A HIGH ASSURANCE WIRELESS COMPUTING SYSTEM (HAWCS®) ARCHITECTURE FOR SOFTWARE DEFINED RADIOS AND WIRELESS MOBILE PLATFORMS

David Murotake, Ph.D. (SCA Technica, Inc. Nashua NH, USA; dmurotak@scatechnica.com)
Antonio Martin (SCA Technica, Inc. Nashua NH, USA, tony.martin@scatechnica.com)

ABSTRACT

In 2004, 2005 and 2006, the authors provided details of wireless network threats discovered during wireless threat analysis studies exposing a potentially serious flaw in the security architecture of software defined radio (SDR), cognitive radios (CR) and wireless mobile platforms. The reconfigurable radio terminal, and the host to which it is attached, are potentially vulnerable to exploitation, malicious reconfiguration and denial of service as a result of Internet based attacks delivered via a wireless signal. These vulnerabilities extend to consumer mobile computing devices with embedded wireless network interfaces including WIFI enabled laptops, PDAs, Smart Phones and Cognitive Radios.

In January 2005, the Joint Tactical Radio System (JTRS) issued Change Proposal CP295, "Exposed Black Side" to address this new class of threats to SDRs (Figure 1). The Software Defined Radio Forum also considered these threats in security related Recommendations published in 2006. This vulnerability was realized in November 2006 with the "Broadcom Exploit" attack affecting world-wide consumer WIFI installations including those from Apple, Gateway, HP, Dell and eMachines. [1, 2]

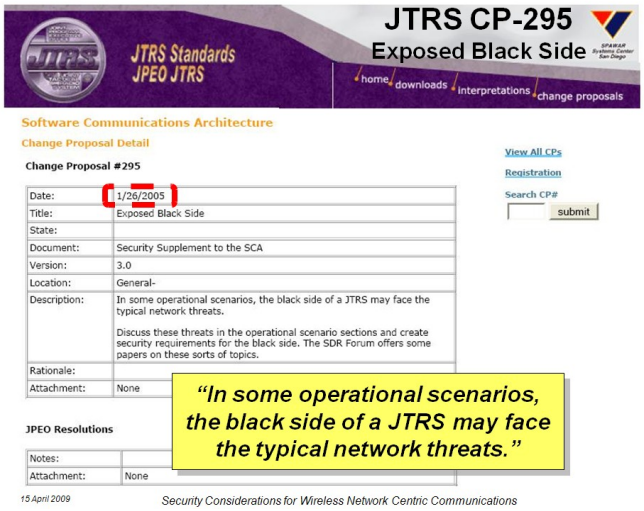


Figure 1

This paper presents an architectural approached called High Assurance Wireless Computing System (HAWCS®) as one way to address such concerns. HAWCS® leverages state of the art separation kernel technology, originally developed for Multiple Independent Levels of Security (MILS) applications, to fortify user end-node integrity and isolate "soft" operating system kernels and applications from network threats such as root kits without the need of additional hardware. HAWCS® addresses CP295 related security flaws in SDR and wireless mobile devices without the need for costly encryption hardware, allowing for greater assurance in mobile eCommerce and endpoint computing.

1. INTRODUCTION

During SDR06, Murotake and Martin showed the architecture of many SDRs and mobile wireless platforms was critically flawed in their approach for a high assurance design [3]. This fundamental flaw in the design of today's wireless mobile device, including many smart phones, wireless laptops and SDRs, allows compromise of the device by network attacks delivered using wireless signals on the radio interface. This allows attackers to easily bypass normal security measures, including firewalls, VPNs and encrypted connections to access points. An example was recently shown at a recent "Black Hat" technical conference; the researchers were able to demonstrate an application for finding vulnerabilities in the iPhone and Android SMS stacks that may lead to remote exploitation of such devices by sending a simple text message. Windows mobile devices are also being tested. [4]

