International Journal of Engineering Technology Science and Research
IJETSR
www.ijetsr.com
ISSN 2394 – 3386
Volume 4, Issue 7
July 2017

An Insight into Performance and Security bugs in IoT Operating Systems : A Literature Review

Mansi Malhotra
Department of Computer Engineering
MKSSS Cummins College of Engineering for Women, Pune, India

Supriya Kelkar
Department of Computer Engineering
MKSSS Cummins College of Engineering for Women, Pune, India

Abstract—The Internet of Things(IoT) devices form the root of the smart development and are capable of carrying out all the tasks by themselves. As the IoT devices are resource constraint and cater to the demands of heterogeneous applications, they require an Operating System(OS).The OS plays a critical role in maintaining the efficiency and security of services in IoT devices. Thus, one must be aware of the desirable features of an OS and the bugs which lead to the degradation of its performance. A lot of effort has been put to eliminate the performance and security bugs but still they have been proved out to be inefficient to deliver the software free of these vulnerabilities. Based on a literature survey, this paper discusses various studies and contributions which have analyzed the root causes of performance and security bugs and also focuses on a research project which led to a bug detection tool which is used to detect and fix a few of these bugs.

Keywords—Operating System; performance bugs; security bugs;

I. INTRODUCTION

IoT is the internetworking of physical objects which enable the devices to collect and exchange information or data. Basically, IoT brings together the sensors ,theanalog data collected,the processors to perform analytics and the security of thedevices. The basic structure of an IoTfrom the industry point of view consists of three layers. The lowermost layer is called the infrastructure layer which consists of sensors and processor. The middle layer is called as the platform layer which consists of an OS which constitutes the kernel, scheduler, APIs and the network stack. Lastly, the application layer which allows the user to use the services of the underlying layers through various tasks. The following figure shows the basic structure of IoT :-

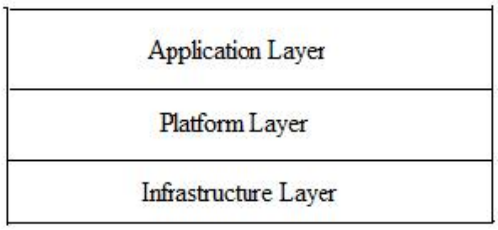

Fig. 1. Basic Architecture of an IoT environment

As the demand for IoT based applications is increasing day-by-day, traditional Windows/Unix OS are unable to meet the demands of IoTapplications. Thus, we require an IoT OS which provides flexibility, portability, light-weight environment with low RAM and ROM memory footprints and with easily configurability and programmability. Some of the existing IoT OS are Contiki, RIOT, TinyOS, LiteOS, FreeRTOS, MantisOS, uClinux[1].

The IoT OS is limited in storage, volume and portability; therefore the quality of software should be paid close attention. There are various kinds of software bugs which greatly affects the power of hardware and may cause a threat to performance and security. Out of the total software bugs recorded by the Software Analysis and Intelligence Lab(SAIL), Canada; two-thirds of the total software bugs consist of the performance bugs and security bugs[2]. Experienced developers are needed to fix performance bugs and security bugs. Also their repair process affects other device attributes

Rest of the paper is as follows: Section II discusses the Earlier work which is divided into two sub-sections in which the first sub-section would focus on the performance metrics of the IoT OS and the second sub-sections would discuss bugs in IoT focussing exclusively on the performance bugs and

International Journal of Engineering Technology Science and Research
IJETSR
www.ijetsr.com
ISSN 2394 – 3386
Volume 4, Issue 7
July 2017

security bugs. Section III discusses the case study of the bug detection tool, Rulede and Section IV presents the conclusion of the paper.

II. EARLIER WORK

A. *Performance metrics for an IoT OS*

Performance metrics are used to evaluate the performance of an IoT OS. These properties include the type of architecture, scheduling, memory management, portability, performance, security and networking. Following is the discussion of each metric in detail:

*1) Architecture:*There are three types of architectures common to all IoT OS:

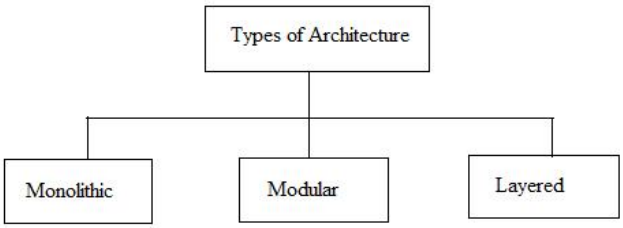

Fig. 2. Different types of architectures for IoT OS

Monolithic kernel has a high memory footprint. Their performance is higher as compared to microkernel as the control does not switch from kernel space to user space. In Modular architecture the modules are loaded as and when required by the application. Failure of even one module leads to the crash of the complete system. The Layered architecture is less modular, highly reliable and less complex than the monolithic kernel[1].

