

# Connected devices sound alarm for embedded security



These *dumb* devices present relatively few security risks. Even if reachable over a network, their code is in ROM, they present few resources to exploit, and the content itself is usually uninteresting intellectually and property-wise. With the addition of first-generation embedded operating systems (RTOSes) and basic TCP/IP networking, these client devices moved toward exploitable *peer* status on public and private networks. The deployment of desktop and enterprise-derived operating systems like embedded Linux, with integrated networking, local file systems, and multi-user/multi-session capabilities, presents new and unprecedented security challenges to embedded designs.

Today, readily available LAN and WAN connectivity is changing the definition of embedded systems. Across the spectrum of embedded design from handheld devices to medical monitors, to industrial control systems to large telephony systems, and for defense applications as well, it is hard today to find an application that is not *connected*.

This connectivity manifests through several media:

By Bill Weinberg

During the last 18 months, even starting before the events of 9/11, security issues have leapt to the forefront in embedded and embedded Linux application design. Factors contributing to this new security consciousness include:

- Embedded/client devices attaining peer status with server systems, in terms of computing power and resources
- An upsurge in content-rich consumer electronics applications, especially those that provide conditional access to audio and video content (MP3/MPEG-based personal audio devices, PVRs, DVRs, STBs, etc.) and the strictures in place and envisioned by DMCA and the Hollings Bill before Congress
- Recent waves of viruses that today attack desktop and server systems and threaten to invade the embedded/pervasive space as it matures
- The silent introduction and acceptance of open source into government agencies, from the Customs Service to numerous science projects, up to the NSA and its Linux efforts

Since 9/11, embedded security concerns have further amplified, companioning a boom in enterprise security efforts and a rise in aerospace and defense design-starts. On the enterprise and infrastructure side, developers building systems as diverse as phone switches, routers, and access equipment are now emphasizing security on a par with other functionality. Indeed, we see every connected device design showing a new level of awareness of the risks inherent in hooking up to public networks.

## Changing face of embedded, pervasive connectivity

In the past, security hardly concerned embedded designers. In the '80s and for most of the '90s, embedded devices were resource-constrained, hardware-focused, and primarily standalone designs. The prototypical embedded system was the *toaster* – a simple, mono-function device with primitive user interfaces, no networking or other connectivity, and of limited intelligence (*just enough* CPU and RAM).

Reprinted from [Applied Computing](#) / Fall 2002

- Ethernet, via standard, low-cost integration of networking into embedded processors
- Similar integration of wireless interfaces into chipsets and reference designs
- Legacy and proprietary interfaces that reach the 'Net through gateways

With this newfound connectivity, embedded systems designers face security challenges usually associated with enterprise and Web applications.

## Embedded security – four focus areas

Applications based on a smart embedded OS such as Linux must address four primary areas of embedded security:

1. Access control, especially wireless access
2. System resilience/robustness in response to untrusted/rogue programs and attacks
3. Security of data streams to and/or from connected embedded devices
4. Conditional access to sensitive and high-value content

## Limiting access

The challenge of limiting network access to embedded devices differs little from the same challenge in enterprise security. Internet appliances and traditional embedded applications alike present opportunities for intrusion and exploitation when they extend their control and configuration interfaces over a network. Devices with visible consoles can fall victim to social engineering, password cracking, and login spoofing, as can systems running telnet sessions. Embedded Web interfaces can be defaced and hacked through Common Gateway Interfaces (CGIs). Embedded Web browsers, e-mail, and other client software need to be secured against malicious content and untrusted downloads.

Secure standalone devices are not immune from attack either. *Toaster* systems need to protect themselves against intrusion over simple user interfaces. Even without a login console, exploits with standard input sequences can offer unwarranted control over a system. Often manufacturers plant *back doors* for QA, line test-

ing, and technical support that can be hidden in plain sight. A typical back door can be an extra serial port (e.g., on a point-of-sale terminal), or a series of undocumented *chords* of buttons can key it into operation. Examples include:

- Overriding mechanical safeguards for printers
- Disabling conditional access in multimedia devices
- Tricks for beating video games

### Focus on policy

The growing market for consumer electronics and personal communications devices (phones, Personal Data Assistants (PDAs), etc.) puts tiny servers and vulnerable clients into the hands of users unaccustomed to security disciplines. These users are likely to regard such good security practices as a nuisance and sometimes disable safeguards altogether.

For embedded *peers* (with workstation-like CPU, memory, file system, and networking), device OEMs, sales channel staff, carriers, and end users also need to mimic the security-conscious behavior of enterprise IT staffers. Embedded platforms can inherit many of the features but also the liabilities of their corresponding enterprise operating systems:

- Embedded Windows derivatives (CE, Stinger, Embedded XP, etc.), which are heir to Windows security architectures
- Embedded Linux implementations that track mechanisms in the standard Linux kernel (like POSIX capabilities)
- Related open source projects (like Medusa and IPs/WAN)

Even proprietary RTOSes reflect the security architectures of stacks they deploy, like BSD and derivatives. Just as these workstation and server releases receive regular patches and security updates, so might embedded devices need to be patched against exploits, viruses, and other security risks.

