SLEDE: Lightweight Verification of Sensor Network Security Protocol Implementations

Youssef Hanna
Dept. of Computer Science
Iowa State University
226 Atanasoff Hall
Ames, IA, 50011
ywhanna@cs.iastate.edu

ABSTRACT

Finding flaws in security protocol implementations is hard. Finding flaws in the implementations of sensor network security protocols is even harder because they are designed to protect against more system failures compared to traditional protocols. Formal verification techniques such as model checking, theorem proving, etc, have been very successful in the past in detecting faults in security protocol specifications; however, they generally require a model. Developing these models is a non-trivial task for an average developer. This task is further complicated by the impedance mismatch between the implementation language and the modeling language. For example, while the dominant implementation language for sensor network applications (*nesC*) uses an event-based paradigm, the modeling language (*Promela*) uses message-driven paradigm. The key goal of this research is to ease the task of verifying sensor network security protocol implementations for the sensor network community by defining an approach for automatically extracting a model from the *nesC* implementations of a security protocol. We contribute the design and implementation of a verification framework that we call *Slede* which emulates our approach to extract a PROMELA model from nesC security protocol implementations. By significantly decreasing the cost of verification, we believe our approach will improve the overall quality of the nesC security protocol implementations.

Categories and Subject Descriptors

D.2.4 [**Software/Program Verification**]: Formal Methods; D.2.4 [**Software/Program Verification**]: Model Checking; F.3.1 [**Specifying and Verifying and Reasoning about Programs**]: Mechanical verification, Specification technique

General Terms

Security, Verification

Keywords

sensor networks, security protocols, model checking, slede

1. INTRODUCTION

A *sensor network* is a collection of small size, low power, low-cost sensor nodes that has limited computational, communication capability and limited storage capacity. These nodes can operate unattended, sensing and recording detailed information about their surroundings. The operating environments for sensor networks are often hostile, requiring mechanisms for secure communication. A number of security protocols for sensor networks have been proposed in the past decade. For example, Perrig *et al.* [19], and Eschenauer and Gligor [6] proposed schemes for pairwise key establishment; Perrig *et al.* [19], and Yang *et al.* [23] proposed schemes for message authentication, etc.

Flaws in security protocols are subtle and very hard to find. In the past, even widely-studied cryptographic protocols are shown to have faults that are detected much later. For example, Meadows [16] showed flaws in selective broadcast protocol by Simmons [22]. Finding flaws in the security protocol implementations for sensor networks is even harder primarily due to two reasons. First, these implementations are more complex because the security protocols for sensor networks protect against more cryptographic failure modes compared to their counterparts. For example, the lack of tamper resistance of sensors makes physical capturing trivial. Second, these implementations are developed for a severely resource constrained environment. Efficiency and code size is more likely to weigh over readability and understandability, which in turn increases the likelihood of inconsistencies and errors.

Automated or semi-automated formal verification techniques such as model checking [5] and theorem proving [15] have recently re-emerged in a big way as plausible alternative to testing techniques. The key advantage of these techniques are two fold. First, a stronger guarantee about the desired properties can be obtained from a formally verified specification or implementation compared to their tested or simulated counterparts. Second, these techniques typically do not require users to write test cases. Despite these two advantages, testing and simulation-based techniques remain attractive to average users, often because most formal verification techniques requires them to develop models in specialized languages before verification. The need for developing yet another representation of the software system requires more time and effort, specialized knowledge, and introduces possibilities of errors due to inconsistencies between the implementation and the model.

In this paper, we present our approach for automatic extraction of verifiable models from *nesC* implementations [7]. The nesC language [7] is the dominant language for the sensor networks paradigm. As the name suggests, it is an extension of C; however, there are few major differences that make automatic extraction of models non-trivial. First and foremost, it is an event-driven language. Sec-

