Informatics System Compromise

Stephen Quinney <squinney@inf.ed.ac.uk>

Thursday 1st December, 2011

Abstract

This document describes the discovery and subsequent investigation of the root compromise of the primary SSH servers running in the University of Edinburgh School of Informatics. It also provides a review of the way in which the systems were managed. The aim of the review is to identify what worked well and to find out where improvements can be made. In particular, the hope is that this will help to reduce the risk of a successful repeat attack and aid in reducing the time it takes to discover any intrusion.

1 Introduction

The University of Edinburgh School of Informatics recently experienced a root compromise of the Secure Shell (SSH) service. There are two primary purposes for this report. The first is to provide an account of the details of the root compromise of the SSH servers. The second is to perform a review of our system administration processes and to consider what requires improvement. Firstly, in Section 3, we describe the basic details of how and when the system compromise was discovered. We then go on, in Section 4, to outline our initial response. Section 5 provides an account of the in-depth investigation into how the compromise was executed and provides details of what to look for if a similar compromise is suspected. Section 6 provides a reasonably comprehensive review of our management processes. For those not interested in all the gory details of the investigation, we recommend skipping directly to this section. An attempt has been made to fairly scrutinise what aspects were done well and what needs improvement. Finally, the results of the investigation and review are combined to form conclusions and a set of proposals which will provide the basis for future work. For those not already familiar with the computing environment within the School of Informatics we recommend reading the background section.

2 Background

The majority of computers within the University of Edinburgh School of Informatics use a managed Linux environment, known locally as DICE. This is designed to support the research and teaching requirements of the School. DICE is based on versions 5 and 6 of Scientific Linux (SL) [1], a free recompiled version of Redhat Enterprise Linux (RHEL) [2]. We currently have approximately 500 managed office machines, 300 student lab machines and 200 servers. The DICE infrastructure provides a common authentication and authorization system based on Kerberos and LDAP. The user home directories are provided using the OpenAFS network filesystem.

The DICE machines are managed using the LCFG configuration management tool [3], [4]. LCFG provides a configuration language and a central repository of configuration specifications, from which individual Unix machines can be automatically installed and configured. The current version of the LCFG software was created locally about 10 years ago. It is a precursor of newer

tools, such as puppet and chef, which share many similar concepts in their design. It continues to be developed by members of the local Computing Team. It is widely used within the University of Edinburgh to manage Linux and MacOSX based computers (a recent estimate put the number of machines at approximately 3700).

As part of the design of the DICE infrastructure we restrict incoming external network access to individual machines using a firewall. The purpose of this is to limit the exposure to outside attackers to a few machines. By default, a machine will have no external network access whatsoever. Specific holes for the network ports associated with a service are only added when necessary. We provide several servers which are accessible by the Secure Shell (SSH) protocol. The intention is that users login to one of these servers and then gain access to local computing resources. The SSH servers are typically known by their DNS CNAMEs, which reflect the user groups who are allowed access (e.g. *staff* or *student*). These CNAMEs act as easy to remember aliases and allow the service to be rapidly moved to alternative hardware when necessary.

3 Discovery

As part of the regular upgrade cycle, the server which hosted the School of Informatics student SSH service (*ratz*) was upgraded to SL5.6 early on Tuesday 4th October and rebooted to install the new kernel at 09:18 on Wednesday 5th. The system subsequently crashed overnight at 04:12 on Thursday 6th with a kernel panic. It was then manually booted at 10:10 but immediately crashed again with another kernel panic before it could complete the boot process. At this point it was left in the crashed state for further investigation.

The DNS CNAMEs were redirected to an alternative server (*rockefeller*) and an announcement email was sent to users so that the SSH service could be restored as soon as possible. An initial investigation was performed the same day but at that stage it was suspected that either a hardware fault had developed because of the reboot or a bad interaction had occurred between the hardware and the SL5.6 upgrade. As a precaution, the system was brought up in single-user mode, networking was started manually and the latest available kernel from SL5 (which was not the default version on DICE) was installed. After that the machine was rebooted once more to bring it up into the normal state and left running overnight to see if any further problems would occur. The machine was connected to the network, the firewall holes remained and access was open to all users.

The kernel panicked again overnight at 03:12 on Friday 7th October. After this further crash, the Redhat bug tracker [5] was checked for kernel problems related to the hardware type. Also memtest86+ was run on the machine to rule out any memory faults. As neither showed any problems the kernel panics (which are preserved in the console logs on a separate server) were inspected more closely. It was noticed that all 3 panics were associated with an attempt to insert a kernel module using the `insmod` command. Although use of the `insmod` command is not unusual during the boot process, it is definitely not normal to see this being used on a server in the middle of the night. On this server it could only have been invoked manually by a user (or user process) which immediately raised suspicion of a deliberate attack against the machine.