The most preferred architecture currently is the Microkernel due to its:

a) Modular nature

b) Easy configurability

c) Efficient utilization of the hardware

d) Caters to the different demands of the heterogenous applications

*Scheduling:*Scheduling is directly related to the system performance.It is very important to choose the right type of scheduler as several system performance factors such as latency, throughput of the OS, waiting time, etc. depend on it.The type of scheduler depends upon the nature of the task to be executed[1].Following are some of its types:

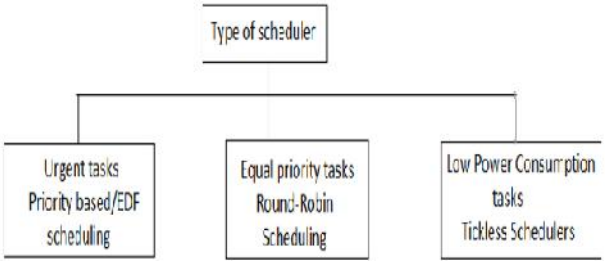

Fig. 3. Different types of schedulers for IoT OS

*2) Memory Management Unit(MMU):*MMU is responsible for controlling and coordinating the distribution of the available memory among the different tasks.The common features of MMU for all OS are the logical-physical address mapping, memory allocation and memory deallocation, memory protection, implementing cache memory,etc.The two distinct features of MMU for an IoT OS are as follows[1]:

a) Supports dynamic memory allocation as in order to maintain the flexibility of run-time allocation

b) It can be implemented as a separate module if the presence of MMU is violating the resource constraint attribute.

*3) Performance:*It is one of the most important metric.The following are the factors which are taken care under it[3] :

a) Efficient resource management : sharing of data and hardware resources among multiple tasks

b) Efficient power utilization: power and energy requirement for a given set of codes while finishing the scheduled instructions in the given time slot

c) Throughput of the OS: number of processes that can be completed by the OS per unit time

d) Complexity of the OS: the design of an IoT OS should be flexible and simple in order to meet the demands of heterogenous IoT applications

*4) Portability:*It refers to the ability of the OS to adjust to the specific needs and requirements of the various applications.Also as the microcontrollers used in devices range from 8 bit to 32 bit, the OS must be compatible with the underlying architecture[1].

*5) Security:*Security refers to maintaining the integrity, confidentiality and the availability of the OS.The security threats to an OS can be classified into two types:

a) Program threats: The kernel and the various processes in an OS do the designated task as

International Journal of Engineering Technology Science and Research
IJETSR
www.ijetsr.com
ISSN 2394 – 3386
Volume 4, Issue 7
July 2017

instructed by the malicious code injected by a malicious user.

b) System threats: It refers to the misuse of the system resouces which are available and the network connections with an intention of causing a disastrous attack on the OS.

6) Networking: The desirable properties of an IoT stack are its flexibility, reliability, internet enabled and light-weight.A support of IPv6 protocol is mandatory in IoT OS as the internetworking of billions of devices is required in IoT environment, with each device having an unique address[1].

B. Bugs in IoT

IoT OS are responsible for maintaining the security and efficiency of the system. According to a study conducted from May 2003 to August 2010 on the Firefox bug database, 4239 performance bugs and 847 security bugs have been reported[4].Also, the distribution of the number of bugs reported quarterly for security, performance and other bugs in a log scale between January 1997 to May 2010 was plotted by Software Analysis and Intelligence Lab(SAIL) which clearly showed that the performance and security bugs constitute two-thirds of the total software bugs[2]. In addition to this, it is very difficult to fix these bugs in comparison to the other software bugs as it is very difficult to detect such bugs and more skilled developers are needed for their repair process. The software is heavily tested before deployment to the customers. Coding consumes only a small portion of the development process but the testing comprises almost fifty percent of the cost of the software development but still the software defects continue to escape detection[5].Following is the detailed description of performance bugs and security bugs :

1)Performance bugs:

Performance bugs are introduced when inefficient code sequences get added in the system and start causing performance degradation and resource wastage. They lead to increased latency and reduced throughput[4]. They are basically the software defects in which small changes in the source code can speed up the software significantly while preserving the functionalities.

