Threat Analysis of GNU Software Radio

Raquel L. Hill, Suvda Myagmar, Roy Campbell

Department of Computer Science
University of Illinois at Urbana-Champaign, Urbana, IL 61801

ABSTRACT

Software defined radio (SDR) technology implements some of the functional modules of a radio system in software enabling highly flexible handsets. SDR devices may be reconfigured dynamically via the download of new software modules. Malicious or malfunctioning downloaded software present serious security risks to SDR devices and networks in which they operate.

In this paper, we analyze threats pertaining to the secure execution of downloaded software. We base our analysis on the open-source implementation of GNU Software Radio. We propose mechanisms to provide the secure execution of software modules that implement radio functionality.

1 Introduction

Software Defined Radio (SDR) is a rapidly evolving technology that is receiving enormous recognition and generating widespread interest. SDR technology implements radio functionality such as modulation/demodulation, signal generation, coding and link-layer protocols in software. Implementing such functionality in software creates highly flexible handsets that can be reconfigured to upgrade and adapt equipment to user preferences and regional regulations. Reconfigurability enables the use of the same equipment in different regions as well as the fast introduction of new services into mobile networks without requiring the purchase of new terminals. While the benefits of reconfigurable radios are enormous, the ability to reconfigure radio functionality with software may lead to serious radio security problems such as unauthorized use of application and network services, unauthorized modification of software and malfunctioning radio equipment. For example, to illustrate the latter, software can be introduced into a device that changes its radio frequency (RF) operating characteristics so that it no longer functions within the regulated constraints (e.g. frequency, power, modulation). Such changes in RF parameters may be used to launch denial of service (DoS) attacks on the hardware or entire wireless network. Additionally, viruses that try to overwrite software modules and system memory may also be introduced. It is essential to provide suitable protection mechanisms that ensure secure, safe, and reliable operation.

Current techniques for ensuring that a radio is functioning within authorized parameters are not applicable for SDR equipment because RF parameters that were once fixed in hardware may now be reconfigured during regular operation [12]. The software and hardware architecture of systems in which SDR technology is deployed must incorporate security measures to isolate and protect radio-critical system elements from improper changes, whether accidental or intentional. Without such protection mechanisms, the increased flexibility and openness of reconfigurable terminals may be misused.

The SDR Forum classifies the collection of software for SDR as follows [9]:

- Radio Operating Environment - consists of the core framework, the operating system, device drivers, middleware, installer and any other software fundamental to the operation of the radio platform.
- Radio Applications - software which controls the behavior of the RF function of the radio. This includes any software defining the air interface and the modulation and communication protocols. It also includes used to manage or control the radio in a network environment.
- Service Provider Applications - software used to support network and other service provider support for the user of the radio. It includes voice telephone calls, data delivery, paging, instant messaging service, video pictures, emergency assistance, and geolocation.
- User Applications - application software not falling into any of the above categories.

Each class of software has its own security considerations, but secure configuration of the SDR device and secure execution of SDR modules are key components of the overall SDR security problem. We focus our analysis specifically on issues that impact the security of the Radio Operating Environment and the Radio Application modules of the GNU Software Radio. GNU Software Radio is an open-source SDR platform that is used currently in both academic research and commercial products [3, 6]. In its current state, GNU Software Radio defines the digital signal processing modules of a radio receiver and the functions to configure the receiver.

In this paper, we:

- Study security vulnerabilities of GNU Software Radio open-source code,
- Investigate threats to secure execution of radio application modules,
- Provide requirements for secure Radio Operating Environment,
- Propose mechanisms to protect memory access and provide preliminary results that describe the overhead of one such mechanism.

The remainder of this paper is organized as follows. Section 2 gives a brief overview of the GNU Software Radio codebase. Section 3 presents our analysis in detail. Section 4 discusses related work, Section 5 presents performance results of our memory protection implementation, and Section 6 concludes the paper.

2 GNU Software Radio

The GNU Software Radio [1] is a code base of free software that defines radio waveforms for a software receiver. Since the GNU Radio receiver utilizes open-source software and a standard PC using Linux, the system is an ideal platform for learning and experimenting with SDR concepts.

