Offensive Technologies Project

Master System and Network Engineering

University of Amsterdam

Firmwall: Protecting hard disk firmware

Jan Laan jan.laan@os3.nl

Niels van Dijkhuizen niels.vandijkhuizen@os3.nl

May, 2014

Abstract

This paper describes our method to protect firmware of SATA and PATA hard disks. This method monitors ATA commands, and blocks potentially malicious commands reading from and writing to the disk. This is done by disabling the classic x86 privileged Input/Output to the hard disk, and by intercepting the `ioctl` system call. This interception is done with a kernel module. Whenever a direct ATA command is issued through the `ioctl` function, it is checked against a whitelist of defined ATA commands. If the command does not exist in this whitelist, it will be blocked and the responsible program will be terminated. This effectively blocks firmware uploads / downloads or manipulation. Our module can be controlled from user space with a manager. It also supports a configurable whitelist for applications that are allowed to perform non-defined commands. We can succesfully block non-defined ATA commands and the performance impact of our module is not noticeable during regular desktop usage.

Contents

1 Introduction **3**

2 Scope **3**

3 Related research **3**

4 Background **4**
4.1 ioctl 4
4.1.1 SG_IO 4
4.1.2 HDIO_DRIVE_TASKFILE 6

5 Approach **6**
5.1 Blocking suspicious ioctls 6

6 Results **7**
6.1 Vendor-specific commands 7
6.2 Prelude IDS 8
6.3 Performance impact 8

7 Future work **8**

8 Conclusion **9**

9 References **10**

Appendix A: Source code **11**

Appendix B: Performance tests **12**
A dd test 12
B cp test 13

Appendix C: SCSI commands **14**

Appendix D: ATA commands **14**

Appendix E: Prelude IDS rule **15**

1 Introduction

Hard drive firmware is an interesting attack vector for rootkit/backdoor creators. The firmware is not checked by regular anti-virus or intrusion detection solutions. Once malicious firmware has been installed, it can run virtually undetected. Publications of NSA internal documents as IRATEMONK [1] or Jeroen Domburg's hard disk rootkit [2] have shown this is a very real threat. In this paper, we present a method to log and block suspicious hard disk access.

2 Scope

There are essentially three methods to detect changes in firmware:

- Sandboxing software
- Comparing cryptographic hashes of earlier made firmware dumps with the 'current' state (signature-based approach).
- Detecting system calls which imply firmware reads/writes (behavioural approach).

The signature-based approach has been researched [3] [4], although there is much work that remains to be done there. This is due to the fact that vendors use proprietary methods to embed their code in hard disks.

Sandboxing is a difficult task, as it is hard to verify results of a sandboxing test. The sandbox is also running within an Operating System, which can still directly access the hardware.

The third method has not yet been researched to our knowledge. We are limiting ourselves to the behavioural approach: monitoring and blocking system calls that might manipulate firmware in a malicious way.

The current Debian testing (Jessie) distribution will be used as operating system of choice, this distribution is running a Linux 3.x kernel. Due to time constraints, this research will be limited to ATA hard disks, as these devices usually store a great amount of sensitive data, and as such are an interesting attack target.

3 Related research

Research by Ariel Berkman [5] has shown it is possible to access the negative cylinders (also known as 'service area' or 'firmware area') on a hard disk. This area can be used to hide data. His proof-of-concept tool can dump this data area. Jeroen Domburg [2] has shown that it is possible to access and modify a hard disk's firmware, and also that it is possible to read out the entire hard disk firmware. He proved that one can alter for example the `/etc/shadow` file once it is in the hard disk's cache. This altered cache will then be used by the operating system, allowing a non-privileged user to gain root access.

