Poster Abstract: KleeNet — Automatic Bug Hunting in Sensor Network Applications

Raimondas Sasnauskas, Jó Ágila Bitsch Link,
Muhammad Hamad Alizai, Klaus Wehrle
Distributed Systems Group, RWTH Aachen University, Germany
{lastname}@cs.rwth-aachen.de

ABSTRACT

We present KleeNet, a Klee based bug hunting tool for sensor network applications before deployment. KleeNet automatically tests code for all possible inputs, ensures memory safety, and integrates well into TinyOS based application development life cycle, making it easy for developers to test their applications.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Reliability, availability, and serviceability; D.2.5 [**Testing and Debugging**]: Testing tools, Symbolic execution

General Terms

Performance, Reliability, Languages

Keywords

Type safety, memory safety, bug finding, TinyOS

1. INTRODUCTION

Currently, sensor network application developers are confronted with a number of domain specific complications. The constrained memory and CPU resources on sensor nodes result in using low-level, type-unsafe languages without dynamic type checking and memory protection. Similarly, because the applications are highly data-flow oriented, the correct exception handling at full coverage is a challenging task. Moreover, due to highly distributed and faulty nature of sensor nodes, some of the program bugs are detected only after the software is deployed.

C language has been the main choice for developing sensor network applications. It provides great flexibility, expressiveness, and in particular, the required low resource footprint. However, the absence of dynamic type checking in C necessitates very careful programming because many sensor OS's do not support memory protection.

Numerous tools exist for error removal in C programs, hence, our first step was to employ them for testing sensor network software written in the widely spread TinyOS platform. We encountered the following problems due to which the available tools are mostly not used at all, and the developers fall back on manual code debugging techniques: (1) Sensor network applications are tightly integrated into the operating system leading to time-consuming manual code modification in order to perform the actual testing; (2) Most of the tools perform only static code analysis with limited support for C semantics; (3) None of the tested tools can offer a push-button bug finding technology and the usage learning curve is mostly too steep for a typical developer.

SenSys'08, November 5–7, 2008, Raleigh, North Carolina, USA.
ACM 978-1-59593-990-6/08/11.

2. RELATED WORK

To the best of our knowledge, currently there are no frameworks for automatic bug detection in sensor network applications *before deployment* at bit-accurate C semantics. For bringing memory safe executions of applications at *runtime*, we only know of the related efforts [2,3], where the sole representative with fined-grained memory safety at C level is Safe TinyOS.

Safe TinyOS adds dynamic memory checks during compilation which allows to catch unsafe pointer and array operations without corrupting the RAM. Overall, this results in 13% increase in the code size and 5.2% increase in CPU usage. As with any dynamic assertion checking, Safe TinyOS can detect program bugs only eventually after the software is deployed. Therefore, still many corner-case bugs circumvent this testing technique. KleeNet, on the other hand offers offline bug detection with automatic code instrumentation. In doing so, it doesn't consume any system resources and ensures memory safety by treating the inputs symbolically i.e. checks any program variable for its all possible input values. As the core engine of KleeNet interprets a virtual instruction set, it cannot detect hardware platform dependent assembly level bugs nor enforce runtime memory safety. Thus, KleeNet complements the beneficial features of Safe TinyOS allowing altogether even more rigorous application testing (see Table 1).

Overall, we make following contributions: First, we integrate an effective bug finding tool into the event-driven TinyOS programming model with usability as a primary goal. Second, we show that, apart from the general checks already available, Klee can easily be extended to incorporate further checks useful for testing sensor network applications. And third, we practically demonstrate that sound testing techniques can be used *throughout* the application development process with minimum manual effort.

3. KLEENET OVERVIEW

Our proposed solution and its prototype implementation is based on *Klee*, the second implementation of EXE [1]. Klee is a symbolic execution tool for bug detection in C

Features	Safe TOS	KleeNet
Automatic code instrumentation	−	+
Target platform independence	−	+
Assembly level bug detection	+	−
Off-line bug detection	−	+
All possible input values checked	−	+
Automatic test case generation	−	+
Runtime safety enforcement	+	−
No additional resource usage	−	+

Table 1: Comparison: Safe TinyOS and KleeNet

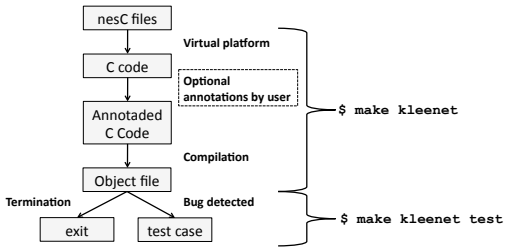

Figure 1: Integration of KleeNet into TinyOS

programs. In contrast to common runtime testing where the program input is (manually) generated, it runs the code on symbolic input initially allowed to take any value. If a bug is detected, Klee automatically generates a test case with concrete values causing that bug. At the current state of its implementation, Klee reports memory reference and division by zero errors.

