Self-Propagating Worms in Wireless Sensor Networks

ACM CoNEXT - Student Workshop

Thanassis Giannetsos
Athens Information Tech.
19.5 km Markopoulo Ave.
Athens, Greece
agia@ait.edu.gr

Tassos Dimitriou
Athens Information Tech.
19.5 km Markopoulo Ave.
Athens, Greece
tdim@ait.edu.gr

Neeli R. Prasad
Department of
Communication, Aalborg Un.
Fr. Bajers Vej 7A5
DK-9220,Denmark
np@es.aau.dk

ABSTRACT

Malicious code is defined as software designed to execute attacks on software systems. This work demonstrates the possibility of executing malware on wireless sensor nodes that are based on the von Neumann architecture. This is achieved by exploiting a buffer overflow vulnerability to smash the call stack, intrude a remote node over the radio channel and, eventually, completely take control of it. Then we show how the malware can be crafted to become a self-replicating worm that broadcasts itself and propagates over the network hop-by-hop, infecting all the nodes.

Categories and Subject Descriptors

D.4.6 [**Operating Systems**]: Security and Protection

General Terms

Experimentation, Performance, Security

Keywords

Von Neumann Architecture, Wireless Sensor Networks, Embedded Devices, Arbitrary Code Size Injection Attacks, Multi-Stage Buffer Overflow, Sensor Worm

1. INTRODUCTION

Recent advances in sensor networks research have shown that an attacker can exploit different mechanisms of sensor nodes and spread malicious code throughout the whole network without physical contact. Such a method is to exploit memory related vulnerabilities, like buffer overflows [2, 3], to launch a worm attack. Since all sensor nodes execute the same program image, finding such a vulnerability can lead to the construction of self-propagating packets that inject malicious code to their victims and transfer execution to that code. If the malware is constructed such as it resends itself to the neighbors of the node by repeating the same process, the attacker can compromise the whole network rapidly and take complete control of it [1, 4]. While this attack is extremely dangerous, there has been very little research in this area.

Our work target sensor devices following the von Neumann architecture. According to this architecture, both instructions and data are stored in the same memory space, allowing the attacker to transfer execution control where the mal-packet is stored. This allows the injection and execution of arbitrary code that did not exist previously in the mote's memory.

2. CHALLENGES OF CODE INJECTION ATTACKS ON SENSOR DEVICES

Buffer overflows are a leading type of security vulnerability. They are the result of programming flaws and are perfect for code injection attacks.

However, achieving that in sensor devices following the von Neumann architecture has some interesting parameters. First, since code injection attacks are based on changing the flow of control in a program, this may lead the sensor to restart itself or go into an unstable state, where further execution of the attack code is canceled. Furthermore, sensor nodes characteristics and constraints limit the capabilities of an attacker, who may want to send large blocks of code that exceed the allowed packet size. Thus, in order to send a meaningful piece of code, one has to break it down and send it through multiple packets. We should also stress that the whole attack code must reside in a *contiguous* memory region so it can be executed. Therefore, the attacker must perform a "*multistage buffer-overflow attack*", where she can manipulate an arbitrary address pointer and modify the data it points to.

3. EXPLOITING BUFFER OVERFLOW FOR CODE INJECTION ATTACKS

In order to send arbitrarily long blocks of code, we are using the "*multistage buffer-overflow attack*". Multistage buffer-overflow is a type of attack that requires several steps of buffer overflow. It allows the attacker to manipulate an arbitrary address pointer and modify the data it points to. So, by sending a number of specially crafted packets that result in consecutive buffer overflows, the attacker has the ability to copy malicious code from one memory location (payload of received message) to another (region pointed by the selected address pointer), and eventually have her attack code stored in a *contiguous* memory region, starting from a memory address of her choice. This type of attack will bypass the limitations of a single buffer overflow, in which the length of the attack code cannot exceed the size of a message payload.