4 Background

To protect a system, various measures have to be taken. There are in general three methods to access a hard disk on I/O level. These are the following:

- The IA32/x86 architecture uses **ioperm/iopl** to set the I/O access level for a program. After the right level is set the program can sent bytes directly to the device. When these methods are allowed, a root user has unlimited access to hardware devices.
- Similar in function to the above is **/dev/port**, though reading and writing with this device is slower. One must have write access to this device in order to use it. Usually only the root user has these rights for security reasons.
- The clean and architecture independent way to access I/O is **ioctl**.

The first two are only used on IA32/x86 machines and they are rarely used in production. Therefor these can be disabled entirely on a hardened system. The third method will be explained in detail in the following section.

4.1 ioctl

Ioctl is a generic system call. Ioctl is an abbreviation for Input-Output control. It facilitates I/O for any connected device. This includes hard disks, sound cards, USB, PCI cards, etc. Exact implementation of the `ioctl` function depends on the device being accessed.

The function has three parameters. The first is a file descriptor, specifying the device to access. The second parameter is a request code which specifies which function to call on for that device. This is also often referred to as an ioctl. The third parameter is implementation-specific, and can differ per request code. This can be a simple value, or a pointer to a complex data structure. For example, the SG_IO ioctl has as a third parameter a pointer to a struct with 22 members, some of which are again void pointer or char pointers.

```
int ioctl(int d, int request, ...);
```

There are two interesting ioctls regarding possible access to hard disk firmware. There are many more, but these provide generic access to a hard disks device. SG_IO is the standard for current Linux kernels (3.x) , while HDIO_DRIVE_TASKFILE has ioctls for the older IDE subsystem which is deprecated nowadays.

4.1.1 SG_IO

SG_IO is the scsi-generic ioctl [6] for accessing a hard disk. An SG_IO call can encompass any SCSI command supported by the device. These commands (opcodes) are specified in the SCSI standard [7]. This standard defines common operations such as READ, WRITE, INQUIRY.

Interesting operations are the SCSI ATA Pass-through 12 ($0xA1$) and 16 ($0x85$) commands. These commands carry an ATA command within them, tunneling one protocol within another. This translation process is described in the SAT-2 standard [8].

The ATA commands, as defined in the ATA standard [9], can be categorized in different groups. The most important groups are:

- **defined**, defined in the standard, devices should implement these
- **reserved**, reserved for use in future standards.
- **obsolete**, were defined, but not used anymore
- **retired**, similar to obsolete
- **vendor specific**, A vendor is free to use this command as he sees fit.

A complete overview of these commands can be found in appendix D. Figure 1 shows the implementation of the above in the Linux kernel. SCSI commands take place at the upper- and middle level, libATA takes care of the translation to ATA and passes these commands to the lower level (the device driver).

