# Security Testing, Vulnerabilities and Exploits in Operating Systems Used in Control System Field Devices

Daniel Peck
Digital Bond, Inc.
Sunrise, Florida
peck@digitalbond.com

**Abstract:** A great deal of effort has been spent on finding vulnerabilities and securing operating systems commonly used on workstations and servers. Embedded devices such as RTU's, PLC's and other field devices used in SCADA and DCS also have an operating system, but the security of these operating systems has been largely ignored. Embedded OS are rarely patched and security mechanisms available in the processors are rarely used.

In this paper the authors review the processors and embedded operating systems commonly used in control system field devices. The available security features and situation with security patching is also discussed. A variety of field device platforms are then subjected to very old attacks that were first used against many workstation and server operating systems more than ten years ago. The results from this legacy attack testing are provided and analyzed.

**Keywords:** ARM, Embedded Linux, Embedded Systems, Embedded Windows, Intel x86, Power PC, VxWorks

## 1 Introduction

Microsoft Windows, Linux and other operating systems get a lot of attention from attackers and security professionals trying to patch and prevent attacks. Conversely, operating systems in RTU's, PLC's and other field devices used in SCADA and DCS rarely are even considered when security is discussed.

There are two sides to this lack of attention. First, there are a number of proactive security measures that can be used by a security aware field device vendor. And second, the vulnerabilities, both latent and known with a patch available, in the embedded operating system are similar to those found in workstation and server operating systems.

In this paper, the processor and embedded operating system from a variety of control system field devices are analyzed.

## 2 Processors

The selection of a processor for an embedded device typically involves computing power, price, power consumption and other price / performance considerations related

to the primary purpose of the embedded device. However the processor choice also limits and determines the security tools and features that a developer will have available for the implementation. In the currently deployed world of embedded devices the choice was usually between an ARM, a Power PC chip or a custom FPGA system. In recent years, Intel's x86 processors have been considered and are increasingly being selected.

The instruction sets implemented by a processor offer variable amounts of security. Some instruction sets are more difficult to exploit because of security features such as memory protection and privileged instructions. While potential vulnerabilities and security protection varies by the processor selected, the choice rarely hinges on an analysis of the security features that one might have over another. That said, the authors contend there are some important security attributes that should be understood and factored into the processor selection decision.

Not surprisingly, the x86 processors tend to have a more developed set of security features because this platform has been the target of more mainstream attacks due to its widespread deployment in PC's and other consumer products.

## 2.1 No Execute (NX) Bit

The no execute bit feature allows data in designated areas of memory to be non-executable. This prevents any code that is loaded into these NX designated memory areas from being executed. This is especially effective to prevent code inserted from an adversary's buffer overflow attacks from being executed. If a vendor knows a memory area is only to be used for data storage, it can be restricted from execution with the NX bit.

The NX feature can and has been emulated in software on various operating systems to have some level of protection even if the underlying processor does not support it. The Linux kernel has two different versions available, Exec Shield and PaX. Intel offers the XD, Execute Disable feature. And ARM's typically label this feature as XN, Execute Never.

## 2.2 Address Space Layout Randomization

Address space layout randomization is a security technique that makes key areas of code and data more difficult to exploit by loading them into different places of memory at each boot. This makes attacks relying on a "return to libc" more difficult, and while it does not do anything to prevent a crash of the system it can be very useful in preventing exploit code from running. This security measure, while first published and implemented in 2001, is still uncommon in all but the most modern operating systems, and is even more rare in embedded systems.

This is, as much as anything, due to design/implementation issues with embedded systems in that in most of them load the entire embedded OS image to a fixed position at boot time, with no external modules. No field devices tested or investigated for this paper implemented this security measure.

## 2.3 Stack Canaries

Stack canaries are another common means of preventing memory corruption. A stack canary inserts a known value into a stack before the return address. This value would typically be overwritten in an overflow attack. The canary value is then checked to make sure it has not been altered before the return pointer is used. Stack canaries are implemented in software and are very easy to implement. They usually only require the developer to pass a given flag to the compiler to activate. However all too often this effective technical security control goes unused.

