J.Emi Karmichael / International Journal of Engineering Research and Applications (IJERA) ISSN: 2248-9622 www.ijera.com
Vol. 2, Issue 4, July-August 2012, pp.447-451

Study And Implementation Of Self Healing Mechanism For The Control Flow Attack For Wireless Sensor Networks

J.Emi Karmichael
Centre For Information Technology And Engineering,
M S University, Tirunelveli

ABSTRACT

Nowadays wireless sensor networks have found their way into a wide variety of applications and systems with vastly varying requirements and characteristics, but all of them have a common element: faults are a normal fact and not isolated events as in traditional networks. Thus, in order to guarantee the network quality of service, it is essential for the sensor network to be able to detect and heal failures. The presented approach aims to employ self-healing services, allowing them to discover, examine, diagnose and react to malfunctions. In sensor application, a malicious code can change the flow of sensor to achieve the attacks. The downloaded malicious code will steal or modify the sensor data. To protect the control flow of sensor, this paper proposes self healing scheme that can detect the attack or the attempt to alter the control flow and recover the sensor application to the normal operation In additional, the original data which is altered by attackers is recovered from the private memory of sensor. Here the private memory is used for storing sensor data as reference, which is used during self optimization time, thus strong security is obtained. In additional, the original data which is altered by attackers is recovered from the private memory of sensor. the selfhealing scheme directly processes application code at the machine instruction level, instead of performing control or data analysis on source code. The implementation show that the self-healing scheme is lightweight in protecting sensor applications.

1.INTRODUCTION

Applications in sensor networks have been researched and developed for years. However, most security work focused on threats to networking and communication protocols. Lessons learned from worm attacks that exploit memory vulnerabilities show that attackers can compromise an entire network without hacking legitimate accounts or breaking protocols. is to protect the control flow of sensor and from the memory fault.[2] In a sensor's simple memory architecture, injected code can alter control flow of a sensor application. To protect the control flow, this paper proposes access control scheme that can detect attacks attempting to alter the control flow and then recover sensor data. Sensors use very simple embedded systems due to cost, efficiency, quality and resource limitations, sensors do not have sophisticated operating systems (OSs) to manage code for safety. Simple Os have been developed for embedded systems. However, they do not distinguish kernel mode or user mode when executing an instruction, and application data is adjacent to system data. Hence, one application routine can easily access the data of the system or other application routines. Furthermore, high-level programming languages have become popular in developing sensor applications because of their convenience for coding and maintenance over assembly languages. Open source based sensor applications have been developed as well. Consequently, applications share more and more common code as they use similar development environments. Memory fault attacks based on the same principle in regular computers become threats to sensor networks[1]. First, sensors do not have architecture to effectively enforce access control in program memory. A few schemes have been proposed to enforce access control in a sensor's data memory by using software-based memory management. These approaches do not prevent exploiting packets from accessing other code segments in the same program memory. Second, sensors do not have an effective recovery mechanism. Illegally accessing instructions in program memory normally causes the crash of the running sensor applications and results in a long restart period[2].The access control code effectively enforces access control in program memory such that the control flow cannot be maliciously altered. The access control code itself is designed to be resilient to control flow attacks that attempt to evade the access control. The scheme provides a self-healing recovery routine to quickly remove a compromised task from the application and restore the sensor to a normal state. The routine cleans up sabotaged data in data memory and releases the resources taken by the compromised task. The scheme works at the machine instruction

level and directly processes an application's machine code instead of the application's source code. The scheme diversifies the protected code images or different sensors.