Here is an example of the trace from one of the kernel panics.

```
BUG: unable to handle kernel NULL pointer dereference at virtual address 0000003
0
 printing eip:
f8bba94f
*pde = 00000000
Oops: 0000 [#1]
```

```
SMP
last sysfs file: /block/ram0/range
Modules linked in: scsi_core(U) nfs(U) fscache(U) nfs_acl(U) lockd(U) sunrpc(U)
bonding(U) ipv6(U) xfrm_nalgo(U) crypto_api(U) dm_mirror(U) dm_multipath(U) scsi
_dh(U) video(U) backlight(U) sbs(U) power_meter(U) hwmon(U) i2c_ec(U) i2c_core(U
) dell_wmi(U) wmi(U) button(U) battery(U) asus_acpi(U) ac(U) parport_pc(U) lp(U)
 parport(U) sg(U) ide_cd(U) pcspkr(U) e752x_edac(U) edac_mc(U) e1000(U) serio_ra
w(U) cdrom(U) tpm_tis(U) tpm(U) tpm_bios(U) dm_raid45(U) dm_message(U) dm_region
_hash(U) dm_log(U) dm_mod(U) dm_mem_cache(U) ata_piix(U) libata(U) sd_mod(U) scs
i_mod(U) ext3(U) jbd(U) uhci_hcd(U) ohci_hcd(U) ehci_hcd(U)
CPU:    1
EIP:    0060:[<f8bba94f>]    Tainted: G      VLI
EFLAGS: 00010286   (2.6.18-238.1.1.el5.inf.1 #1)
EIP is at mp_cache_proc_write+0xf/0x2d [scsi_core]
eax: 00000000   ebx: 0000000c   ecx: f8bc4100   edx: f8bbb56f
esi: f8bbb56f   edi: f7ccf000   ebp: f7ccf3f8   esp: f3f87e90
ds: 007b   es: 007b   ss: 0068
Process insmod (pid: 6253, ti=f3f87000 task=f7f56aa0 task.ti=f3f87000)
Stack: f8bc0980 f7ccf3c0 f8bbaba5 f8928032 c043f7c4 f3f87ed8 f3f87efc f8bc0980
       00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
       00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
Call Trace:
 [<f8bbaba5>] res_setup+0x41/0x1e5 [scsi_core]
 [<f8928032>] res_wake+0x32/0x57 [scsi_core]
 [<c043f7c4>] sys_init_module+0x1ae0/0x1c9d
 [<c0473933>] __kmalloc+0x0/0x72
 [<c048c8f3>] dput+0x72/0xed
 [<c04902aa>] mntput_no_expire+0x11/0x6a
 [<c0475779>] filp_close+0x4e/0x54
 [<c0404f4b>] syscall_call+0x7/0xb
 =======================
Code: e0 25 00 f0 ff ff 8b 00 83 78 0c 00 78 0b 85 d2 7e 07 89 d0 e9 e8 fe ff ff 89 d0 c3 56
EIP: [<f8bba94f>] mp_cache_proc_write+0xf/0x2d [scsi_core] SS:ESP 0068:f3f87e90
 <0>Kernel panic - not syncing: Fatal exception
```

The next stage was to check which users had been logged into the server overnight 6th/7th October. Due to the time it takes for DNS changes to propagate it was expected that a small number of people would login to this server via the ssh CNAME. Examination of the `last` log showed entries for two users - **A** and **B** - neither of whom were logged in at or near the time when the kernel panic occurred. As the only way to get shell access on the machine is using ssh the sshd "Accepted" entries in syslog were examined. This showed entries for 4 users - **A**, **B**, **C** and **D**. Although entries are not always reliably stored when a system crashes, the difference between these two lists raised suspicions that something had erased the `last` entries for users **C** and **D**. This is a common technique when using compromised accounts which is intended to cover any tracks.

As a precaution in case of attack (or other abuse by users), the SSH servers have process accounting enabled and logs are kept for 30 days of all processes executed on the machines along with the user names and the times. The data is fairly simplistic in terms of the process name stored (just the file-name itself, no path) but it is still very useful for diagnosing problems. This means that in addition to knowing which users had logged in we have some idea of what they had been running. As well as shell logins there was also the possibility that user cron jobs

would be running so not every user would appear in the `last` log.

In the time period from the reboot for the latest kernel up to the crash on the Friday morning, commands were run for the following 4 normal users: **A**, **B**, **C** and **D**, none of whom had registered cron jobs. Close examination of the processes executed suggested that the sessions for users **A**, **B** and **C** were all consistent with normal expected usage. However, the session for **D** immediately raised suspicion: it began at 02:48 and there were 5 calls to a program named `chrn`, which is not a standard system command. This could have been a perfectly legitimate program stored in the user's home directory but immediately after the calls to chrn at 03:06 the *root* user started two bash shells. During this period the *root* user also began running autoconf "configure" scripts, and used the GCC C compiler along with various other tools for building software. Five minutes after these events the kernel panicked which gave a clear indication that the system had been compromised in some way.

Going back further in the process accounting logs it was clear that the same user was logged in immediately prior to the crash early on Thursday 6th October.

4 Initial Response

Once it was clear that *ratz* had been compromised, all external firewall holes for the machine were closed and access was restricted to members of the Informatics Computing Team.

The network interfaces were left "up" so that investigation was easier. This did expose the machine to the slight risk of the exploit code sending more information to the attackers. However, with the kernel panics having happened it was a fairly safe assumption that the exploit code was no longer functional.

At this point the user account which was used to compromise the system was disabled and the user was contacted via email (as their email is handled externally). They were required to change their password before access was returned. On questioning the user admitted to having a very weak password and also not running any anti-virus software on their personal computers. Immediately afterwards the user reinstalled their personal computers and installed anti-virus software to prevent reinfection. Examination of the contents of their home directory revealed no evidence that showed they had introduced the root-kit or used setuid *root* scripts to become the root user. Taking everything into account we have no reason to suspect the user was in anyway involved with the attack. They had purely been used as an access route: their password had either been acquired through a key-logging virus on their personal computer or by password cracking software.

A message was also sent to the University Incident Response Team explaining that one of our primary SSH servers had been compromised.

5 Further Investigation

Once the initial discovery stage was complete and the compromised account was blocked, the investigation moved on to gaining a deeper understanding of what had occurred. There clearly had to be a method of easily elevating the user privilege and there had to be some sort of root-kit which was installed to achieve the attacker's aims. To understand the extent of the system compromise these needed to be discovered and dissected. All other systems then needed to be checked for evidence of similar attacks. Only with a good understanding of the attack could the risk to our wider infrastructure be gauged and thus a suitable response plan prepared.

5.1 The Privilege Escalation Route

Following up on the suspicious `chrn` program, which had previously been spotted in the process accounting logs, a quick search of the entire file-system was done on *ratz*. The command used was:

```
find / -mount -name chrn
```

and this revealed the following files:

```
/usr/lib/locale/nb_NO.gb18050/chrn
/usr/share/system-config-ldap/chrn
/usr/share/system-config-ldap/tools/chrn
```

Each of these 3 programs is setuid *root* and is executable by all users. This is the privilege escalation route which was used to go from being a normal user to having full *root* privilege.