GNU Software Radio Application modules are written in C++, while the functions that configure the RA modules into a functioning radio are written in Python [4]. We refer to the latter as the ROE function of GNU Radio. The GNU Software Radio ROE configures the radio by constructing a flow graph of signal processing modules (SPMs). Each SPM is a vertex in the graph. Shared buffers connect adjacent SPMs and are used by the SPMs to pass data from one module to the next. After configuring the flow graph, the ROE invokes the scheduler which initiates the execution of the first SPM. The scheduler is a GNU RA function that defines the sequence of execution for the SPM in the flow graph.

The ROE and the flow graph execute within a single process and share a single address space. Therefore, the SPMs in the flow graph may overwrite the data produced by other SPMs. The SPMs may also introduce code that reconfigures the function of the radio.

3 Threat Analysis

In this section, we analyze the security of the Radio Operating Environment (ROE) and Radio Application (RA) modules for GNU Software Radio. Regarding the ROE, we examine how the ROE configures the GNU radio and initiates the execution of the RA modules. Regarding the RA modules, we examine how RA modules interact and share data. We also summarize how these modules may program RF parameters and describe the security threat of such programming.

The ROE for GNU Software Radio provides functions to construct and execute the dataflow graph. Any RA module may define a graph object, add vertices to the graph and initiate the graph's execution. Therefore, it is possible for any newly downloaded module to reconfigure the function of the SDR device. To prevent the SDR device from being reconfigured by malicious code, the SDR graph must be protected, and trusted system modules must construct, update and execute the graph. The current GNU Software Radio framework doesn't provide such functionality.

In addition to the risks associated with the instantiation of the execution graph, there are serious risks associated with the execution of modules in the graph. First, all signal processing modules run in the same address space. Therefore, any malicious module can read or write any data in the entire address space. For any meaningful security to exist between modules, the memory accesses of each module must be isolated and limited to it's own data. Additionally, the shared buffer that connects adjacent signal processing modules may be overwritten by a malicious module or a module that contains software errors, resulting in buffer overflow. To protect against buffer overflow, mechanisms must be employed that limit the amount of data that is written to the size of the shared buffer.

Coupled with the risk of RA modules making unauthorized memory accesses are the security threats that result from the programmability of Radio Frequency (RF) parameters such as modulation, frequency and power. We summarize programmability of each RF parameter and describe the type of attack that may be launched.

- *Modulation* The potential DoS attack against a device that uses software modulation is the improper change of the modulation format. This attack has limited impact and renders only the individual unit inoperable.
- *Frequency* The design and manufacturing tolerances for RF filters and resonators limit the extensibility of this hardware past specified limits [7]. Therefore software enhancements within this area may allow wireless devices to transmit and receive on various frequencies above or below a certain frequency (high or low pass filters) or within a certain range (band filter). Interference occurs when a device is programmed to transmit on a frequency for which it is not authorized, thus enabling the device to jam the signals from other nodes that are using the same frequency.
- *Output Power* Like frequency, output power is limited by inherent mechanical limitations of the hardware design. Of the three RF parameters, power is least likely to be affected by SDR technologies [7]. Realistic security threats are mostly limited to scenarios where the device operates at maximum power when it should operate at a reduced power state. Operating at maximum power may enhance an individual users performance, but degrade the overall performance of the communications system. Additionally, other users within the network may be forced to operate at elevated power levels, thereby reducing their battery life. Therefore a node that operates at maximum power for

extended periods may cause denial of service for nodes that raise their power levels and prematurely expend their batteries.

To overcome the aforementioned threats in secure execution of downloaded software, in the following sections, we provide security requirements for the ROE and propose the use of various memory protection mechanisms to protect against buffer overflow and the corruption of shared data.

3.1 Security Requirements for GNU Software Radio ROE Configuration Function

Software Defined Radio technology facilitates enriched communication capabilities via the use of reconfigurable radio devices. The ability to reconfigure a SDR device may lead to unauthorized modification of radio function. Therefore, reconfiguration of a SDR device must be limited to trusted and authorized agents. To this end, we present requirements for the secure configuration of SDR devices.

- *Separate Processes*The GNU Software Radio ROE configuration and RA functions share a process and address space. Thus, there is no way to control how, when and by whom the GNU SDR device is configured. The first step to addressing this problem is to ensure that the ROE and the RA are separate processes.

- *Trusted Configuration* Users and service providers who once trusted the function of hardware are now required to trust that software components have been properly configured. To facilitate this trust, the SDR device must be able to verify that the configuration function is correct. The SDR device should also be able to prove to external entities that it is running the appropriate ROE and RA code.

