Dumbots: Unexpected Botnets through Networked Embedded Devices

Kwang-Hyun Baek, Sergey Bratus, Sara Sinclair, Sean W. Smith

Dartmouth College Computer Science
Technical Report TR2007-591

May 26, 2007

Abstract

Currently, work on botnets 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. In this paper, we present both an empirical demonstration on a widely deployed multimedia box, as well as an evaluation of the deeper potential of these *dumbots*.

1 Introduction

When it comes to subverting networked machines for distributed use (such as creating botnets), the primary arena tends to be commodity personal computers. We see this tendency from both the attack and defense perspectives: a malicious attacker building a botnet targets PCs, and system administrators focus their defense on these same machines (both because they are a primary target and because they are vital to the day-to-day operations of the organization).

What qualities of commodity PCs make them so attractive to botnet developers? We posit the following characteristics are essential for botnet devices.

- **Ubiquity:** There are an estimated 900 million PCs currently in use today, with over 230 million (about 25%) in the United States [3]. This means that an attacker has a huge number of target nodes spread throughout the world for creating a massive distributed network of bots.
- **Insecurity:** Most of these machines, especially those located in home environments, are administered by average users without a specialized knowledge of security principles or best practices. As a result, many of these machines are vulnerable to known attacks, even in enterprise environments.
- **Availability:** Most of this machines are on for long periods of time, and remain online via a broadband internet connection. Furthermore, these machines are usually not burdened with intense computation or network traffic by their owners, so they are both accessible and available for use by an attacker without the owner noticing.

The community also focuses defense efforts on PCs because they are the visible, tangible manifestation of computing today. Users consciously touch them, run programs on them, and (potentially) install patches. Thanks to continuing attacks and mass media coverage, the risk and urgency of PC security is probably part of everyone's consciousness now.

This focus on commodity PCs diverts attention from another fruitful arena for attackers: networked embedded systems. We compare them to PCs according to the three characteristics we presented earlier:

- **Ubiquity:** We see small networked computers with lightweight or embedded operating systems in all sorts of scenarios: in grocery stores, factories, automobiles, hospitals, power substations, and in unexpected areas of the home and office.

- **Insecurity:** We know that these devices suffer from a variety of existing security vulnerabilities (see Section 2).
- **Availability:** Finally, most of these networked embedded devices, once deployed, are meant to be always connected and powered on.

We argue that devices that integrate invisibly into our lives are more susceptible to attack because the need to protect them has not yet become part of the mass consciousness—when it comes to computing, these devices are invisible.

If embedded or lightweight networked systems are ubiquitous, vulnerable, and available, what prevents them from becoming a target for attackers developing botnets? As more and more attention is drawn to the security of PCs, will future attackers turn to these lightweight devices to satisfy their needs?

In this paper, we consider the possibility of botnets composed of embedded systems. We describe a practical instantiation: 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 botnets we could create with these devices. By subverting an unexpected, 'invisible' networked system, we expose a new arena for botnet development—and defense.

2 Related Work

As we noted, much of the research on botnets has been subverting desktop PCs for use in DDoS attacks (e.g., [9]). More recently, researchers have noted the ability of bots to spy on the PCs' owners [12]; when aggregated, the personal data harvested from an entire botnet could prove very valuable to the attacker. (Indeed, there exist online markets where criminals can rent or buy the use of botnets because the aggregated information is so valuable [13, 1, 21].)

Researchers have identified vulnerabilities in a variety of embedded systems, particularly in enterprise environments [17]. The deployment of RFID technology in supply chains presents serious concerns for security and privacy [8]. The sensitive nature of networked patient care equipment and power supply devices in the electric infrastructure have also brought attention to their security; Koopman notes that these devices are extremely cost-sensitive, and cutting corners on security can lead to big market advantages [16].