It is difficult to detect such bugs. They are also costly to diagnose due to their non-fail symptoms. Below some of the root causes and fixing techniques of performance bugs are discussed:

a) Root causes of performance bugs:

- Uncoordinated functions – Almost a third of the overall performance bugs are caused by inefficient function-call combination which is composed of efficient individual functions. The valid function call takes a detour to generate results and thus lead to degradation of performance[3].
- Synchronization issues – The unnecessary synchronization which occurs between the threads intensifies thread competition which leads to performance degradation.
- Lacking memory management – Dynamic memory allocation is preferred in IoT OS. Hence, the heap space which lies below the stack space is allocated dynamically. Sometimes, this heap space conflicts with the above stack space which leads to memory overflow and thus performance degradation[3].
- Code redundancy and logic defects –The application codes and the core codes develop redundancy which affects the storage space adversely. The logic defects are the useless and error operations which lead to wastage of CPU time. Thus, both these factors lead to performance degradation[3].
- API misuse – Many of the IoT OS make use of C and C++ programming. Many APIs are used to perform memory and string related operations such as “memcpy”, “sprint” and “strcat”. The misuse of these APIs leads to memory overflow, memory over-read, memory leak, etc[4].

b) Fixing performance bugs:

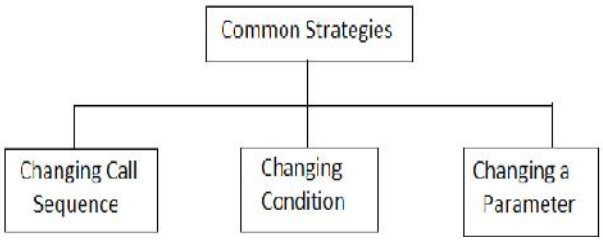

Fig. 4. Common strategies for fixation of performance bugs

There are some simple and easy to implement common strategies used to fix these bugs. Fig. 4 shows some of the methods used. Let us emphasise each of them.

- Changing call sequence – This strategy is used to fix performance bugs caused due to uncoordinated function calls and skippable functions. This technique involves reorganizing and

International Journal of Engineering Technology Science and Research
IJETSR
www.ijetsr.com
ISSN 2394 – 3386
Volume 4, Issue 7
July 2017

replacing a sequence of function calls in the source code[3].

- Changing Condition – In this strategy, different conditions are added or modified to remove the performance bugs. The code units which do not always generate useful results are conditionally skipped[3].
- Changing a parameter – Some of these bugs can easily be removed by changing or modifying some parameters in the code. Sometimes changing one or more function structures or adjusting the configuration parameters can also remove these bugs[3].

Thus, the performance bugs can be fixed by making simple changes in the application codes. So, it is easy to fix the bugs but the detection of such bugs quickly still proves out to be a challenge.

2) Security bugs:

Security of an OS refers to ensuring the integrity, confidentiality and availability of an OS. The attackers exploit the security vulnerabilities of the system which lead to catastrophic results[4].

These bugs can be introduced by various techniques. One of them is exploiting the shared memory. Also, if malicious program gets introduced in the system, the hackers can have access to and can read the sensitive information and compromise privacy.

Some of the common security bugs are as follows:

- Heartbleed–It is one of the security bug in OpenSSL cryptography. Here, maliciously crafted inputs are designed to exploit a lack of bound checking so that they could read ports of memory which are not intended to be accessible.
- Dangling pointer bugs – These types of bugs get introduced when a pointer makes a virtual function call. Here, a different address may be called due to the vtable pointer being overwritten.
- Code injection – It is basically inserting a piece of code into the vulnerable system by the malicious user and changing the action of execution of the program.
- Directory traversal – The aim of such a bug is to gain unauthorized access to file system by corrupting an application.
- Race conditions – This type of scenario occurs when a system attempts to perform two or more operations at the same time. This leads to a degradation in the performance of the system which eventually leads to making the system vulnerable and prone to attacks by viruses, worms and hackers.
- Cross-site scripting – This security vulnerability enables the attackers to inject client-side scripts into web pages. So, in this way the sensitive information of a user can be viewed by other users. The attackers make use of this vulnerability to bypass the access controls.