- *Policy Driven Configuration* To address the risks that are associated with the programmability of RF parameters, network administrators and regulators must define policies that specify the operating constraints of SDR devices. The GNU Software Radio ROE must provide mechanisms for evaluating and enforcing these policies.

3.2 Protecting Memory Access

One of the main vulnerabilities of the secure execution RA modules is the integrity of the runtime memory. Faulty or malicious software components may perform unauthorized memory access through buffer overflow. Several types of protection techniques exist to prevent buffer overflow such as dynamic runtime check, address protection, and software fault-isolation. These mechanisms are implemented as stand-alone system software, kernel extensions, compiler extensions, and loader extensions. Although no mechanism can prevent all unauthorized attempts to access memory, these mechanisms along with good programming techniques will help to provide the best possible solution for such attacks [13].

Dynamic runtime check primarily relies on the safety code being preloaded before a software component is executed. This preloaded component can either provide safer versions of the standard unsafe functions, or it can ensure that return addresses are not overwritten. Source code of the software component is not needed. LibSafe is an example of such solution; it protects a process against the exploitation of buffer overflow vulnerabilities in process stacks [2]. LibSafe intercepts all calls to library functions that are known to be vulnerable. A substitute version of the corresponding function implements the original functionality, but in a manner that ensures that any buffer overflows are contained within the current stack frame.

Compiler tools allow bounds checking to go into compiled code automatically, without changing the source code. These compilers generate the code with built-in safeguards that try to prevent the return address from being overwritten, as most attacks occur this way. StackGuard detects and defeats smash stacking attacks by protecting the return address on the stack from being altered [5]. It places a canary word next to the return address whenever a function is called. If the canary word has been altered when the function returns, then some attempt has been made on the overflow buffers.

Software-based fault isolation techniques modify the machine code of a program during load time to instrument all critical accesses such as memory read/write, jumps, calls and returns to point to valid and allowed addresses. Sandboxing is an isolation technique that inserts a code before every unsafe instruction. This code sets the upper bits of the target address to the correct segment identifier to ensure that the address falls in the logically separate portion of the software component within its address space [21].

4 Related Work

In this section, we discuss previous research related to security of software defined radio (SDR). We have categorized this work into three main groups. The first group proposes security frameworks with key management and various encryption algorithms for secure download and electronic commerce [15, 16, 17, 19]. The second group proposes schemes for securing a specific feature of software download, the download connection [11, 20]. The third group proposes radio spectrum management for global roaming [18, 14].

Lachlan Michael et al. [15, 16, 17]propose a framework for establishing secure download for SDR that includes employment of tamper resistant hardware and four different cryptographic techniques: secret key encryption, public key encryption, cryptographic hashing, and digital signature. They assume the existence of tamper-resistant hardware that provides secure storage for the terminal keys. This work also assumes that software is created and distributed by hardware maker. Sugita et al. [19] propose an electronic commerce scheme for SDR that utilizes the ability of SDR to switch between different security algorithms for electronic commerce. The following issues are identified, but without precise specifications: (a) encryption of download channel, hardware key, and terminal ID

to prevent illegal copying of the downloaded program; (b) certification against alteration of the downloaded program. While the work within this group addresses the secure software download problem, our work assesses the problems that prevent the secure execution of downloaded code.

Brawerman et al. [11] propose a lightweight LSSL protocol that uses less bandwidth than SSL to securely connect a manufacturer's server and SDR devices for downloading radio configuration files. In addition to securing the download connection, their secure protocol includes mutual authentication, public/private key mechanisms for data encryption and decryption, and fingerprint calculations to check data integrity. Uchikawa et al. propose a secure download system that uses the characteristics of the field programmable gate arrays (FPGAs) composing the SDR. The wiring of configuration logic blocks on FPGAs can be arranged in many different ways (astronomical number), enabling high security encipherment to prevent illegal acquisition of software using replay attack [20]. These works assume that SDR devices download software only from their manufacturers, and they focus on confidentiality and authentication aspects of SDR security. We are assume that software can be provided by a third-party, and certified software may be faulty.

