Lightweight Remote Image Management for Secure Code Dissemination in Wireless Sensor Networks

An Liu, Peng Ning
Department of Computer Science
North Carolina State University
Raleigh, NC 27695, USA
Email: {aliu3, pning}@ncsu.edu

Cliff Wang
U.S. Army Research Office
Research Triangle Park, NC 27709, USA
Email: cliff.wang@us.army.mil

***Abstract*—Wireless sensor networks are considered ideal candidates for a wide range of applications. It is desirable and sometimes necessary to reprogram sensor nodes through wireless links after they are deployed to remove bugs or add new functionalities. Several approaches (e.g., Seluge, Sluice) have been proposed recently for secure code dissemination in wireless sensor networks, all as security extensions to the state-of-the-art code dissemination system named *Deluge*. However, existing approaches all focused on securing the propagation of code images, but overlooked the security vulnerabilities in other image management aspects such as rebooting and erasing code images.**

In this paper, we identify the security vulnerabilities in epidemic image management in all existing solutions to secure code dissemination in wireless sensor networks. Such vulnerabilities allow an attacker to reboot a sensor network to undesirable images or erase critical images, exposing the network to security risks. We then develop a sequence of lightweight techniques to address these vulnerabilities. Our approach takes into consideration the limited resources on current sensor platforms, and removes the security vulnerabilities without introducing significant overhead. To evaluate the feasibility of our approach, we implement the proposed approach as a remote image management system named *Seluge-ImageMan*, which is intended to work with *Seluge*, a security extension to Deluge for injecting new code images. We perform a substantial set of experiments in the WiSeNeT sensor testbed, which consists of 72 MicaZ motes, to assess the performance overhead of Seluge-ImageMan. The experimental results indicate that our approach introduces very light overhead while completing the secure remote code image management solution for wireless sensor networks.

I. Introduction

Wireless sensor networks are considered ideal candidates for a wide range of applications, such as industry monitoring, data acquisition in hazardous environments, and military operations. It is desirable and sometimes necessary to reprogram the sensor nodes through wireless links after they are deployed due to the need of fixing bugs or adding new functionalities.

The process of propagating a new code image to the sensor nodes in a network is referred to as *code dissemination*. Several code dissemination protocols [6], [12], [15], [21], [23], [24] have been proposed for wireless sensor networks. Deluge [12] is generally accepted as the state of the art for code dissemination in wireless sensor networks, and has been included in recent TinyOS distributions [3].

In addition to propagating code images to sensor nodes, it is also necessary to manage the disseminated code images in sensor networks. Examples of the management features include rebooting sensor nodes to a new code image, probing the status of an active image, and erasing images that are no longer needed.

Several approaches have been proposed recently for secure code dissemination in wireless sensor networks [7], [8], [13], [16]. All of them are security extensions to Deluge [12]. However, they all focused on securing the propagation of code images, but overlooked the security vulnerabilities in the image management aspects of code dissemination, such as rebooting and erasing code images.

In this paper, we investigate the security issues in remote image management for code dissemination in wireless sensor networks. To maximize the impact on real-world applications, we focus on Deluge, the most commonly used code dissemination system in wireless sensor networks.

We first identify the security vulnerabilities in epidemic image management in all existing solutions for secure code dissemination in wireless sensor networks [7], [8], [13], [16]. Though these vulnerabilities are inherited from Deluge [12], not addressing them makes all the existing secure code dissemination approaches vulnerable to security attacks. Indeed, such vulnerabilities allow an attacker to reboot a sensor network to undesirable images or erase critical images, exposing the network to security risks. We then develop a sequence of lightweight techniques to address these vulnerabilities. Our approach takes into consideration the limited resources on current sensor platforms, and removes the security vulnerabilities without introducing significant overhead.

To evaluate the feasibility of our approach, we implement the proposed approach as a remote image management system named *Seluge-ImageMan*, which is intended to work with Seluge [13], a security extension to Deluge for injecting new code images. We perform a substantial set of experiments in the WiSeNeT sensor testbed, which consists of 72 MicaZ motes, to assess the performance overhead of Seluge-ImageMan. The experimental results indicate that our approach introduces very light overhead while completing the secure remote code image management solution for wireless sensor networks.

The contribution of this paper is three-fold. First, we identify the security vulnerabilities in epidemic image management in all existing secure code dissemination solutions [7], [8],

TABLE I
CODE IMAGE MANAGEMENT COMMANDS IN DELUGE [11]

Command	Functionality	Impact
Inject	Inject and propagate code image	epidemic
Reboot	Reboot all nodes to a specified code image in the external flash	epidemic
Erase	Erase the specified code image from the external flash	epidemic
Ping	Get info about the running image and images in the external flash	local
Reset	Reset the version for the specified image in the external flash	local
Dump	Dump the specified code image	local

[13], [16]. Second, we develop lightweight techniques to address these vulnerabilities and protect the epidemic image management in wireless sensor networks. Third, we implement a lightweight and secure image management system, Seluge-ImageMan, which works with Seluge to provide a complete and readily available solution for secure code dissemination in wireless sensor networks.

The remainder of this paper is organized as follows. Section II describes code image management in Deluge. Section III identifies the security vulnerabilities in Deluge not addressed by current solutions. Section IV discusses assumptions and threat model. Section V presents our lightweight secure image management techniques. Section VI gives the experimental evaluation of Seluge-ImageMan in the WiSeNeT testbed. Section VII discusses related work, and Section VIII concludes this paper.

II. CODE IMAGE MANAGEMENT IN DELUGE

Deluge assumes a deployment environment in which a PC is used to inject or manage code images in a network of sensors. In other words, the network owner interacts with the PC, which then interacts with the nearby sensor nodes to disseminate or manage the code images.

