SDR SECURITY THREATS IN AN OPEN SOURCE WORLD

Bill Hart (Green Hills Software, Santa Barbara, CA)

ABSTRACT

The software development community is rapidly accepting the use of open source software in business and mission critical applications. At the same time, software attacks are taking an increasing toll on business and society. Such assaults have recently been detected in embedded software, foretelling a trend toward malicious attacks on every day digital devices.

Are software-defined radios vulnerable to such attacks? What would motivate the perpetrators? What are the potential costs if attacks are successful? Are security methods in place that would reduce the risk of such attacks?

Open source software is by definition the product of many contributors. It is possible for one or more contributors to insert malicious code into compilers, operating systems and other software building blocks. Once inside a software-defined radio, a Trojan horse could exert direct control over hardware encryption devices and other security components, allowing an attacker to monitor communications or conduct other forms of sabotage.

An attack against a consumer radio could expose its user to invasion of privacy or direct economic loss. A successful attack against a telecommunications network could cause economic damage or system outage amongst a population of users. On a more sinister note, malicious code in a military radio could provide an enemy with real-time intelligence for tactical or strategic battlefield advantage.

Software security methods are spelled out under the Common Criteria standards. To an extent, open source operating systems have already been scrutinized under Common Criteria methods. How well have they fared? Can they be expected to improve in the foreseeable future?

These questions will be addressed in the context of the growing acceptance of open source software. Security assessments must take into account the threat environment, the vulnerability of the target, and potential loss due to a successful attack, and acceptable levels of risk. The software-defined radio community has a compelling need to understand software security and the benchmark by which it is measured.

1. INTRODUCTION

It could be argued that information warfare is as old as warfare itself. Throughout the ages the interception or alteration of communications has been used for military, political, economic, and sometimes social gain. Today vast resources are dedicated to both securing ones own communications and undermining the communications security of ones adversaries. As software-defined radios emerge as a new communications technology, we must understand their underlying security vulnerabilities and harden these systems accordingly.

A software-defined radio is fundamentally a computer system. The Software Communications Architecture has at its core a Posix-conformant operating system, often an Embedded or Real-time Operating System. It is critical that a designer understand the central role of the operating system in implementing system security.

Open source operating systems have recently begun to appear in embedded systems. Most prominent are embedded versions of the Linux operating system. Linux can be attractive to developers for a number of reasons. It is available for download at little or no up front cost, there is a large community of developers continually fixing or enhancing the technology, the available source code provides transparency, and metrics indicate it approaches the quality of many commercial software offerings.

However it can be shown that embedded Linux operating systems fall short in the area of security. This is a critical shortcoming in many SDR applications. System designers should understand and weigh these shortcomings before allowing embedded Linux to control a radio.

2. COMMUNICATIONS AND ESPIONAGE

It is impossible to identify the first time in history one adversary spied upon another. The ancient Greeks tell of spying amongst Gods and men, there are over 100 biblical references to spying, and Sun Tzu references spying in “The Art of War”, written in 500 BC. It can be said is that the gathering of intelligence is a very aged practice. One of the

primary methods of spying is to intercept or corrupt the communications of ones adversary.

The stability of nations, economies and societies depend on secure communications. The amount of effort dedicated to penetrating that security for military, political, economic or social gain cannot be overstated.

Today every major nation dedicates substantial resources to the interception of communications. Equally important, they dedicate resources toward securing their own communications. The United States National Security Agency employs more than 30,000 people who are dedicated to these two endeavors.

In the non-government arena, communications might be compromised for either economic gain or social purposes. Economically, corporations transmit sensitive information over a variety of channels and depend on security to protect financial transactions, competitive strategies or key intellectual property. Finally, individual citizens depend on secure communications to assure privacy and safety.

During World War II the battles of both the Atlantic and Pacific turned on the interception and decoding of military communications by the Allies. Identity theft strikes 9.9 million people a year at a cost of more than $5 billion[1]. To ignore communications security is to invite military, political or financial disaster.

Whatever the intended application, software-defined radio architects must design security into their systems. They must expect that adversaries are going to go to great lengths to penetrate their systems. To assume anything less is fallacy.

"History has taught us: never to underestimate the amount of money, time, and effort someone will expend to thwart a security system. It is always better to assume the worst. Assume your adversaries are better than they are. Assume science and technology will soon be able to do things they cannot yet. Give yourself a margin for error. Give yourself more security than you need today. When the unexpected happens, you'll be glad you did." [2]

3. OPERATING SYSTEMS AND SECURITY

An operating system controls the core of an SCA-compliant software-defined radio. The operating system controls all of the system's hardware functions. It also serves as the foundational platform for system security. If the operating system does not perform as it should, or if it is intentionally compromised, then the system's security architecture can be penetrated.