According to FCC, the radio spectrum should be protected by proving the compliancy of the *combination* of SDR hardware and software to the local radio regulations before this combination is brought to the market. A SDR security architecture that enables separate software and hardware certification is proposed in [18, 14]. This architecture is contains: (a) automatic calibration and certification unit (ACU) that ensures that the output spectrum is compliant with regional radio regulations; (b) built-in GPS receiver for terminal location check; (c) radio security module (RSM) that manages the life-cycle of the downloaded software.

Some of the security issues and threats are discussed in the SDR Forum documents [12, 8, 10, 9]. Among them [12] mentions use of sandboxing as a possible approach to prevent harm from potentially malicious software. We elaborate on this idea even further and evaluate the use of fault isolation in our GNU Software Radio implementation.

5 Evaluation

In the previous sections we emphasized the importance of protecting memory spaces shared among software modules that compose a software radio system. As a first step towards providing a fully secure execution environment for SDR, we designed and implemented security mechanisms to protect access to the shared buffers in the GNU Software Radio flow graph. Our solution utilizes the encapsulation capability of object-oriented programming and a LINUX system function for privileged memory access. We defined a new buffer class that allocates and manages buffers in more secure manner than the current GNU software radio platform. A buffer allocated for signal processing can be read or written only by authorized signal processing modules that are registered as readers or writers of the buffer. When a read/write operation is attempted, the buffer verifies the identity of the requester. For write operation, the buffer also checks whether there is enough space to write the requested number of elements into the buffer, thus preventing buffer overflow. We use the `mprotect` system function to control accesses to a region of memory containing shared buffers and to prevent malicious or faulty code from obtaining a random pointer within the process's address space and overwriting it. `mprotect` assigns desired access permissions for the memory pages containing the given memory region. If a disallowed access is attempted, the program receives a segmentation violation. In our implementation the access permission of a shared buffer remains read-only until an authorized write occurs. We've performed a preliminary evaluation of our memory protection mechanism. The evaluation setup is as follows:

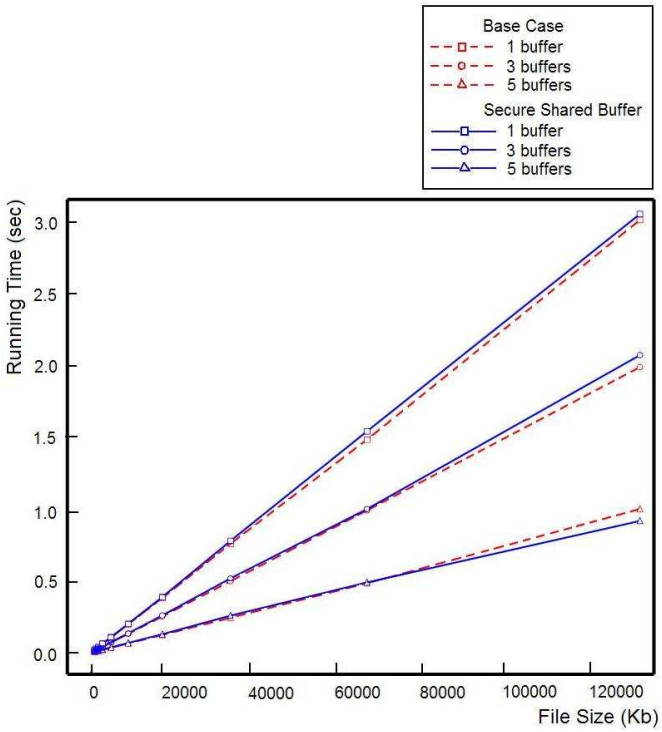

Figure 1: Runtime overhead.

- Since the main overhead due to new security features lies in memory accesses, we want to observe the effects of increasing number of shared signal processing buffers and size of input data on the running time of the system. We tested the radio system without RF front-end hardware because delays due to radio transmission and digital-to-analog conversion are orthogonal to our study.
- We used input files containing voice data. These data were recorded using GNU radio.
- We used signal processing blocks that applied simple

mathematical functions on input data such as multiplication by a vector. As the number of signal processing blocks increased, the number of shared buffers increased proportionally.

Figure 1 illustrates running time of the radio system in two different cases: a) unmodified base case; b) security mechanism in place. For each of these cases we run the radio system with three different flow graph constructions: 1 buffer, 3 buffers, 5 buffers. There are six test suites. We execute each test suit with 10 different input files of varying sizes, ranging from 100 Kb to 128 Mb. The dotted lines indicate the performance of the base case; the solid lines indicate the performance of the radio system with shared buffer protection.