### System resilience

Software and hardware architecture of an embedded device determine the types of possible exploits, containment strategies, and overall consequences of security breaches, independently of access controls. Multi-user/multi-session OSes like embedded Linux can offer more opportune points of vulnerability, through shells, network daemons, Web interfaces, scripting languages, etc., but they also feature greater resiliency and stronger defenses against rogue programs, Trojan horses, and other programmed exploits.

Buffer overflows and other exploits in a traditional embedded OS immediately yield access to all program instructions, user and system data, and I/O devices in the *flat* address spaces common to RTOSes and embedded executives. The virtually addressed protected program and data in Linux/POSIX processes isolates programs from one another. This model also helps to limit the scope of exploit damage. Exploit-inspired failures can be isolated to single, restartable processes instead of requiring costly system-wide rebooting (see Figure 1 on next page).

### Network stream security

Embedded devices that leverage public networks or that have data and control streams that flow outward through gateways to those networks are on a par with enterprise systems in terms of stream security. Legacy embedded nodes, like many existing desktop and server installations, do not leverage the inherent security mechanisms available in the protocols they employ, or they use software that predates mechanisms like available IPSec in IPv4 and IPv6, SSL, and so on.

Use of simple or proprietary datagram-based networking is no guarantee of security. Such ad-hoc schemes are not immune to sniffing and spoofing, giving crackers access to data and control information for the client device, the server it attaches to, or both.

Embedded designers need to understand their options for authentication and encryption of network streams, make informed choices about applying those options to data and control over their fabric of choice, and provide and document configuration options to their customers – whether integrators, carriers, administrators, or end-users.

### Content protection

Very much unlike the enterprise world, many embedded applications need to deny even trusted users access or offer only conditional access to secure content. Examples include:

- Protection schemes for digital media like music CDs and DVDs
- Pay-per-view resources like cable TV movies, transaction control information on point-of-service (POS) devices, classified information on battlefield and other military applications
- Trade secrets embodied in program code itself in a variety of industrial applications

Many encryption schemes exist for the developer and content provider alike; with long public key encryption methods favored for generic document and message protection (DES, PGP, etc.), and various proprietary, industry- and application-specific methods like DeCSS for DVDs and extremely robust encryption for military and intelligence applications. In the area of content protection, again, obscurity is no guarantor of security, as shown in the recent newsworthy cracking of DeCSS.

In the commercial arena, conditional access and content protection tend to aggregate with industry-specific middleware, leveraging Java security or schemes built into platform specifications like the Multi-media Home Platform (MHP).

### Performance impact

TANSTAAFL – *There Ain't No Such Thing As A Free Lunch*. Embedded security doesn't come for free. Encryption/decryption and processing of secure protocols exacts a toll in throughput and overall system performance. Policy brokering strategies, like Medusa, can induce system call delays and other overhead that saps 10-15 percent of throughput. A nimble node on a high-speed network, nominally approaching wire speed, and server applications able to service hundreds of unsecured connections can slow to a crawl by needing to perform block encryption and/or decryption on data streams and content.

Such extravagances as scaling memory, increasing CPU clock speed, and adding nodes to the network, unavailable in most embedded applications, do address performance challenges in the desktop and enterprise. One solution is to offload security-related computation to coprocessors, just as DSPs handle signal processing, and network processors handle packet forwarding. Security processors (SPUs) support either *look-aside* processing wherein the SPU performs operations on behalf of the main CPU as part of control-plane processing, or *in-line* security processing with data and control streams traversing the SPU itself.

In the absence of such specialized hardware, the distributed architecture of many network and telecommunications applications provides opportunities for sharing the encryption load, as line cards and system blades boast multiple, powerful CPUs with



enough spare cycles to *crunch* various parts of the encrypted streams as they come in.

### Open source vs. obscurity – Which is most secure?

The events of 9/11 have also reignited the debate over whether open or proprietary systems are inherently more secure, with opinion-makers from the public and private sectors weighing in on both sides of the issue.

Traditionally, embedded systems have relied on simplicity and secrecy to ensure security. Simplicity was that of the *toaster* cited above. Secrecy came from the small audience for embedded OS platforms, little or no access to source code, the relative obscurity of embedded hardware architectures, and the isolated nature of embedded devices themselves. This privileged, arcane position for embedded applications is approaching the end as developers leverage more COTS hardware and software in devices that are becoming pervasive throughout society.

