Attacking and Defending Networked Embedded Devices

Kwang-Hyun Baek
Dartmouth College
Hanover, NH 03755
jbaek@cs.dartmouth.edu

Sergey Bratus
Dartmouth College
Hanover, NH 03755
sergey@cs.dartmouth.edu

Sara Sinclair
Dartmouth College
Hanover, NH 03755
sinclair@cs.dartmouth.edu

Sean W. Smith
Dartmouth College
Hanover, NH 03755
sws@cs.dartmouth.edu

ABSTRACT

Currently, work on malware attack and defense focuses primarily on PCs. However, as lightweight computing devices with embedded operating systems become more ubiquitous, they present a new and very disturbing target for botnet developers; and as embedded devices become more integrated and networked with general-purpose computing, they can easily become the launching point for many attacks on the enterprise network in which the embedded devices are deployed. In this paper, we showcase a variety of practical attacks we were able to launch on an enterprise network using already widely deployed set-top-boxes. We identify challenges associated with securely deploying and managing these embedded multimedia boxes. We also present a solution to these attacks by hardening these embedded devices, using the Enforcer LSM and SELinux.

Categories and Subject Descriptors

H.4 [**Embedded Systems Security**]: Trusted Computing

1. INTRODUCTION

As networked embedded systems become more ubiquitous and integrated into an enterprise network, they present a very enticing launching point for remote attacks on the network. Indeed, we see small networked computers with lightweight or embedded operating systems in all sorts of scenarios: in grocery stores, factories, hospitals, power substations and other SCADA systems, kiosks, and in unexpected areas of the home and office. Because they are less powerful and limited in their computational power and functionality, the need to protect them has not yet become part of the mass consciousness. Moreover, the networked embedded systems run a single application; they are meant to be deployed "as-is" without many changes in their configuration. Thus, updates and patches for these embedded devices are much less frequent than PCs, and they often suffer from variety of *existing* security vulnerabilities.

At the same time, we see a trend in embedded devices to use commodity software, including commodity operating system like embedded Linux. The commodity OS that runs on these embedded devices is often customized in their configuration, and unwanted modules are removed from it, so that the kernel configuration and the built-in modules match the purpose of the device. Moreover, the user-level software stack is often greatly reduced to both accommodate the limited hardware environment and remove undesirable or unnecessary software that can be exploited by attackers.

Nevertheless, because of the complexity of commodity OS and inter-dependencies of different functionality, removing certain functionality may not be feasible. Moreover, it is hard to be assured that all the unwanted functionality is removed from the platform even with costly whole-system analysis. Finally, removing kernel modules and user-level binaries in this way may not be able to stop the attackers from gaining all the functionality they need by exploiting functionality that is already available in the system in an unexpected way.

In this paper, we present a case study, how we have remotely transformed off-the-shelf media systems (widely deployed at our university and elsewhere) into general-purpose devices under our control, and the implications of the botnet we could create with these devices. In doing so, we also demonstrate that securing embedded systems by ad-hoc removal of unnecessary functionality is not enough, and that the principle of least privilege—allowing only the necessary *privileges* to perform the task at hand—is a better way to protect the embedded systems. We prototype our solution using *Trusted Platform Module (TPM)*, *the Enforcer Linux Security Module (LSM)*, and *Security Enhanced Linux (SELinux)* to protect the integrity of the software stack and guarantee the principle of least privileges.

2. RELATED WORK

2.1 Attacking Embedded Systems

A number of recent Blackhat and Defcon talks have been devoted to devices surreptitiously placed in enterprise environments by penetration testers or attackers. The associated techniques make use of the fact that a familiar entertainment or appliance device does not attract undue attention (at least not until after the damage has been done). Devices featured include a modified version of the DreamCast

game console [4] and a sniffer disguised as a UPS device [14]. Similar work subverted innocuous devices, such as printers, that are already a visible part of the enterprise computing environment [3, 2], and showed some of the ways in which a device with limited capabilities could be targeted as an entry-point for an attacker or a passive observer in an organization. (In contrast, in our attack we discuss the ease with which a large number of similar media devices could be subverted without resorting to sophisticated kernel exploitation techniques and used in aggregate to create a much larger, scarier threat than a one-off version).