The results of our evaluation show that our memory protection mechanism doesn't add significant overhead to the running time of the radio system; the overhead is negligible.

6 Conclusions

In this paper, we have analyzed threats to the secure execution of downloaded software for GNU Software Radio. In the current implementation of GNU Software Radio, all downloaded software modules share a single address space. When these modules are inserted in the execution graph, adjacent modules are connected via a shared buffer. The use of a single address space and shared buffer introduces such security threats as buffer overflow and unauthorized read/write of memory.

We have designed and implemented a secure buffer class that protects access to and manipulation of shared buffers that connect signal processing modules in the GNU Software Radio flow graph. Our preliminary results show that the overhead of the protection mechanisms that we employ is negligible.

For future work, we plan to incorporate additional memory protection mechanisms into our implementation. In addition, we plan to define a policy framework that will enforce the authorized construction, update and execution of the flow graph. The framework will also support the specification, evaluation and enforcement of polices that control the programmability of RF parameters.

7 Acknowledgments

This work is funded by the Office of Naval Research through the National Center for Advanced Secure Systems Research (NCASSR). We would like to thank Eric Blossom for his feedback throughout the duration of this project.

REFERENCES

[1] GNU Software Radio project. http://www.gnu.org/software/gnuradio/.

[2] Libsafe, project, AvayaLabs. http://www.research.avayalabs.com/project/libsafe/.

[3] NCASSR Software-Defined Radio research. http://www.ncassr.org/projects/sdr.html.

[4] Python. http://www.python.org.

[5] StackGuard, project, IMMUNIX. http://www.cse.ogi.edu/DISC/projects/immunix/StackGuard/.

[6] Vanu Software Radio. http://www.vanu.com/index.html.

[7] Report on issues and activity in the area of security for software defined radio. SDR Forum Approved Document, SDRF-02-A-0003, 2002.

[8] SDR system security. SDR Forum Approved Document, SDRF-02-A-0006, November 2002.

[9] A structure for software defined radio security. SDR Forum Input Document, SDRF-03-I-0010, May 2003.

[10] SDRF wireless security. SDR Forum Input Document, SDRF-04-I-0023, April 2004.

[11] A. Brawerman, D. Blough, and B. Bing. Securing the download of radio configuration files for software defined radio devices. In *Proceedings of the 2nd International Workshop on Mobility Management & Wireless Access Protocols*, October 2004.

[12] R. Falk, J. F. Esfahani, and M. Dillinger. Reconfigurable radio terminals - threats and security objectives. SDR Forum Input Document, SDRF-02-I-0056, November 2002.

[13] S. Grover. Buffer overflow attacks and their countermeasures. *LINUX Journal*, March 2003.

[14] C. F. Lam, K. Sakaguchi, J. Takada, and K. Araki. Secure download system based on software defined radio composed of FPGAs. In *Proceedings of IEEE Vehicular Technology Conference*, October 2003.

[15] L. B. Michael, M. J. Mihaljevic, S. Haruyama, and R. Kohno. Security issues for software defined radio: Design of a secure download system. *IEICE Trans. Commun.*, January 1999.

[16] L. B. Michael, M. J. Mihaljevic, S. Haruyama, and R. Kohno. A framework for secure download for software-defined radio. *IEEE Communications Magazine*, July 2002.

[17] L. B. Michael, M. J. Mihaljevic, S. Haruyama, and R. Kohno. A proposal of architectural elements for implementing secure software download service in software defined radio. In *Proceedings of the 13th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications*, 2002.

[18] K. Sakaguchi, C. F. Lam, T. D. Doan, M. Togooch, J. Takada, and K. Araki. ACU and RSM based radio spectrum management for realization of flexible software defined radio world. *IEICE Trans. Commun.*, December 2003.

[19] M. Sugita, K. Uehara, and S. Kubota. Flexible security systems and a new structure for electronic commerce on software radio. In *Proceedings of the 13th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications*, 2002.

[20] H. Uchikawa, K. Umebayashi, and R. Kohno. Secure download system based on software defined radio composed of FPGAs. In *Proceedings of the 13th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications*, 2002.

[21] R. Wahbe, S. Lucco, T. Anderson, and S. Graham. Efficient software-based fault isolation. In *Proceedings of the Symposium on Operating System Principles*, 1993.