Commands: Deluge manages code dissemination through six commands, as shown in Table I. These commands can be divided into two categories: *epidemic commands* and *local commands*. The epidemic commands have impact on all sensor nodes in the network; any epidemic command can change the status of the entire wireless sensor network. The epidemic commands include "Inject", "Reboot", and "Erase". The local commands can only affect the node directly connected to (or one hop away from) the PC. The local commands are introduced for convenience, and are not an essential part of code dissemination.

Metadata of Code Images: Deluge uses the external flash on sensor nodes to store multiple code images. Assume Deluge allocates m code image slots in the external flash and uses index number 0 through $m-1$ to identify them. Each code image slot can store one code image. We use IMG_i, $0 \leq i \leq m-1$, to represent the code image stored in slot i. To facilitate the management of code images on sensor nodes, Deluge introduces two metadata structures: `Node Description` and `Image Description`. As Figure 1 shows, `Node Description` includes the information of the current running code image on the sensor node, while `Image Description` consists of the information of a code image in the external flash. Thus, on each sensor node, there are one copy of `Node Description` and m copies of `Image Description`.

Node Description
uid
vNum
imgNum
crc

Image Description
uid
vNum
imgNum
numPgs
numPgsComplete
crc

Fig. 1. Metadata structures in Deluge

In these metadata structures, `uid` is a unique identifier of code image. `vNum` is the version number, and reflects the freshness of the code image. `imgNum` is the index of code image slot in the external flash. `numPgs` is the total number of pages of code image IMG_{imgNum}. `numPgsComplete` is the number of pages received by this sensor node for IMG_{imgNum}. `crc` is the cyclic redundancy checksum of `Node Description` or `Image Description`.

Advertisement and Epidemic Commands: Deluge uses advertisement packets to control the epidemic behaviors triggered by commands "Inject", "Reboot", and "Erase". Each node periodically broadcasts its own metadata structures through advertisement packets to declare the information of the current running image and those in the external flash. Each advertisement packet contains the `Node Description` and one `Image Description` of the sender. Once a node receives an advertisement packet, it compares the `vNum` field in its own metadata structures with the ones in the advertisement packet. If the `vNum` of its own `Node Description` is smaller than the corresponding one in the advertisement packet, a "Reboot" command has been issued. If the `vNum` of its own `Node Description` is the same as the corresponding one in the advertisement packet, but the `vNum` of the receiver's `Image Description` is smaller than the corresponding one in the advertisement packet, an "Erase" or "Inject" command has been issued, depending on whether the `numPgs` is zero or not.

III. SECURITY VULNERABILITY IN DELUGE IMAGE MANAGEMENT

Deluge was not designed with security in mind. A number of approaches [7], [8], [13], [16] have been developed to provide security protection for Deluge in hostile environments. However, all these approaches focused on the protection of the "Inject" command, but overlooked the other image management issues.

Vulnerability of "Reboot" and "Erase" Commands: The "Reboot" and "Erase" commands are both epidemic commands, which affect all nodes in the network. Despite the existing security solutions for securing the "Inject" command, there is no mechanism to authenticate the "Reboot" and "Erase" commands to all sensor nodes. As a result, an attacker may exploit this vulnerability to boot the entire network to

a wrong code image, or erase an important image from all sensor nodes. For example, to boot to a wrong image IMG_i, the attacker simply needs to send out a fake advertisement packet with a large enough version number (i.e., larger than the one in the `Node Description` on all sensor nodes) and a wrong image number i. Any receiving node will accept this `Node Description` due to the large version number, update its own `Node Description`, and reboot to image IMG_i.

Similarly, to erase a code image IMG_i (except for the golden image) on all sensor nodes, the attacker simply needs to send a fake advertisement packet with a large enough version number in `Image Description` and 0 in fields `numPgs` and `numPgsComplete`. Any receiving node will accept this `Image Description` and erase the code image accordingly.

Note that even if the advertisement packets are authenticated (e.g., using cluster key as in [7], [13]) between neighbor nodes, the attacker can still launch such attacks by compromising a single node and injecting faked commands through the compromised node.

Vulnerability of Local Commands: The "Ping" and "Dump" commands only obtain, but do not change, the status of the sensor network. Thus, the attacker can make limited impact on the network even if he can exploit these commands. The "Reset" command can only change the status of the node connected to the PC directly. All three local commands can be authenticated using cluster keys as suggested in [7], [13] to eliminate external attacks. Even if the attacker can compromise a node and get the cluster key, the impact is still constrained in a small region.

Need for Lightweight Secure "Reboot" and "Erase": The above discussion suggests that it is necessary to provide security mechanisms to protect the remaining two epidemic commands ("Reboot" and "Erase") in order to have truly secure code dissemination in wireless sensor networks. The fundamental reason for the vulnerability is the lack of broadcast authentication. However, straightforward solutions (e.g., digital signatures) immediately introduce significant performance overhead. In the rest of this paper, we investigate lightweight mechanisms that can authenticate "Reboot" and "Erase" efficiently.

IV. Assumptions and Threat Model

Assumptions: We assume the source of the code images, i.e., the *base station*, is a powerful node (e.g., a laptop). The base station is trusted and cannot be compromised by attackers. We assume that sensor nodes are resource constrained. A sensor node can perform a limited number of Public Key Cryptographic (PKC) operations, but cannot perform many such operations due to limited energy. Each node has enough memory (i.e., external flash) for storing a few program images. We assume that the propagation of code images is protected by Seluge [13] to have a specific target. (Other schemes are possible, but may lead to small variations in the proposed approach.) We assume the base station has a private/public key pair. Each node in the sensor network is pre-configured with the base station's public key.

Threat Model: We assume attackers has powerful nodes (e.g., laptops). Attackers can launch both *external* and *internal attacks*. In an external attack, the attacker does not control any node in the network, but can eavesdrop packets, inject bogus packets, and replay packets. The attacker can also launch wormhole attack [10], Sybil attacks [22], and Denial of Service (DoS) attacks. In an internal attack, the attacker can compromise some sensor nodes, and attack the rest of the network by using the sensitive information in the compromised nodes. The attacker can also control the compromised nodes not to cooperate with other nodes (e.g., inject bogus packets, drop packets).