A number of recent Blackhat and Defcon talks have been devoted to devices surreptitiosly 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 [11] and a sniffer disguised as a UPS device [19]. Similar work has been done to subvert innocuous devices, such as printers, that are already a visible part of the enterprise computing environment, [10, 7]. That work showed some of the ways in which a device with limited capabilities could be targeted as an entrypoint for an attacker or a passive observer in an organization. (In contrast, in our paper we discuss the ease with which a large number of those devices could be subverted 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 [20]. 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 botmasters 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 [6] 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.

3 Our Work: Dumbots

In this section, we discuss the work that we have done as part of a collaborative security initiative to evaluate a particular lightweight device prevalent on our local network. Here and in the rest of the paper, we consider "embedded devices" to be special-purpose computers that have limited computation, storage, and/or input/output capabilities. Through our exploitation of this one type of device in an easy-to-reproduce, scalable way, and our subsequent exploration of its capabilities as a member of a botnet (particularly, its capabilities within its context of an enterprise environment), we will show that embedded devices could easily become the hosts of the next generation of botnets.

3.1 The Environment

Exploring our local infrastructure, we noticed a multimedia box quietly lurking in the corners. (Out of courtesy to collaborators who let us have access to a few sample boxes, we will not reveal here the specific purpose of these boxes, nor the specific vendor.) Our university has deployed approximately 450 of these boxes, across different subnets in the dorms and administrative buildings (we discuss the details of their network configuration in Section 4). The boxes are equipped with comparatively slower processor (compared to that of commodity PCs) and 100 MBit Ethernet interface, and an NVRAM flash memory chip where the OS image is stored. These systems are meant to be deployed on the network, "as is", and require minimal administration.

Upon analysis, we discovered found that each of the multimedia boxes contains the following:

- a custom Linux 2.4 kernel,
- a custom BusyBox[1] shell,
- 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.

The boot loader process unpacks the root filesystem from the image stored in NVRAM 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 in the flash drive, we have found that writing to the flash drive and not restoring the flash drive back to the original state prevented the device from booting again, an apparent integrity protection measure to prevent tampering with the stored OS image.

3.2 Subversion

We now explain in detail how we gained the control of these boxes and how we made the necessary network tools available in the multimedia boxes to turn them into powerful bots.

[1]BusyBox is a UNIX shell replacement designed for small and embedded systems that combines tiny versions of many common UNIX utilities into a single small executable. More information about BusyBox can be obtained at `http://www.busybox.net/`.

Telnet We saw the first opportunity of attacking the box when we discovered that the boxes are running a telnet server for remote management. We found the box's default password on the Internet[2] and was able to gain root access to the box. Because the boxes mount a read-only image from its flash drive on each boot, any changes to the root password are reverted back to the default password upon each reboot. The vendor may have wanted a way to revert to the default password if the administrator forgets the password he set for the box. Another reason for this 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.

Installing Network Penetration Tools We wanted to put powerful network tools such as `tcpdump` and `arp-sk` in our media boxes. As we expected, most non-essential tools and libraries did not exist on the box, except `wget` and a minimal version of `ping`. However, after finding out the processor type, we were able to cross-compile most of the tools remotely and download the tools onto the box using `wget`. Moreover, although the box lacked `chmod` to add execution bits to the downloaded files (`wget` strips the execution bit), we were able to preserve execution bits of the downloaded binaries using `tar` since `tar` preserves the permission bits [3].

Nevertheless, due to the limited storage available in the media box, even though we were able to cross-compile most of the tools we needed, we were not able to put these tools in the media box. The media box had only about 20 kilobytes of free space in the ramdisk. The contents of the flash drive are protected via signature-based integrity protection. During our analysis, we found that writing to the flash drive made the device unbootable.

Storage and NFS Fortunately, the media's kernel also included support for mounting the *Network File System (NFS)*. We simply exported the directory containing all the cross-compiled tools to the media boxes. Making the tools available this way enabled us to quickly add and remove new tools to all media boxes. The NFS support also enabled us to add virtually unlimited storage to the multimedia boxes: We made our tools to store all their logs directly in the NFS mount. In Section 4.3, we discuss how mounting NFS affects our performance. We found that the box's performance is good enough for most of our attacks even when we use NFS to store tcpdump log.