Stack canary implementations have been available in Visual Studio since 2003, and implementations have existed for GCC since 1997 - - a standard method has been in place since GCC 4.1. This should have a big impact on code security since GCC can compile for nearly any architecture and operating system. Many development environments use GCC as its backend, but often security options are hidden.

# 3 Embedded Operating Systems (OS)

The security, or lack thereof, of embedded OS is not surprising given the environment they typically were deployed in. They were designed for harsh, relatively inaccessible and demanding environments, such as control systems, where reliable operation was paramount. They were not designed for malicious or sloppy environments where unexpected or inappropriately formatted data or commands reached an interface. If the authors were forced to characterize the embedded OS's in broad strokes, we would call them difficult to exploit, but often very easy to crash or make malfunction.

The embedded OS choice is important and will affect the potential security of the device, not that there is a single correct choice for all situations. Almost as important as the choice of embedded device is the development teams' understanding and use of the security related tools available in the embedded OS / processor combination.

So why worry about OS choices, aren't attackers usually looking at the application level? Yes and no, security and attempts to compromise security tend to move up and down the stack fluidly. A trend in the control system community is to focus and rely on addressing based access control data link and network layers as the security mechanism. MAC and IP addresses are easily changeable by even a novice attacker, and therefore most attackers can attempt to compromise the lower layers in the stack. If an attacker can compromise a device at the data link, network or transport layer then there is no need to attack the application layer because code execution in the network stack normally has system privileges.

## 3.1 Unknown Applications and Debugging Interfaces

Embedded OS are typically distributed to device vendors as a prebuilt image for a processor. The device vendor then adds functionality to this image, communicates to other device boards or components, and adds other capabilities to form a product. The embedded OS image that is distributed often has additional code and capabilities that are not used and should not be shipped in the final product. Some of these items may not even be documented by the embedded OS vendor nor known by device. As an example,

JTAG pins that are unintentionally left on production runs and allow end users to find far more about the internal workings of the device than intended.

Many embedded OS ship with services that may not be required by the device. Field devices often are seen with unnecessary TFTP, FTP and Telnet servers that are preconfigured and ready to run from the embedded OS image. Sometimes the servers require a firmware modification to activate the servers, while other times the servers simply require connecting to a nonstandard port that they're always running on. In other embedded OS, a hidden configuration in a management interface can turn on or off and configure the servers. These servers are usually just included to make like easier for the developers. They don't receive much attention, but they can be leveraged for exploits.

For example, VxWorks and embedded Linux based systems often have a minimal shell installed that can be interacted with once code execution is gained. In general these services/applications tend to be an easy target because they are far removed from the core functionality of a device. The service is the more likely to be an afterthought for development and QA teams who are pushed by timelines. The services are often ignored by the device vendor because they are not part of the functionality that is required and will be used in the embedded OS.

At S4 2009, the authors documented vulnerabilities in field device management interfaces [1]. Vulnerabilities were found in the FTP server included with some VxWorks images, as well as the web management interfaces of various VxWorks and FPGA based devices. A year later most or all of these vulnerabilities likely still remain in deployed field devices, as many of them still exist in the firmware images available from the vendors.

## 3.2 Patching

Patching embedded systems is historically rare in the control system community. The PLC's, RTU's and other field devices follow a deploy and don't touch approach. This is due to a variety of legitimate technical reasons and many cultural reasons. However as the embedded systems have become more interconnected to other IP networks, albeit with security perimeters, and more similar from a hardware and software perspective to frequently attacked targets such as PC's, this no patching approaches warrants another look.

Definitive patch information for embedded OS is difficult to obtain, as patches are issued to device vendors, not end users, and vendors often do not patch their own firmware. Security patching of the embedded tends to wait until a device vendors major firmware release, and then the security patches are rarely documented. It is often a silent fix that hides the security ramifications of the upgrade decision from the end user.