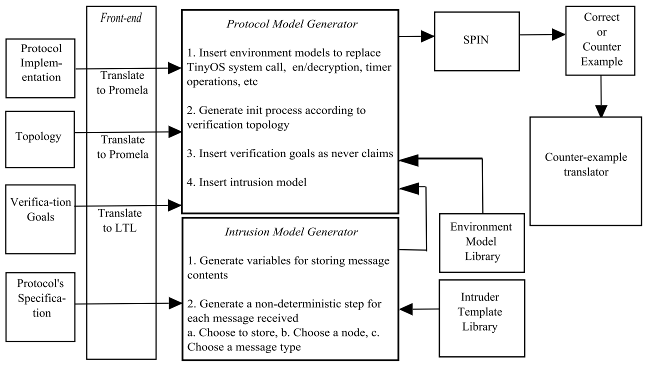
Front-end
Protocol Implem-entation
Translate to Promela
Topology
Translate to Promela
Verifica-tion Goals
Translate to LTL
Protocol's Specifica-tion
Protocol Model Generator
1. Insert environment models to replace TinyOS system call, en/decryption, timer operations, etc
2. Generate init process according to verification topology
3. Insert verification goals as never claims
4. Insert intrusion model
Intrusion Model Generator
1. Generate variables for storing message contents
2. Generate a non-deterministic step for each message received
a. Choose to store, b. Choose a node, c. Choose a message type
SPIN
Correct or Counter Example
Counter-example translator
Environment Model Library
Intruder Template Library

Figure 1: Overview of the Slede Verification Framework [10]

ond, it provides a component model for enhanced modularity compared to imperative languages like C. Third, it provides a concurrency model based on tasks and events. We also describe the design and implementation of our tool prototype *Slede* [10] that applies our techniques for model extraction to automatically extract the PROMELA model [13], verify it against the properties specified as Linear Temporal Logic (LTL) formula [20] using the SPIN model checker [13], and map the results back to the implementation domain. By decreasing the cost of verification, we believe that our approach will improve the overall quality of the nesC security protocol implementations.

Outline: Section 2 illustrates our approach for automating the model extraction from nesC applications and describes our existing framework. Section 3 discusses related work. Future work is discussed in Section 4. Section 5 describes the criteria for evaluating our research. Section 6 concludes.

2. APPROACH

Our verification framework *Slede* [10] is illustrated in Figure 1. The front end generates an abstract syntax tree of the protocol implementation, which is then passed to the protocol model generator responsible for automatically extracting verifiable PROMELA models [13].

In order to reduce the size of the generated model, *Slede* provides a lightweight annotation language that can be used to provide hints to the verification framework. The input to the framework is the protocol implementation in nesC annotated with the objectives, the topology as well as the protocol specification required to generate the intruder. The annotation language is described in Section 2.3.

Given the annotated protocol implementation, *Slede* generates a PROMELA model from the input files. During the model generation, all system calls (calls to pre-implemented modules in TinyOS) are replaced by calls to environment models provided as *Slede*'s libraries. (i.e. calls responsible for sending/receiving, LED manipulation, etc). The model is then given as input to the SPIN model checker, which verifies whether the model violates the objectives which have been translated into LTL formulas. If the objectives are satisfied, the protocol is verified as secure. Otherwise, SPIN produces a counter example that violates the security objectives in the domain language.

Our model extractor is built on top of nesC compiler [18] designed to support the TinyOS project [12]. TinyOS is the operating system used for sensors. During compilation, nesC compiler creates a call graph of nesC applications, which our compiler uses in the model extraction phase.

2.1 Model Creation

The main problem in the translation is that the implementation of security protocols usually includes many constructs that have no equivalent constructs in the verification languages. For example, secure hash functions, encryption and decryption use a lot of constructs that are far from what current verification techniques offer. Moreover, since nesC is built on C language, using pointer arithmetic in the implementation of sensor network protocols is common. So, when translating from nesC to PROMELA, an equational theory is required to be associated with implementation constructs with no equivalent in PROMELA as well a pointer analysis technique that traces pointer in the program. However, the problem with the equational theory is that it may lead to inaccurate verification results. The reason is that abstracting nesC constructs into PROMELA constructs might cause some discrepancy from the actual behavior of the protocol, which makes the verification of the model erroneous.

Our solution to this problem is based on the feature provided by SPIN, versions 4.0 and later, that allows embedding C code inside the PROMELA model. The global variables of every nesC module of the protocol (module is similar to a class in OOP) are included in the state vector using the SPIN's `c_state` construct, and almost all statements of the protocol are embedded one by one as C code using the `c_code` construct. Thus, no equational theories associated with function symbols nor pointer analysis techniques are required because the operations themselves are included in the model. Verifying security properties is done by tracing the values of the state vector variables, whose values are manipulated by the embedded nesC code, thus preserving the behavior of the protocol as defined in the implementation. Some statements still need to be abstracted (i.e. TinyOS code for sending messages). Abstraction is discussed in the next subsection.