V. Secure Remote Image Management

In this section, we present our approach for secure remote code image management. Our techniques are applicable to remote image management in general. However, to make these techniques concrete, we use Deluge image management as a specific target in this paper.

Deluge includes three epidemic commands (i.e., "Inject", "Reboot", and "Erase") in advertisement packets for code dissemination and remote image management. The protection of the "Inject" command has been addressed in recent papers [7], [8], [13], [16]. Thus, we focus on the protection of "Reboot" and "Erase" in the following discussion.

As discussed earlier, the fundamental reason for the vulnerabilities related to "Reboot" and "Erase" is the lack of broadcast authentication. A naïve approach is to have the source node sign the metadata structures in each advertisement packet using a digital signature. Unfortunately, each receiving node has to verify the signature from the source node whenever it receives a new advertisement packet. This not only leads to long delay due to the expensive signature verification, but also introduces the possibility of DoS attacks.

In the following, we present a sequence of lightweight schemes to protect "Reboot" and "Erase". Each later scheme addresses the limitation of the previous one and improves over it. The treatments of the "Reboot" and "Erase" commands are similar. The difference is that a code image may be rebooted multiple times, while each image can be erased only once. This difference implies that the mechanism for "Reboot" is more complex than that for "Erase". In this section, we use "Reboot" as the main target to develop our approach. The resulting approach is also applicable to "Erase" with simplification.

A. Scheme 1: Key Chain-based Image Management

Our approach is based on a simple observation: To reboot or erase a code image, we only need to authenticate to all sensor nodes a reboot or erase *signal* for a code image that has *already* been distributed to all nodes. This is in contrast to authenticating a block of new data (e.g., inject a new code image), where we not only need to authenticate the existence of the data, but also its content.

Based on this observation, we propose a lightweight approach to authenticate the "Reboot" command using one-way key chains. The basic idea is to use a dedicated one-way key chain for each code image IMG_i to authenticate instances of rebooting to this image, with each key representing a reboot instance.

Key Chain Generation: Before injecting a new code image IMG_i, $i \geq 0$, into the network, the source node generates a one-way key chain for IMG_i, as shown in Figure 2. Specifically, the source node randomly generates a key $K_{i,n}$ as the last key for the key chain, and repeatedly performs a one-way function F to compute all the other keys: $K_{i,j} = F(K_{i,j+1}), 0 \leq j \leq n-1$.

$$K_{i,0} \xleftarrow{F} K_{i,1} \xleftarrow{F} \cdots \xleftarrow{F} K_{i,n-1} \xleftarrow{F} K_{i,n}$$

Fig. 2. One-way key chain for code image IMG_i (F: One-way function)

With the one-way function F, given $K_{i,j}$, anybody can compute all the previous keys $K_{i,l}, 0 \leq l \leq j$, but it is computationally infeasible to compute any of the later keys $K_{i,l}, j+1 \leq l \leq n$. Thus, with the knowledge of F and the initial key $K_{i,0}$, which is called the *key chain commitment*, a sensor node can authenticate any key in the key chain by merely performing one-way function operations.

Commitment Distribution: In order to use the one-way key chain to authenticate the "Reboot" commands for code image IMG_i, we need to distribute the key chain commitment $K_{i,0}$ to all the sensor nodes. Note that the code image IMG_i has to be disseminated to all nodes before they can reboot to IMG_i. Thus, it is natural and convenient to distribute the commitment $K_{i,0}$ along with the code image when it is injected into the network. For example, Seluge [13] bootstraps the injection of a new code image with a signature packet, in which various parameters (e.g., version number) are included and authenticated. Thus, the signature packet is an ideal place to distribute and authenticate the key chain commitment in the case of Seluge. The same argument applies to other approaches such as [7], [8], [16], which all use a signature packet to bootstrap the injection of a new code image.

Reboot through Key Disclosure: When the source node needs to reboot all sensor nodes to a code image IMG_i, it includes an undisclosed key in the key chain for IMG_i along with its index in an advertisement packet. This advertisement packet will be propagated throughout the sensor network due to the epidemic property of Deluge [12].

Assume the sensor nodes have rebooted to code image IMG_i for j_i times. (Before the first reboot to IMG_i, $j_i = 0$, and only the key chain commitment $K_{i,0}$ has been disclosed.) We use the disclosure of key K_{i,j_i+1} to signal the reboot to IMG_i for the (j_i+1)-th time. In the key chain for code image IMG_i, all keys before K_{i,j_i}, including $K_{i,j+i}$, have been disclosed. Assume the current running code image is IMG_x. If the source node wants to reboot all sensor nodes to code image IMG_i, it first gets the current `Node Description` meta-data structure

$$\texttt{uid}_x \| \texttt{vNum} \| x \| j_x \| K_{x,j_x} \| \texttt{crc},$$

then constructs the new `Node Description` for the "Reboot" command

$$\texttt{uid}_i \| \texttt{vNum} + 1 \| i \| j_i + 1 \| K_{i,j_i+1} \| \texttt{crc'},$$

and finally broadcasts the new `Node Description` in an advertisement packet.

Verification of "Reboot" Commands: Once a node receives the above `Node Description` in an advertisement packet, it first verifies if key $K_{i,j+1}$ has already been disclosed. Then it compares $H(K_{i,j+1})$ with $K_{i,j}$, which is disclosed in an advertisement packet for the last "Reboot" command. If $H(K_{i,j+1}) = K_{i,j}$, the "Reboot" command is authenticated, and the node updates its own `Node Description` with the new `Node Description` and reboots to IMG_i. This node will also broadcast the new `Node Description` in its own advertisement packets. As a result, the new `Node Description` will propagate to the whole network, and all nodes will reboot to IMG_i.