### **3.2.1** Linux

The Linux OS is frequently patched, although many of these patches do not have a security ramification, and the average developer does not understand many of the security patches. In fact it is sometimes difficult to determine if a patch affects security and silent security patch fixes are a current subject of debate in the Linux community. It

is even more difficult to determine which updates affect the security of an embedded Linux build. Consider the recent SCTP vulnerability, CVE-2009-0065, as an example. The applicability is dependent on the whether the developer uses a modular or a monolithic kernel; this could determine if they need to perform a full recompile/rebuild of kernel.

Patch management also has tradeoffs involved in deciding whether to use an embedded distribution or to build up from the kernel source. When building from a distribution the vendor is a step removed from the kernel developers and security patches may be delayed or undistributed based on the thoughts of the distribution developers/integrators. But other tradeoffs come in from building directly from the kernel source and integrating patches and support applications in house. Security information is often difficult to discern, with critically ratings varying between distributions and are often unrealistically low.

The Linux 2.6 kernel, of course not counting supporting applications which are too numerous and varied to account for in this paper, has had 385 vulnerabilities since 2003.

### 3.2.2 VxWorks

VxWorks updates, like many embedded system updates, are usually distributed as images by the vendor. There are only a small number of updates. A typical device vendor will likely publish less than five updates over a field device's lifespan. These updates may contain major security changes, but they are seldom documented to the end user as the updates to the OS from Wind River.

All VxWorks updates have to be passed through and verified by the device vendor that their code works correctly before being distributed. Silent security patches, or at the very least minimal information, tends to be more common in VxWorks based systems than other embedded operating systems.

### 3.2.3 Embedded Windows

While still somewhat uncommon in the control system device space, embedded Windows is growing in popularity. Some vendors have moved their future development to Intel x86 chipsets and embedded Windows. And while Windows as an HMI/backend server platform is changing the way that control system operators approached patch management, it is still unclear if embedded Windows on field devices will do the same. Patching these systems will have similar issues that other vendor provided Windows systems have. Embedded windows codebase is not very different from traditional desktop/server installs of a Windows OS. The codebases are absolutely massive, general purpose systems, and as such they have security issues. Each month Microsoft releases security updates, and those for the embedded versions are typically released a few weeks later.

However the larger problem with this again occurs with the delay between Microsoft issuing a security patch and the field device vendor testing, applying and making that patch available to their customers. To their credit Microsoft has provided methods to make this update process much easily for end users by providing automatic update

mechanisms, though the authors doubts that these will be used in control system environments anytime soon.

# 4 Embedded OS Security Testing

Over the past two decades most consumer based OS have suffered from and corrected a number of vulnerabilities. It was interesting how the same problems affected different operating systems; a case of common programming errors by different teams. The question is had embedded OS suffered from the same problems? And if they had, were the problems addressed?

The authors performed the testing using freely available tools, primarily the python packet crafting library Scapy, along with various shell utilities. Sulley was used as the fuzzing framework for specific application protocols included in embedded system builds.

## 4.1 IP Size Field and TCP Windows

The IP size field is the first example of a common error that led to vulnerabilities. Many network stacks do not evaluate parameters and process invalid data coming into the stack. In the IP Size Field example, an error would be allocating memory based on a 0 sized IP packet. If this allocation goes unchecked a null pointer could attempt to write to/or read from, or a valid memory location could be returned (behavior for requesting 0 bytes of memory is undefined by most implementations) and heap control segments could be written thereby causing a crash.

The TCP window exists as a form of congestion control on TCP networks. Each end of a connection advertises the maximum size buffer of outstanding data that the sender should allow to exist without being acknowledged by the receiver. This sometimes gives a glimpse into the memory setup of the receiving system, telling at attacker how much memory has been allocated for that connection. With this knowledge an attacker could send excessive amounts of data to potentially overflow this buffer.

The field devices tested for this paper were not vulnerable to this attack.