2. DESCRIPTION OF PROBLEM

Virus/Malware/Worms and other such infections are on the increase. Technological security advancements are falling behind as "zero day" exploits become prevalent and system infections become impossible to detect. Except for limited research and development, security industry dollars are primarily focused on reactive solutions, not proactive. Patches and signature definitions are days, if not weeks behind active exploits. Attacks have not been just confined or limited to individual infected machines tied into "bot

nets" numbering into the thousands, utilized primary by individuals for financial gains. This playground is evolving as state sponsored activities are on the rise in the form of attacks on multiple US government facilities and agencies as seen as in Titan Rain attacks. [5]

2.1. Device Driver Exploits

In 2006, security researchers Dave Maynor and Jon Ellch demonstrated an exploit leveraging flaws in the device drivers for Apple's Atheros WiFi chipset allowing for root access without the target even being connected to a network. This attack translated to the Broadcom chipset and allowed for attacks on Windows based systems as well. [1,2] Broadcom inadvertently released a reference driver with a buffer over flow exploit; the companies utilizing the chipset in their computers (Dell, HP, Gateway, etc), simply used the reference drivers, porting them to their respective platforms.

Wi-Fi chipsets are constantly scanning for available networks as soon as power is applied, whether connected to a network or not. The exploit leveraged a buffer overflow by broadcasting a malformed SSID or network access point identifier. The malformed SSID was received by the target and the attacker was able to inject executable code at the device driver level, allowing for instant root access and bypassing all firewalls and virus checking mechanisms.

2.2 Vulnerable Monolithic OS/System Kernels

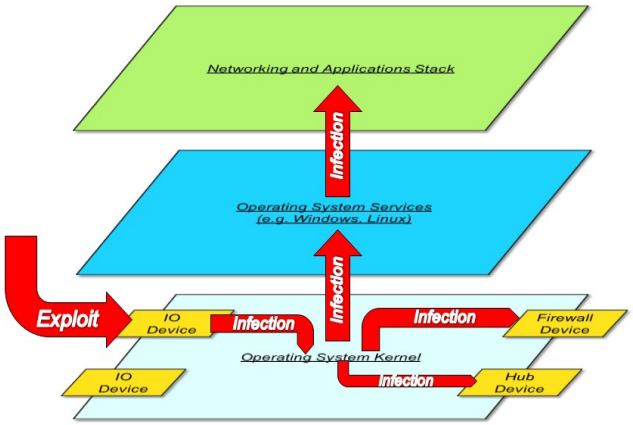


Figure 2

Exploitation of device divers allows an attacker to tunnel into the heart of the system, usually at root level privileges (Figure 2). If such an attack is realizable, the integrity of the whole system (and the security of its applications and information) may be called into question. Since a successful attack at any single point of entry can then subsequently corrupt the entire platform (including corruption of USB drivers, firewalls, crypto drivers, virus protection programs, etc.) the first successful exploit can result in a "catastrophic" security breach. In the vernacular of the hacker, "the system is owned".

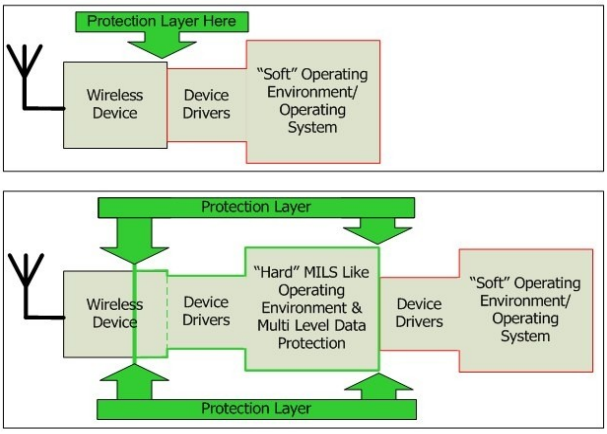


Figure 3

Current means of computational operation revolve around complex and relatively insecure monolithic operating systems and applications. The large size of their code base makes it nearly impossible to attain high level of assurances. [6] As a result, system integrity is always questionable. The monolithic operating system incorporates all operating system functionality into a single system or code group such as Unix, Linux, OS X and the various Windows operating systems. Even if a system is momentarily "stable", the addition of a new or updated device driver or application may introduce a flaw.. Viruses and other attacks can penetrate a system and hide themselves within and even under the operating environment.