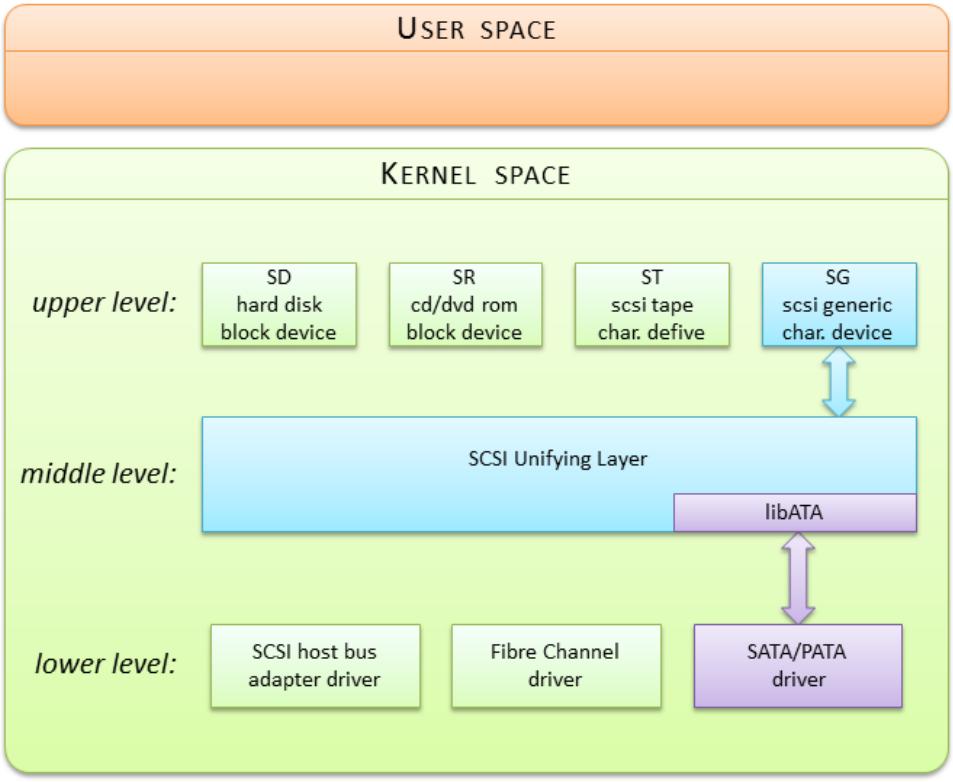

Figure 1: Linux storage driver layers

4.1.2 HDIO_DRIVE_TASKFILE

The functionality of HDIO_DRIVE_TASKFILE is generally similar to SG_IO ioctl. The difference is that SG_IO is for /dev/sdX devices, and HDIO_DRIVE_TASKFILE for /dev/hdX devices. The kernel option CONFIG_IDE_TASKFILE_IO needs to be set in order to use this ioctl. This option was removed somewhere in the 2.6-series kernel and the use of direct ATA/IDE is deprecated in the 3.x kernels.

5 Approach

As mentioned in the previous section, there are in general three methods to send direct IO commands to a hard disk under Linux. Using `iopl`/`ioperm`, using `/dev/port`, or using `ioctl`.

The first two are rarely used in normal operation, and can be blocked by the Grsecurity patches [10]. The third one, `ioctl`, is a frequently-used system call, and hence cannot be blocked entirely. The approach of this research is to monitor calls to `ioctl` and block (potentially) malicious system calls.

5.1 Blocking suspicious ioctls

A loadable kernel module is created to facilitate monitoring and blocking `ioctl` calls. This module listens to all `ioctl` calls. When a suspicious `ioctl` call is detected, it is blocked, otherwise the call is performed as usual. This process is visualized in figure 2. The module patches the system call table so that all calls to the `ioctl` function will pass through the module's custom `ioctl` function. This custom function will then decide whether to pass the call through to the real `ioctl` function or to block the call.

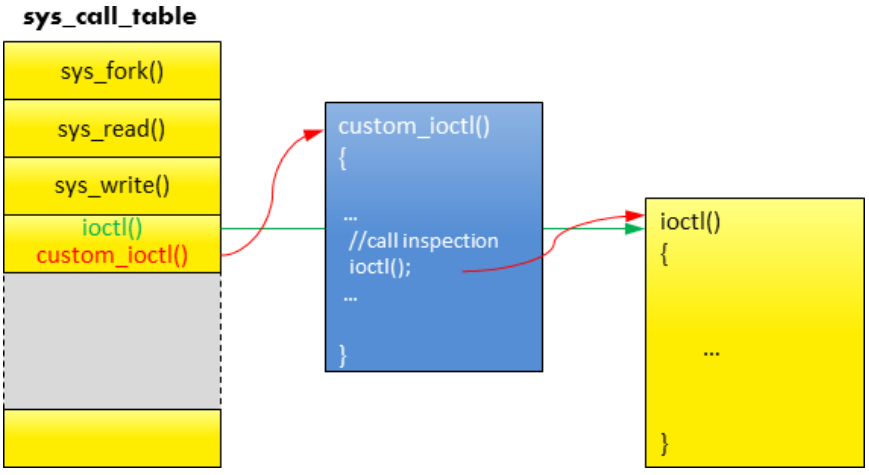

Figure 2: ioctl interception

Defined ATA commands are not classified as suspicious, and these are whitelisted by the module. All other commands (reserved, obsolete, retired and vendor specific) are considered suspicious. These do not occur in normal operation. As such, when an application performs one of these calls, it is suspicious by default. A list of all commands, categorized, can be found in appendix D.