2.RELATED WORK

Various journals are referred to know the sensor and its attacking methodologies from the given references papers at below . Attacks on the data collected without appropriate authentication of the nodes an attacker can impersonate a node to send fake data. An attacker not part of the network can tamper with the data. While there exists many Data authentication is a difficult problem in WSN, therefore data tempering by a malicious node is a difficult problem. Secure data aggregation protocols have been proposed to solve those issues. In a physical intrusion detection alarm system, the authority using the system would be willing that the alarms reported are secret, i.e. the messages passing would not to acknowledge the detection of the intruder. This for example would allow the authority to catch the intruder in the act. . At mean while, it runs protection code to enforce access control in program memory. This will match the public memory and private memory and recover the data if fault occurs. Like wise it will self optimist the sensor node and recover its original data. Many computer attacks exploit memory vulnerabilities in current computer systems. Various vulnerabilities have been identified in software, such as stack overflow, format string error, double-free error, heap overflow, return-to-libc, etc. These vulnerabilities are exploited to overwrite critical data in memory to launch control flow attacks [3] and data flow attacks [4]. Control flow attacks manipulate control data to change the flow of code execution. Return addresses and function pointers are two major types of control data that attackers are interested in altering and exploiting. In a typical "stack smashing" attack, return address in stack is overwritten to the address where injected codes are executed when the function (corresponding to the current stack frame) returns. When target program's control data are modified, attackers can execute injected malicious code or out-of context library code at the memory address pointed by the altered control data. Data flow attacks do not alter the control flow, but rather manipulate non control data to cause security breach in software. Many real-world software applications are susceptible to data flow attacks [4]. In such attacks, attackers examine the software to find out "which data within a target application are critical to security other than control data, whether the vulnerabilities exist at appropriate stages of execution that can lead to eventual security compromises, and whether the severity of security compromises is equivalent to that of traditional control data attacks. This paper focuses on control flow attacks that alter control flow to execute an unexpected sequence of instructions.

3.OVERVIEW OF POSSIBLE ATTACKS

Wireless sensor network security is many-fold, there are various ways to attack them. It is commonly assumed that wireless sensor networks are based on non tamper resistant devices, i.e. an attacker can easily collect a few nodes to analyze or modify them. However, as the network is large, possibly made of hundreds or thousands of devices, an attacker cannot tamper with all the devices. This is a basic assumption in security protocols designed for wireless sensor networks. An attacker can chose to attack the network, the data or directly the nodes that are described in paper [5].

3.1 MEMORY FAULT ATTACK

Many computer attacks exploit vulnerabilities due to memory fault in current computer systems. Such attacks can be categorized as control flow attacks. Attackers can overwrite control data to alter control flow via exploiting vulnerabilities of format string error, double-free error, heap overflow, return-to-lib, etc is given in paper [1] Attackers can alter control flow to execute injected malicious code or to bypass conditional branches or invoke indirect jumps.

3.2 CONTROL FLOW ATTACK

Attackers can alter the control flow via many well known buffer overflow techniques. In sensor nodes, attackers could find more approaches as the sensor's architecture is very simple. Attackers can directly overwrite kernel data or registers that are memory-mapped. The program memory of the processor is write-protected such that the application code can reliably work in the field. One of the attacks targeting this architecture is to alter the control flow of a sensor application that as been refered in paper [1].

4.ATTACK MODEL

In this paper, we do not consider attacks that simply capture nearby sensors. Instead, we examine attacks that send malicious packets to exploit vulnerability in remote sensors. Such attacks help attackers obtain more control over remote sensors that are not in their nearby areas. Such attacks can effectively threaten a network of tens or hundreds of sensors. We assume attackers can obtain source code or binary image of sensor applications, find exploitable coding errors, and develop exploiting packets offline, ahead of launching attacks. Researchers have found techniques that use non

executable data carried in exploiting packets to redirect the control flow to achieve certain attacks. First, a malicious packet is injected into a vulnerable sensor. Since the sensor is not aware if the packet is malicious or not, it will put the packet in a buffer in data memory. Then, when the packet is being processed in the sensor, the packet exploits vulnerability in code. The exploitable vulnerability varies, but leads to altering the control flow so that the data carried in the packet can misuse the application code. The misuse of the application code is carried in a chain of operations. Each unit in the chain consists of two steps and uses a part of data in the injected packet to accomplish a part of the attack. As the injected packet does not carry any code, each unit in the chain must use a part of the application code, and also ensure that, when it finishes, the control flow is altered to the next address of application code that can be used by the next unit in the chain. The first step in a unit of the misuse chain loads some data in the injected packet into registers. Because registers are used for passing parameters to functions in sensors, the loaded data will be used as the parameters in the second step. Then, the second step invokes a function in the application code with the loaded parameters to accomplish a specific part of attack. Finally, after the chain of misused operations completes, the attack exits. The attacking packet could simply alter the control flow to the RESET interrupt to restart the sensor, or release the control flow to let the sensor regain the control.