Key Chain Update: The one-way key chain for a given image may run out after multiple rounds of rebooting. A simple solution is to have the source node to generate another key chain for this code image, and distribute the commitment using a signature packet (without injecting a new code image). All sensor nodes that receive this packet will use the new key chain for rebooting purposes. Other more complex solutions are possible. For example, we may use multi-level key chains proposed in [19], [20] to address the key chain update problem.

Limitation: In this approach, an attacker cannot trigger the nodes to reboot to wrong code images, because it does not know the undisclosed keys in the key chains. However, the only value reflects the time order between the disclosed keys in different key chains is the version number `vNum` in `Node Description`, which is not authenticated by the source node. By manipulating this version number, the attacker can make nodes reject certain "Reboot" commands from the source node. The whole network may run inconsistent code images if the attacker launches such attacks. We address this limitation in the next scheme.

B. Scheme 2: Total Ordered Key Chains

The reason for the above attack is that the attacker can manipulate the version number in `Node Description`. To avoid authenticating the content of the `Node Description` (and increase the overhead as a result), we should not use version number `vNum` to indicate the freshness of `Node Description`. To solve this problem, we introduce a total order among the keys in different key chains, so that the receiving node can determine the freshness of received `Node Description` based on disclosed keys and their indexes only (rather than the version numbers).

Definition 1 (newer than $\gg$ and older than $\ll$ relations): Suppose K_{i_1,j_1} is the j_1-th key in the key chain for code image IMG_{i_1} and K_{i_2,j_2} is the j_2-th key in the key chain

for code image IMG_{i_2}. K_{i_2,j_2} is *newer than* K_{i_1,j_1}, denoted $K_{i_2,j_2} \gg K_{i_1,j_1}$, if

- $j_2 > j_1$, or
- $j_1 = j_2$ and $i_2 > i_1$.

Alternatively, we say K_{i_1,j_1} is *older than* K_{i_2,j_2}, denoted $K_{i_1,j_1} \ll K_{i_2,j_2}$ if $K_{i_2,j_2} \gg K_{i_1,j_1}$. □

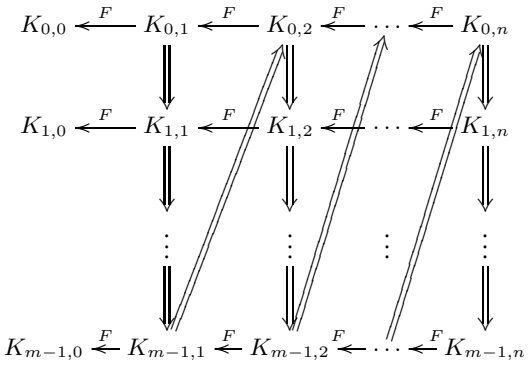

Fig. 3. Order among keys in different key chains

The "newer than" (or alternatively, the "older than") relation is a total order among the keys in all key chains, which we "force" on these keys. It defines the key disclosure order for different code images. The source node can simply follow this order to disclose keys to authenticate the "Reboot" commands included in advertisement packets. As Figure 3 shows, the disclosed key for a newly released "Reboot" command should always be newer than the key in the previous "Reboot" commands. For example, if key $K_{1,1}$ has been disclosed in the last round of reboot, and the source node wants to reboot the whole network to code image 3, it needs to include key $K_{3,1}$ in the new "Reboot" command. However, if the source node wants to reboot to the golden image (i.e., image 0), it needs to use key $K_{0,2}$ in the new "Reboot" command. Any receiving node will be able to determine whether the received "Reboot" command is newer or not by comparing the new key with the one received for the last reboot.

Because of the total order among the keys in different key chains, any receiving node can easily compare the freshness of different "Reboot" commands. As a result, the attacker cannot launch the wormhole attack described earlier.

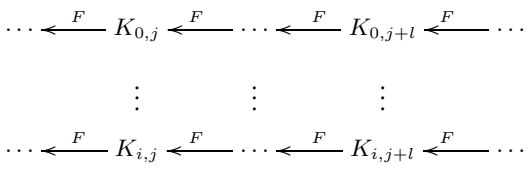

Fig. 4. Multiple hashing in reboot management

Limitation: Though better than Scheme 1, Scheme still has some limitations. Consider Figure 4. Assume the current active code image in the network is IMG_i, and the last active code image is IMG_0. Thus, the keys disclosed for rebooting to these images are $K_{i,j}$ and $K_{0,j}$, respectively. After sensor nodes reboot to IMG_0 l times without rebooting to IMG_i, the disclosed keys for IMG_0 and IMG_i are $K_{0,j+l}$ and $K_{i,j}$, respectively. If the source node wants to reboot the network to IMG_i, it needs to disclose key $K_{i,j+l}$. In order to verify $K_{i,j+l}$, all nodes will have to hash it l times. A large l will introduce long delays. More importantly, this leaves a security vulnerability: An attacker may send a "Reboot" command with a very large index. Even if the attacker does not have the right key to cause a real reboot, it will force all receiving nodes to perform a large number of hash operations for key verification.

C. Scheme 3: Virtual Key Chains with Sliding Window

A straightforward solution to address the above problem is to use a sliding window for each key chain, as Figure 5 shows. Assume sensor nodes have rebooted to code image IMG_i j times. After rebooting to other images a number of times, when sensor nodes receive a command to reboot to IMG_i again, only the key with index less than $j + (w - 1)$ will be verified, where w is the window size. Thus, the attacker cannot force the nodes to perform a large number of hash operations by forging a large index value.

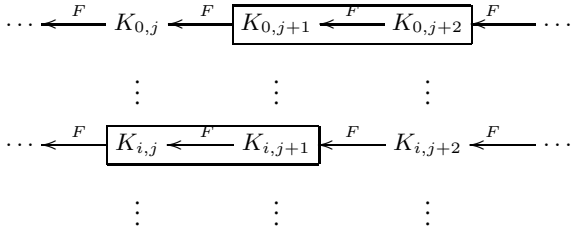

Fig. 5. Sliding window for one-way key chain (window size $w = 2$)

However, the simple sliding window approach may lead to rejection of authentic "Reboot" commands. Consider Figure 5, where the windows size is $w = 2$. Assume the currently active code image is IMG_i, with a disclosed key $K_{i,j+1}$. Also assume that the last time when IMG_0 was rebooted, the key $K_{0,j}$ was used. Now if the source node wants to reboot to IMG_0 again, the key $K_{0,j+2}$ must be used. However, it is already outside of the sliding window at $K_{0,j}$, and all receiving nodes will reject $K_{0,j+2}$.