A quick inspection using the `strings` command suggested they are small process launchers which can be used by normal users to start processes running as the *root* user. Alongside the `/usr/lib/locale/nb_NO.gb18050/chrn` file a program named `os` was found which is a simple bash script which could be used to launch a new bash shell. As well as gaining *root* privileges this means that no command history is retained in the calling shell.

One of the interesting aspects of this attack is that each of these directories appears to be a "normally" named directory in the system. The names are consistent with other valid entries (e.g. `/usr/share/system-config-date/` or `/usr/lib/locale/nb_NO.utf8/`) and would probably be missed in a visual inspection of the contents of the parent directories. Notably, neither of them is actually owned by any RPM. This seems to be a deliberate ploy by the root-kit to aid in avoiding detection: a verification of the contents of the RPMs installed on the system would return no errors. They are effectively hidden in plain sight.

To look deeper into the state of the file-system, a full review of all setuid programs was done using this command:

```
for i in $(find / -mount  -perm -u=s,o=x); do\
   ( rpm -qf $i >& /dev/null ) || echo $i;\
done
```

This revealed another user-accessible setuid program which was not owned by any package: `/usr/bin/quota-add`. This appears to be yet another way for a normal user to launch a shell as *root*. Unlike the other process launchers found it launches the bash shell itself.

A correct setuid program would typically carry out authorization checks before actually doing anything (e.g. `sudo`) but clearly these checks could be subverted by replacing the file with a modified version. To ensure that this had not occurred the previous approach of checking files was taken further still. It was used to verify the setuid scripts which are provided as a part of RPMs in case any have been modified:

```
rpm --verify $(for i in $(find / -mount  -perm -u=s,o=x); do\
     rpm -qf --queryformat '%{NAME}\n' $i;\
done | sort | uniq )
```

This did not reveal any more programs which could be used to maliciously escalate user privileges.

5.2 The Root-kit

Examination of the directories on *ratz* which contained the `chrn` script revealed the location of the main root-kit to be the `/usr/share/system-config-ldap/` directory. The contents of the root-kit give a lot of information about the extent of the problem. In particular, the timestamps on the files seem to be entirely genuine and consistent. They reveal that the initial break-in occurred on Thursday 28th Dec 2010. At that time the system was running version `2.6.18-194.3.1.el5.inf.1` of the linux kernel.

The root-kit appears to be an amalgamation of several root exploit toolkits which are readily available on the internet. One notable program found, named `p-2.4g`, was found in a sub-directory named `hacked`. Examination of this compiled executable using the `strings` command shows that it is from the *Phalanx2* Linux root-kit (version 2.4g). This root-kit has been around in various forms for several years, since at least 2008 when it was noted by the *Internet Storm Centre* (ISC) [9] and US-CERT [10]. The *Phalanx2* root-kit has been used in a number of high profile exploits such as the recently discovered intrusion into the Linux kernel infrastructure [7]. According to the documentation this rootkit is detectable using the *rkhunter* tool [12].

The rootkit contains a kernel-devel RPM taken from the Centos5 distribution for the `2.6.18-194.32.1.el5` Linux kernel version. This seems to indicate that the attack is against specific versions of the RHEL5 kernel, which is also used on SL5 and Centos5. The existence of this package, the building of source code by root seen in the process accounting and the failure of `insmod` with the more recent versions of the Linux kernel leads to the conclusion that an extra kernel module was being built and installed as part of the attack. The root-kit directory contains several sub-directories which hold kernel modules with different names:

```
bak/bond_core.ko
bak/dm_raid10.ko
bak/usbkey.ko
k1/bond_main.ko
k1/key.ko
k1/scsi_core.ko
k2/scsi_core.ko
k3/scsi_core.ko
```

Those in the `bak` directory are from Thursday 28th December 2010 when the kernel was `2.6.18-194.3.1.el5.inf.1`, those in `k1` are from Thursday 27th January 2011 when the kernel was upgraded to version `2.6.18-194.32.1.el5`, those in `k2` are from Thursday 6th October 2011 when the kernel was upgraded to `2.6.18-238.1.1.el5.inf.1` and those in `k3` are from Friday October 7th 2011 when the kernel version was `2.6.18-274.3.1.el5`

The contents of the modules were examined with the `strings` command and the `usbkey.ko` and `key.ko` modules revealed that it is based on code from "The Hacker's Choice" website [14]

Further examination of the scripts in the rootkit and the process accounting logs showed usage of a `vlogctrl` script. This is part of the *vlogger* code [15] which is distributed on The Hacker's Choice website . It describes itself thus:

> "THC-vlogger, an advanced linux kernel based keylogger, enables the capability to log keystrokes of all administrator/user's sessions via console, serial and remote sessions (telnet, ssh), switching logging mode by using magic password, stealthily sending logged data to centralized remote server. Its smart mode can automatically detect password prompts to log only sensitive user and password information.!"

This code was first released in 2003 and presumably has been updated for the later kernel versions. As yet we have not discovered the location of the source code for the kernel modules

but clearly it must be on the system somewhere to allow automated rebuilds whenever the kernel version is updated.

Clearly for a long period of time the attackers had the ability to acquire usernames, passwords and other sensitive information and silently send this back to a central location.

To check if the root-kit was all stashed in one particular location or spread around the file-system a search was done to find all files which are not owned by RPMs. This revealed that there are over 5000 files of this type. Thankfully, nearly all of these are in the `/var` directory and are almost certainly legitimate log and cache files. There are about 50 which are elsewhere in the file-system, after a visual inspection of this shorter list these two stuck out:

```
/bin/rlog
/bin/startlog
```

There are no calls to `startlog` in the process accounting logs on *ratz* but there is a single call to `rlog` in each boot sequence. This points to the method used by the root-kit to ensure the exploit is restarted with each system reboot.

Again, this seems to be an attempt to hide files in plain-sight, as they look like legitimate file names. Only by spotting that they are not part of any RPM do they become apparent. They are also the types of process names that might legitimately appear in a standard boot sequence so are unlikely to raise any alarms.