The Tools We Used Our goal was, using remote network access to deployed boxes, to transform these multimedia boxes into platforms for sniffing, intercepting, relaying, and injecting traffic. We found that the Linux 2.4 kernel included the support for the BPF packet capture and filtering architecture [15], as well as support for raw sockets.

This enabled us to use the `libpcap` [14] library for packet sniffing and the `libnet` [18] library for shaping and injecting arbitrary packets. Using these two popular libraries, we were able to make the following tools available in the multimedia boxes:

- `arp-sk` for *Address Resolution Protocol (ARP)* poisoning attack to sniff and impersonate the local gateway on our switched subnets,
- `fragrouter` for packet forwarding and custom packet filtering,
- `dsniff` for targeted password sniffing and man-in-the-middle attacks,
- `socat` and `netcat` to create and forward SSL connections in and out of the box,
- and finally, `tcpdump` to log the sniffed traffic.

[2] In another application domain, the power grid, we've 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 default passwords, to prove the point.

[3] Another way of adding executable bit is `cp -p /bin/ls /tmp/cmd; cat tool > /tmp/cmd`. Also note that `ld-linux.so`, which can be used to load arbitrary ELF-formatted file into memory, is not available in our boxes.

These tools allowed us a significant degree of control over the local subnets where each box was placed.

Normal Operation When we enhance the media boxes with our network tools, we do not modify NVRAM, nor kill any processes, nor modify the kernel. Indeed, we were able to use the media box for its normal functionality while we were connected to it remotely. As a consequence, we do not expect users to notice when our dumbots are in operation.

4 The Potential of Dumbots

Having described how we transform a deployed media box into a "dumbot," we now describe some applications for a botnet obtained from systematically transforming all of them.

4.1 Controlling the Local Network

With the network tools available on each one of our multimedia boxes, we can achieve a considerable degree of control over the LANs on which they are placed. In particular, these boxes are typically placed next to the users' desktops, connected to the ports of the same switched environment, and are actually in the same broadcast domain as these desktops. From the attacker's perspective, the policy of putting each desk on its own switch LAN complicates getting access to each user's PC. However, deploying an "invisible" media box next on each desk, next to the PC, provides one-stop shopping to get around this obstacle. (The media box also prevents us from having to worry about how to discover and exploit vulnerabilities in a wide variety of user PCs with varying configurations and defenses.)

We consider some specific examples:

1. *Sniffing.* The media box can sniff traffic on its local subnet. Since our network is switched, this requires interposing our box between the target and its local gateway. We achieve this by a two-way ARP poisoning attack combined with packet forwarding transparently to the victim, and use `arp-sk` and Linux kernel IP stack packet forwarding (also supported by the box's kernel and enabled via `/proc/sys/net/ipv4/ip_forward`).

   Since, in general, media boxes are expected to be deployed inside the firewall of an organization and close to the target machines, the media box bot will be able to thwart some standard network confidentiality measures such as *Virtual Private Networks (VPNs).*

2. *Traffic Manipulation.* The box can actively manipulate the traffic to and from the victim–e.g., by filtering, faking, or mangling it. We use a patched version of `fragrouter` together with `arp-sk` to achieve this attack.

   Some of the victim's outgoing traffic such as centrally logged alerts can be quietly dropped or modified on the fly, and outgoing packets can be fragmented to evade known detection rules. Responses to the victim's DNS queries can be spoofed (we used `dnsspoof`, a part of the `dsniff` suite for our spoofing). Higher level protocol traffic can be transparently redirected with `socat`, creating and efficient filter/proxy combination.

3. *Simple DDoS.* The media boxes can generate TCP connection attempts, such as HTTP requests, to overload a local or a remote server. Since these boxes are within the enterprise firewall, they can mount a local DDoS attacks that, if mounted from outside, a border firewall or IDS might suppress. Furthermore, in a more sophisticated scenario, it can be used to overload a local Intrusion Detection System or other network activity monitor.