One way to compensate is to have the source node disclose additional keys in the key chains that will be pushed out of the sliding windows due to a new "Reboot" command. However, this method will introduce additional communication overhead as well as more complications in the protection of these key disclosures.

We propose a more efficient method to address this problem. Intuitively, we would like to "simulate" multiple key chains for different code images using only one key chain. We assume that there are m slots for code images on each sensor node. In other words, each node can have up to m code images. We can use a single one-way key chain, which is called the *global key chain*, to replace the multiple key chains, as shown in Figure 6. By arranging the global key chain in a zigzag way, we simulate a *virtual key chain* for each code image. Specifically, each

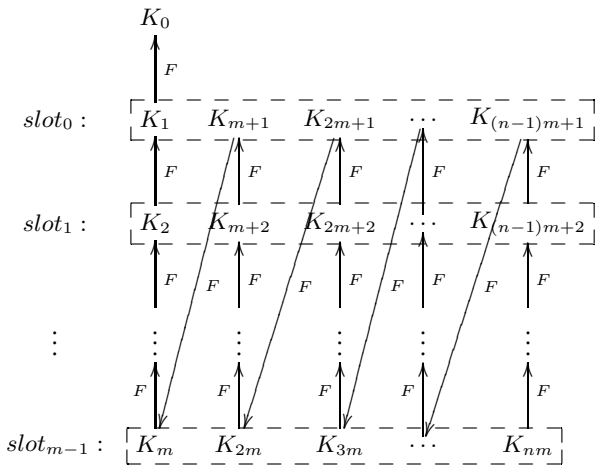

Fig. 6. Simulating multiple virtual key chains using a global key chain

code image IMG_i has a virtual one-way key chain consisting of $K_{i+1}, K_{i+1+m}, ..., K_{i+1+(n-1)m}$. Each sensor node only needs to save one commitment of the global key chain for all the virtual key chains.

When the source node needs to reboot the network to IMG_i, it broadcasts the next undisclosed key K_x in the global key chain that is also in the virtual key chain for IMG_i. There is no need to disclose keys in other virtual key chains at all. The sensor nodes can always verify the later disclosed keys in other virtual key chains, since all keys in virtual key chains are also ordered in the same global key chain.

An additional assumption we make in Scheme 3 is that there are a maximum number (m) of code images on each sensor node. This allows us to simulate multiple virtual key chains using one global key chain. This is a reasonable assumption, since all sensor platforms have an upper limit on the number of code images they can save. For example, TinyOS assumes a maximum of 3 code images on each node. The convenience offered by this assumption allows us to greatly simplify our scheme.

D. Analysis

Now we provide analysis of our final scheme, scheme 3.

Security Analysis: Our approach can properly authenticate all "Reboot" commands to the entire network. Due to the one-way property of the one-way key chain, even if the attacker knows the key chain commitment and all disclosed keys, he does not have access to any undisclosed keys, and thus cannot forge "Reboot" commands. The attacker may compromise some sensor nodes. However, as long as he does not compromise the source node, the attacker will not be able to forge any "Reboot" command. In addition, since there is a total order among all the keys used for reboot management, all sensor nodes will honor the most fresh key and determine the active code image accordingly. The epidemic property of Deluge will guarantee that all "Reboot" commands will be delivered to all reachable sensor nodes. As a result, all reachable nodes will run the same code image consistently.

By using the sliding window method, attackers cannot force the receiving nodes to perform a large number of hash operations by forging a large index for the disclosed key. Thus, even though our approach requires additional communication and computation, due to the above reason and the light overhead of hash operations, it does not expose the network to DoS attacks.

Performance Analysis: Our approach introduces very light performance overhead. To reboot different code images n times, the source node needs to perform hash function operations for at most $m \cdot n$ times to generate the global key chain. The source node needs to include this commitment (8 bytes) in a signature packet, possibly piggy-backed in a signature packet for injecting a new code image, and distributes it to the network. Thus, each reboot action requires at most m hash operations and $\frac{1}{n}$ of a signature operation in preparation.

To reboot to a given code image, the source node needs to include a previously undisclosed key and an index in an advertisement packet. This adds about 10 bytes to the advertisement packet. In normal situations, each node that receives a "Reboot" command needs to perform at most m hash operations in order to verify the disclosed key. Due to the efficiency of hash functions, our approach introduces very little overhead into the rebooting process.

In addition to theoretical analysis, we evaluate the performance overhead of our approach in a real testbed consisting of 72 MicaZ motes. The results are presented in Section VI.

E. Secure Erase Management

The above schemes are developed with reboot management as the main target. The complication is mainly due to the reason that each code image may be rebooted multiple times. In contrast, remotely managing erasure of a certain code image can be performed in a much simpler way. Indeed, no key chain is necessary to authenticate the "Erase" command. When a new code image IMG_i is injected into the network, we only need to randomly generate an authenticator, for example, e_i. We can then compute its hash image $F(e_i)$, where F is a one-way function. This value $F(e_i)$ can be included in the signature packet and authenticated to all the sensor nodes. When the source node needs to erase IMG_i, it only needs to broadcast e_i throughout the network. The sensor nodes can then compute $F(e_i)$ and compare with the value they receive from the signature packet. Since only the source node knows e_i, a match in the comparison is sufficient to authenticate the "Erase" command to the sensor nodes.