Having found that the call to `rlog` was not normal, a verification of the contents of the `/etc/rc.d/init.d/` directory was done. This is the most likely place from which to make the call to `rlog`, particularly as LCFG handles most of the boot-sequence itself. The following command was used:

```
rpm --verify $(rpm -qf --queryformat '%{NAME}\n' * | sort | uniq ) \
             | grep -F /etc/rc.d/init.d
```

This revealed the following modified files:

```
S.5.....    /etc/rc.d/init.d/sshd
```

This highlights a very significant issue with files stored in the `/etc/rc.d/init.d/` directory. They are, in essence, executable scripts which are only ever used for starting daemons and yet they are considered to be "conffiles" by RPM. Being a "conffile" means that the file is considered to be a system configuration file and as such is expected to be modifiable by a system administrator. The primary purpose of this system is to avoid software updates overwriting files which have been altered by the administrator. This in turn means that a verification of the RPM contents does not flag up the change in contents as being a problem. It also typically means that the changes made by the attacker will not be overwritten by an upgrade of the package.

Looking at those scripts the relevant lines are:

```
/etc/rc.d/init.d/sshd:rlog && [ -f /etc/sysconfig/sshd ] && . /etc/sysconfig/sshd
```

This matched up with the evidence in the process accounting logs which showed the exploit code starting at about the same time as the SSH daemon.

One other interesting thing found in a couple of the scripts in the root-kit is the usage of "*what is my IP?*" services. The scripts use both `http://www.whatismyip.org/` and `http://www.whatismyip.com/` to check the current IP address. These are legitimate services which can be very useful in scripts run by authorised users but clearly any unusual connection activity to these sites could be indicative of a system compromise. Regular connections to these sites are something which could be monitored using a firewall.

5.3 Checking Other Systems

Immediately after detecting the root-kit on *ratz* we used the evidence found to scan all the other DICE servers. Only the second SSH server - *rydell* - access to which is restricted to staff and post-graduate students, was also found to have been compromised. Further to this, we also checked for any machines which were still running the version of the Linux kernel which was known to be exploitable and rebooted them to a newer version.

5.3.1 Staff SSH Server

As soon as we identified that the staff SSH server had also been compromised the service was moved to another server (by changing the DNS CNAME entry) and the users were informed. We allowed users with existing connections 3 hours grace to finish their work and log off, after that time a few user sessions were forcibly disconnected. After that point all firewall holes were closed for the server and access was restricted to the Computing Team.

The scan of the file-system looking for setuid *root* binaries found one at `/usr/bin/at-start`, alongside this there is a non-suid binary, named `at-view`, which is for removing incriminating entries from various system logs. A comment in the script claims it is "*Logclean by Tick*". It checks the following log files:

```
/var/log/messages
/var/log/dmesg
/var/log/kern.log
/var/log/kernel/errors
/var/log/kernel/info
/var/log/kernel/warnings
/var/log/warn
/var/log/secure
/var/log/auth.log
/var/log/user.log
/var/log/security.log
/var/log/authlog
/var/log/syslog
/var/log/sulog
/var/log/htmlaccess.log
/var/log/httpd/access_log
/var/log/httpd/error_log
/var/log/httpd/ssl-access_log
/var/log/httpd/ssl-error_log
/var/log/httpd/ssl-request_log
/etc/httpd/logs/access_log
/etc/httpd/logs/error_log
/etc/httpd/logs/ssl-access_log
/etc/httpd/logs/ssl-error_log
/etc/httpd/logs/ssl-request_log
/var/apache/logs/access_log
/var/apache/logs/error_log
/var/apache/logs/ssl-access_log
/var/apache/logs/ssl-error_log
/var/apache/logs/ssl-request_log
/var/log/xferlog
```

```
/var/log/proftpd.log
/var/log/daemons/info
/var/log/daemons/warnings
/var/log/daemons/errors
```

The script also appears to have the ability to modify `utmp`, `wtmp` and `lastlog` files.

This shows the huge benefit which can be gained by sending important log and audit information directly to a separate central server where it can never be altered by an attacker.

On this machine the scan of the file-system showed that the root-kit was stored in the `/usr/lib/stunnel/` directory.

Within the root-kit the following kernel modules had been built:

```
bak/afs_keys.ko
bak/e1000_bond.ko
bak/openafs_core.ko
k1/afs_keys.ko
k1/e1000_bond.ko
k1/openafs_core.ko
```

The following scripts in the `/etc/rc.d/init.d` directory had been modified:

```
S.5....T  c /etc/rc.d/init.d/autofs
S.5.....    /etc/rc.d/init.d/sshd
S.5....T  c /etc/rc.d/init.d/syslog
```

In this case the root-kit is started using a binary named `rlogd`:

```
/etc/rc.d/init.d/autofs:if rlogd && [ -r $confdir/autofs ]; then
/etc/rc.d/init.d/sshd:rlogd && [ -f /etc/sysconfig/sshd ] && . /etc/sysconfig/sd
/etc/rc.d/init.d/syslog: rlogd && [ -x /sbin/syslogd ] || exit 5
```

In the root-kit directory there is a sub-directory named `XguYiJehTaOD.p2/` which contains a lot of files (56) named like `.sniff-8904`. These files contain information collected by the *Phalanx2* root-kit and are **complete** traces of SSH login sessions. This shows that the attacker could have access to much more information than just usernames, passwords and SSH keys.

5.3.2 Old CVS/SVN Server

There is one further server which we have setup to provide SSH access to all DICE users. This server provides access to our CVS and SVN repositories. Earlier in the year we had experienced problems with the machine on which it was hosted - *stockholm* - related to an unsuccessful attempt to replace the SSH daemon with a trojanned version. After that episode the machine was replaced with another and it was decommissioned as it was rather old hardware. The interesting aspect of this is that the machine remained turned-off since that point and had not had the disks wiped. We decided it was worth checking the machine based on the evidence we had collected from the SSH servers. When we reactivated the machine we discovered that it did indeed have the root-kit installed and, as the kernel was the old exploitable version, it was still able to become active. This gave us a good opportunity to examine the behaviour of the root-kit.