2.2 Abstractions

Including every nesC statement in the PROMELA model might lead to unnecessary growth in the size of the model. For example, when a sensor wants to send a message, it calls some TinyOS *library* for sending the message that takes the message and broadcasts it. Translating such a library is firstly irrelevant to our goal of verifying the security protocols. It is not our goal to verify that the TinyOS sending library is working properly. Secondly, message sending in sensor networks is implemented in a totally different way than it is done in PROMELA. While sending a message in sensors requires some nesC code for physical broadcasting, sending in PROMELA is usually done by putting some variable in a channel visible to all processes and then another process receives the variable by dequeuing it from this channel. Thus, translating the sending from nesC to PROMELA will not help in emulating message sending in the verification phase.

In our model extractor, we avoid translating any *library* of the libraries of the TinyOS. Instead, whenever there is a call to a certain command in a TinyOS library, we substitute it with corresponding predefined PROMELA instructions. For example, whenever there is a call for sending a message, the compiler replaces the call by setting the value of some variable `Environment` with the value of the sent message. We do not use PROMELA channels since the type of the message variables used in the nesC implementation is different from the predefined types of PROMELA used in defining PROMELA channels. Receiving is done by reading the value of the variable `Environment` and then setting it to `null`. We plan to abstract other TinyOS libraries like libraries for turning the sensor led on and off, detecting light, etc.

2.3 Annotation Language and Intruder Model Generation

Intruders that are totally independent of the protocol implementation might lead to state explosion. One reason behind this is that different protocols might have different structures of the messages they send. Even though the structure for messages in TinyOS `TOS_Msg` used by all nesC applications have a fixed size, different protocols partition messages according to the need. For example, one protocol might divide the message into 3 variables, one for sender, one for receiver and one for the data. Another protocol might have a different partitioning with an additional partition for time stamp. Thus, letting the intruder guessing all possible partitions non-deterministically will increase the state space of the model for no good reason. If there is a flaw in the protocol, then one of the guesses of the intruders about the partitioning of the message will eventually cause the flaw to happen. And, if there is no flaw in the protocol, then for no guess by the intruder SPIN would find a flaw in the protocol. Thus providing the intruder with the knowledge of the partitioning of the message should help decrease the size of the model.

The annotation language helps reduce the size of the generated model by giving hints to the verification framework. In order to verify a protocol, besides the source code, *Slede* requires the topology, property to be verified and protocol specification required for intruder model generation.

```
1  /*@
2    @ message Ping mapsto IntMsg{
3    @  int data mapsto info;
4    @  }
5    @ message Ack mapsto IntMsg{
6    @  int sender mapsto src;
7    @  int receiver mapsto dest;
8    @  }
9    @ node SensorM A{ }
10   @ node SensorM B{ A; }
11   @ protocol p {
12   @  (SensorM, SensorM, Ping)
13   @  (SensorM, SensorM, Ack)
14   @ }
15   @ A.snd(Ping m)-><>!Intruder.knowsData
16   @*/
```

Figure 2: An Example Verification Configuration

An example of the annotation language is shown in Figure 2. The message declaration `Ping` (lines 2-4) is message mapped to the structure `IntMsg` in implementation. It has one field `data` mapped to field `info` in implementation. Two nodes are involved in the protocol where the topology is linear (lines 9-10). The message sequencing of the protocol should also provided to *Slede* (lines 11-14). The objective of the protocol is that if the intruder doesn't understand the data sent by node *A* (line 15). Note that the properties we verify against are always safety properties, because our goal is always to ensure that something bad doesn't happen (i.e. the secret value not revealed to the intruder). The complete grammar for the annotation language is in [10].

2.4 Emulating Event-Driven Paradigm

Sensor networks applications use the event-based paradigm. However, PROMELA uses message-passing paradigm. Thus, in order to emulate the event-based paradigm we add a call to event handlers between every two statements in the model [14]. This solution seem to work fine on small protocols; however, we have not tested the consequence of such approach when dealing with large protocols.