So, till now some of the root causes and fixation techniques of performance bugs have been highlighted.

Also,security bugs have been discussed in detail. Now in next section a tool under research by the Beijing

Information Security Test and Evaluation Centre, China would be summarized in brief.

III. CASE STUDY OF RULEDE

This section would discuss a bug detection tool called ‘Rulede’ developed in Beijing, China. They had conducted their study on 23 bugs found in three IoT OS – Contiki[6], RIOT OS[7] and TinyOS[8]. This tool has potential of detecting 45 bugs in the target OS. There are four checker rules which are used to detect 2 performance bugs and 2 security bugs[4].

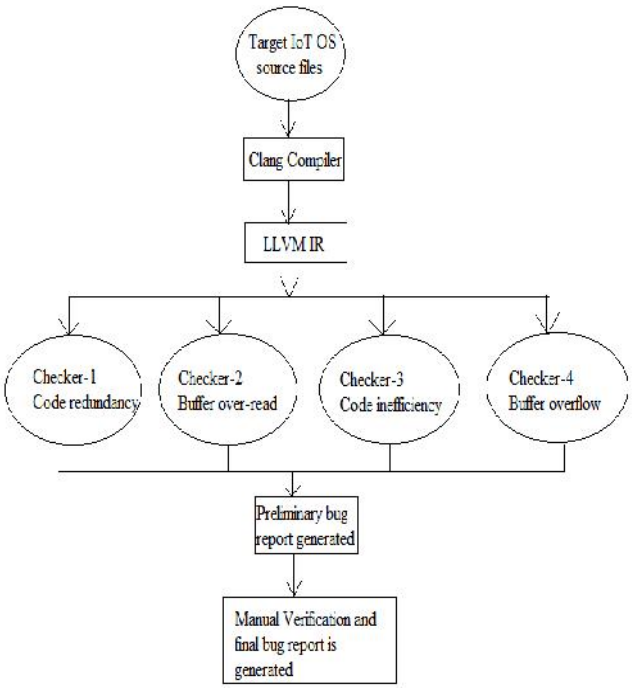

Fig. 5. Workflow of tool “Rulede

International Journal of Engineering Technology Science and Research
IJETSR
www.ijetsr.com
ISSN 2394 – 3386
Volume 4, Issue 7
July 2017

Rulede is built on LLVM(Low Level Virtual Machine) compiler and uses Intermediate Representation(IR) that is both in front-ends and back-ends.

Rulede consists of two work phases – Automated Analysis and Manual Verification. These two phases are explained in detail. Fig.5. shows the complete workflow of this tool :

1)Automated Analysis:

This phase is again divided into two stages:-

a) Language Process Module:

Here, the different source codes of the target IoT OS is submitted to the clang compilers. The Clang Compilers convert these files into LLVM IR. The Clang compilers are developed where LLVM is used as backend. The Clang can compile only those applications which can be built by GCC. So, basically it will modify the GCC files to Clang and add some compiler options. These compiler options specify the location of the CPU information files and platform files[4].

b) Rule-based bug detection tool

This module consists of four checkers according to four bug detection rules. LLVM compiler is used for implementation of these checkers which makes use of intra-procedural control-flow analysis and data-flow analysis and data-flow analysis API[4]. Following is the detailed discussion about the four checkers:

- Detection and fixing checker - 1

It deals with the storage, code redundancy and CPU time issues which reduce the performance of the system.

Problem:

The study of code redundancy and inefficient loop structure bugs in target OS revealed that it becomes inefficient when the step in a loop structure is small, but the string to be processed is long. This decreases the run speed of a program. Also, the performance degradation occurs when the same operation to the same string may occurmore than once in a file.

Remedy:

The loop structure used several times should be written in a function so as to save storage. Also, in order to save CPU time the loop step should be set to a greatervalue[4].

- Detection and fixing checker - 2

This deals with the memory over read bug. It is important to fix this bug as it decreases the system performance and constitutes one of the performance bug.

Problem:

This type of bug mostly occurs in memory and string operations. It is important to fix this bug as it overruns the buffer boundary and reads the adjacent memory.