## 4.2 IP Fragmentation

These days IP fragmentation is often used by attackers to evade detection and prevention security controls, but it can and has also been leveraged to cause memory errors. The fragmentation process exists so that the IP data can travel over any number of data link layers that have varying limits on maximum frame size.

Overlapping fragments are handled differently on different systems and may allow an attacker to overwrite data that has already been determined acceptable with control characters, size information or other methods that turn the previous packet into a dangerous one.

There have been various denial of service conditions caused by IP fragmentation. As recently as last year Microsoft had a Windows' vulnerability that was caused by sending a large number of incomplete fragments and lead to the receiving stack keeping large

buffers of memory in use and not being able to support future connection requests. Attackers would send an initial IP packet with a larger size, or several fragmented packed from the middle of a large packet, and then keeping the connection in a "wait" state.

Fragmentation can also be combined with the IP header size to send fragments that add up to memory larger than the specified size in the IP header, leading to potential buffer overflows, and memory corruption. First made public in late 1996, and commonly known as the "ping of death", the vulnerability exists because of incorrect assumptions made about packet size. The IP specification states that an IP packet can be no larger than 65,535 bytes. By sending a large initial IP packet and specially crafting fragmented packets, the final packet is written to memory past the end of the memory reserved for the packet, thereby overwriting control information and causing memory corruption. At the time this attack first became public, nearly every network stack in existence was vulnerable and quite a few legacy systems still are.

For the most part the devices tested held up well to this attack, as they should for a nearly 15 year old attack that was patched by most vendors within hours and days of information becoming available. The field devices tested were designed and built in the last five years, so there are likely many older deployed OS stacks vulnerable to the "ping of death".

## 4.3 SYN Flooding

SYN flooding is an attack, intentional and otherwise, that has existed since the early days of TCP. \There are various ways to mitigate the problems caused by it, but it is still a problem that is difficult to solve. The problem occurs when a device is flooded with connection requests for one or multiple other devices. The receiving device allocates memory for each of these connections that will never fully occur, potentially leading to a denial of service condition. This is an example of an asymmetric attack since the sender has to use very little computational power/memory in comparison to the receiver. Since many field devices with embedded OS have significantly lower computing power than a PC, this type of attack is more likely to succeed.

Figures 1 to 4 show the results from SYN flood testing field devices. The X-axis represents the number of SYN packets and the Y-axis represent response time in seconds.

As you can see from the graphs some of the systems respond favorably to SYN flooding, however others begin dropping packets and increasing connection times quickly. Worse yet, in a few cases all services on these devices ceased to respond and had to be hard rebooted to return the device to a usable state. This crosses over into being not only a security issue, but also a reliability issue, as this sort of problem could be caused by a malfunctioning router, or other misconfigured devices, blinding operators of data and making services unusable.