2.3 Conventional Endpoint Defenses Are Inadequate

In today's mobile devices and endpoints (including SDRs and CRs), the vulnerability of I/O device drivers and monolithic system kernels makes the conventional approach of defending mobile platforms and endpoints through the use of software firewalls and virus checking programs inadequate against today's cyber threats. Many operating system based protection mechanisms work by detecting malicious actions or patterns described in a data base of malware, updates to which must be continuously maintained. The firewall examines incoming and outgoing data requests and decides if the information should be allows to flow or not. Virus checking programs also examine flowing (Email scanners for example) and static data (hard drive) looking for known and undesired data patterns. More advanced operating system based tools are examining state and behavior, looking for malicious actions from known sets of rules. These reactive means of

protection are fundamentally flawed and are becoming increasingly ineffective. If the operating system is the target of the attack, the attack must be first allowed inside before a firewall or virus checking program can recognize and stop the attack. The first thing many infections will do is disable firewalls and virus checking programs, prevent updates and then download a more advanced payload to take over a computer. (Figure 2)

Some of the most serious attacks employ "root kits" which cannot effectively be detected and removed by operating system level applications. Rootkits embedded themselves into, run parallel to and even underneath (by virtualizing) an operating system kernel. By successfully exploiting any input/output (I/O) device driver, the system kernel itself can be compromised. Even encryption (either software based methods, or hardware encryption using drivers hosted by the monolithic system kernel) are not sufficient to provide system security. The mobile platform must provide always-on, in-line, non bypassable security to achieve effective isolation of the vulnerable components from the protective mechanisms (firewall, malware detectors), networking stacks and applications (Figure 3).

3. MEMORY SEPERATION

HyperVisors provide an environment where monolithic kernels are virtualized. The concept of virtualization has been around since at least the 1960 with IBM's CP-40 and soon after CMS. [7] IBM continued to utilized virtualization and in the mid to late 90s, as computing resources increased, server based virtualization began to pick up steam with other vendors. In the late 90s, virtualization technologies appeared in the desktop environment and in recent years have made headway into the embedded space.

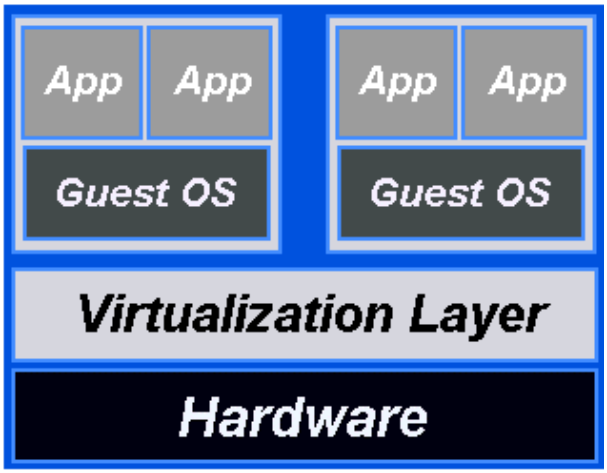


Figure 4

Hosted virtualization or type 2 hypervisors, have a standard operating system (Windows, OS X, Linux) within which a virtual machine runs as an application(s) that hosting another operating system. (Figure 4). This solution begins to approach a solution for some of the problems with securing a monolithic kernel. An operating system is contained in the virtual machine; all I/O into and out of the virtualized operating system can be inspected from the host operating system and the system state of the contained operating system can be monitored. There are some weaknesses to this approach (when considered from a HAWCS® point of view):

1. The data flowing into the contained operating system must first traverse through the hosting operating system's drivers and IP stack. Any vulnerabilities in the processing of the data allows an attacker exploit and take over the host and then the contained operating system.
2. The contained operating system runs within a virtual machine. It is possible an attacker could cause a memory overflow in the virtualized OS into the virtual machine application, escalating privileges to eventually assume control of the host OS.