Considering that these boxes will be distributed throughout the organization's subnets and likely expected to retain their capability to download updates, they will be well-poised to take advantage of subnet-wide trust relationships. For example, they can be used to "shadow" desktop machines otherwise protected by firewalls or router access control lists.

4.2 Global Tasks for the Botnet

Various publications analyze the uses of botnets in online crime (e.g., [13, 1] and many others). Our dumbots easily lend themselves to these types of applications as well. They may scan remote machines for services or vulnerabilities; relay network attacks, to hide the origin of the attack; carry out low-bandwidth application logic attacks such as sending specially crafted requests or queries; intercept or interfering with confidential traffic within the organization; extrude data (especially in the presence of bandwidth monitoring rules in the IDS, watching traffic to and from target machines but not for ubiquitously deployed commodity devices); and forward spam. They may also enable traditional botnet applicatons, such as private IRC channel control and instant messaging.

Given their prevalence in university environments, our dumbots could also be used to systematically spoof illegal music download requests from every user PC on campus, which would add some interesting wrinkles to RIAA legal action.

4.3 Performance

Having established the general feasibility of using our media boxes as nodes of a botnet, we now estimate their performance for various uses that such a botnet can be realistically put to.

We limit our discussion to a botnet of dumbots deployed throughout a single organization. Although one can expect organizationally and geographically diverse botnets of embedded devices to become feasible (perhaps even in the near future), this is not yet the case today. Thus one of the major features of a typical botnet, diverse distribution of its nodes through multi-domain IP space, is not a part of our scenario, and we will not consider botnet activities associated with typical for-profit botnets that exploit the difficulty for their targets to block a large part of the geographically diverse nodes. In particular, we will not consider outward DDoS by bandwidth consumption, since the worst that can happen is that our dumbots will saturate the organization's own link.

Instead, we concentrate on the damage that a botnet of dumbots can do primarily *within* the organization. We consider the case of dumbots distributed throughout the organizational network and sharing it with desktops, servers, printers, network equipment etc.

We consider two use cases:

- **case 1** Internal DoS attack on a target within the organization, and
- **case 2** Targeted sniffing of communications, not immediately obvious to the victim.

Despite the perceived weakness of a dumbot platform, such as limited CPU power and memory, we demonstrate that it can in fact be an efficient traffic generator for case 1, and a traffic forwarder (since sniffing on a switched network involves forwarding packets between the sniffed parties, such as a desktop or a printer and the LAN gateway, and will be immediately obvious if sniffed sessions grind to a halt or slow down to a crawl because of the sniffer's insufficient capacity to handle the traffic flowing through it) for case 2. For this, we measure the performance of our individual dumbot and compare it with that of a typical laptop.

We compare our media box with a laptop configured with Pentium M 1.86GHz processor and 1 GB of RAM, and a Gigabit ethernet network card. Despite the fact that our media boxes have a weaker CPU than the laptop, we show that the network performance of each box is good enough to be used for a powerful net.

Rate of HTTP Requests To demonstrate the DoS capability of each media box, we wrote a simple program that tries to make HTTP requests to a remote server as fast as possible, using `wget`. We ran the same program on a laptop to compare it with the media box's performance. The laptop was able to make 94.34 requests per second, whereas the media was able to make 37.88 requests per second. Given the limited computational power that is available in the media box, we found the media box to be more capable than we had first thought.

