ensures that updates can only originate from a single trusted source. The transmission overhead of our approach amounts to a single hash value per fragment. The processing overhead is the time to compute a single hash per fragment plus a single digital signature verification per program image. This approach allows us to *amortize the cost of a single digital signature over multiple fragments, the cost of a hash over several packets*, and *exploits the hash chain construction to allow for both incremental program verification and pipelining.*

We are currently implementing our progressive verification algorithm to enhance an existing code update protocol with security. We will evaluate the secure protocol's performance, on a sensor testbed at CMU, to quantify the impact of our security mechanism with respect to propagation latency and energy usage in the presence of malicious nodes. We will study the impact of our scheme with pipelining turned on/off, progressive vs. wholesale image verification, etc.

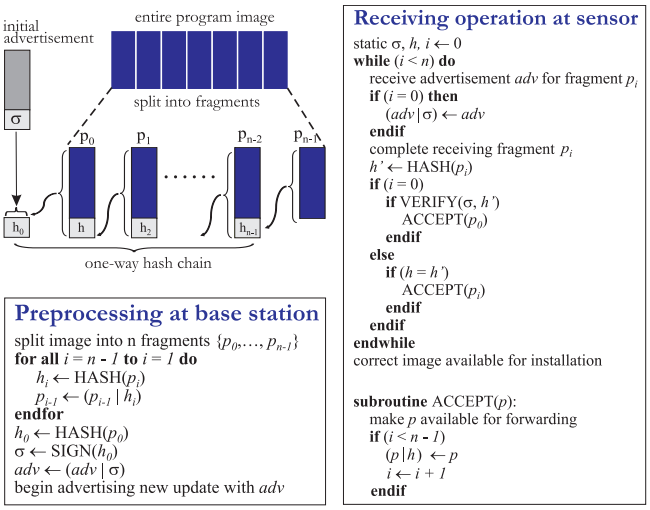

Figure 1: Progressive verification for code updates.

There are additional optimizations that we might make, for further resource conservation and efficiency. Our focus here has been on securing update protocols to ensure that correct sensors will never install unauthorized code updates. In our future work, we will aim to secure other aspects of the code update process, e.g., integrity of advertisements and of requests for missing fragments. Our ultimate aim is to develop a new update protocol that is robust against a range of threats, including sinkhole or selective forwarding attacks that could partition/impede the update process.

4. REFERENCES

[1] R. Gennaro and P. Rohatgi. How to sign digital streams. *Information and Computation*, 165(1):100–116, 2001.

[2] V. Gupta et. al. Sizzle: A standards-based end-to-end security architecture for the embedded internet. In *PERCOM '05*, pages 247–256, 2005.

[3] J. W. Hui and D. Culler. The dynamic behavior of a data dissemination protocol for network programming at scale. In *SenSys '04*, pages 81–94, New York, NY, USA, 2004. ACM Press.

[4] C. Karlof et. al. TinySec: a link layer security architecture for wireless sensor networks. In *SenSys '04*, pages 162–175, New York, NY, USA, 2004. ACM Press.

[5] S. S. Kulkarni and L. Wang. MNP: Multihop network reprogramming service for sensor networks. Technical Report MSU-CSE-04-19, Department of Computer Science, Michigan State University, East Lansing, Michigan, May 2004.

[6] A. Perrig et. al. SPINS: security protocols for sensor networks. *Wireless Networks*, 8(5):521–534, 2002.

[7] A. Seshadri et. al. SWATT: Software-based attestation for embedded devices. In *IEEE Symposium on Security and Privacy*, pages 272–282, 2004.

[8] T. Stathopoulos et. al. A remote code update mechanism for wireless sensor networks. Technical Report CENS-TR-30, University of California, Los Angeles, Center for Embedded Networked Computing, November 2003.

[9] R. Watro et. al. TinyPK: securing sensor networks with public key technology. In *SASN '04*, pages 59–64, New York, NY, USA, 2004. ACM Press.