2009 International Conference on Advances in Recent Technologies in Communication and Computing

Safe Contiki OS: Type and Memory Safety for Contiki OS

Tomsy Paul and G. Santhosh Kumar

Department of Computer Science, Cochin University of Science and Technology, Cochin, Kerala, India – 682022
tomsypaul@hotmail.com san@cusat.ac.in

***Abstract*-Embedded systems, especially Wireless Sensor Nodes are highly prone to Type Safety and Memory Safety issues. Contiki, a prominent Operating System in the domain is even more affected by the problem since it makes extensive use of Type casts and Pointers. The work is an attempt to nullify the possibility of Safety violations in Contiki. We use a powerful, still efficient tool called Deputy to achieve this. We also try to automate the process.**

***Keywords*-wireless sensor networks, type safety, memory safety, Deputy, Contiki OS.**

I. INTRODUCTION

Untrapped memory errors are particularly harmful in sensor networks. The major reasons are: 1)It is easy to accidentally corrupt OS code; 2) Since memory objects are tightly packed, walking even one element past the allotted space is likely to corrupt RAM; 3) Deployed motes are difficult to debug, so it can be very hard to tell when memory corruption is the root cause of a serious deployment problem. Safety violations can even let others' programs intrude into a sensor node [1].

Safety violations are inevitable in WSNs. A feasible yet reliable approach is to avoid them by detection during application development. Our work focuses on the Safety issues pertaining to a popular Operating System for WSNs – the Contiki OS [2, 8]. Our aim is to make it free of all safety violations. We make use of a powerful analyzer for C programs viz, Deputy [3], that provides safety checking for both Type and Memory safety. The motivation behind the work was the successful implementation of Safe TinyOS [5], which used the same tool, Deputy to make Tiny OS, another popular OS for WSN, free of safety violations.

This paper is the continuation of our work in [10] with the addition of an attempt to automate the process and the provision of making safety an optional feature.

The rest of the paper is organized in the following manner. Section II gives a background of the work by describing the Contiki OS – the subject of the work, the Deputy – the tool used and the Safe Tiny OS – the motivation behind this work. Section III will give the details of the implementation and the current status of the work. Section IV provides with some examples of the possible safety violations in Contiki and the modifications we have done. And finally Section V will conclude the paper by describing the expected final output of the work.

II. BACKGROUND

A. Contiki OS

Contiki is an open source, highly portable, multi-tasking operating system for memory-efficient networked embedded systems and wireless sensor networks. A typical Contiki configuration is 2 kilobytes of RAM and 40 kilobytes of ROM. Contiki consists of an event-driven kernel, on top of which application programs can be dynamically loaded and unloaded at run time.

Contiki provides IP communication, both for IPv4 and IPv6. It was developed by a group of developers from industry and academia lead by Adam Dunkels from the Swedish Institute of Computer Science. Contiki's protothreads, have been used in many different embedded systems. Its idea of using IP communication in low-power sensor networks has lead to an IETF standard and an international industry alliance. The uIP embedded IP stack, is today used by hundreds of companies. Contiki programs are written in the C programming language. To ease software development for Contiki, Instant Contiki provides a single-file download that contains all necessary tools and compilers for developing software for Contiki.

B. Deputy

Deputy is a C compiler that is capable of preventing common C programming errors, including out-of-bounds memory accesses as well as many other common type-safety errors. It is designed to work on real-world code, up to and including the Linux kernel itself. Deputy verifies that your program adheres to these invariants through a combination of compile-time and run-time checking. Deputy is implemented using the CIL infrastructure for C program analysis and transformation, and it uses gcc as a back end. It was implemented in OCaml and the code size is around 20,000 lines.

Deputy uses the information provided by the programmer to ensure Type and Memory Safety. The information is given to Deputy through certain keywords called Annotations. Most of the Annotations of Deputy are self explanatory and suits the different situations that a C programmer commonly faces while using constructs like Pointers and Union that are highly prone to safety violations. The details of different Deputy Annotations can be found elsewhere [4].

978-0-7695-3845-7/09 $26.00 © 2009 IEEE
DOI 10.1109/ARTCom.2009.126

C. *Safe Tiny OS – The Motivation*

Safe Tiny OS was first presented in [6], which proposed some modifications to the existing Tiny OS tool chain. A Tiny OS [9] application is an assembly of components plus a small amount of runtime support. Typically, components are written in nesC, a dialect of C. The nesC compiler (ncc) translates an assembly of components into a monolithic C program, which is then compiled and optimized by gcc.