Automatic code instrumentation. To provide a user-friendly solution, we use grammar based automatic code instrumentation. We extend the ANTLR [4] based GNU C grammar to automatically insert symbolic annotations (i.e. to mark the memory locations to be checked by Klee) in the C source code. The user only needs to provide a high level configuration stating the variable names that have to be checked inside the code.

Struct type checking. The received sensor data is initially available only as a pointer to an untyped bit stream and is later repeatedly casted to different structure types, making the code vulnerable to type errors. We have extended the functionality of Klee to check the struct type equality during pointer casting operations. This check is optional, nevertheless, KleeNet warnings are useful and facilitate program comprehension.

Integration into TinyOS. We have added a virtual *KleeNet* platform based on the TinyOS *null* platform to integrate Klee into TinyOS. This approach allows to easily extend our platform by adding modules that automatically mark inputs from sensors and incoming packets as symbolic.

In order to cover all possible program control flow paths during testing, we have extended our virtual platform with an automatic event signaling mechanism. Once an application is booted, all implemented events are signaled and processed.

Figure 1 shows an overview of KleeNet's build process. First, a user can optionally specify in a configuration file which variables should be marked as symbolic. Second, all incoming packet buffers are also marked symbolic automatically. Finally, Klee compiles and interprets the instrumented code and terminates when no bug is detected. Otherwise, a test case with real input values is automatically generated, causing the deployed sensor network application to follow the same path and hit the same bug.

We apply code instrumentation at C level, therefore, our approach can easily be integrated into any other sensor network development platform and operating system.

4. PRELIMINARY EVALUATION

We first checked the BlinkFail application from the TinyOS source repository used to test the Safe TinyOS toolchain installation. After marking the array index variable as symbolic, we immediately detected the known out of bound pointer error and a concrete test case leading to it.

Then we checked several applications without annotating the source code and rapidly detected possible division by zero errors. They occurred when received network data was processed without sanitization - a typical mistake made by novice programmers.

Overall, after our initial tests, we have confirmed the following key benefits of KleeNet:

Usability: A programmer can test the code with minimum manual effort and without any previous knowledge about the checking tool.

Coverage: KleeNet covers all possible execution paths and checks all possible data values before application deployment.

Integration: KleeNet is invoked by simply adding an extra build flag enabling the *permanent* code checking during the application development process.

Efficiency: It is fast for everyday use.

5. CONCLUSION AND FUTURE WORK

It is essential to fully test sensor network applications, i.e. taking all possible input values into account, where the cost of occurring undetected errors after deployment could be fatal. We have demonstrated that it is possible to close the gap between the testing and development community by providing a user-friendly and automatic bug finding tool which is strongly integrated into the system development life cycle.

Our work is in progress and incorporating further useful checks in Klee, such as runtime monitoring of long loops and computationally intensive tasks by adding time annotations, is future work. Similarly, verifying the distributed behavior of sensor network protocols, such as correct state transitions, remains to be addressed.

6. REFERENCES

[1] C. Cadar, V. Ganesh, P. M. Pawlowski, D. L. Dill, and D. R. Engler. EXE: automatically generating inputs of death. In *CCS '06: Proc. of 13th ACM conf. on Computer and communications security*, 2006.

[2] N. Cooprider, W. Archer, E. Eide, D. Gay, and J. Regehr. Efficient memory safety for TinyOS. In *SenSys '07: Proc. of the 5th international conference on Embedded networked sensor systems*, 2007.

[3] R. Kumar, E. Kohler, and M. Srivastava. Harbor: software-based memory protection for sensor nodes. In *IPSN '07: Proc. of the 6th international conference on Information processing in sensor networks*, 2007.

[4] T. J. Parr and R. W. Quong. Antlr: a predicated-ll(k) parser generator. *Software: Practice and Experience*, July 1995.