Su et al. discussed subverting modern mobile phones using worms that spread through Bluetooth [15]. However, mobile phones are different from the media systems we are considering in several ways. Their threat model often targets the mobile phones themselves, such as the personal information stored in the phone, whereas we are trying to use our media system as a launching point to access data on other devices. Moreover, mobile phones do not use IP for routing (with few exceptions that use 802.11); thus, it is harder to use them to attack the infrastructure that uses IP routing, which is most prevalent today. Finally, mobile phones are mobile and dynamic. Thus, it may be challenging for the attacker to administer them since it is hard to predict their location and targets.

SCADA (Supervisory Control and Data Acquisition) systems have been another area of intense security research [1] in recent years. SCADA systems are used in a variety of critical infrastructures around the country, including in most utilities, such as power and water. However, much of the concern with SCADA security is not focused on the actual embedded devices deployed as part of the system, but rather on the protocols by which those devices relay their data to central servers, or the servers from which the system is controlled. Instead of examining security concerns in systems that include embedded devices, we consider in this paper the security problems that the embedded devices themselves present to the system.

2.2 Securing Embedded Systems

Trusted Computing

There is an ongoing effort to put trusted hardware like TPM on an embedded platforms, such as PDAs and mobile phones by Trusted Computing Group, an industry consortium [17]. TPM is a certified chip that performs an authenticated boot, measuring hardware and software components of the platform it is mounted on [18]. It can perform the attestation of its configuration to a remote party, so that the remote party can verify the attester's hardware and software configuration. Moreover, it can store cryptographic keys, and the access to these keys can be restricted by the TPM depending on the measurements from the authenticated boot. Thus, if the measured configuration of the hardware and software component is different from the one that the key is *wrapped* to, the TPM will deny the access to the key.

Seshadri et al. proposed using software-based attestation for sensor network device (SWATT) without any special hardware [13]. SWATT uses randomized access pattern to compute the checksum of the running code in such a way that if the attacker were to produce the correct checksum, the checksum calculation takes observably longer. The embedded systems we are considering—set-top boxes and kiosks—are much more powerful than the sensor node, and we suspect that these systems will soon be equipped with trusted hardware, so we simply used a TPM-enabled platform.

Mandatory Access Control

NSA developed Security-Enhanced Linux (SELinux) [9, 19], which enforces Type-Enforcement Mandatory Access Control (MAC) through system call hooks. Jaeger et al. analyzed that SELinux can provide the full mediation of all Linux system calls [8, 7]. SELinux MAC policy labels each process, user, and files into domains, defines a policy for each domain, including domain transitions.

Tomoyo [16] Linux implements simple pathname-based Type Enforcement mandatory access control for embedded Linux. It uses domain transition history to determine the correct domain of the subject. Unlike SELinux, its policy uses simpler file system attributes (read, write, execute for a specific path) rather than mimicking all the system call hooks. It has smaller memory footprint and suffers less performance degradation than SELinux, and does not require static inode attributes. However, Tomoyo Linux cannot enforce as fine-grained and comprehensive the policy as SELinux. Furthermore, even though coming up with a working SELinux policy that accommodates multi-application commodity PCs and servers, profiling single-application embedded system with limited functionality is much easier.

3. BOTNET THROUGH SET-TOP BOXES

In this section, we discuss our experience of attacking networked embedded systems that are deployed in our network.

3.1 Description of the Set-Top Boxes

During the examination of our local network, we noticed hundreds of networked set-top boxes deployed in dormitories and administrative buildings, across different subnets. Each of these boxes is equipped with comparatively slower process (compared to that of commodity PCs) and 100 megabit Ethernet interface, and an SDRAM and a flash memory chip, where the OS image is stored. These boxes are meant to be deployed on the network, "as is", and require minimal administration.