5..IMPLEMENTATION

Self-healing scheme is to handle control flow attacks. It has two modules (a) access control module that enforces the control flow of a running task, and (b) recovery module with additional memory that recovers the control flow of the sensor application from a compromised task. The execution of a sensor application is managed by the task scheduler of the sensor's OS. When a sensor receives a packet, a task will be dispatched by the task scheduler to process the packet. Once the task finishes, the execution of the application will return to the task scheduler so that the next pending task can be dispatched. The self-healing scheme embeds small blocks of access control code in all code segments in the program memory.

In a normal situation, all code segments being accessed by a task are in fact determined by the sensor application. Hence, each task has a pre-determined control flow. A non-compromised task should not have any abnormal access to a code segment that is not in its control flow. Thus, the access control code will allow the execution of any regular task. If packet exploits vulnerability in the code of the running task, we consider the task to be compromised. The vulnerability of the running task in fact allows the exploiting. Then, the access control code hands over the compromised task to the recovery routine that cleans the compromised task and returns the execution to the task scheduler for the next pending task. Both the task scheduler and the recovery routine are protected with access control code to prevent attackers from exploiting them. The intuitive arrangement of interactive graphical elements (windows, toolbars, menus, etc.) makes it easy to view and access the many powerful capabilities of ModelSim. VHDL includes facilities for describing logical structure and function of digital systems at a number of levels of abstraction, from system level down to the gate level. It is intended, among other things, as a modeling language for specification and simulation. We can also use it for hardware synthesis if we restrict ourselves to a subset that can be automatically translated into hardware. Easy-to-use wizards step you through creation of more complex HDL blocks. The wizards show how to create parameterizable logic blocks, test bench stimuli, and design objects. The source window templates and wizards benefit both novice and advanced HDL developers with time-saving shortcuts.

Control flow analysis component. It identifies CNs that includes the code of interrupt routines, and application routines. It also identifies and restructures the data memory layout with task related memory and non-task-related memory. Recovery code insertion component. It appends the recovery routine to the original application code. It also fills NOPs to all empty addresses in the code memory. Access control code insertion component. It assigns a random mark to each CN and inserts the access control code to enforce access control in code memory. The safety of the access control code is based on the fact that both marks and code are stored in the write protected code memory and cannot be modified.

6.OVERVIEW OF SELF-HEALING ARCHITECTURE

The recovery first releases resources allocated to the compromised task, then releases the compromised task from the kernel, and finally guides the kernel to execute the next pending task. As kernel routines are very crucial in a system, restarting the whole system is the ideal, safe and straightforward response to eliminate kernel attacks in sensors. Hence, in this paper, we focus on recovering the system when application tasks are being exploited. Because it is possible that an exploited function may affect other functions of the same task, the recovery is task-based. In this section, we first discuss the idea of the recovery approach normally used in preemptive OSs and then the recovery approach for the non-Preemptive OS in

sensors. Attackers can overwrite control data to alter control flow via exploiting vulnerabilities of format string error, double-free error, heap over flow, return-to-libc, etc. Attackers can alter control flow to execute injected malicious code or to bypass conditional branches or invoke indirect jumps. Control flow analysis component. It identifies CNs that include the code of interrupt routines, TinyOS routines, and application routines. It also identifies and restructures the data memory layout with task related memory and non-taskrelated memory. The recovery code insertion component generates the recovery routine and attaches it to the original code. It appends the recovery routine to the original application code. Italso fills NOPs to all empty addresses in the code memory. Finally, the access control code insertion competent safeguards the unprotected code and also diversifies the protection code to ensure releases the compromised task from the kernel, and finally guides the kernel to execute the next pending task. Each individual sensor obtains a unique protected

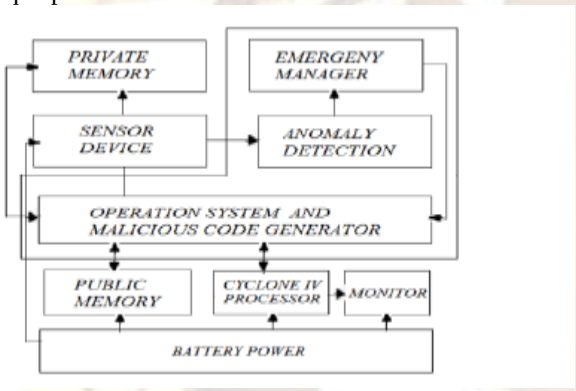

FIG 1 Self Optimization Of Sensor Node that each individual sensor obtains a unique protected code image.