The operating system executes its machine instructions in supervisor mode because it must have access to all CPU, memory and I/O resources across the entire system. This means that any errant code, whether it was placed in the operating system inadvertently or through malicious intent, has access to sensitive data or critical peripherals such as storage and cryptographic devices. For this reason all code in the operating system must be held to a standard of trust and security that is at least as high as the most critical application or device in the system.

It is worth noting that many traditional embedded and real-time operating systems do not attempt to separate operating system code and application code. Instead, application code also executes in supervisor mode and therefore also has full access to system resources. This further challenges system security, as all code in the system, including application code, must be held to the highest security standards.

More modern operating systems partition the operating system and application code into separate virtual memory partitions. Applications operate in the processor's "user mode" and do not have access to system resources except for those granted by the operating system. The burden of security is on the operating system itself. This allows developers to use the access protocols of the operating system to develop a more verifiable security architecture.

4. DESIGNING TO THE COMMON CRITERIA

In order to verify that the operating system can be trusted it must be both designed and verified to meet a set of security criteria. This is the foundational logic behind the Common Criteria, an international standard that allows designers and consumers of systems to specify security features and then measure confidence in their implementation.

The specification of security features includes a Protection Profile that identifies the operational environment, security threats, and accepted risks for a given class of system. It sets forth requirements for the system to counter the security threats. An operating system designer uses the Protection Profile as a functional guide to designing a system's security architecture.

Once an operating system is designed and implemented it must be verified. The Common Criteria assigns Evaluation Assurance Levels (EAL) which designate levels of confidence that the architecture measures up to its security claims. These are ranked from EAL1 to EAL7, with the higher numbers indicating greater confidence. The process by which higher rankings are achieved involves the inspection of requirements, design, test and results documents. It also involves the formal auditing of the

software code, evaluation of the configuration management process, and methods by which requirements are through the development and test process. Finally, at the highest EAL levels, mathematical methods and proof may be required to make certain there are no hidden flaws or backdoors left open to exploitation.

According to the Common Criteria, "EAL4 is the highest level at which it is likely to be economically feasible to retrofit to an existing product line."[3]

EAL4 is generally achieved by verifying best commercial practices including the use of development environment controls and automated configuration management. It also requires a review of interface and low level design documentation and a moderate amount independent testing.

EAL4 operating systems are appropriate for users who "require a moderate to high level of independently assured security in conventional commodity [operating systems] and are prepared to incur additional security-specific engineering costs."[3]

Products that have been verified at EAL4 include such commodity operating systems as Windows XP. Linux has so far only achieved an EAL2 rating. However some commercial suppliers have announced plans to try to drive Linux toward EAL4 in the next couple of years.

Commodity operating systems such as Linux or Windows generally employ a "perimeter" defense. Privileged access is granted by way of passwords or permissions. But once past these barriers the attacker has full reign over the entire system. This is the military equivalent of posting guards on the fence, but with no further defenses for sensitive installations inside the perimeter.

4. VERIFICATION: SIZE MATTERS

It is far more economical and feasible to verify a small operating system than a large one. Today's approach to designing a secure operating system involves reducing the size of the operating system to a minimal kernel and moving everything else (device drivers, communications stacks, file systems) into user mode virtual address spaces. With less code in supervisor mode the scope of the verification process is greatly reduced. It is much more feasible to verify these operating systems to EAL 6 or EAL7, thus providing greater assurance of security.

This requires that the operating system be of a modular instead of a monolithic design. With monolithic operating systems it is difficult to separate the individual functions because there are too many interdependencies. In a monolithic operating system such as Windows or Linux, a large amount of code executes in supervisor mode, driving it beyond the practical reach of detailed verification. With millions of possible logical branches and interactions, all possibilities cannot be tested.

Certain embedded operating systems are designed as modular – or microkernel – operating systems. The core functions of the operating system are easily isolated from the peripheral functions such as device, file and communication management.

Microkernel operating systems can be verified to EAL6 or above. With a properly designed microkernel operating system, "perimeter" security penetrations are restricted to user partitions and deeper access to the system is blocked. Most important, this can be verified to the highest levels.

Many communications systems require operating systems with higher than EAL4 ratings, including defense and other highly critical systems. Mission-critical SDR's must be regarded as prime targets for espionage. An SDR's operating system security must be verified, otherwise all parts of the radio are open to penetration. Microkernel operating systems can be verified to the highest levels of trust, such as EAL6 or EAL7.