The module has a simple management interface, which can perform two actions. The first one allows the module to be turned on or off. The second one is a whitelist interface. A user can add a specific executable to the module's whitelist. Calls from an executable in the whitelist will always be performed, and not blocked by the module, regardless of their suspiciousness. Note that this interface is unsafe. An attacker with root access can use this to add his own executable to the whitelist, or to turn off the module entirely. This interface should be removed or replaced in a production version.

6 Results

Listing 1 shows the output of the created kernel module. Lines 2 and 3 indicate a blocked system call. The program OCZ Toolbox is also killed. The 0x2285 is the SG_IO ioctl call which contains the non-defined 0xFE (vendor specific) command. At line 5 OCZ Toolbox is executed again. Because the program is now in the whitelist, the non-defined calls made by it are no longer blocked.

```
1  2014-05-21 11:21:49 | Blocking / Monitoring: Enabled
2  2014-05-21 11:21:55 | Intercepted: ioctl(7, 0x2285) from
       PID 2761, exe: /root/OCZToolbox
3  2014-05-21 11:21:55 | Blocked non-defined command: 0xFE
4  2014-05-21 11:22:25 | Added to Whitelist: /root/
       OCZToolbox
5  2014-05-21 11:22:29 | Whitelist match, skipping
6  2014-05-21 18:46:19 | Whitelist:
7  2014-05-21 18:46:19 |  - /usr/sbin/idle3ctl
8  2014-05-21 18:46:19 |  - /root/OCZToolbox
9  2014-05-21 18:58:51 | Removed from Whitelist: /usr/sbin/
       idle3ctl
10 2014-05-21 19:00:22 | Blocking / Monitoring: Disabled
11 2014-05-22 09:32:16 | HDFW: Unloaded
```

Listing 1: Kernel module output

6.1 Vendor-specific commands

Two distinct vendor-specific ATA commands were intercepted. Western Digital uses the vendor specific command $0x80$ for communication with its hard disks. This is illustrated by the idle3-tools[1] and fwtool[2]

[1] idle3 tools: http://idle3-tools.sourceforge.net/
[2] fwtool: http://spritesmods.com/?art=hddhack

programs. OCZ uses the vendor specific command $0xFE$. This is illustrated by the OCZ Toolbox[3] program.

No other in-use vendor-specific commands were identified, and we have not found any Linux tools from other vendors.

6.2 Prelude IDS

A simple rule for the Prelude IDS system [11] has been created, which listens for blocked `ioctl` calls originating from our module. The module logs information to the syslog, which will be read by Prelude. The rule is shown in appendix E. When our module blocks a call, Prelude detects this and registrates this in it's database. The result can be seen in the PreWikka web-interface, see figure 3.

Classification	Source	Target	Analyzer	Time
1 x **Denied ioctl access** 2 x Credentials Change (**succeeded**)	n/a	127.0.1.1	PAM (firmwall.os3.local) HDFW (firmwall.os3.local)	22:56:51 - 22:50:59
1 x **Denied use of iopl** (failed) 2 x Remote Login (**succeeded**)	53530831.cm-6-4a.dynamic.ziggo.nl	127.0.1.1	grsecurity (firmwall.os3.local) sshd (firmwall.os3.local)	22:55:58 - 22:50:59

Figure 3: Prelude IDS / Prewikka interface

6.3 Performance impact

For most ioctls, the overhead introduced by the created kernel module is one single if-statement and an extra context switch, created by passing the ioctl through to the real ioctl function.

We performed some simple checks to test the impact of the created module in normal operation. Test results can be found in appendix B. From these results it can be concluded that performance impact is negligible. No significant performance differences were found.