The investigation into this system revealed that the kernel module installed by the attacker is capable of hiding all the root-kit processes from normal process listings (e.g. using `top` and `ps`) and it had also hidden itself from the kernel module listing (found using `lsmod`). It also

appears that it intercepts system calls so that it can "hide" the location of the root-kit directory. This means that whilst it is active any usage of standard Unix tools such as `find` or `ls` will not show the existence of the directory.

We ran the *chkrootkit* tool (version 0.49) [11] on this system and it did spot the trojan Linux Kernel Module:

```
Checking 'lkm'... You have     1 process hidden for readdir command
chkproc: Warning: Possible LKM Trojan installed
```

So this shows that there is real value to be gained from regularly running intrusion detection tools such as *chkrootkit*.

We also considered running the *rkhunter* tool [12] but time-pressure and the complexity of the configuration (when compared with *chkrootkit*) combined with our unfamiliarity with the tool meant that we did not use it during the investigation. In the future we plan to regularly run both as there clearly is a lot of scope for false-positives and issues being missed by any single tool.

A search of the file-system revealed that the root-kit was installed in the `/usr/share/doc/gfs-utils-1.0.0` directory. The contents of this directory were very similar to that found on *ratz*. The kernel modules which were built at various times were named:

```
k1/dm_core.ko
k1/nfs_core.ko
k1/parport_core.ko
k2/dm_core.ko
k2/nfs_core.ko
k2/parport_core.ko
```

Again, these are all believable enough at a first glance but none of them are real Linux kernel module names.

A further search in the file-system for binaries which were setuid *root*, runnable by normal users and not owned by any RPM revealed the existence of the `/usr/bin/newusr` file. This appears to be used for launching processes as the *root* user. Examination of the binary using the `strings` command suggests this is very similar to the `chrn` binary found on *ratz*.

The files in the `/etc/rc.d/init.d/` directory were verified and the following modified file was found:

```
S.5.....  c /etc/rc.d/init.d/crond
```

In this case the script used to start the root-kit code was named `startlog` and was stored in the `/bin` directory. Again it was inserted into the init-script in the same way as on *ratz*.

```
startlog && [ "$t" != "UNSET" ] && export CRON_VALIDATE_MAILRCPTS="$t"
```

In the root-kit directory is a shell script named `romeo` and associated with this is a sub-directory, named `.unsent`, within which were found a large number of files (126) named with timestamps like `romeo.20111012170801`. The script appears to be used to handle SSH keys that have been intercepted. Although we cannot make any guarantees, the contents of the "unsent" sub-directory clearly suggests that they were not successfully sent off the system.

6 Discussion

6.1 What Did We Do Right?

In a situation like this it is very easy to focus entirely on all the negatives and only examine what went wrong, why that happened and where the fault lies. It is genuinely useful though to take stock of the entire situation and see what was done well. If nothing else the hope is that this should be useful to other sites who may wish to avoid going through a similar "learning process".

6.1.1 Restricted SSH Access

To limit our exposure we have always restricted external SSH login access to a small number of machines. The most likely purpose of this attack seems to have been to collect sensitive data from connections: there does not appear to have been any deeper intrusion into our network. It is relatively easy and quick to scan for open ports on an entire network of machines and identify those listening for SSH connections. Consequently we can infer that if external SSH login access had been available widely across our servers and desktops the effects of this attack could have been truly devastating. So it is very clear that limiting SSH access to our network is a very simple and effective way of limiting our exposure to attacks and helps reduce the potential for serious damage.

On the SSH servers we have the SSH daemon configured to only permit access for real users. This effectively blocks SSH login access to all "system" accounts: these are routinely attacked in an attempt to find weak points in our infrastructure. In the system logs there are a large number of failed login attempts for a huge variety of common Unix system account user names which demonstrates just how prevalent this attack is and that our defences are effective. Normally the entries in the password file ensure that it is not possible to login to those accounts by setting an appropriate shell but without this additional protection a simple misconfiguration could allow access to an attacker. Additionally to this, we further restrict login access for real users by using the `pam_access` PAM module so that only those users with the correct roles are allowed to access the servers. This shows that we do have real effective protection against unauthorized access to accounts which should never need to be accessible via SSH. However this, of course, does not protect against a valid user account being compromised elsewhere. If the user exposes their account information on an insecure network or has a virus on their own personal computer their username, password and the locations of target systems can be intercepted.

One issue that our security review did uncover is that we currently have a large number of user accounts that are "live" but pending deletion. In many cases it is still possible to login to these accounts even though they are no longer required by the original people with whom they were associated. This is a weak point as unauthorized access could occur and without the original person still being around there really is no-one who might notice any unusual behaviour. These user accounts which are pending deletion could in future be put into a "nologin" group for which SSH login access can be easily banned.

6.1.2 Kerberos Authentication

The evidence found on the systems indicates that the main aim of the root-kit was to collect usernames, passwords and other sensitive data, such as SSH keys. Within the School of Informatics we have been using Kerberos authentication for a long time: this attack has proved the true value of that system in several ways. Firstly, if the passwords had been stored locally on the servers as a file or in a Network Information Service (NIS) directory the attacker would have instantly gained access to all user passwords (albeit in an encrypted form). When the attacker

has *root* access having shadowed passwords, for instance, is of no benefit. Secondly, the use of Kerberos makes it possible to securely authenticate directly with the KDC using `kinit` before making an SSH login to the server. This means that no password is transmitted to the SSH server and thus it could not be intercepted by the kernel module which had been inserted by the attacker. There are additional benefits to this approach such as having single-sign-on which means that multiple logins can be made without further password prompts. Given the clear benefits we should do more to promote to our users this mode of access to the SSH servers.