Upon analysis, we discovered that each of the set-top boxes contains following:

- a custom Linux 2.4 kernel,
- a custom BusyBox shell[1],
- a minimal web server for web-based configuration,
- a telnet server for remote management,
- a set of update scripts,
- and a copy of `wget`, which is used by the update scripts to download the upgrades.

[1]BusyBox is a UNIX shell replacement designed for small embedded systems that combines tiny versions of many common UNIX utilities into a single small executable.

The boot loader process unpacks the root filesystem image stored in the flash memory chip and mounts it on a ramdisk. The free space on the ramdisk is limited to around a hundred kilobytes. Even though there is around tens of megabytes of free space available, we found that writing to the flash drive and not restoring the flash drive back to the original state prevented the device from booting again. This integrity protection uses a database containing the MD5 hash of the entire flash drive and some of the stored files. From observing the hexadecimal dumps of the code, we found that the MD5 hash database is signed using a 1024-bit RSA key. However, the public key to verify the signature of the file is also contained in the image itself. Thus, an attacker can insert its own public key and recalculate the hash of his modified code and the matching signature, to bypass the integrity protection mechanism.

3.2 Root Access

Surprisingly, we were able to gain root access to the set-top box via telnet, using the default password we found from searching the Internet[2]. Because the box mounts and unpacks the root filesystem from the read-only image, any changes to the root password are reverted back to the default password upon each reboot, as are any changes to disable telnet server in the init scripts. The vendor may have wanted a way to revert to the default password if the administrator forgets the password he set for the box. The reason for running a telnet server can be that, lacking keyboards and displays, these boxes can only configured via a network connection—and the goal to minimize end-enterprise administration pushes this task to the remote vendor.

3.3 Regaining Removed Functionality

Even though we gained root access to the machine, we found that most basic binaries found in commodity Linux systems, such as `chmod` and `insmod`, were not available in the set-top boxes. Moreover, when we used `wget` to fetch statically cross-compiled tools like `tcpdump`, `wget` stripped the execution bits to the downloaded files. Thus, eliminating `chmod` from their system was apparently an intentional effort to stop attackers from executing arbitrary binaries.

However, although the box lacked `chmod` binary, we found three ways to turn on the execution bits of the downloaded files. First, we used `tar`, which was available in the box, since tar preserves the permission bits of the files in the tarball. Second, we were able to add the execution bit by simply overwriting an existing executable with `cat`. Finally, we discovered that the box's kernel also included support for *Network File System (NFS)*, so we simply exported the directory containing cross-compiled tools to the media boxes from a remote server. Making the tools available via NFS enabled us to quickly add remove new tools to all media boxes under our control, and it also gave us a lot more storage than what the boxes are capable of.

In order to demonstrate the potential of a botnet consisting of these types of networked embedded devices, we transformed these set-top boxes into platforms for sniffing, intercepting, relaying and injecting traffic into our university's switched network. We found that the Linux 2.4 kernel included yet another unexpected functionality—the support for Berkeley PF packet capture and filtering architecture [6], as well as support for raw sockets. This unexpected functionality enabled us to use the `libpcap` [5] library for packet sniffing and the `libnet` [12] library for shaping and injecting arbitrary packets. Using these two popular libraries we were able to launch man-in-the-middle attacks to achieve (1) sniffing of the traffic on our university's switched subnets, (2) logging of sniffed data in our NFS share, and finally (3) packet filtering, modification and injection, These tools allowed us a significant degree of control over the local subnets where each box was placed.

[2]In another application domain, the power grid, we have seen one vendor actually brag about how their products are more secure because their default passwords are slightly less obvious—and include a nice summary table of brands, models, and defaults passwords, to prove the point.