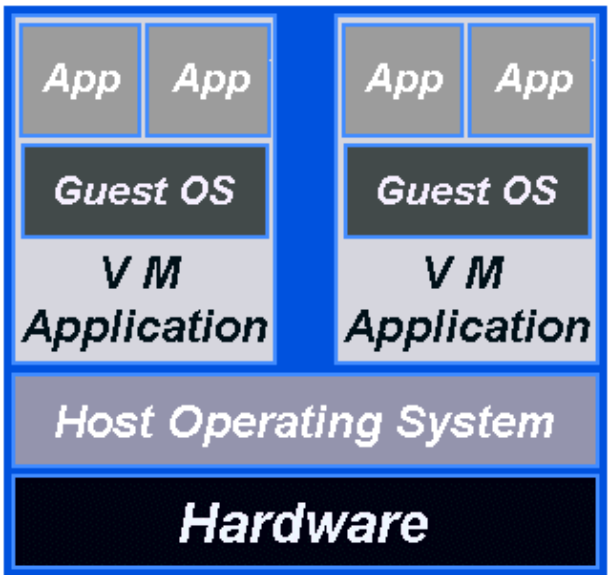


Figure 5

A second form of virtualization is the "bare metal", type 1 hypervisor. (Figure 5). This approach does not contain a host operating system but only a very slim virtual machine on which the virtualized operating system runs. This type of virtualization offers enhanced performance over the type 2 (less memory abstractions) but has its own set of flaws:

1a. In some designs, the network I/O and drivers are hosted by the contained operating system. Thus it is still subject to attacks on the drivers.

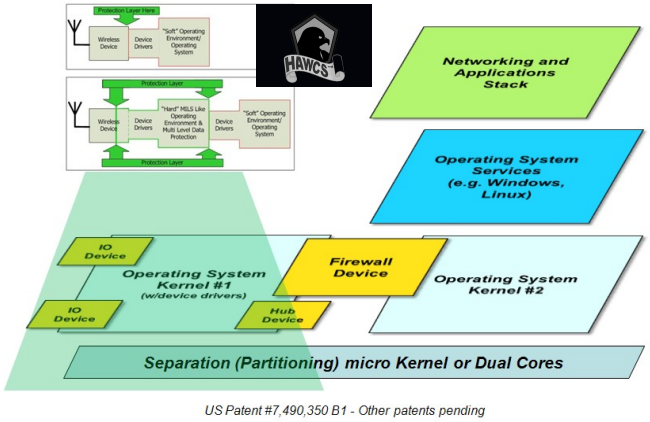
1b. In other designs, the device drivers are hosted in the virtual machines (as apposed to a memory map pass through), these are subject to attack allowing the subversion of the host kernel.

2. I/O is not inspected prior to entering the contained OS.

The advantage of this type of approach is the enhanced performance and the limited kernel footprint for the virtual machine helps to limit the potential exploits and increase performance within the conainted operating system.

Figure 6

Separation and virtualization can also be achieved using modern "separation kernels", such as those used in Multiple Independent Levels of Security (MILS) or DO-178B high assurance separation kernels. The United States Government is supporting the development of MILS which allows various levels of classified memory containers, operating in parallel on the same processor and memory. In the past, to handle multiple levels of security, a system(s) had to employ an independent processor and memory for each security level or more simply, multiple separate machines. As depicted in Figure 6, the typical conceptual usage for MILS technology is to separate various information classifications to prevent leaks. In the course of MILS development, separation and virtualization micro-kernels with very high Evaluated Assurance Levels (EAL) were developed and are being deployed as publicly available, embedded real time operating systems. Much like the hypervisor, the MILS kernel separates processing groups or elements that are used to segregate information of different levels; yet it can also be used for other purposes.



US Patent #7,490,350 B1 - Other patents pending

Figure 7

4. HAWCS® ARCHITECTURE

In HAWCS®, a separation kernel or virtualized memory partitions are used not as security containers but to create an in-line, non bypassable security-in-depth architecture. (Figure 7) This is accomplished by using the memory partitions to separate functionality (as opposed to security levels) and then to define and limit exactly how data is allowed to flow from one partition to the next.