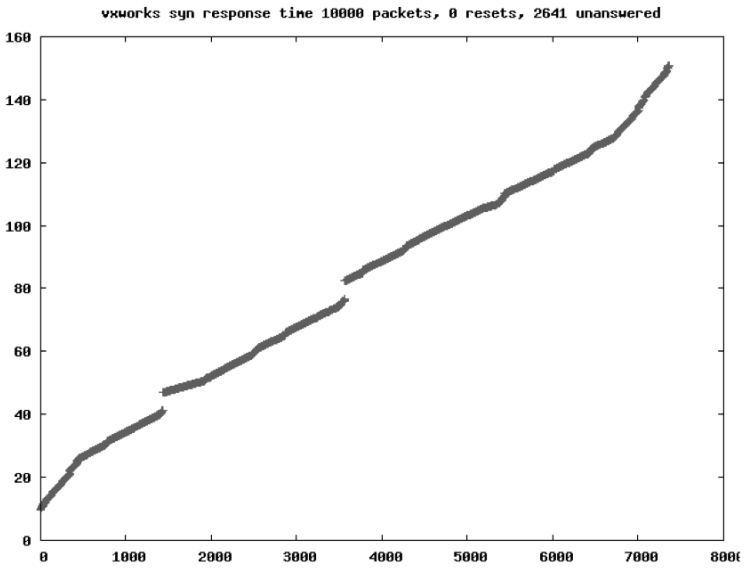


*Figure 1 – SYN Flood Testing on VxWorks Field Device*

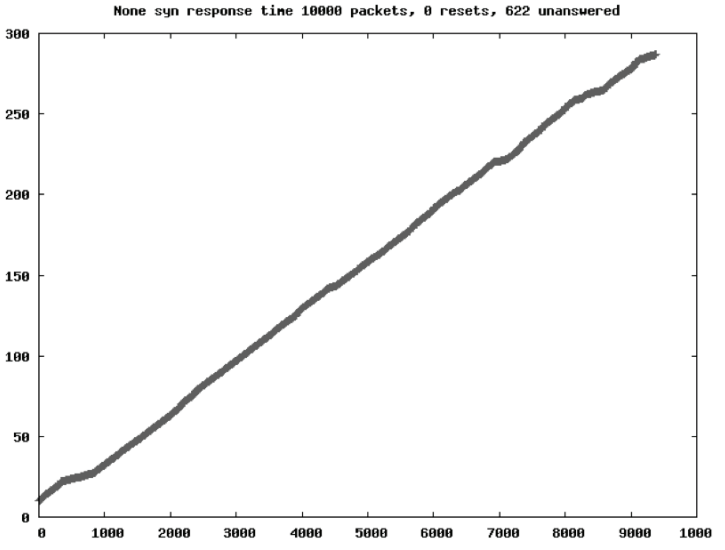


*Figure 2 – SYN Flood Testing of Embedded Linux Field Device*

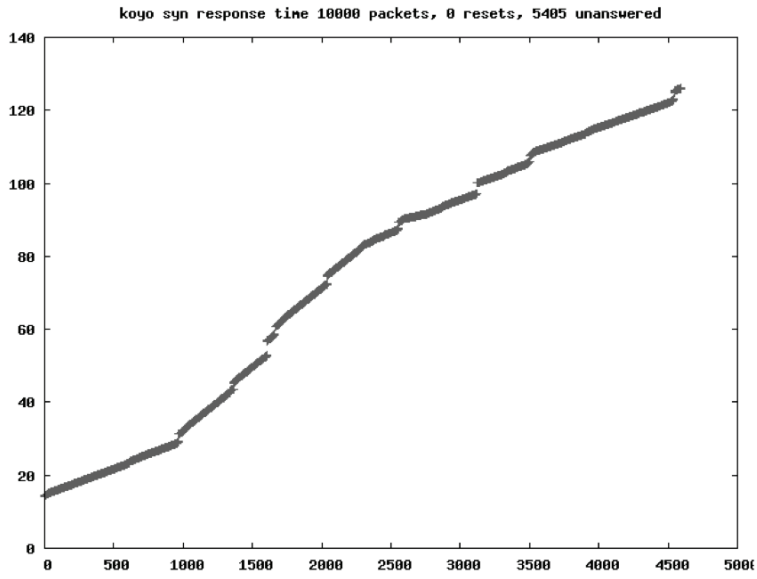


*Figure 3 – SYN Flood Testing of Koyo (NIOS OS, a derivative of uclinux)*

There are several mitigating technologies for SYN flood attacks. One of the most popular of which is SYN cookies, in which the receiver doesn't allocate memory until the sender has SYN/ACK'd and the connection is established.

## 4.4 TCP Sequence Number Predictability

Another very old attack that is still prevalent in embedded systems is TCP Sequence Number predictability. This was another attack that was common in the mid 1990's and was used in several high profiles and very publicized attacks. Due to shortcuts in providing randomization to the initial sequence number used in creating the TCP connection, most commonly starting from a set base number and incrementing by a fixed amount each second and with each connection, this connection can be spoofed. This is especially critical in control systems as certain services are only open to trusted hosts, and the TCP sequence number predictability could allow an attacker to inject data into the TCP stream. An attacker could leverage this vulnerability to run untrusted commands, corrupt data, and possibly exploiting vulnerabilities that would otherwise only be open to trusted hosts.

