SLEDE: Event-Based Specification of Sensor Network Security Protocols

Youssef Hanna Hridesh Rajan
Dept. of Computer Science, Iowa State University
226 Atanasoff Hall, Ames, IA, 50011, USA
{ywhanna, hridesh}@cs.iastate.edu

ABSTRACT

The semantic gap between specification and implementation languages for sensor networks security protocols impedes the specification and verification of the protocols. In this work, we present *SLEDE*, an event-based specification language and its verifying compiler that address this semantic gap. We demonstrate the features of *SLEDE* through an example specification of the μTesla, secure broadcast protocol for sensor networks.

1. INTRODUCTION

A *sensor network* is a collection of small size, low power, low-cost sensor nodes that has some computational, communication and storage capacity. These nodes can operate unattended, sensing and recording detailed information about their surroundings. Finding flaws in the new security protocols for these networks is harder compared to traditional protocols because they protect against more cryptographic failure modes.

Unlike traditional computers, sensor networks are not used for general-purpose computation. The typical usage is for data collection and control of local environments where a node has to react to changes in the environment rather than being driven by interactive or batch processing [3]. The nodes in these networks are thus event-driven. To address the domain specific needs, the languages need to be event-driven. The implementation language designs have responded to the need. The dominant language in this domain, *nesC* [3] supports the event-driven paradigm. Most security protocols in sensor networks are designed to work in an event-driven paradigm, whereas existing specification languages are either imperative or use a message passing style. There is thus an impedance mismatch between the specification and the implementation paradigm, which may be a potential reason for the proliferation of informally specified cryptographic protocols.

This work is a step towards filling the semantic gap between specification and implementation languages for sensor networks security protocols. We have developed an event-based specification language, *SLEDE* (for Specification Language for Event Driven Environments). In the rest of this document, we briefly describe *SLEDE* and its verifying compiler that internally uses the SPIN model checker [4]. We use a cryptographic protocol called μTesla [6] for illustration.

FSE'06 Portland, OR

2. THE SLEDE LANGUAGE

In this section, we present the specification of an example protocol called μTESLA [6] in *SLEDE* to demonstrate the syntax of the language (See Figure 1, comments appear after %). μTESLA is a protocol for securing message broadcasting between a sender (e.g., base station) and multiple receivers (e.g., ordinary sensor nodes) in a sensor network.

The key component of a specification is the node definition (nodes `Sender` (Lines 1-17) and `Receiver` (Lines 18-30)). As shown in the figure, a node may contain zero or more state declarations (`keyChain` (Line 5)), command declarations (`StdControl.start()` (Line 6-8)) and event declarations (`Timer.fired()` (Lines 9-16)). In μTesla protocol, the sender has a one-way key chain and a timer that fires every a predefined time interval. At every time interval (Lines 9-16), sender sets the message authentication code of the message to the hash of the current key and the message itself (Line 11) and then sends it (Line 12). The hashing library and the node implementing the broadcast of messages and are not included due to space limitation. At every 2 time intervals, the sender sends the current key it used to send messages (Line 14) and advances to the next key in the key chain (Line 15).

On the receiver side (Lines 18-30), `Receiving.receiveMsg(msg_t msg)` event handler (Line 22) is triggered whenever a message is received, which in turn stores the message in a buffer. The event handler `Receiving.receiveKey(Key kNew)` (Lines 23-29) is triggered when a new key is received. The event handler delivers the buffered messages after applying the hash function on the incoming key to verify that it comes from the authenticated sender (Lines 25-28); otherwise, it signals an error event (Line 24). The protocol starts from the node Main (Lines 31-32). Similar to *nesC*, the wiring of components is done using the `configuration` declaration (Lines 34-38).