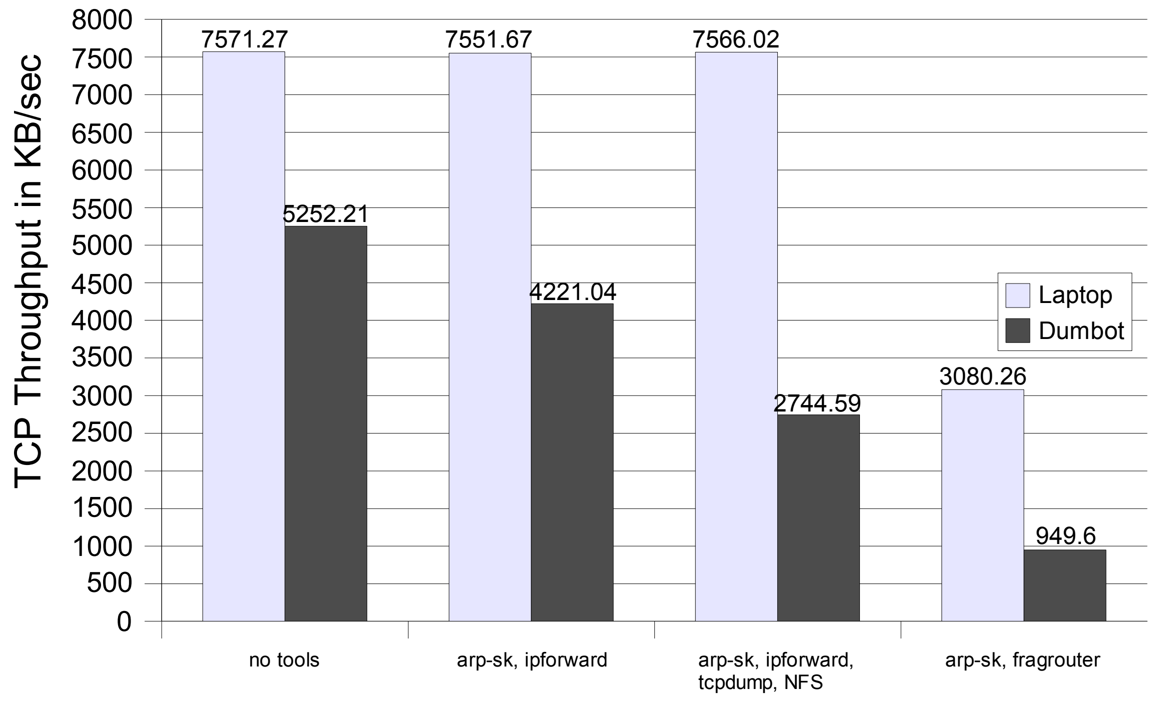

Figure 1: Throughput measurement between a remote sever and a dumbot. As a comparison, we also show the throughput measurement between a remote server and a laptop. We demonstrate how running each network tool affects the throughput between a remove server and a dumbot. The measurement "no tools" shows the maximum throughput a dumbot could obtain in this particular connection. The measurement "arp-sk, ipforward" shows the throughput when the dumbot mounts a man-in-the-middle attack using `arp-sk` and `ipfoward`. The measurement "arp-sk, ipforward, tcpdump, NFS" shows the throughput when the dumbot mounts a man-in-the-middle attack and records the captured traffic in an NFS mount. Finally, the measurement "arp-sk, fragrouter" shows the throughput when the dumbot uses `fragrouter` instead of kernel `ipforward`, to filter and mangle the traffic between the victim and remote server.

Throughput Measurement We used the `ttcp` tool for estimating the maximum bandwidth between a bot and a remote server beyond the local gateway. First, we measured the throughput between the media box and the remote server without running any network tools as a comparison. Similarly, we also measured the throughput between a laptop and a remote server as a comparison. We then ran the tools to show the performance effect they had. To demonstrate a simple man-in-the-middle attack, we used `arp-sk` with Kernel-level IP fowarding [4]. We compare this simple man-in-the-middle attack to the case when we ran `tcpdump` and logged the sniffed data in the NFS-mounted storage in order to demonstrate the media box's sniffing capability. Finally, the last entry in the graph shows when the media box is manipulating the traffic using `fragrouter`. Figure 1 shows the result in kilobytes per second.