Safe TinyOS suggests the use of Deputy along with another tool cXprop [7] meant for optimization. The modified tool chain used Deputy and cXprop between ncc and gcc. The developers had to modify ncc to ignore the Deputy annotations. The output from Deputy was fed to cXprop and then given to gcc to generate the final safe machine code. The modifications required to the source code to become safe, on average, were only 0.74%, the target code size was increased only by 13% and the target data size was *decreased* by 2.3%.

III. SAFE CONTIKI

The source code of Contiki consists of nearly one thousand two hundred files, including both the OS core and the application programs. The OS core itself consists of around 300 files. The work comprises of annotating each pointer access in all these files and recompiling with Deputy. Once the core is free of Type/Safety errors, we can go for the safety of applications. After this step, we will set Deputy as the default compiler of Contiki. Programmers will have to adapt to the annotations of Deputy, which are very simple and easy to learn.

We use Instant Contiki version 2.2.1. and Deputy version 1.1; both being the latest versions.

The flow graph of Safe Contiki development is shown in Fig 1.

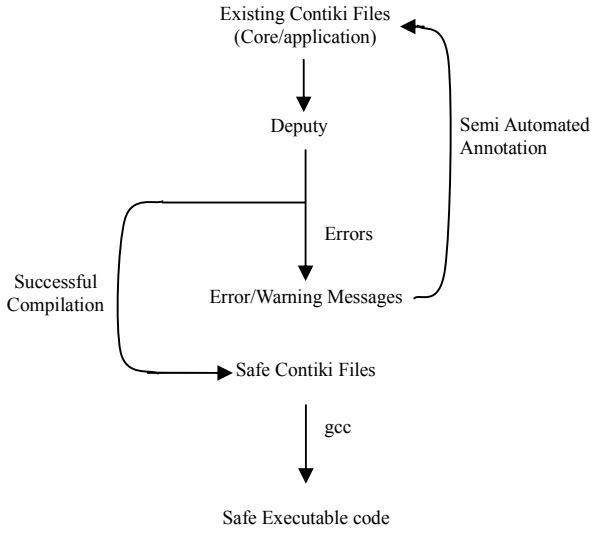

Fig.1 Safe Contiki Development

In Contiki, there can be 4, or in some cases 5, makefiles associated with each application. This facilitate the use of make command to build applications for different platforms and CPUs by simply changing the arguments to make. We started the implementation by replacing gcc with deputy in the makefile for the 'native' processor in the cpu directory. Then we started the making of 'hello-world' in the examples directory. Since the making of this application initially depends on more than 80 different source files, we have to fully annotate all these files to compile the program 'safe'ly. Currently we have annotated about 70 files and the total annotations are 397 which consists of 31 COUNT, 146 SAFE, 179 TC, 12 NTS, 2 TRUSTED, 10 NTDROP, 16 BOUND and 1 NTEXPAND. Once the annotations of the hello-world related files are over, we can go the next example and so on. After annotating the examples, there would be only a few core files left which will be found out and annotated. The developers of Safe TOS also had followed a similar strategy of incremental annotation.

Since Contiki make use of pointers in an extensive manner, the work is more laborious, compared with the development of Safe TOS. The better part of this work is that not much change was required in the tool chain; a simple replacement of gcc with deputy was enough. In the case of the former, they required the modification of the ncc.

Currently, we have not included cXprop in the tool chain, compromising with a little larger target code. Our primary goal is Safety, and we do not bother about optimization, for the time being. However, as mentioned in [6], the inclusion of cXprop in the tool chain would not be a tough task and could be done at a later stage.

We have given the provision for making Safety an optional feature. The reason is that many of the traditional Contiki programmers might be reluctant towards the use of Safety features. We introduce a new make variable called SAFETY. Only if its value is set to 'yes' will we go for safe compilation; otherwise, normal compilation. A problem related with this was the handling of the Deputy annotations during unsafe compilation. gcc was used for the unsafe compilation and the Deputy annotations were simply strange words to gcc resulting in Syntax errors and/or lexical errors. The solution was to use the –D flag of gcc to erase the Annotations when the source files are fed to gcc.

The performance of the Safe Contiki will be evaluated considering the three factors of Code Size, Data Size and the Speed of Execution. These give the requirements of ROM, RAM and CPU. Once these are measured with some sample applications for the Safe version, we can compare it with the corresponding values for the unsafe version.

From the very beginning of the work we were contemplating on the possibility of automating the process of Annotation, as soon as we realized the huge amount of manual work to be done. We found that annotations like TC, SAFE and NTS could be inferred from the error message, in *many of* the cases. These were also the most prominent Annotations used in the work. A rough estimate on the gain of

automating the process revealed that Automation was feasible. We designed a software tool that provides suggestion by analyzing the error message from Deputy. If the suggestion is accepted, it does the modification in the source file and also updates the log files. If the suggestion is incorrect, manual annotation can be done. This has considerably reduced the labor.