Attacks	Throughput (KB/sec)
none	5252.21
(1) simple MitM	4221.04
(2) NFS logging	2744.59
(3) filter, mangle	949.6

Table 1: Throughput measurements between the target victim and a remote server in Kilobytes per second. The first measurement shows the effect on bandwidth when we launch simple man-in-the-middle sniffing attack that logs the sniffed traffic in the local filesystem. The second measurement shows the same attack, except that we log the sniffed result in the NFS mount. The third attack shows the effect on bandwidth when we actively modify the traffic at the set-top-box.

Moreover, we found that the box's performance is good enough for most of our attacks even when we had the set-top boxes write to the NFS mount. We used the `ttcp` tool for estimating the maximum bandwidth between a target and a remote server beyond the local gateway while the set-top box is performing the attacks. Table 1 shows the throughput measurements. First, we measured the throughput between the and the remote server without running any network tools as a comparison. Note that, using NFS, even when we "enhance" the media boxes with our binaries, we do not modify SDRAM, kill any process, or modify the kernel. Indeed, we were able to use the set-top box for its normal functionality, while we were connected to it remotely. As a consequence we do not expect users to notice when we are running our binaries.

3.4 Automating Botnet Creation

We have demonstrated that our media box's kernel includes support for many network stack features central to attacking its home network. In particular, it supported raw Ethernet frame injection, capture and forwarding. Moreover, it was easy to download and run the cross-compiled versions of tools— `fragrouter`, `fragroute`, `arp-sk`, and `dnsspoof`, the multipurpose relaying tool `socat` and the like. Given unlimited access to the fundamental Linux system calls that all of these tools rely on, we chose among the existing network attack tools at will. Moreover, it was easier to make these tools available in the set-top-box due to the combination of

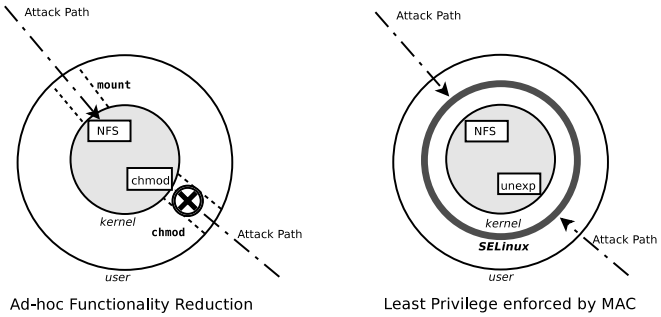

Figure 1: Ad-hoc functionality reduction versus least privilege enforced by SELinux. *Removing* all the unwanted functionality is difficult because of the complexity and interdependencies of the functionality within a commodity OS. For embedded systems, it is easier to leave all the functionality but *permit* only the necessary privileges, using a strong MAC policy.

programs that are already present on the box, such as `wget` and `tar`, and `telnet`, which are intended to be used by its firmware update scripts.

Making a widestread botnet of these set-top-boxes is little more than a shell script and a few statically compiled C programs to compensate for the shell commands absent from the platform's BusyBox shell. Thus, propagating the code to another media box is a simple programming exercise. Although we stopped short of creating a worm that would propagate our campus network through these set-top-boxes, we found no technical obstacles preventing this scenario.

Our subversion of the set-top boxes via such "unexpected" functionality demonstrates that an ad-hoc approach to reducing functionality is not enough to guarantee the security of these embedded boxes and are too error-prone to be used in real systems. Even if we deal with our findings, attackers will be able to find yet another clever way to reuse existing functionality to achieve what they want. Thus, more robust security mechanism needs to be in place that can guarantee information flow and integrity of the software even when the extra functionality is not entirely eliminated from the system.

4. SECURING EMBEDDED SYSTEMS

In this section, we present our solution to enhance security of the networked embedded platforms.