Discussion As we expected, as we ran more sophisticated attack, the media box's throughput degraded much more drastically than the laptop's due to the limited computational resources that is available in the media box. Despite this weakness, the media boxes have several advantages over the traditional PCs. As we briefly discussed earlier, these media boxes are constantly powered on and connected to the network. Even

[4] `/proc/sys/net/ipv4/ip_forward`.

when the user presses the power button, the device is still powered on and remotely accessible. On the other hand, more PCs are now laptops; thus, they are powered down or disconnected from the network when not in use. Moreover, these media boxes receive less scrutiny from the administrators because of their limited computational resources, the static nature of the media box, and the small number of services that runs on the box. For example, an organization usually has some central security mechanism in place to protect the systems that are owned by the organization—e.g., automatic patch update, anti-virus software, distributed firewall, etc. However, most of these security mechanisms are aimed more towards commodity PCs and servers, and they are not suitable for protecting embedded devices. Finally, since all the media boxes have the exact same hardware and software stack, once an attacker successfully attacks one of these media boxes, he can easily attack all the boxes of the same kind he can find.

In the Large A back-of-the-envelope estimate of our botnet capability shows that its maximum bandwidth at its present size of approximately 450 nodes would add up to a theoretical maximum of around 18Gbits/sec (the maximum for a single node being 40Mbits/sec). Of course, they will never achieve this maximum, for many reasons – for example, being within the same organization, they will at most overload its outgoing "pipe".

When we consider overall bandwidth, an enterprise network of these dumbots does not compare to to geographically well-spread PC-based botnets of over 10,000 nodes with actual bandwidth consumption capacity over 10GBits/sec, However, recent industry white papers suggest that actual botnets attacks show a trend toward specialization and targeting specific application logic and architecture rather than raw DDoS, needing fewer nodes[5]. The position of these media boxes within enterprise infrastructure makes our dumbots valuable asset for staging such an attack scenario.

5 Countermeasures

Clearly, the threat of botnets on embedded devices is a real one. How can we work to combat the problems that result in device vulnerability? What can we do to reduce the risk we currently face?

Network Infrastructure One way to limit the risk of embedded device compromise is to control the flow of data to and from that device over the network.

For example, an enterprise could put the embedded devices in a separate Virtual LAN (VLAN) and isolate it from other parts of the internal network that carry sensitive data. However, most homes and small businesses do not have the infrastructure or expertise to support VLANs—and users expect such boxes to "just work" once plugged into the wall network socket. More simply, an organization might deploy a firewall in the devices to block all traffic between them and the outside world—but many of these devices rely on the ability to contact other parties or to "phone home" for their basic functionality, such as receiving software updates or new data.

Better Key Management We could instead examine the authentication used between an embedded device and its control server, particularly the way in which the two manage authentication keys; the door to our exploitation of our dumbots was a default root password. If all remotely-accessible services were protected by a strong authenticator, it would be much harder to subvert the devices. Generating strong, unique authenticators and distributing them throughout a large organization is a very difficult task, however; the key management and deployment literature is rich with schemes and analysis, particularly in the domain of sensor networks [5], [4]. Furthermore, our dumbots are still powerful enough to support public-key

[5] "As a result [of DDoS solution deployment], custom-tailored attacks to web-servers' application logic and back-end infrastructure are emerging. These attacks rely less on bandwidth and more on logical and architectural weaknesses of the site's themselves. Since individual bot- power is high in these cases, attackers find they can use smaller botnets, without hiding the true IP addresses of the bots.", *The Prolexic Zombie Report*, `http://www.prolexic.com/zr/jan2007.pdf`

operations; having dumbots authenticate remote upates via SSH based on a baked-in public key would be feasible and a significant improvement.

Better Awareness On a higher level, what makes our dumbots possible is the fact that commodity embedded systems, with exploitable software, are being deployed throughout enterprise infrastructure—but by being specialized devices, rather than "computers," they tend to fall below the radar of system administrators and security officers. (Indeed, one the main reasons we wrote this paper is to help change that.)

We could also consider scenarios in which devices fetch updates at a regular basis instead of having them pushed from the server, which would only require the devices to know the server's key, but this method reduces flexibility of the system and may not be useful in all scenarios.