IV. Examples

As we started with the implementation we could find out a lot of portions of Contiki susceptible to safety violations. This section depicts a few of them. First we describe the violations and then we mention the Deputy annotations we gave.

A. Possible Safety violations found in Contiki

Possible Violation #1 static uint8_t *packetbufptr;
This is the declaration of packetbufptr in the file pcketbuf..c. This pointer points to the packet buffer used in the sending and reception of radio packets. The vulnerability here is that an application developer can accidentally use this pointer to access any location in the entire memory of the sensor node, possibly corrupting it. It is a typical example of Memory safety violations in C. We must set a limit on the area accessible through this pointer.

Possible Violation#2 char *name;
From mac.h which contains declarations for Medium Access Control layer functions and variables. This is the declaration of a variable to hold the name of a MAC driver. A typical C character string. If the user accidentally modifies the null character, an strcpy–like function might corrupt the entire memory.

Possible Violation#3 void *item
From list.c which is a basic linked list library. This is a parameter to list_push which adds an item to the beginning of a list. This declaration is prone to both Memory and Type safety violations.

B. Annotating the Violations

Possible Violation #1 static uint8_t *packetbufptr;
Gave the COUNT annotation,
static uint8_t * COUNT (PACKETBUF_SIZE) packetbufptr;
where PACKETBUF_SIZE is a macro defined in packetbuf.h

Possible Violation#2 char *name;
Gave the NTS annotation,
char * NTS name;

Possible Violation#3 void *item
Gave the BOUND annotation,
void list_push(list_t list,void *BOUND(__auto,__auto) item);
Since it was difficult to give the apt Annotation for item, we left it to Deputy using BOUND Annotation.

V. Conclusion

The Contiki OS, which uses pointers and type casts extensively, is an ideal platform for the application of Safety measures. As Deputy could free Tiny OS from Safety issues, we expect it can provide more assistance to Contiki, just like a real deputy. So far, a few possible safety violations have been discovered and corrected. The use of Automated Annotation has lead to considerable reduction in the manual work load and boosted up the speed of the process. As the work completes, we will also be able to assess the cost of safety in terms of speed of execution, code size and data size and in terms of the total modifications needed.

Acknowledgment

We thank the Almighty for His endless blessings. Sincere thanks to John Regehr and David Gay for sharing their experiences with the development of Safe Tiny OS and to Zach Anderson and Jeremy Condit for help on Deputy. Special thanks to Adam Dunkels for his interest in the work.

References

[1] Aurélien Francillon and Claude Castelluccia, “Code injection attacks on harvard-architecture devices” In *Proceedings of the 15th ACM Conference on Computer and Communications Security, Alexandria, Virginia, USA October 27 - 31, 2008* pp 15-26

[2] Adam Dunkels, Bjorn Gronvall and Thiemo Voigt “Contiki - A Lightweight and Flexible Operating System for Tiny Networked Sensors” In *Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks,16-18 November 2004, Tampa, FL, USA* pp 455 - 462

[3] (2007)The Deputy Project. http://deputy.cs.berkeley.edu

[4] (2007)The Deputy On-line Manual http://deputy.cs.berkeley.edu/manual.html

[5] (2008)The Safe TinyOS. http://docs.tinyos.net/index.php/Safe_TinyOS

[6] Nathan Cooprider, Will Archer, Eric Eide, David Gay and John Regehr “Efficient memory safety for TinyOS” In *Proceedings of the 5th international conference on Embedded networked sensor systems, Sydney, Australia* Nov. 2007 pp 205-218

[7] Nathan Cooprider and John Regehr. “Pluggable abstract domains for analyzing embedded software”. In *Proc. of the 2006 Conf. on Languages, Compilers, and Tools for Embedded Systems (LCTES), Ottawa, Canada,* June 2006, pp 44 -53.

[8] The Contiki OS, http://www.sics.se/contiki/

[9] Philip Levis, David Gay, Vlado Handziski, et al “T2: A second generation OS for embedded sensor networks”. Technical Report TKN – 05 – 007, Telecommunication Networks Group, Technische Universitat, Berlin, November 2005.

[10] Tomsy Paul and G. Santhosh Kumar “Safe Contiki OS” In *Proceedings of the National Conference on Education and Research (ConfER 2009) Cochin, India* organized by IEEE CS and CSI division V 11-14 March, 2009.