If there isn't a suitable length string check before their function is called, then there is a threat to the safety of the memory.

Remedy:

Replace the "memcpy" API with the "strncopy" API as the "memcpy" standard format is :memcpy(dest, src, length) and if suitable length check isn't carried out before the function call, then the memory overread occurs. The "strncopy" copies the string to the maximum of either the number of characters present in the string or to the position of the first null[4].

- Detection and fixing checker - 3

It is used to save CPU time

Problem:

The formatted data is written into the destination string buffer using the "sprintf" API. It converts a fixed number of characters and copies the characters into the string buffer. But this complete operation reduces the speed of CPU.

Remedy:

Instead of using the "sprintf" API we can make use of special arithmetic to make the program run faster[4].

- Detection and fixing checker - 4

This checker is used to resolve the problem of buffer overflow.

Problem:

Buffer overflow problem occurs due to the errors generated in the memory and string operations. The syntax of the "strcat" is :char *strcat(char *dest, const char *src).

This API is designed to avoid the problem of buffer overflow. Many times an inappropriate value is assigned to the third parameter of the function which leads to the buffer overflow.

Remedy:

Replace the third parameter of the “strcat” function with - " sizeof(dest)-strlen(dest)-1"[4].

2)Manual Verification:

A preliminary bug report is generated after the IR files are checked by Rule-based static checkers. They are then passed to the second stage of work phase i.e. Manual Verification. In this step the documents of the target OS are required in order to verify the report.Finally, a report is generated which consists of the type of bug detected and the kind of bug fixation techniques used to resolve the bug.

Thus, from the above discussion of the tool "Rulede" it is clear that it is able to detect and resolve two performance bugs i.e. code redundancy and code inefficiencyand two security bugs i.e. buffer overflow and buffer over-read. This tool makes use of the LLVM compiler in the backend as it is a good analyser tool with high speedand good analysis ability[9].

IV. CONCLUSION

This paper started off by discussion of differentperformancemetrics for the IoT OS. The later sections focussed on the performance and security bugs and their root causes which lead to the performance degradation of the OS. Finally, a bug detection tool ‘Rulede’ was also highlighted.

It is required that the IoT service quality ensures security, efficiency and privacy as there are around a billion devices which make use of this technology. There are a few limitations of studying such bugs. It becomes difficult to repair the performance and security bugs due to the limitation on resource consumption, hardware speed and complexity of software. Also, in rectifying these bugs, the changes made in the source code can affect the functionality of other software in the system. Hence, experienced developers and experts are required to fix such bugs.

It is required that the developers and software experts have a good understanding of the impact of the different bugs on various project aspects. This would certainly improve the software quality research and practice. Very less research is going on to fix these bugs. This paper has also highlighted a tool named Rulede, developed in Beijing, which is exclusively used for bug detection. On similar lines, more research needs to be carried out to develop the detection rules and fixing techniques as it would lead to a better, efficient and bug free IoT development in the future.

References

[1] Padmini Gaur, Mohit P. Tahiliani “Operating Systems for IoT Devices: A Critical Surver”, 2015 IEEE Region 10 Symposium, Year 2015, pp.33-36

[2] Adams B, Hassan A E. “Security versus performance bugs:a case study on firefox”, Proceedings of the 8th Conference on mining software repositories, ACM, 2011, pp.93-102

[3] derstanding and Detecting Real-World Performance Bugs” , ACM SIGPLAN Notices,2012, Beijing, China, pp. 77-87

[4] Hongliang Liang, Qian Zhao, Yuying Wang “Understanding and Detecting Performance and Security Bugs in IOT OSes”, 2016 17th IEEE/ACIS International Conference in Software Engineering, Artificial Intelligence , Networking and Parallel Distributed Computing(SNPD), Year 2016, pp. 413-418.

[5] Biruk Mammo, Ricardo Rodriguez, Todd Austin, Valeria Bertacco, William Arthur “Schnauzer : scalable profiling for likely security bug sites”, Proceedings of the 2013 IEEE/ACM International Symposium on Code Generation and Optimization(CGO), Year 2013, pp.1-11Guoliang Jin, Joel Scherpelz, Linhai Song, Shan Lu, Xiaoming Shi “Un

[6] http://www.contiki-os.org/

[7] http://www.riot-os.org/

[8] http://www.tinyos.net/

[9] http://clang.llvm.org/