4.1 Information Flow and Least Privilege

We argue that using the principle of least privilege is a more systematic approach to securing the system. Removing functionality may leave undesirable attack paths that the engineers did not expect during their development phase. Because the functionality that networked embedded systems achieves is simple, single-application oriented, it is not difficult to profile what the expected behavior of the system should be. As we mentioned earlier, SELinux provides full mediation of Linux system calls and can provide very fine-grained policy. We simply allow only the necessary permissions to achieve the functionality that the networked embedded systems. Furthermore, after we define the domains of all the necessary components, we group all the other components as undefined, and restrict permissions from the undefined domain to any of the marked domain. Thus, there may be available functionality in the system, but our restrictive SELinux policy will deny access to it (see Figure 1).

To prove our point, we profiled an expected behavior of an E-voting machine, where the platform goes through a series of domain transitions from kernel to init to an init script which loads the E-Voting binary and domain transition to the E-voting domain. We profiled the policy so that E-voting domain can only be entered via this expected sequence. Moreover, we allow only the set permissions necessary to run the E-voting software, which resulted in a platform even though login console is available (functionality), no one can login to run a shell (privilege).

4.2 Integrity Protection

In order to fully guarantee the information flow, we need to protect the integrity of the SELinux policy and the kernel image, as well as important configuration files for the system.

As mentioned earlier, the set-top box's integrity protection mechanism can be subverted since there is no protection for the public key within the compressed image[3]. Moreover, once the system boots, no check is made on the binaries the platform loads, leading to time-of-check, time-of-use (TOCTOU) attack. Thus, until the system reboots the attacker may carry out his attack by modifying the system files. This TOCTOU attack may be more detrimental to the embedded systems than commodity PCs since embedded systems are expected to go through power cycle much less frequently than the PCs.

We use the Enforcer LSM [11, 10] with TPM to make the integrity protection more robust. Enforcer LSM is similar to the set-top box's integrity protection mechanism, except that it does load-time check of the important files *while* the system is running rather than just during the boot-time. Moreover, it relies on the TPM's authenticated boot to measure the public key that is used to verify the database of the hashes. It relies on the TPM to deny access to a protected key if a wrong key is used to verify the database. Furthermore, this protected key in the TPM is used for the authenticating to the network, so that only the boxes that are running certified kernel image and enforcing the correct database of hashes can use the key to authenticate to the network.

5. IMPLEMENTATION

We upgraded Enforcer code to run on Linux 2.6.20. We customized a static kernel image including Enforcer LSM, SELinux, and the support for hardware that the platform needs to boot and perform E-voting, including its graphics card support and USB input devices. We achieved 4.8 MB of uncompressed kernel image and 2.0 MB of compressed kernel image. Because we were not able to find the latest

[3]As we were testing the boxes, we made several systems unbootable. In order not to make too many systems unbootable, so we decided not to carry out the actual attack

version of TPM on a simpler hardware that is designed for networked embedded device, we used a commodity PC to run the prototype E-voting software; thus, we had to include the support for SCSI and various hardware support necessary to boot the commodity system. We argue that if the kernel is customized for a dedicated single board computer (SBC), the kernel image will shrink even more, reducing the footprint of the kernel image on the disk. For example, we removed part of the kernel we believe is unnecessary for SBCs, such as SCSI, IDE controllers, and the kernel's size shrank few hundred kilobytes more, resulting in a compressed image of approximately 1.8 MB. The memory footprint of the entire system, 27MB, is bigger than the set-top-box's—15MB—which uses an optimized 2.4 Linux kernel and binaries. Nevertheless, our system, which has not been optimized for embedded systems, can still fits in the 35 MB SDRAM in the set-top box, making it possible to upgrade the existing kernel to our more secure one.

6. CONCLUSION AND FUTURE WORK

We presented a case study of practical attacks on an enterprise network using networked embedded devices. We believe that the next frontier for botnet developers may be lightweight embedded devices like the set-top boxes we explored. They are ubiquitous but largely invisible in our day-to-day lives. We view our subversion of media boxes as one case among many possible instantiations. In particular, we note that embedded device developers are increasingly turning to generalized commodity software, particularly operating systems, for a low-cost way to get their products to market; customization always results in a higher price. In the case we examined, the developers had not disabled unnecessary functionality in an operating system (versions of which are also used on desktop PCs). We presented our solution using Enforcer LSM and SELinux. Our solution provides integrity protection and the principle of least privilege to the development of the set-top boxes, in the hope of raising the bar of embedded system security.