6 Conclusion

The threat that botnets pose today are a challenge to researchers and industry specialists; combating these networks of malicious computers consumes resources in many domains. We believe that the next frontier for botnet developers may be lightweight embedded devices, which 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 posit that this is the scenario with many modern embedded systems, which means that such devices are inherently susceptible to the problems that have plagued PCs for years. While we can envision solutions that reduce the risk in the context of full-fledged PCs with specialized tasks (i.e., SELinux and its strong implementation of the principle of least privilege via types [2]), we must further explore the extent of these vulnerabilities and examine the best ways to mitigate them within the constraints under which embedded devices run.

7 Acknowledgement

This work was supported in part by Sun Microsystems, Intel Corporation, and the NSF (CNS-0524695). The views and conclusions do not necessarily represent those of the sponsors.

References

[1] Paul Bächer, Thorsten Holz, Markus Kötter, and Georg Wicherski. Know your Enemy: Tracking Botnets. http://honeynet.org/papers/bots/, March 2005.

[2] Lee Badger, Daniel F. Sterne, David L. Sherman, and Kenneth M. Walker. A Domain and Type Enforcement UNIX Prototype. *Computing Systems*, 9(1):47–83, 1996.

[3] Computer Industry Almanac, Inc. PCs In-Use Surpassed 900M in 2005, May 2006.

[4] Wenliang Du, Jing Deng, Y.S. Han, Shingang Chen, and P.K. Varshney. A Key Management Scheme for Wireless Sensor Networks Using Deployment Knowledge. In *Infocom*, volume 1, page 597, March 2004.

[5] Laurent Eschenauer and Virgil D. Gligor. A Key-Management Scheme for Distributed Sensor Networks. In *CCS '02: Proceedings of the 9th ACM Conference on Computer and Communications security*, pages 41–47, 2002.

[6] John D. Fernandez and Andres E. Fernandez. Scada systems: vulnernabilities and remediation. *Journal of Computing Sciences in Colleges*, 20:160–168, 2005.

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

[8] Xingxin Gao, Zhe Xiang, Hao Wang, Jun Shen, Jian Huang, and Song Song. An Approach to Security and Privacy of RFID Systems for Supply Chain. In *IEE International Conference on E-Commerce Technology for Dynamic E-Business*, pages 164–168, September 2004.

[9] David Geer. Malicious Bots Threaten Network Security. *IEEE Computer*, 38(1):18–20, 2005.

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

[11] Aaron Higbee and Chris Davis. DC Phone Home. In *Black Hat USA Security Briefings*, August 2002.

[12] Thorsten Holz. Spying with Bots. *LOGIN*, 30(6):18–23, December 2005.

[13] Nicholas Ianelli and Aaron Hackworth. Botnets as a Vehicle for Online Crime. Technical report, CERT Coordination Center, December 2005.

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

[15] 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.

[16] Philip Koopman. Embedded System Security. *IEEE Computer*, 37(7):95–97, July 2004.

[17] Philip Koopman, Jennifer Morris, and Priya Narasimhan. Challenges In Deeply Networked System Survivability. In *NATO Advanced Research Workshop On Security and Embedded Systems*, 2005.

[18] Mike D. Schiffman. *The Libnet Packet Construction Library.*

[19] Spyde$\tilde{1}$, AutoNiN, and Mystic. The UPS (Undetectable Packet Sniffer). In *Defcon 11*, August 2003.

[20] Jing Su, Kevin K. W. Chan, Andrew G. Miklas, Kenneth Po, Ali Akhavan, Stefan Saroiu, Eyal de Lara, and Ashvin Goel. A Preliminary Investigation of Worm Infections in a Bluetooth Environment. In *Proceedings of the Workshop on Rapid Malcode (WORM)*, Fairfax, VA, November 2006.

[21] SwatIt.org. Bots, Drones, Zombies, Worms and Other Things That Go Bump in the Night. http://swatit.org/bots/, 2003.