We should also aim to eradicate SSH public key access to the front-line servers. This authentication method is of little benefit to our users since file-system access would additionally require the entry of the password. Although an SSH public key using a good modern encryption system is effectively "uncrackable", they still present a security risk. It is well known that SSH keys are often used on multiple systems and the private keys are often not well protected by users. This has been the direct entry mechanism for a number of cases of system compromise for high-profile targets over the last few years. Another issue is that whilst we can centrally enforce a password change (and require any new password is different) we cannot enforce a change of private keys since they are maintained independently individual users.

We have also discovered that our use of AFS for home directories creates some potential for users to mistakenly leak their private keys. AFS has directory-based ACLs rather than the traditional Unix-style file-based ACLs and this has occassionally led to users allowing anyone in the world with an AFS client to read their private keys. We should regularly check the user home directories for insecure private keys and educate users appropriately when they are found.

One way in which we could enhance the security of our authentication process would be to introduce the requirement of a secondary factor. This type of system is typically based on using a physical device. At the current time this is unlikely to be easily workable for our normal user logins. As this type of system is taken up by more businesses, such as banks, it will become easier to introduce into Informatics systems. At this stage we should seriously consider it as part of the standard *root* access process. This would be done through the PAM stack and it should be noted that this does nothing to prevent unauthorised privilege escalation through security flaws.

6.1.3 AFS Home Directories

Our use of AFS home directories was especially valuable in reducing the impact of this attack. The attacker was limited to the access permissions of the compromised user account and, if the attacker was aware that Kerberos and AFS were being used, the access permissions of users with current sessions. On our systems, having access to the *root* account does not automatically grant complete access to the available file-systems. This is a distinct advantage of AFS compared to the traditional-style of NFS usage where the *root* user is able to read and modify any NFS file-system to which the machine has mount access. Even with "root-squashing" to prevent the user directly modifying files it does not own, *root* on the client can still use a tool, such as `su`, to become any other user and then read or change their files.

6.1.4 Secure Access Only

Although this attack was carried out via SSH it remains one of the best available technologies available for connecting to our systems, the usage of SSH is not in any way a mistake. The usage of a secure, encrypted, communication system is absolutely essential in the modern world. We do not permit any access by unencrypted technologies such telnet, rlogin, and ftp which permit the acquisition of sensitive data, such as usernames and passwords, whilst in transit. This clearly protects our users from many risks on foreign networks. The other side of this issue is that we must not automatically assume that our users are always totally safe just because

they are using an encrypted connection. There is still a major risk related to the security of the external machines from which connections are made, viruses which introduce keylogging software are probably the biggest issue.

6.1.5 Console Logs

All our servers have serial consoles attached which are accessed through site-specific central servers. This allows easy management when reboots, installs, etc., are required. The big benefit though in this case was the storage of console logs over an extended period of time. Typically when a system crashes due to a kernel panic some trace output will be sent to the console showing where the error occurred and the trigger process. Without the console servers the output would have only gone to the local screen (e.g. nowhere if nothing was attached). In this case it became evident that all of the panics, although initially looking different, were in fact being triggered by the attempted insertion of a kernel module with the `insmod` tool. Without the console logs the detection of the intrusion could potentially have taken much longer.

6.1.6 Process Accounting

By keeping process accounting logs we were able to rapidly detect the compromise once we were aware that some sort of problem was occurring. This is not currently the default on all of our servers. As it is very easy to enable and does not add unnecessary load to the machines we should enable it everywhere. It should be noted that as these logs are stored locally on the server they are not completely safe against attack but they do provide a useful secondary information source.

6.1.7 Monitoring Failed logins

It is not clear where the compromised users password was acquired but clearly we want to make it as difficult as possible for attackers to use our systems for this purpose. We have been running the *fail2ban* system for some time. This monitors system logs in real-time looking for failed logins and then uses tcpwrappers to block access to hosts which cause a lot of failures in a short period of time. This massively reduces the attractiveness of our servers to attackers whilst only rarely causing problems for genuine users. One slight shortcoming of the current monitoring is that it is only done on a per-host basis. An attacker who was determined to gain unauthorized access to a particular account could do so by using a wide-range of IP addresses (e.g. if they have a botnet at their disposal). We should consider augmenting the login failure monitoring to check for attacks on specific accounts (this can be done with the `pam_tally` PAM module). We could also consider aggregating the login failure data across all servers which are accessible by SSH by using a central database.

6.1.8 Installed Software

The way in which the installed software on all our machines is managed is excellent. All required software packages are installed as RPMs and the whole installation process is done using automated tools. The list of software packages and the versions are precisely specified, security updates are applied weekly once they have been successfully tested. There is the possibility that having gained *root* privileges an attacker could use the `rpm` or `yum` tools to install additional software. A benefit of the way in which we manage our installed software is that any additional packages, manually-installed using the `yum` or `rpm` tools, would be automatically removed.

6.1.9 Minimal Services Enabled

We only enable the minimal number of services required for a particular system. This includes not starting the *internet daemon* (xinetd) which is used to manage network connectivity for other software. The result of this is that we do not have any unnecessary daemons listening and we do not make it easy for attackers to enable new services. The greater the number of daemons running the greater the opportunity is for attackers to find software bugs which are exploitable. Many daemons run with root privileges for at least part of their process lifecycle (to acquire access to particular network ports, for example) so it is possible to use them to gain *root* privileges if there is a security flaw. Also, although we use a firewall to block incoming external network access on unnecessary ports we cannot assume that an attack will not come from an internal machine. Thus it always makes sense to disable all unneeded network daemons.

6.1.10 Auto-Reboots

All of our desktop machines run software which automatically schedules a reboot whenever one is required by a software upgrade (e.g. for a new version of the Linux kernel). Where machines are configured to sleep for periods of inactivity the machines will even wake-up to handle the reboot. This means that within a few days of an update being released onto our systems the majority are running that version without any human interaction. This avoids the potential for desktop machines to be accidentally left running an old version for too long. A number of our servers have also been configured to use this facility and we might want to consider if this could be extended further to any more of our servers.