VI. Implementation and Evaluation

A. Implementation

We have implemented our secure image management scheme as an extension to Seluge [13], which is a secure code dissemination system based on Deluge [12]. For the sake of presentation, we call our secure image management system *Seluge-ImageMan*. Following Seluge, Seluge-ImageMan has both base station side and sensor side programs. The base station side programs are a set of Java programs expected to run

on a PC. They extend the Seluge Java tools to construct and inject new code dissemination packets into sensor networks, to reboot sensor nodes to a specified code image, and to erase a specified code image. The sensor side program is written in nesC [9] and runs on regular sensor nodes.

We use the 64-bit truncation of SHA-1 as the hash function F, which provides sufficient pre-image resistance and has been used previously (e.g., [8], [13]). For digital signature, which is required by Seluge, we use ECDSA over the 160-bit elliptic curve `secp160k1` [4]. On the base station side, we use the JCE provider in the Bouncy Castle Crypto APIs [1] for key generation and signature generation. On sensor nodes, we use ECDSA and SHA-1 in TinyECC [18] for signature and key verification.

We add the following functionalities in the Java tools on the base station side: (1) Generation and storage of the global key chain, (2) extraction of the reboot key for a specific code image from the global key chain, and (3) construction and broadcast of advertisement packets for "Reboot" and "Erase" commands. In the sensor side program, we modify the `DelugeM` module of the Seluge nesC library to perform "Reboot" and "Erase" key verification.

TABLE II
CODE SIZE (BYTES) ON MICAZ.

	ROM	RAM
Deluge	22,226	1,123
Seluge	45,258	2,278
Seluge-ImageMan	46,348	2,555
TinyECC in Seluge/Seluge-ImageMan	13,044	426

Table II shows the ROM and RAM usage of Seluge and Seluge-ImageMan on MicaZ motes. The code size of Deluge and that of TinyECC are also included as references. It is easy to see that Seluge-ImageMan only slightly increases the ROM and RAM consumptions compared with Seluge. Seluge and Seluge-ImageMan do increase both the ROM and RAM consumptions compared with Deluge, but the majority of the ROM increase is due to TinyECC.

B. Experimental Evaluation

We have provided an analysis of the security and performance properties of Seluge-ImageMan in Section V. In the following, we report the experimental evaluation of Seluge-ImageMan in a network of MicaZ motes [2]. For comparison purposes, we perform the same set of experiments with Deluge [12] and Seluge [13], since Seluge-ImageMan is an extension to Seluge and Seluge is an extension to Deluge.

Evaluation Metrics: We use two performance metrics in our evaluation: *Propagation delay* and *communication overhead*. There are three propagation delays in our evaluation: *Code* propagation delay, *reboot* propagation delay, and *erasure* propagation delay. Code propagation delay is the time required to finish disseminating a code image to all the nodes in the network. Reboot propagation delay is the time to reboot all the nodes in the network to a specified code image. Erasure propagation delay is the time to erase a specified code image from all the nodes in the network. As mentioned in [12], for performance reasons, Deluge requires that every node keep its radio on. Thus, these delays are closely related to the energy consumption required by code dissemination. The communication overhead is measured as the total number of packets transmitted by all the nodes during a code dissemination, which is also related to power consumption.

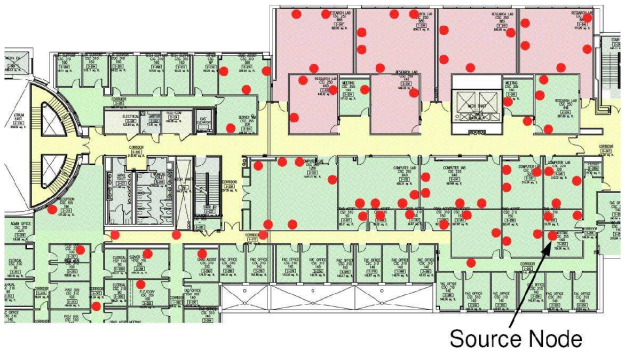

Fig. 7. The WiSeNet testbed (72 MicaZ motes; 152.5 feet $\times$ 97 feet).

Testbed: We perform the experiments in the WiSeNeT sensor testbed, which has 72 MicaZ motes at the time of experiments. Figure 7 shows the layout of the testbed. The sensor nodes are deployed on the second floor of Engineering Building II at North Carolina State University. The testbed area includes offices, labs, server rooms, and corridors, covering an area of 152.5 feet $\times$ 97 feet. Each node is equipped with an Ethernet programming board to provide remote access to the node. We only use the programming boards to gather evaluation results from the sensor nodes; they do not interfere with the radio communication between sensor nodes at all. We set the transmission power level of the MicaZ radio module (CC2420) as 0dBm.

Experiment Setup: We need to configure a number of parameters for Deluge, Seluge, and Seluge-ImageMan for the experiments. All parameters are set the same for the three schemes. We set the data packet payload size as 102 bytes. The advertisement packet is a special packet that contains `Node Description` and `Image Description`. Since we place the disclosed key and index for "Reboot" commands in `Node Description` and the commitment of global key chain into `Image Description`, the size of advertisement packet becomes 108 bytes. Although the maximum packet payload size defined in IEEE 802.15.4 [14] is 102 bytes, the actual maximum packet payload size of MicaZ can be up to 116 bytes. To integrate the security mechanisms and the Deluge propagation mechanisms, we have to make certain change to another parameter. Deluge by default uses a 2ms gap between the transmission of two data packets. However, a SHA-1 hash verification operation takes about 15ms. Thus, we increase the transmission gap from 2ms to 17ms to accommodate this requirement. Besides these parameters, all the other parameters remain the same as the default configuration in Deluge.

We use the code images of 4 programs (`Blink`, `RfmToLeds`, `CntToLedsAndRfm`, and `TestDrip`) as the

input of the “Inject”, “Reboot”, and “Erase” commands. The original sizes of these programs are 1,532, 9,442, 10,050, and 10,670 bytes, respectively. After compiled with Deluge, Seluge, and Seluge-ImageMan, their code images range between 25,602–26,494 bytes, 46,304–47,232 bytes, and 47,202–48,132 bytes, respectively. In each experiment, we first inject a new code image at the source node (see Figure 7), and then reboot the source node to the injected code image. After all nodes reboot to the new code image, we erase the new code image from the source node. For each program, we perform the same experiment 10 times and take the average for the performance metrics.