Control Flow of Code Injection Attack

Figure 1 illustrates the execution flow upon reception of the k-th mal-packet. As mentioned previously, a number of packets need to be sent for the whole attack code to be copied in the target region. Thus, each mal-packet needs to alter control flow several times in order to allow further reception of packets. Figure 1 also shows the specially-crafted packet sent at the end of the attack for activating the injected malware. Details of the operations that take place, are provided below:

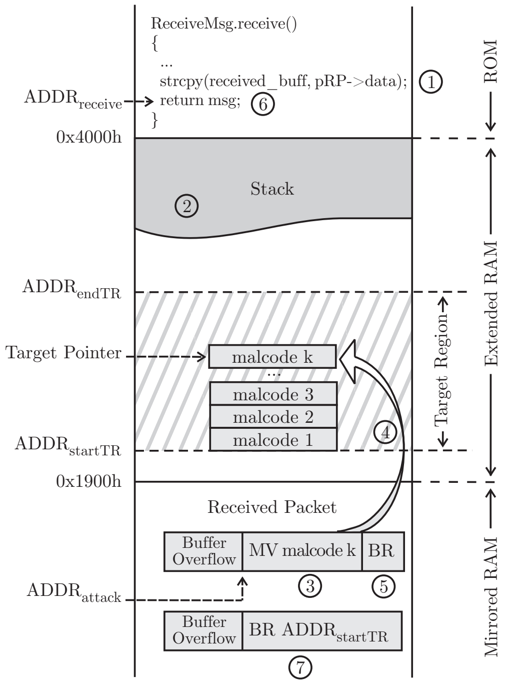

Figure 1: Control flow under multistage buffer-overflow attack.

1. Vulnerable function *strcpy()* is called from the reception routine.

2. A buffer overflow occurs resulting in the overwrite of the return address ($ADDR_{receive}$), stored in the stack frame of the *strcpy()*, with the starting address $ADDR_{attack}$ of the attack code. $ADDR_{attack}$ points to the `MOV` instructions contained in the packet's payload.

3. When *strcpy()* finishes its execution, control flow is redirected to $ADDR_{attack}$ memory address.

4. `MOV` instructions are executed for copying malcode bytes to consecutive memory addresses starting from where the target pointer (TP) points at the time.

5. The `BR` instruction that occupies the last 4 bytes of the mal-packet payload is executed in order to restore program's control flow.

6. Program execution continues normally. This is accomplished by setting the program counter to point to $ADDR_{receive}$ memory address of the receive function.

7. Once the attack code is stored in the target region, the last specially-crafted packet is sent for activating it. Its payload contains a `BR` instruction that is executed for setting the instruction pointer to the starting address of the target region, $ADDR_{startTR}$ ($0x2574h$ in our case).

4. PERFORMANCE EVALUATION

In order to evaluate the performance of the "sensor worm", we deployed several nodes in random topologies on the floor of an office building.

What we can depict from Figure 2 is the time needed by the worm to reach 100% of sensors in a particular neighborhood. As we can see, the propagation delay is really low and determined by the success or failure of the broadcast transmissions.

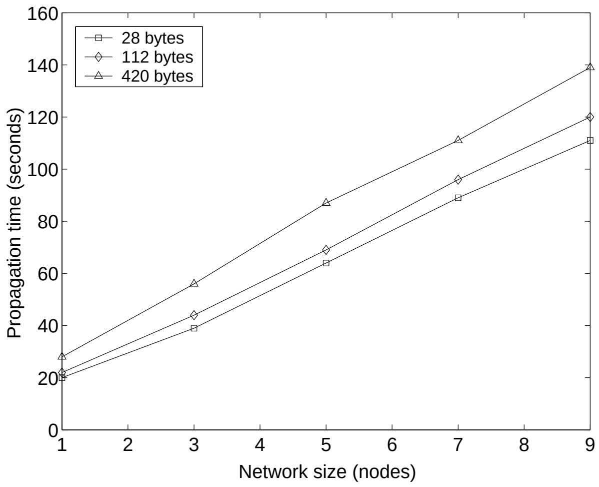

Figure 2: Infection time for different sizes of malicious code. The self-propagation code is also accounted for.

5. ACKNOWLEDGEMENTS

The research leading to these results has received funding from the European Community's Seventh Framework Programme ($FP7/2007-2013$), Call reference $SEC-2007-1$, under Grant agreement no: 217925.

6. REFERENCES

[1] A. Francillon and C. Castelluccia. Code injection attacks on harvard-architecture devices. In *15th ACM Conference on Computer and Communications Security (CCS)*, Alexandria, VA, USA, 2008.

[2] T. Goodspeed. Exploiting Wireless Sensor Networks over 802.15.4. In *Texas Instruments Developper Conference*, 2008.

[3] T. Goodspeed. Exploiting Wireless Sensor Networks over 802.15.4. In *ToorCon* 9, San Diego, 2007.

[4] Q. Gu and R. Noorani. Towards self-propagate mal-packets in sensor networks. In *WiSec '08: Proceedings of the first ACM conference on Wireless network security*, pages 172–182, Alexandria, VA, USA, 2008.