6.2 What Did We Do Wrong?

There were a number of things which could have been done better that allowed the compromise to occur. A number of factors also contributed to the exploit continuing to run far longer than we would have liked. In an ideal world we would avoid ever falling victim to a *root* exploit but we are very aware that this is not achievable whilst still providing useful services for our users. We have to make it very difficult for an attack to be successful. Where a system break-in does occur we must ensure that the damage is limited and detection is quick and easy to avoid long-term exposure. This section details the main factors:

6.2.1 Handling of Previous Attack

The earliest date-stamps for the root-kit matches with a series of problems (including some kernel panics) we experienced with the SSH servers early in January 2011. Those problems were not thoroughly explained at the time. That previous attack had involved a root-compromise and the replacement of the SSH daemon with a trojanned version. In that instance the attack was unsuccessful as we have a locally patched version of the OpenSSH daemon which takes some non-standard options, this caused the trojanned daemon to fail. The affected machines were reinstalled after that break-in and we believed that the issue had been resolved because the kernel panics ceased occurring. It is now obvious that we should have carried out a more thorough investigation and tracked down the compromised user account which was the access route. We should also have done more to harden and monitor the SSH servers.

6.2.2 Logging

The initial discovery of the system compromise was based on the contents of the process accounting logs and entries in the main syslog. This clearly shows the importance of having several

layers of logging. The `last` log on its own would not have been sufficient so the addition of process accounting was hugely beneficial. However, we were lucky that the process accounting logs and main syslog were not modified or destroyed by the attacker. Although we had a recently deployed central log server available we were not yet using it to collect syslog entries from this server. All DICE machines which are running F13 and SL6 use the new *rsyslog* daemon and are configured by default to send copies of all system log entries to the central log server. This is clearly the right thing to do and we should consider extending that service by enabling the centralised logging using the old *syslog* daemon on SL5 for, at least, important servers.

We also were not storing the log files for long enough on these servers. We only had 1 month of history which was insufficient to fully analyse the extent of the problem. For such important systems we need to increase the storage to something like 6 months.

The `last` log is only stored on the local disk and is clearly a primary target for attackers. Similarly the process accounting logs are only stored on the local disk. It does not seem to be possible to easily transfer additions to this data directly to a central server in a "live" fashion which is essential for reliable problem detection. We will need a better method of logging and auditing user logins and activity to a separate server. One possible solution would be to use the Linux audit daemon [6].

6.2.3 Weak Passwords

As mentioned previously, the user with the compromised account revealed that they had a weak password. This showed up the fact that there was a flaw in our password policy which allowed users to set very weak passwords when using certain methods (e.g. the `kpasswd` tool). The standard password setting method on a DICE machine, which uses the `passwd` command, did not permit weak passwords. This is because the password changes on a DICE machine go through the Linux PAM stack and had to satisfy various *cracklib* checks (e.g. minimum length and number of character classes) before being accepted. Consequently this means that the majority of DICE accounts did not have weak passwords,

Furthermore this raised awareness of the fact that we had a lot of disused and dormant accounts which had not been fully deactivated and were still externally accessible.

6.2.4 Installed Software

Up until fairly recently we have deliberately chosen to make the software installations almost identical across all systems using a particular platform (e.g. SL5). This means that all servers have a large set of packages installed which are only really required for desktop installs. The main advantage of this approach is that management of the lists of packages is very simple and it is very easy to avoid problems such as package dependency conflicts.

By doing this, we have been effectively providing a large suite of useful tools for any attacker which makes the escalation of an attack easier than we would like. In particular, all servers have the entire GNU Compiler Collection (GCC) installed which makes it possible to easily build kernel modules without installing additional tools. The root-kit did contain a lot of simple tools (mostly based around *busybox*) but it did not include any compiler tools. Although it is not impossible for them to be installed by the attacker the size of the files would have made this much more awkward and liable to detection

The sheer volume of software installed on the servers also made it very easy for the attacker to hide "extra" files and difficult for administrators to scan once the intrusion had been discovered. A smaller install would be easily monitored, it could then be frequently checked for any obvious changes. Although most of the root-kit was "hidden" by the Linux kernel module it was necessary for the attacker to keep some files unhidden. This was necessary to enable easy

escalation of privileges from the compromised user account and also to start the various tools at boot-time.

Further to this, installing a lot of unnecessary software results in a lot of unnecessary programs which are setuid *root* and executable by normal users. Whilst the most important setuid scripts in a system are typically well written and secure (e.g. the `passwd` command) that may not be the case for less well-used software. These clearly provide a "soft" target for any attacker and make privilege escalation much easier.

One particularly glaring issue that the compromise of these systems has raised is that the version of the Linux kernel which the servers were running was too old. This was not due to any failure to provide an updated version of the SL5 kernel package, an update had been available for some time. It was felt that the aim wherever possible should be to minimise the downtime of important servers in favour of avoiding disruption to user sessions. It is clear that we need to revisit the way we trade-off risk of system compromise against availability.

To totally avoid the issue of what software we have installed we could completely change the way in which the SSH servers function. Currently we allow full shell logins for our users so that they can carry out basic activities, such as reading email, on the SSH servers. Clearly this is very convenient for our users but could be opening us up to unnecessary security risks. As an alternative we could run a "bastion" SSH service which, once access has been authorised, restricts users to only being able to login to another internal machine.

7 Proposals for Improvements

It is useful to distil the previous discussions into a set of distinct proposals. Some of these proposals are for small, very specific, tasks which need to be done immediately, others will require more effort and potentially a great deal of discussion. The existence of an item in this list of proposals does not automatically mean that it is guaranteed to be done but it will at least be given some consideration. Those which have already been completed since the system compromise was detected have been marked as such. The others are either ongoing or will form parts of new development projects, in particular many will be in DevProj #224 (System Security Enhancements).

There are a number of guides available (e.g. [16], [17] and [18]) which contain very useful advice on how to improve the security of Linux, and in particular RHEL, systems. These have been consulted and used to develop this set of proposals.