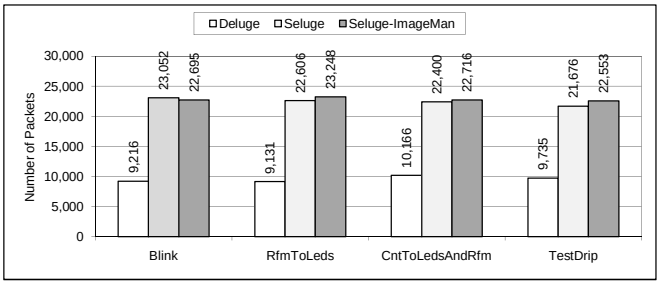

Fig. 8. Communication overhead (number of packets)

Communication Overhead: Figure 8 shows the communication overhead in all three code dissemination systems for the test programs. Deluge has the smallest communication overhead, while Seluge and Seluge-ImageMan have similar communication overheads, which are much larger than that of Deluge. Note that the code image includes both the target program and the code for the dissemination system (i.e., Deluge, Seluge, or Seluge-ImageMan). Since Deluge has no security mechanism, while Seluge and Seluge-ImageMan have TinyECC, Cluster Key, and Message Specific Puzzle modules, the code image generated for Deluge is much smaller than the one generated for Seluge or Seluge-ImageMan. Thus, the number of packets transmitted during the code image propagation in the network for Deluge is much smaller than that for Seluge or Seluge-ImageMan. An important thing to note is that Seluge and Seluge-ImageMan have similar communication overheads, mainly because the additional code size introduced by Seluge-ImageMan over Seluge is small.

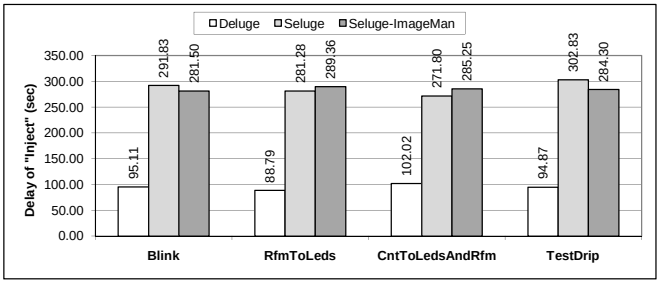

Fig. 9. Code propagation delay

Propagation Delays: Figure 9 shows that Seluge and Seluge-ImageMan have very similar code propagation delays.

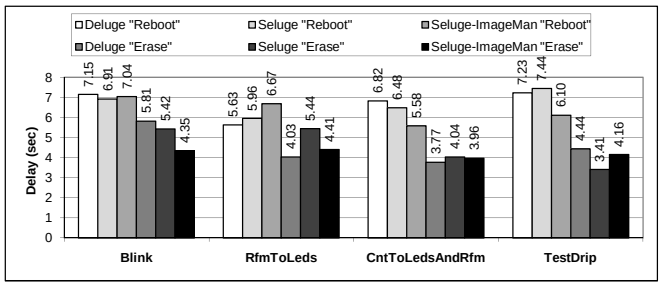

Fig. 10. Reboot and erasure propagation delays

In fact, they perform the same kind of operations during injection of new code images, though Seluge-ImageMan entails a slightly larger image size than Seluge for the same target program. Seluge and Seluge-ImageMan lead to much larger code propagation delays than Deluge. This is because the image sizes generated for Seluge and Seluge-ImageMan are significantly larger than that for Deluge.

Figure 10 shows the reboot and erasure propagation delays. It seems strange at first glance that none of Deluge, Seluge, and Seluge-ImageMan can guarantee smaller reboot or erasure propagation delay than the other two schemes for all 4 test programs.

We take a closer look at these delays. For reboot propagation delay, the additional verification step of Seluge-ImageMan compared with Seluge on each node is up to m hash operations. Since a SHA-1 hash operation takes about 15ms, the additional verification of Seluge-ImageMan takes at most 45ms, assuming each node has 3 code image slots (default value on TinyOS). The advertisement packet of Seluge-ImageMan has 20 more bytes than that of Seluge. Given the 250 kbps data rate of MicaZ’s radio chip CC2420 [5], the additional communication time of Seluge-ImageMan’s advertisement packet is about $\frac{20\times 8}{250} = 0.64ms$. These additional costs will not cause observable changes of reboot and erasure propagation delays. In contrast, the dynamics of wireless communication have more impact on reboot and erasure propagation delays.

The experimental results indicate that Seluge-ImageMan has very similar performance overhead to Seluge.

VII. Related Work

Code dissemination is a critical issue to enable efficient tasking of wireless sensor networks. A few code dissemination protocols (e.g., [6], [12], [15], [21], [23], [24]) have been developed recently to propagate new code images using the ad-hoc wireless network formed by sensor nodes. In particular, Deluge [12] uses an epidemic protocol [17] for efficient advertisement of code meta data and spatial multiplexing for efficient propagation of code images.

Integrity of disseminated code images is crucial to ensure the success of code dissemination. Lanigan et al. proposed a protocol named Sluice [16] and Secure Deluge [8] were independently proposed to integrate signature and hash functions to provide efficient authentication of disseminated code images. Deng et al. proposed a scheme to improve the DoS-resilience

of secure code dissemination by integrating signatures, hash chains, and hash trees [7]; the use of hash trees allows each packet to be immediately authenticated upon receipt. Seluge is the most recent approach for protecting the dissemination of new code images [13]. By carefully arranging code dissemination data items and their hash images in packets, Seluge provides immediate authentication of each packet upon receipt, without disrupting the efficient propagation mechanisms used by Deluge. Please refer to [13] for further discussion of the similarity and difference between these schemes.