On environments which heavily use `ioctl` calls, there can be an impact on performance, however this has not been verified.

7 Future work

The ATA standard and the Linux kernel are both quite complex. While we feel that we have a good understanding of the relevant parts, and implemented our module accordingly, it is entirely possible that we missed an entrance to an ATA hard disk which is not being checked by our module.

While the module we have created succesfully blocks SCSI Pass-through `ioctl` calls to ATA hard disks, intercepting calls to native SCSI hard disks have not been looked into.

We tested our module on a Desktop computer that is connected directly to the internet. Unfortunately we did not see real world ATA hard disk exploits on this machine. It would be very interesting to see if our module combined with a Grsecurity patched Linux kernel gets triggered in a honeypot setup.

[3]OCZ Toolbox: http://ocz.com/consumer/download/firmware

Our method could be implemented in a cleaner and more coherent way using a security framework. We believe seccomp_bpf is a potential candidate framework. OSSEC [12] and Samhain [13] could help to guarantee the integrity of the binaries we use in our whitelist.

8 Conclusion

The created proof-of-concept shows it is feasible to monitor and block unwanted system calls to hard drives while maintaining a workable system.

As a standalone solution the proof-of-concept has limited use, it should be used in addition to existing hardening methods.

There is still room for improvement, as mentioned in the future work section. The current management interface is insecure. However, with some modifications the created kernel module can be turned into a good addition to Linux device security.

9 References

[1] Der Spiegel. Interactive Graphic: The NSA's Spy Catalog. http://www.spiegel.de/international/world/a-941262.html, dec 2013.

[2] Jeroen van Domburg. Hard Disk Hacking. http://spritesmods.com/?art=hddhack.

[3] Loïc Duflot, Yves-Alexis Perez, and Benjamin Morin. Run-time firmware integrity verification: what if you can't trust your network card. *CanSecWest/-core11, Vancouver (Canada)*, pages 9–11, 2011.

[4] Yanlin Li, Jonathan M McCune, and Adrian Perrig. Viper: verifying the integrity of peripherals' firmware. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 3–16. ACM, 2011.

[5] Ariel Berkman. Hiding data in hard-drive's service areas. 2013.

[6] The Linux SCSI Generic (sg) HOWTO. http://www.tldp.org/HOWTO/SCSI-Generic-HOWTO/.

[7] SCSI standard, T10. http://www.t10.org.

[8] Mark Overby. SCSI / ATA Translation 2, INCITS 465. http://www.t10.org/members/w_sat2.htm.

[9] ATA standard, T13. http://www.t13.org/.

[10] Grsecurity. http://grsecurity.net/.

[11] Prelude IDS. https://www.prelude-ids.org/.

[12] OSSEC IDS. http://www.ossec.net/.

[13] Samhain IDS. http://www.la-samhna.de/samhain/.