The testing found numerous embedded systems to still be vulnerable to these types of attacks. Of the devices profiled in this testing, 50% were found to have ISN that were trivial to guess and could easily be attacked, and 25% were found to have ISN numbers that could be guessed without too much trouble. All of these devices also run services in clear text, without any real validation, allowing data to easily be injected from a spoofed host. Embedded systems tended to be trivial to exploit relative to current general purpose operating systems.

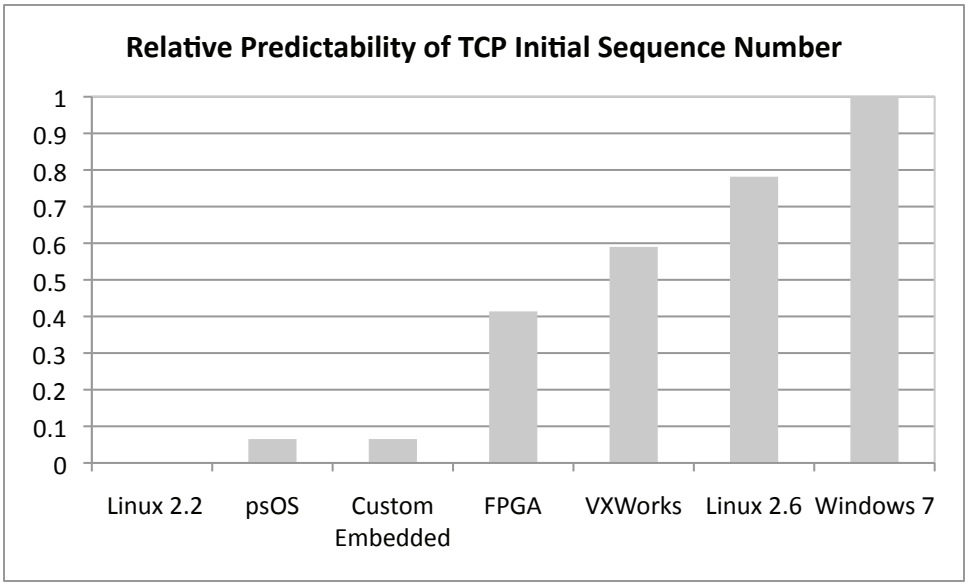


*Figure 4 – TCP Initial Sequence Number Predictability*

# 5 Conclusions

Not surprisingly many security features are not implemented in embedded systems. Some of this can be understood due to resource issues, however these devices fall far behind the security curve and are closer to security obsolecense. This may be understandable for devices that were deployed before the vulnerabilities were discovered. However it is less acceptable for new embedded systems that are deployed with software slightly changed from previous versions and deployed onto new hardware. This allows vulnerabilities to languish in the same codebase over multiple revisions.

A security development lifecycle and security testing is still lacking by embedded device vendors. With the life cycle of these devices, older vulnerabilities may be out of the mainstream thought of developers vendors and operators so these test suites have to be continuously updated. To that end we've developed a small suite of tools as an example of how to test these vulnerabilities and can perform tests for each of the vulnerabilities discussed in this paper and will be available on digitalbond.com.

---

**About the Author** – Daniel Peck is a Senior Security Researcher and leader of the Offensive Security Team at Digital Bond. He is a key member of the U.S. Government funded Portaledge and Quickdraw research projects. Prior to joining Digital Bond, Mr. Peck was a security researcher at SecureWorks and a frequent presenter at leading security events such as BlackHat, Defcon and the Pentagon Security Forum.

## References

[1] Daniel Peck and Dale Peterson, Leveraging Ethernet Card Vulnerabilities in Field Devices, 2009 Proceedings of the SCADA Security Scientific Symposium, Digital Bond Press