As shown in figure 8, a memory partition holds the device drivers for the network interface. If there are multiple network I/O devices, each can have its own memory partition or they can reside as a group within one. Placing each driver into its own partition adds complexity and overhead (yet another context switch) but adds greater containment, protecting the other drivers from a cascading attack. It would be possible for a buffer overflow attack on one driver to overwrite other another, giving an attacker access to protected networks. Network device driver communication with other memory partitions is restricted; the network driver partition must only exchange information (status, configuration, data) with the memory location containing the system's protective mechanisms.

The protection layer, as shown, contains a network firewall and a malicious data inspector. These processes accept incoming data from the network driver memory partition(s), inspect the data, take appropriate action when an issue is found and forward the traffic on to the appropriate memory partition with the contained operating system. Likewise, it inspects outgoing information from the contained operating system to the network device drivers. In this way, all data is inspected and information cannot flow directly between the contained OS and the device drivers.

The protection layer can abstract cryptographic functionality away from the contained operating system. It can provide virtual private network capabilities (like IPSEC) to secure communication and also encrypt/decrypt

data stored on disk (hard drive, USB, etc), protecting static information.

The operating system is contained within its own memory partition. Any infection trying to "escape" cannot disable the firewall or virus checking functions. Like the device drivers, the only outlet for network data (and file storage is chosen) is to the protection layer for inspection.

Yet another memory partition (not shown) can be established with read access to all other partitions. It would host a system monitor, actively scanning the memory and state of the processes running in the various partitions (like Windows, the network drivers, firewall) for abnormal behavior. Rootkit like infections within the contained operating system would have a more difficult time hiding from such a malicious behavior / pattern inspection tool.

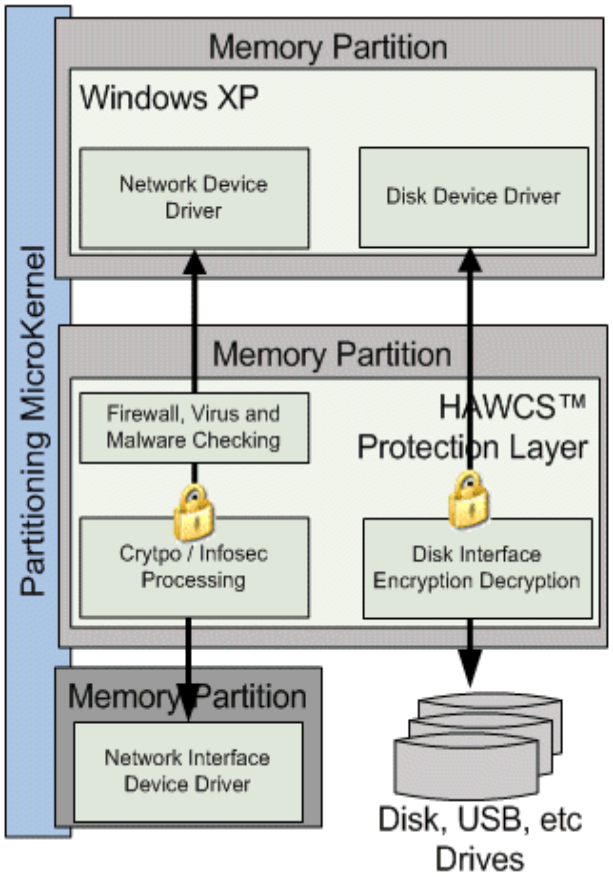


Figure 8

Such a protection architecture and supporting mechanisms and services carry a cost. Virtualization of an operating system increases memory abstractions and thus access latencies. Moving data between memory partitions in a limited and secure manner will degrade performance and multiple memory partitions require more kernel context switching and require more memory. With today's multi-core, multi-gigahertz and multi-gigabyte of memory machines, it is questionable how much a problem this presents to the normal user.