However, none of the above approaches provides protection for the other epidemic image management commands such as "Reboot" and "Erase". The research in this paper complements these approaches by addressing these issues.

VIII. Conclusion

In this paper, we identified the security vulnerabilities in code dissemination that have not been addressed in previous research. The root of these vulnerabilities is the lack of efficient broadcast authentication of "Reboot" and "Erase" commands. We then developed a sequence of lightweight techniques to allow efficient authentication of these commands by taking advantage of efficient cryptographic hash functions. Our approach takes into consideration the limited resources on current sensor platforms, and removes the security vulnerabilities with very light overhead. We have implemented our approach as a remote image management system named Seluge-ImageMan, which is intended to work with Seluge [13], a security extension to Deluge for injecting new code images. Seluge and Seluge-ImageMan together provide a complete solution for secure code dissemination in wireless sensor networks.

Acknowledgment

This work is supported by the National Science Foundation under grants CNS-0721424 and CAREER-0447761, and by Army Research Office under Wang's staff research grant W911NF-04-D-0003. The contents of this paper do not necessarily reflect the position or the policies of the U.S. Government.

References

[1] Bouncy castle crypto apis. http://www.bouncycastle.org.

[2] MICAz: Wireless measurement system. http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MICAz_Datasheet.pdf.

[3] TinyOS: An open-source OS for the networked sensor regime. http://www.tinyos.net/.

[4] Certicom Research. Standards for efficient cryptography – SEC 2: Recommended elliptic curve domain parameters. http://www.secg.org/collateral/sec2_final.pdf, September 2000.

[5] ChipCon. 2.4 GHz IEEE 802.15.4/ZigBee-ready RF Transceiver. http://www.chipcon.com/files/CC2420_Data_Sheet_1_4.pdf.

[6] Crossbow Technology Inc. Mote in-network programming user reference, 2003.

[7] J. Deng, R. Han, and S. Mishra. Secure code distribution in dynamically programmable wireless sensor networks. In *Proceedings of the Fifth International Conference on Information Processing in Sensor Networks (IPSN '06)*, April 2006.

[8] P. K. Dutta, J. W. Hui, D. C. Chu, and D. E. Culler. Securing the deluge network programming system. In *Proceedings of the Fifth International Conference on Information Processing in Sensor Networks (IPSN '06)*, April 2006.

[9] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler. The nesC language: A holistic approach to networked embedded systems. In *Proceedings of Programming Language Design and Implementation (PLDI '03)*, June 2003.

[10] Y.C. Hu, A. Perrig, and D.B. Johnson. Packet leashes: A defense against wormhole attacks in wireless ad hoc networks. In *Proceedings of INFOCOM 2003*, April 2003.

[11] J. Hui. *Deluge 2.0 - TinyOS Network Programming*, July 2005. http://www.cs.berkeley.edu/~jwhui/research/deluge/deluge-manual.pdf.

[12] J. W. Hui and D. Culler. The dynamic behavior of a data dissemination protocol for network programming at scale. In *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys '04)*, November 2004.

[13] S. Hyun, P. Ning, A. Liu, and W. Du. Seluge: Secure and dos-resistant code dissemination in wireless sensor networks. In *Proceedings of the Seventh International Conference on Information Processing in Sensor Networks (IPSN '08)*, April 2008.

[14] IEEE Std 802.15.4-2003. IEEE standard for information technology – telecommunications and information exchange between systems – local and metropolitan area networks – specific requirements – part 15.4: Wireless medium access control (MAC) and physical layer (PHY) specifications for low-rate wireless personal area networks (LR-WPANs).

[15] S.S. Kulkarni and L. Wang. MNP: multihop network reprogramming service for sensor networks. In *Proceedings of the 25th International Conference on Distributed Computing Systems (ICDCS '05)*, pages 7–16, June 2005.

[16] P.E. Lanigan, R. Gandhi, and P. Narasimhan. Sluice: Secure dissemination of code updates in sensor networks. In *Proceedings of the 26th International Conference on Distributed Computing Systems (ICDCS '06)*, July 2006.

[17] P. Levis, N. Patel, D. Culler, and S. Shenker. Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks. In *Proceedings of the 1st Symposium on Network System Design and Implementation (NSDI '04)*, March 2004.

[18] A. Liu and P. Ning. TinyECC: Elliptic curve cryptography for sensor networks (version 1.0). http://discovery.csc.ncsu.edu/software/TinyECC/.

[19] D. Liu and P. Ning. Efficient distribution of key chain commitments for broadcast authentication in distributed sensor networks. In *Proceedings of the 10th Annual Network and Distributed System Security Symposium (NDSS'03)*, pages 263–276, February 2003.

[20] D. Liu and P. Ning. Multi-level μTESLA: Broadcast authentication for distributed sensor networks. *ACM Transactions in Embedded Computing Systems (TECS)*, 3(4):800–836, 2004.

[21] V. Naik, A. Arora, P. Sinha, and H. Zhang. Sprinkler: A reliable and scalable data dissemination service for wireless embedded devices. In *Proceedings IEEE International Real-Time Systems Symposium*, pages 277–286, December 2005.

[22] J. Newsome, R. Shi, D. Song, and A. Perrig. The sybil attack in sensor networks: Analysis and defenses. In *Proceedings of IEEE International Conference on Information Processing in Sensor Networks (IPSN 2004)*, April 2004.

[23] N. Reijers and K. Langendoen. Efficient code distribution in wireless sensor networks. In *Proceedings of the 2nd ACM International Conference on Wireless Sensor Networks and Applications (WSNA '03)*, pages 60–67, September 2003.

[24] T. Stathopoulos, J. Heidemann, and D. Estrin. A remote code update mechanism for wireless sensor networks. Technical Report CENS-TR-30, UCLA, Center for Embedded Networked Computing, November 2003.