The figure shows how SLEDE contributes in filling the semantic gap between the implementation and the specification language of security protocols in sensor networks. There are two benefits from having the specification language similar to the implementation language. First, it is easier to understand and write in a specification language that is similar to the implementation language in terms of constructs, computation models, etc. Second, it is easier to

```
1 node Sender { % Node declaration
2   %Provided and used interfaces
3   provides interface StdControl;
4   uses interface Timer;
5   OneWayKeyChain keyChain; %States
6   command void StdControl.start() { %commands
7     call Timer.start();
8   }
9   event result_t Timer.fired() { %event handlers
10    %every 1 time firing, set MAC of msg and send it
11    msg.MAC = hash(keyChain.current,msg); %library command call
12    call Transmission.broadcast(msg);
13    %every 2 firings, bdcst key and move key to next in keychain
14    call Transmission.broadcast(key); % command call
15    currentKey = Next(keyChain);
16  }
17 }
18 node Receiver {
19   Key k; % Revealed Key
20   Buffer b; %Message Buffer
21   %Stores the message in buffer
22   event result_t Receiving.receiveMsg(msg_t msg) { ...  }
23   event result_t Receiving.receiveKey(Key kNew) {
24     if (hash(kNew) != k)  signal Error.error() %signal statement
25     if (hash(kNew) == k) {
26       % set k=kNew, deliver msg from buffer if msg.MAC==hash(k,msg)
27       call Transmission.deliver(msg)
28     }
29   }
30 }
31 node Main { % Program starts from this node
32 }
33 %configuration responsible for wiring components
34 configuration sender { }
35 implementation {
36   components Main, Sender, Comm, TimerM, ClockM;
37   Main.StdControl -> TimerM.StdControl; ...
38 }
39 % A simple example objective
40 objective {
41   Sender.Transmission.broadcastMsg(msg)=>
42       Reciever.Transmission.deliver(msg) ||
43       Receiver.Error.error()
44 }
```

Figure 1: Specification of μTESLA

generate an implementation, either automatically or manually, from a protocol specification written in such a language.

Figure 1 also demonstrates how security objectives of the protocol are presented in SLEDE. This simple objective (Lines 40-44) checks that any secure broadcast message sent by the sender is either received by a receiver correctly, or an error event is generated. Note that the events used in this specification are also used to describe the objectives of the protocol, which gives an easier and more expressive way to represent objectives of the protocol in terms of user-defined events.

At this point, SLEDE is only providing the means to verify properties of the security protocols with no presence of attacks. We are working on SLEDE so that the user can specify the types of attacks he/she wants to verify the protocols against.

3. THE SLEDE COMPILER

The verifying compiler for *SLEDE* is built on the SPIN model checker [4]. The input language for the compiler is SLEDE and its target language is PROMELA, the input language for SPIN. The compiler translates the event handlers of SLEDE into processes. The commands are in-lined in the processes. The communication between the nodes of the protocol is simulated using channels. Events and commands are modeled as Boolean variables. Throwing an event or calling a command are translated to setting of the corresponding Boolean variable. Finally, the objectives are translated to Boolean formulas that may use the event/command Boolean variables.

4. RELATED WORK

The common authentication protocol specification language (CAPSL) developed by Millen et al. [5], is closely related. The motivation for the CAPSL project was that it is difficult to apply most cryptographic protocol verification mechanisms. They argued that the reason for this difficulty is that a protocol has to be re-specified for each verification technique that is applied to it and translating published description to the input of the verification tool is difficult [1]. CAPSL project solves this problem by developing a two-layered language design, where higher-level specification is translated to the CAPSL intermediate language (CIL). CAPSL allows clear specification of security properties in the style of Dolev and Yao [2]; however, it is also a message driven specification language that does not fit the sensor network paradigm very well.

5. CONCLUSION

Impedance mismatch between current specification and implementation languages for sensor network security protocols makes specification and verification of the protocols hard. We proposed *SLEDE* as a new event-based specification language for sensor network security protocols that fills this mismatch. We presented an example protocol μTesla in *SLEDE* that demonstrated the syntax of the language. The verifying compiler for *SLEDE* that is built on SPIN model checker was described to reveal how verification is achieved.

6. ACKNOWLEDGEMENTS

This work is supported in part by NSF Grant ITR-0627354. The discussions with Wensheng Zhang were very helpful.

7. REFERENCES

[1] G. Denker and J. Millen. CAPSL integrated protocol environment. In *DARPA Information and Survivability Conference and Exposition (DISCEX'00)*, pages 207–221, Hilton Head, South Carolina, Jan 2000.

[2] D. Dolev and A. C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, IT-29(12):198–208, mar 1983.

[3] 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 *PLDI '03: Proceedings of the 2003 conference on Programming language design and implementation*, pages 1–11, 2003.

[4] G. J. Holzmann. The model checker spin. *IEEE Transactions on Software Engineering*, 23(5):279–95, May 1997.

[5] J. K. Millen. CAPSL: Common authentication protocol specification language. In *NSPW '96: Proceedings of the 1996 workshop on New security paradigms*, page 132, 1996.

[6] A. Perrig, R. Szewczyk, V. Wen, D. Culler, and J. Tygar. Spins: security protocols for sensor netowrks. In *Proceedings of ACM Mobile Computing and Networking (Mobicom'01)*, pages 189–199, 2001.