7. ACKNOWLEDGEMENT

This work was supported in part by Sun Microsystems, Intel Corporation, the NSF (under grant CNS-0524695), by the Bureau of Justice Assistance (under grant 2005-DD-BX-1091). The views and conclusions do not necessarily represent those of the sponsors.

8. REFERENCES

[1] J. D. Fernandez and A. E. Fernandez. SCADA Systems: Vulnernabilities and Remediation. *Journal of Computing Sciences in Colleges*, 20:160–168, 2005.

[2] FX of Phenoelit. Attacking Networked Embedded Systems. In *Black Hat Europe, Amsterdam*, August 2003.

[3] J. Hernandez, J. Sierra, A. Gonzalez-Tablas, and A. Orfila. Printers Are Dangerous. In *IEEE 35th International Carnahan Conference on Security Technology*, pages 190–196, October 2001.

[4] A. Higbee and C. Davis. DC Phone Home. In *Black Hat USA Security Briefings*, August 2002.

[5] V. Jacobson, C. Leres, and S. McCanne. *Libpcap*. Lawrence Berkeley Laboratory, University of California, August 1996.

[6] V. Jacobson and S. McCanne. The BSD Packet Filter: A New Architecture for User–Level Packet Capture. In *Proceedings of the Winter 1993 USENIX Conference*, January 1993.

[7] T. Jaeger, A. Edwards, and X. Zhang. Consistency Analysis of Authorization Hook Placement in the Linux Security Modules Framework. *ACM Trans. Inf. Syst. Secur.*, 7(2):175–205, 2004.

[8] T. Jaeger, R. Sailer, and X. Zhang. Analyzing Integrity Protection in the SELinux Example Policy. In *12th USENIX Security Symposium*, August 2003.

[9] P. Loscocco and S. D. Smalley. Meeting Critical Security Objectives with Security-Enhanced Linux. In *Proceedings of the 2001 Ottawa Linux Symposium*, July 2001.

[10] J. Marchesini, S. Smith, O. Wild, J. Stabiner, and A. Barsamian. Open-Source Applications of TCPA Hardware. In *20th Annual Computer Security Applications Conference*, Dec. 2004.

[11] J. Marchesini, S. W. Smith, O. Wild, and R. Macdonald. Experimenting with TCPA/TCG Hardware, Or: How I Learned to Stop Worrying and Love The Bear. Technical Report TR2003-476, Department of Computer Science, Dartmouth College, Dec. 2003.

[12] M. D. Schiffman. *The Libnet Packet Construction Library*.

[13] A. Seshadri, A. Perrig, L. van Doorn, and P. Khosla. SWAtt: SoftWare-based Attestation for Embedded Devices. In *Proceedings of IEEE Symposium on Security and Privacy*, May 2004.

[14] SpydeĪ, AutoNiN, and Mystic. The UPS (Undetectable Packet Sniffer). In *Defcon 11*, August 2003.

[15] J. Su, K. K. W. Chan, A. G. Miklas, K. Po, A. Akhavan, S. Saroiu, E. de Lara, and A. Goel. A Preliminary Investigation of Worm Infections in a Bluetooth Environment. In *Proceedings of the Workshop on Rapid Malcode (WORM)*, Fairfax, VA, Nov. 2006.

[16] Tomoyo Linux Project.

[17] TCG Mobile Trusted Module Specification, June 2007.

[18] Trusted Platform Module main specification version 1.2 revision 94, Mar. 2006.

[19] C. Wright, C. Cowan, S. Smalley, J. Morris, and G. Kroah-Hartman. Linux Security Modules: General Security Support for the Linux Kernel. In *Proceedings of the 11th USENIX Security Symposium*, pages 17–31, Berkeley, CA, USA, 2002. USENIX Association.