Code image. The two memory areas that are public and private are used. If public get attack then private can be used to recover the sensor data which is altered by malicious code. Here for sample develop the malicious code and then alter the control flow. After that recovery module will recover the sensor module and sensor data. If a packet exploits vulnerability in the code of the Running task, we consider the task to be compromise. The redirection will be captured by the access control code at the end of the destination code segment, because the execution of the code segments deviates the normal control flow of the task. The recovery first releases resources allocated to the compromised task, then releases the compromised task from the kernel, and finally guides the kernel to execute the next pending task. The two memory areas that are public and private are used. If public get attack then private can be used to recover the sensor data which is altered by malicious code. Here for sample develop the malicious code and then alter the control flow. After that recovery module will recover the sensor module and sensor data. If a packet exploits vulnerability in the code of the Running task, we consider the task to be compromise. The redirection will be captured by the access control code at the end of the destination code segment, because the execution of the code segments deviates the normal. In the block diagram, the sensor node will sense the input data and the sensed data will store in the public and private memory. Here the access code will control the access and it will check whether fault is occurred or not. If any memory fault occurs then the private memory will recover the data.

7.RECOVERY OF A SENSOR NODE DATA

The recovery, it first releases resources allocated to the compromised sensor data, then releases the data from the kernel, and finally guides the kernel to execute the next sensor data to original position. In this section, we first discuss the idea of the recovery approach normally used in recovery OSs and then the recovery approach for the Tiny OS in sensors has been developed. Attackers can overwrite control data to alter sensor data via exploiting vulnerabilities of format string error, double-free error, heap over Attackers can alter control flow to execute injected malicious code or to bypass conditional branches or invoke indirect jumps. The recovery code insertion component generates the recovery routine and attaches it to the original code. Finally, the access control code insertion component safeguards the unprotected that done in [1] and additionally the protection code to ensure the original data that each individual sensor .

8.CONCLUSION AND FUTURE SCOPES

The overhead of the self-healing scheme in program memory and how much it affect the execution of normal routines will be examined. That enforces access control in the control flow of sensor applications and recovers the sensor data using the additive memory, when a control flow attack and memory fault attacks are captured. The security analysis shows that the scheme self- optimizes the sensor node and its data from various attack. Finally restore the sensor to a normal state. In the future, the study of preventing the attackers to intrude inside the sensor application is derived on new trends. The current self-healing scheme simply releases memory and recover the data from private memory taken by a compromised task. On next step the memory protection scheme can be implemented for more confidential areas.

REFERENCES

[1] Christopher Ferguson, Qijun Gu, Hongchi Shi "Self-healing Control Flow Protection in Sensor Applications", March 16 2009, Zurich, Switzerland.

[2] Harald Vogt, Matthias Ringwald, Mario Straser , "Intrusion Detection and Failure Recovery in Sensor Nodes" , at ETH Zurich, Switzerland.

[3] A. Smirnov and T. Chiueh, "DIRA: Automatic Detection, Identification and Repair of Control-Data Attacks," Proc. Ann. Network and Distributed System Security Symp., 2005.

[4] C. Kruegel, E. Kirda, D. Mutz, W. Robertson, and G. Vigna, "Automating Mimicry Attacks Using Static Binary Analysis," Proc. USENIX Security Symp., 2005.

[5] "Self-Healing Methodology in Ubiquitous Sensor Network", Giljong Yoo, and Eunseok Lee, p.p 3, February, at School of Information and Communication Engineering Sungkyunkwan University.

[6] A. One, "Smashing the Stack for Fun and Profit," Phrack Magazine, http://www.phrack.com/issues.html?issue=49&id= 14#article, 1996.