Appendix A: Source code

The following function is the custom `ioctl` function created. This function sits before the real `ioctl` function and filters out potentionally suspicous hard disk calls. The entire source code of the kernel module and the management interface, as well as a few management scripts and the created Prelude IDS rule can be found on Github at https://github.com/janlaan/firmwall

```
/*
 * Our ioctl replacement
 * Most of the time just passes through the requests to
     the real ioctl function.
 * Intercepts, and blocks non-defined ata commands,
     unless the requesting executable is whitelisted.
 */
asmlinkage int custom_ioctl(int fd, int request, void*
    arg)
{
   int suspicious = 0;
   sg_io_hdr_t* io_hdr;
   unsigned char* cdb;
   unsigned int command = 0x9999;
   struct hdio_taskfile* taskfile;
   if(request == SG_IO) //transfer data to device
   {
          suspicious = 1;
          io_hdr  = (sg_io_hdr_t*) arg;
          cdb = io_hdr->cmdp;
          if(cdb == NULL)
          {
             //invalid command, cdb should not be null.
             suspicious = 0;
          }
          else if(cdb[0] == SG_ATA_12)
          {
             command = cdb[9];
          }
          else if(cdb[0] == SG_ATA_16)
          {
             command = cdb[14];
          }
          else
          {
             int scsi_cdb_code = cdb[0];
             printk(KERN_WARNING "[HDFW] Non-ATA 12 or 16
                 command, SCSI CDB command: %s [0x%X]\n"
                 , CDB_OPCODE[scsi_cdb_code],
                 scsi_cdb_code);
             suspicious = 0; //dangerous assumption
          }

   }
```

```
    if(suspicious == 1)
    {
        int level = 9999, i;
        for(i = 0; i < ATA_DEFINED_SIZE; i++)
        {
            if(command == ATA_DEFINED[i])
            {
                level = 0;
                continue;
            }
        }
        if(level > 0)
        {
            int whitelisted = -1;
            int requestingpid = current->pid;
            char * exename = "";
            exename = current_exename(exename);
            whitelisted = check_whitelist(exename);
            if(whitelisted != 0)
            {
                printk(KERN_CRIT "[HDFW] Intercepted: ioctl
                    (%d, 0x%X) from PID %d, exe: %s\n", fd,
                    request, requestingpid, exename);
                printk(KERN_CRIT "[HDFW] Blocked non-defined
                    command: 0x%X\n", command);
                send_sig(SIGKILL, current, true);
                return EINVAL;
            }
            else
            {
                printk(KERN_WARNING "[HDFW] Whitelist match,
                    skipping\n");
            }
        }
    }

    return real_ioctl(fd, request, arg);
}
```

Listing 2: Source code of ioctl interception function

Appendix B: Performance tests

A dd test

Without our kernel module active:

```
-=firmwall=-(root)/# /etc/init.d/hdfw_service stop; for
    i in 1 2 3 4; do dd if=/dev/zero of=/dev/sdb1 bs=1G
    count=1 oflag=direct; done
HDFW stopped

1+0 records in
```

```
1+0 records out
1073741824 bytes (1.1 GB) copied, 11.2396 s, 95.5 MB/s
1+0 records in
1+0 records out
1073741824 bytes (1.1 GB) copied, 11.2419 s, 95.5 MB/s
1+0 records in
1+0 records out
1073741824 bytes (1.1 GB) copied, 11.2186 s, 95.7 MB/s
1+0 records in
1+0 records out
1073741824 bytes (1.1 GB) copied, 11.2078 s, 95.8 MB/s
```

Listing 3: dd Performance test, HDFW off

With our kernel module active:

```
-=firmwall=-(root)/# /etc/init.d/hdfw_service status;
    for i in 1 2 3 4; do dd if=/dev/zero of=/dev/sdb1 bs
    =1G count=1 oflag=direct; done
HDFW kernel module is loaded

1+0 records in
1+0 records out
1073741824 bytes (1.1 GB) copied, 11.225 s, 95.7 MB/s
1+0 records in
1+0 records out
1073741824 bytes (1.1 GB) copied, 11.2439 s, 95.5 MB/s
1+0 records in
1+0 records out
1073741824 bytes (1.1 GB) copied, 11.2216 s, 95.7 MB/s
1+0 records in
1+0 records out
1073741824 bytes (1.1 GB) copied, 11.2371 s, 95.6 MB/s
```

Listing 4: dd Performance test, HDFW on

B cp test

Without our kernel module active:

```
-=firmwall=-(root)/# /etc/init.d/hdfw_service stop; for
    i in 1 2 3 4; do time cp -R ./fakefiles/ /mnt/sdb1/;
     rm -R /mnt/sdb1/fakefiles; done
HDFW stopped


real    0m10.367s
user    0m0.032s
sys     0m5.792s


real    0m10.396s
user    0m0.016s
sys     0m5.828s
```

```
real    0m10.593s
user    0m0.028s
sys     0m5.780s


real    0m9.928s
user    0m0.024s
sys     0m5.796s
```