3. RELATED WORK

There is a significant body of research on software model checking. Bandera [4] is a tool for model checking concurrent Java programs by generating models from Java to different verification tools such as SPIN and SMV to detect problems such as deadlocks. Java PathFinder [11] is similar to Bandera in model checking concurrent Java programs for deadlocks and assertion violations. Verisoft [8] verifies implementations of concurrent systems written in C/C++ in order to find bugs in the code. A survey of different model extractors from C code is in [21].

Verification of security protocols from implementation was tackled by Bhargavan *et al.* [1]. They verify implementation of security protocols for web services written in F#. They compile the implementation of the protocol to ProVerif [2], a resolution-based theorem prover for cryptographic protocols. Goubault and Parrennes [9] translate cryptographic protocols written in C to Horn clauses to be verified using theorem provers. Unlike our approach that provides support for the nesC language, this approach is useful only for C implementations; however, the insights described by Goubault-Larrecq and Parrennes [9] could be used to enhance the underlying verification technique for Slede.

CMC [17] is another tool that provides model checking of C code. However, instead of extracting a model from the code, CMC generates the state space of a program by directly executing its C/C++ implementation. The goal of CMC is to find bugs in the implementation of a protocol, while ours is to find a security flaw in the protocol by checking if the protocol is satisfying security goals like secrecy.

4. FUTURE WORK

In this section, we describe some of the challenges that we will be dealing with in the future.

4.1 Modeling Unbounded Topology

Unlike other networks, sensor networks might have unbounded topology. In other words, the number of nodes in a sensor network could be arbitrarily large (in terms of thousands of nodes). Model checking such a network without facing a state explosion is quite hard.

We will deal with the unbounded topology problem by setting some limit on the number of the nodes in a protocol and decomposing the resulting network into smaller ones that are easier to verify. We believe that putting a bound on the number of nodes shall not affect the efficiency of the verification. Besides, when the number of nodes is *large*, the network is naturally divided into smaller networks with different topologies that communicate with each other. Thus, verifying those small network by decomposing it into smaller networks should remain an efficient way for verifying the large network, while avoiding the problem of state explosion. We are in the phase of investigating different techniques of network decomposition (i.e. [3]).

4.2 Scalability

For small protocols, our prototype seems to work fine [10]. One challenge to face is to deal with large protocols (1000+ lines of code). Extracting models from such protocols is a problem because the verifier might never halt due to the huge size of the extracted model. We believe that more abstraction is required in order to reduce the size of the model. For example, annotations on method calls to encryption/decryption functions may be added similar to Goubault-Larrecq and Parrennes's approach [9]. In the current prototype of Slede, we have not explored these abstraction techniques.

5. EVALUATION

Evaluation of this research project would be done using two criteria. First criterion is the size of the state vector. We need to make sure that the translation of large protocols will not lead to a state explosion with large protocols. Second criterion is the ability to find flaws. In our previous work [10], we ran *Slede* on small implementations of protocol with known flaws (i.e. Needham-Schroeder Public Key protocol) and *Slede* was able to detect the flaw. The next phase will be to verify real implementations of such protocols and check the ability of *Slede* to detect the flaws. Later, we will run the model extractor on protocols with no known flaws and check if SPIN detects new flaws with the least amount of false-negatives (if any).

6. CONCLUSION

We propose *Slede* [10], a verification framework that allows verification of security protocols for sensor networks through automatic model extraction from nesC implementations of the security protocols. We believe *Slede* will help in improving the overall quality of the nesC security protocol implementations. We are working on ways to reduce the state space of the model, abstract different TinyOS libraries as well as dealing with the unbounded topology problem by decomposing the networks into smaller ones that will yield smaller models.

7. AKNOWLEDGEMENTS

This work is supported in part by the National Science Foundation under grant CT-ISG: 0627354. The author thanks Hridesh Rajan for his guidance in this research.

8. REFERENCES

[1] Karthikeyan Bhargavan, Cedric Fournet, Andrew D. Gordon, and Stephen Tse. Verified interoperable implementations of security protocols. In *CSFW '06: Proceedings of the 19th IEEE Workshop on Computer Security Foundations*, pages 139–152, Washington, DC, USA, 2006. IEEE Computer Society.