5. HAWCS® REQUREMENTS

Always invoked: Protection services must always be invoked. The security layer or partition must be the first environment booted after the partitioning microkernel boots. There should be no easy way to turn protection services off. The partitioning microkernel should be a high assurance kernel, such as those employed in MILS or DO-178B certified RTOS. The memory partitioning must utilize a hardware memory management unit (MMU).

Inline and non-by passable: There must be no way for a network attack to bypass protection services or mechanisms in the security layer. An example of a bypassable security function can be found in USB based hardware firewalls. The issue becomes the operating system must be trusted to route the traffic to the device. In the case of a successful exploit, this routing can be corrupted. Such corruption can render the protection worthless. Device drivers for different network interfaces should be contained in separate memory protected regions.

Detection: The secure system must incorporate an effective detection service capable of identifying possible intrusion or tamper incidents.

Logging: The system must log critical incidents, such as detection (or possible detection) of intrusion and tamper and the countermeasures taken.

Reporting: When queried by a higher level monitoring function in a network-centric system, the protection services must authenticate the query, and upon successful authentication, report critical incidents. When queried by an authorized entity, security services must be capable of reporting part or its entire log file / incidents, such as intrusion or tamper detections. The report may be to a user console.

Defensive Mechanisms Location: The placement of defensive services and mechanisms must be such that they form a non-bypassable barrier between the source and the point of attack. Trying to stop an attack at the point being attacked fails the premises of defense in depth.

Ability to Inspect and Intervene: The monitoring and control applications must be able to inspect all running processes, memory mapped I/O and memory in common memory areas, including the virtualized partition(s) containing the system kernel, OS services and utilities,

device drivers, firewalls, network stacks and applications of the protected part of the mobile platform or endpoint.

Containment and Countermeasures: When an attack is realized, it must be contained in such a manner so as to cause as little harm to the protected system as possible. If so configured, security system shall attempt countermeasures through removal of malicious software and repair of damage caused by the attack.

6. CONCLUSIONS

Attacks on mobile wireless devices, using delivery of malicious software delivered via radio signals to wireless devices, are now a reality. Systems such as WIFI enabled laptops, wireless mobile devices and smart phones have been successfully "hacked", validating the concerns of the authors from previous years. In recognition of these threats, JTRS Change Proposal CP-295 was issued in January 2005. However the threat extends to many SDRs, CRs and mobile wireless platforms. The patented HAWCS® architecture, demonstrated to US Government officials in 2006, provides one approach to providing non-bypassable security-in-depth which "fixes" the flawed design approach making today's mobile platforms vulnerable to this class of exploits.

This paper presented is an overview of High Assurance Wireless Computing System (HAWCS®) (US Patent # 7,490,350 and pending application #20080016313).

7. REFERENCES

[1] R Lemos, *Maynor reveals missing Apple flaws*, Security Focus, Mar 2007, http://www.securityfocus.com/news/11445

[2] R Naraine, *Alarm Raised for Critical Broadcom Wi-Fi Driver Flaw*, http://www.eweek.com/c/a/Security/Alarm-Raised-for-Critical-Broadcom-WiFi-Driver-Flaw, November 2006.

[3] D. Murotake and A. Martin, *A High Assurance Wireless Computing System (HAWCS®) for Software Defined Radio*, SDR Forum Technical Conference,

[4] C Mulliner, C Miller, *Fuzzing the Phone in your Phone*, Black Hat Technical Security Conference: USA 2009, June 2009

[5] N Thornburgh, *The Invasion of the Chinese Cyberspies (And the Man Who Tried to Stop Them),* Time, August 2009 http://www.time.com/time/magazine/article/0,9171,1098961-1,00.html

[6] W. M. Vanfleet et al,. *MILS :Architecture for High-Assurance Embedded Computing*, CrossTalk, The Journal of Defense Software Engineering, August 2005 http://www.stsc.hill.af.mil/crosstalk/2005/08/0508Vanfleet_etal.html

[7] R. J. Creasy, The Origin of the VM/370 Time-Sharing System, IBM Journal of Research and Development, Vol. 25, September 1980