Listing 5: cp Performance test, HDFW off

With our kernel module active:

```
-=firmwall=-(root)/# /etc/init.d/hdfw_service status;
    for i in 1 2 3 4; do time cp -R ./fakefiles/ /mnt/
    sdb1/; rm -R /mnt/sdb1/fakefiles; done
HDFW kernel module is loaded

real    0m10.771s
user    0m0.064s
sys     0m5.880s


real    0m11.217s
user    0m0.004s
sys     0m5.820s


real    0m10.160s
user    0m0.040s
sys     0m5.800s


real    0m9.996s
user    0m0.040s
sys     0m5.772s
```

Listing 6: cp Performance test, HDFW on

Appendix C: SCSI commands

A complete list of SCSI commands can be found at the T10 website at http://www.t10.org/lists/op-num.htm. For this research only the ATA PASS-THROUGH commands are inspected ($0x85$ and $0xA1$).

Appendix D: ATA commands

The following list of all ATA commands was extracted from the ATA standard at http://www.t13.org. They are categorized by usage type.

Table E.1 - Command matrix

	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xA	xB	xC	xD	xE	xF
0x	C	R	R	C	R	R	R	R	C	R	R	R	R	R	R	R
1x	O	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E
2x	C	O	O	O	C*	C*	C*	C*	R	C*	R	R	R	R	R	C*
3x	C	O	O	O	C*	C*	C*	C*	C	C*	R	R	O	R	R	C*
4x	C	O	C*	R	R	R	R	R	R	R	R	R	R	R	R	R
5x	O	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R
6x	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R
7x	C	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E
8x	V	V	V	V	V	V	V	F	V	V	V	V	V	V	V	V
9x	C	O*	C	R	E	E	E	E	E	E	V	R	R	R	R	R
Ax	C	C	C	R	R	R	R	R	R	R	R	R	R	R	R	R
Bx	C	C*	R	R	R	R	R	R	A	A	A	A	A	A	A	A
Cx	F	V	V	V	C	C	C	C	C	O	C	O	C	C	R	R
Dx	R	C*	M*	M*	M*	R	R	R	R	R	C	E	E	E	C	C
Ex	C	C	C	C	C	C	C	C	C	E	C*	R	C	C	O	C
Fx	V	C	C	C	C	C	C	V	C	C	V	V	V	V	V	V

Key:
C = a defined command.
R = Reserved, undefined in current specifications.
V = Vendor specific commands.
O = Obsolete.
E=a retired command.
F=If the device does not implement the CFA feature set, this command code is Vendor specific.
A=Reserved for assignment by the CompactFlash‰ Association
M=Reserved for the Media Card Pass Through Command feature set.
* indicates that the entry in this table has changed from ATA/ATAPI-5, NCITS 340-2000.

Figure 4: ATA commands categorized

Appendix E: Prelude IDS rule

The below rule can be added to the Prelude IDS system. When our module is running, any alerts will be logged by Prelude.

```
# HDFW: Loadable Linux kernel module for monitoring and
    blocking suspicious hard disk activity.
# This is the Prelude-LML ruleset beloning to HDFW.

regex=Intercepted: ioctl\([0-9]+, 0x[0-9]+\) from PID
    ([0-9]+), exe: (.*); \
 classification.text=Denied ioctl access; \
 id=901; \
 revision=1; \
 analyzer(0).name=HDFW; \
 analyzer(0).manufacturer=http://www.os3.nl; \
 analyzer(0).class=Integrity; \
 assessment.impact.severity=high; \
 assessment.impact.completion=failed; \
 assessment.impact.type=file; \
 assessment.impact.description=Process ID $1 ($2) tried
    to access a storage device via ioctl; \
 source(0).process.name=$2; \
 source(0).process.pid=$1; \
 last;
```

Listing 7: HDFW Prelude rule