5. THE LINUX OPERATING SYSTEM

The Linux operating system was developed by Linus Torvalds as a free alternative to the Microsoft Windows operating system. Linux was designed as an "open source" system. The Linux source code is shared across a Linux development "community". Anyone can contribute to the operating system. Enhancements and improvements are then shared with every Linux user.

Although Linux was originally developed as an operating system for desktop systems or file servers, there are now versions that are being used in embedded systems. This establishes Linux as an alternative for use in software-defined radios and other critical communications systems. The question then arises as to the trustworthiness of Linux in defense and mission-critical systems.

In discussing Linux security, three areas of concern can be identified. First, the Linux kernel does not scale down to any reasonable size. Second, Linux system is developed using an ad hoc software engineering process, making it difficult to trace requirements and designs to actual code and verification tests. Third, the thousands of contributors to the Linux code base mean that adversaries could identify weaknesses or insert malicious code into the operating system code.

The size of the Linux kernel makes it very difficult and costly to verify. Although build configurations vary, an embedded Linux kernel is minimally measured in hundreds of thousands of lines of code. (In contrast, the NSA estimates that a certifiable microkernel should have no more than a few thousands lines of code for successful verification.)

Companies that extensively validate operating system code are familiar with the complexities and costs. Green Hills Software has validated its INTEGRITY Real-time Operating System to the highest levels of DO-178B flight safety standards. INTEGRITY is also being certified to EAL6 or EAL7. Costs of such validations approach $1000 per line of code. And in contrast to Linux, INTEGRITY is based on the more advantageous microkernel architecture.

Because Linux is the product of thousands of contributions, very little formal design documentation exists. While evolutionary development has its merits, the Common Criteria mandates that verifiable levels of security be supported by a strong software engineering process and traceability from design documents to code to test results. The dynamic nature of Linux development makes this an overwhelming challenge.

Any code that operates in the processor's supervisor mode has access to the entire system. It is critical that a verification trail exist not only for the Linux kernel, but also for any non-kernel Linux code that also executes in supervisor mode. This includes hardware adaptation layers, device drivers, communications stacks, and file systems.

The final concern involves easy access to the Linux source code base by adversaries. A considerable amount of Linux code is contributed or maintained outside of the US. It is argued by the Linux community that "many eyes" can catch an attempted insertion of malicious code. However one must question whether Linux contributors are both qualified and motivated to spot such insertions amongst the hundreds of thousands of lines of code. Even if some of the obvious security flaws could be intercepted, one has to assume that a well organized attempt to insert a security bug would be disguised. A determined adversary could hide malicious code by dividing it amongst several modules so that each code fragment appears innocent, but in combination they create a security breach.

A General Accounting Office report recently expressed warnings about unknown sources of software used in defense systems. It recommends that information on software suppliers be collected in order to assess risks associated with the use of foreign software [4]. While the international efforts behind Linux are admirable to many, the concerns express by the GAO must be taken into account by developers of defense systems.

Defense contractors routinely operate under strict security rules. Their engineering personnel are generally required to be US citizens or permanent residents, and are often required to hold security clearances. If this is the case, why would anyone allow the most security critical software in the system to be authored by an anonymous and international collection of programmers?

Open source software development and deployment may be outwardly attractive to many developers. But the Common Criteria requires proof that the operating system is designed, developed and tested to a set of requirements. When it comes to verification, the burden of proof is on the open source community to provide evidence of Linux security in accordance with the Common Criteria.

6. CONCLUSION

Defense communications systems are likely targets for security attacks. Software-defined radios must incorporate secure operating systems in order to defend against such attacks. The Common Criteria is the accepted method for assigning confidence levels in operating system security.

Until the open source community achieves sufficient Common Criteria EAL rankings, the Linux Operating System is not appropriate for defense critical systems including military software-defined radios. Software-defined radios employed in other applications, such as commercial telecommunications, public safety or for personal use, may also have critical security requirements. The open source community must be held to the same security standards as all other software development organizations.

[1] Federal Trade Commission, *Press Release*, September 3, 2003

[2] Bruce Schneier, *Why Cryptography is Harder Than It Looks*, published essay, 1996

[3] ISO/IEC 15408, *Common Criteria for Information Technology Security Evaluation, Part 3: Security assurance requirements, August, 1999, Version 2.1, CCIMB-99-033*

[4] United States General Accounting Office, *DEFENSE ACQ UISITIONS Knowledge of Software Suppliers Needed to Manage Risks*, GAO-04-678, 2004