This trend towards pervasiveness shines a light on the technologies and practices of the proprietary commercial OS business. The companies that are developing proprietary platform software have a limited number of developers and cannot guarantee security because so few *eyeballs* ever get to see the code involved. By contrast, open-source embedded OSes benefit from broad community disclosure, quick exposure and repair of exploits, and an emphasis on standard practices.

Detractors of open source are quick to point out the holes and exploits recently uncovered in Linux (e.g., the recent IRC-based Netfilter and various buffer-overflow exploits) and also periodically published and patched for the BSD family (Open/Net/FreeBSD, etc.). They claim that the exposed code bases are an invitation to hack and to crack.

The open software position is that closed-source suppliers have a stake in hiding all knowledge of implementation and history of exploits. A base position of plausible denial then provides commercial opportunities for service contracts, after-market distribution of patches, updates, and service packs. They are as quick to cite the thousands of known (and presumably more unknown) security holes in Microsoft products, discovered only through deployed use and costly breaches. They question how many more undocumented security holes exist in platforms like VxWorks, pSOS, and the 200-odd RTOS products on the market.

### Security as game theory

Reasonable people can agree to disagree about whether open or closed approaches are more secure. A very powerful argument, however, for open systems security proceeds is as follows: both open and closed systems are subject to attack and have been cracked with regularity by *hawks*. Access to source code may offer some small advantages to potential exploiters, hawks, but binary-only products have fallen victim as often to malicious hacks. The real advantage to open source lies with the *doves*. While there is no guarantee that the open source community doves will repair security holes and exploits in a given time period, there does at least exist the *potential* for such amelioration. With closed systems, there is no potential for broad-based defensive efforts outside the vendor's own team, and security holes often remain unplugged for prolonged periods, even years!

### What now?

After sheltering for decades in proprietary and obscure designs and practices, embedded systems are emerging into the world of connected, mainstream information technology. The volume of

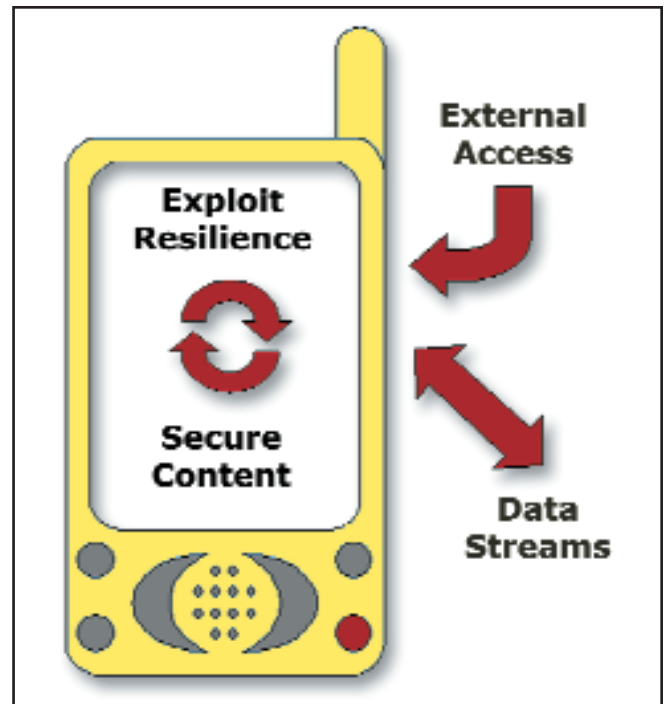


Figure 1

pervasive devices is surging past enterprise and infrastructure applications. Digital cell phones and other handheld applications, automotive telematics, digital video entertainment, information appliances, and a myriad of other more traditional embedded applications outnumber the servers and desktop computers to which they connect. No longer can these pervasive devices rely on secrecy and obscurity to offer secure and reliable services. To meet these needs and to bring their applications to market faster, embedded developers are turning to standards-based security and more often to open systems and open source.



**Bill Weinberg** is director of strategic marketing and evangelism for MontaVista Software, Inc. Bill focuses his 16+ years of industry experience on advancing MontaVista and embedded Linux in today's dynamic pervasive computing marketplace.

Bill's background includes extensive embedded and real-time experience with expertise in OS and software tools. Previously, he managed Java, embedded Web technologies, and alliances programs at Lynx Real-Time Systems. As Brazil country manager at Acer, he spearheaded introduction of Pentium technology and Internet-ready PCs. As lead product marketer at Microtec Research, he introduced embedded C++, ANSI C, and monitor-based real-time debug technologies.

Throughout his career, Bill has been a featured speaker at industry conferences and a frequent contributor to electronics and telecommunications publications.

You can contact Bill via e-mail at [bill@mvista.com](mailto:bill@mvista.com). For further information about **MontaVista Software, Inc.** and its products, visit [www.mvista.com](http://www.mvista.com).