[2] Bruno Blanchet. An efficient cryptographic protocol verifier based on prolog rules. In *CSFW '01: Proceedings of the 14th IEEE workshop on Computer Security Foundations*, page 82, Washington, DC, USA, 2001. IEEE Computer Society.

[3] Edmund Clarke, Muralidhar Talupur, Tayssir Touili, and Helmut Veith. Verification by network decomposition. In *Fifteenth International Conference on Concurrency Theory (CONCUR 04)*, pages 276–291. Springer-Verlag, 2004.

[4] James C. Corbett, Matthew B. Dwyer, John Hatcliff, Shawn Laubach, Corina S. Preanu, Robby, and Hongjun Zheng. Bandera: extracting finite-state models from java source code. In *ICSE '00: Proceedings of the 22nd international conference on Software engineering*, pages 439–448, New York, NY, USA, 2000. ACM Press.

[5] Jr. Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model checking*. MIT Press, Cambridge, MA, USA, 1999.

[6] L. Eschenauer and V. Gligor. A Key-management Scheme for Distributed Sensor Networks. *The 9th ACM Conference on Computer and Communications Security*, pages 41–47, November 2002.

[7] David Gay, Philip Levis, Robert von Behren, Matt Welsh, Eric Brewer, and David Culler. The nesC language: A holistic approach to networked embedded systems. In *PLDI '03*, pages 1–11, 2003.

[8] Patrice Godefroid. Model checking for programming languages using verisoft. In *POPL '97: Proceedings of the 24th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 174–186, New York, NY, USA, 1997. ACM Press.

[9] Jean Goubault-Larrecq and Fabrice Parrennes. Cryptographic protocol analysis on real C code. In Radhia Cousot, editor, *Proceedings of the 6th International Conference on Verification, Model Checking and Abstract Interpretation (VMCAI'05)*, volume 3385 of *Lecture Notes in Computer Science*, pages 363–379, Paris, France, January 2005. Springer.

[10] Youssef Hanna and Hridesh Rajan. Slede: A domain specific verification framework for sensor network security protocol implementations. Technical Report 07-09, Computer Science, Iowa State University, 2007.

[11] Klaus Havelund. Java pathfinder, a translator from java to promela. In *SPIN*, page 152, 1999.

[12] Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David E. Culler, and Kristofer S. J. Pister. System architecture directions for networked sensors. In *Architectural Support for Programming Languages and Operating Systems*, pages 93–104, 2000.

[13] Gerard J. Holzmann. The model checker SPIN. *IEEE Trans. Softw. Eng.*, 23(5):279–295, 1997.

[14] Gerard J. Holzmann and Margaret H. Smith. A practical method for verifying event-driven software. In *ICSE '99: Proceedings of the 21st international conference on Software engineering*, pages 597–607, Los Alamitos, CA, USA, 1999. IEEE Computer Society Press.

[15] Chin-Liang Chang; Richard Char-Tung Lee. *Symbolic Logic and Mechanical Theorem Proving*. Academic Press, Inc., Orlando, FL, 1973.

[16] Catherine Meadows. The NRL protocol analyzer: An overview. *Journal of Logic Programming*, 26(2):113–131, 1996.

[17] Madanlal Musuvathi, David Park, Andy Chou, Dawson R. Engler, and David L. Dill. CMC: A Pragmatic Approach to Model Checking Real Code. In *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation*, December 2002.

[18] nesC Compiler. http://sourceforge.net/projects/nescc.

[19] 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.

[20] Amir Pnueli. The temporal logic of programs. In *The 18th Annual Symposium on the Foundations of Computer Science*, pages 46–57, New York, 1977. IEEE.

[21] Bastian Schlich and Stefan Kowalewski. Model checking c source code for embedded systems. In *IEEE/NASA Workshop Leveraging Applications of Formal Methods, Verification, and Validation (ISoLA 2005)*, sep 2005.

[22] Gustavus J. Simmons. How to (selectively) broadcast a secret. In *Proceedings of the IEEE Symposium on Security and Privacy*, page 108, 1985.

[23] H. Yang, F. Ye, Y. Yuan, S. Lu and W. Arbaugh. Toward Resilient Security in Wireless Sensor Networks. *ACM MOBIHOC*, May 2005.