1. Require all users to change their passwords:

There is evidence that one of the Linux kernel modules which was inserted was able to log all session activity and collect sensitive data such as usernames and passwords. The root-kit also had the capability to send these to an external host. Although we cannot be sure this actually occurred we have to assume the worst scenario and we must force everyone to change their DICE passwords. All DICE accounts which have not changed their password after a cut-off date must be disabled. To avoid the attacker resetting passwords at a later date the disabled accounts will only be re-enabled when the user provides extra identification information. **[DONE]**

2. Improve password strength checking:

Whilst reviewing the compromised user account we discovered that they had a weak password. This revealed a flaw in our password strength checking which permitted a small number of users to set bad passwords. As the passwords are stored in an encrypted

form we do not know which users have bad passwords. Consequently, prior to requiring all users to change their passwords we must improve our password strength policy. In the short term this can only be done by setting a minimum length and minimum number of character classes. We plan to revisit the DICE password policy next year, this could, for example, give users the option to use fewer character classes in longer passwords. We must also add documentation which clearly states the DICE policy. **[DONE]**

3. **Promptly handle old accounts:**

   We currently have a lot of old DICE accounts which are no longer required. These present a particular security risk as no-one would spot any unusual activity. We should aim to disable them much more quickly (at least block the possibility of a login) and have them deleted as soon as possible.

4. **Disable dormant accounts:**

   There are likely to be many DICE accounts which although still associated with someone who is entitled to an account are rarely used, if at all. Similarly to the old accounts described above these present an increased risk to our system compared with active users. We should regularly identify and disable those accounts which have not been used for an extended period of time. Re-enabling a dormant account would require the user to provide additional identification information.

5. **Block SSH public key access:**

   The primary SSH servers have always allowed users to gain access using SSH public key authentication. Due to our use of Kerberos and AFS this provides little benefit and has the potential to cause confusion amongst our users. Also, if private keys are not well protected there is a risk of exposure to attackers. Taking this all into account it seems reasonable to block SSH public key authentication. We will still permit it internally between DICE machines where it can be useful (e.g. for automated backup scripts). **[DONE]**

6. **Develop a risk profile for servers:**

   We have traditionally had a habit of treating most DICE servers in the same way, regardless of their role. We should look at all our servers and identify their risk profile. This would be evaluated based on the level of exposure (e.g. firewall holes, number of users with access) and the potential value of the system to any attacker (e.g. the KDCs contain all the passwords). Systems should be classified as low, medium or high risk. For those which are determined to be high risk we should look at ways in which the security measures could be improved. Obviously some hardening may be too expensive in terms of effort so we will need to evaluate the cost-benefit ratio.

7. **Thoroughly investigate all issues:**

   We should always thoroughly investigate issues which may be related to a system compromise (e.g. any kernel panic should be fully understood). We should never assume that a kernel panic is just "odd behaviour" that is safe to ignore, particularly on high risk machines. This means that we need to develop some forensic investigation procedures for handling suspected compromised machines.

8. Enhance logging:

We only had one month of syslog and auth logs and they were stored on local disk of the compromised servers. This is not sufficient for examining longer term user and system behaviour. We should keep 6 months worth of important logs for the SSH servers. We are lucky that we did not have the contents of the various log files completely wiped by the attacker, the "last" log was edited for instance. We should investigate ways of securely transferring this valuable information to an external server. In particular, the centralised log host facility should be extended to all SL5 machines. [**DONE**]

9. Apply updates in a more timely manner:

This episode has demonstrated that on systems where there is a high risk of user accounts being compromised we need to treat all local exploits as seriously as remote exploits. The SSH servers ran with an old kernel for a long time, whenever a new kernel is available for these systems we should aim for them to be rebooted within a few days. Similarly, they were some of the very last machines to receive the SL5.6 update. We might also want to consider reinstalling these machines at each minor update to ensure consistency. For updates which require a reboot we have an "autoreboot" system. This is used for desktop machines and can also be used for the majority of servers, consideration should be given as to whether this can be used even more widely. [**DONE**]

10. Reduce the installed set of packages:

It is clear that the attacker used the compilation tools installed on the system to build Linux kernel modules. These tools along with many others are not required for normal usage of the SSH servers and should be removed. [**DONE**]

11. Block the loading of kernel modules after boot-time:

It is clear that this root-kit inserted a Linux kernel module. There is no reason on most of our servers to allow module loading after boot-time so it should be disabled wherever possible. Note that it is still theoretically possible to compromise a Linux kernel via the `/dev/mem` device file but that is significantly more difficult compared to loading a module. [**DONE**]

12. Regularly check kernel versions:

We have a lot of DICE machines so it is very easy for an administrator to miss an upgrade. We should have an automated system which regularly monitors the versions of the Linux kernel in use on the DICE machines. Any machines that are particularly out-of-date can then be easily identified. [**DONE**]

13. Regularly monitor the file-system:

It is clear that we need to monitor the file-system of our servers for unauthorized changes. In particular we should monitor:

1. All setuid *root* scripts which are runnable by other users.
2. All files which are not owned by RPMs

3. Important directories such as `/etc/`

This can be done with intrusion-detection software such as AIDE (Advanced Intrusion Detection Environment) [13] which is a file and directory integrity checker.

14. Regularly sweep for root-kits:

We should regularly run software to check for the evidence of root-kits having been installed, e.g. *chkrootkit* [11] and *rkhunter* [12]. There is a fairly high possibility of false-positives and some issues may not be detected by a particular detection strategy. Consequently, as these tools function quite differently, running more than one check would definitely be useful.

15. Improve file-system partitioning:

As an attacker is reliant on setuid *root* scripts to elevate privileges we should restrict where these can be used within the file-system. There is no typically need for this type of script beyond the standard binary directories (e.g. `/usr/bin`, `/usr/sbin`, `/bin`, `/sbin`). In particular the `/var` directory contains a lot of regularly changing data files within which it would be very easy to hide malicious files from a system administrator. On external-facing machines the `/var` and `/tmp` directories should be on separate partitions which are mounted with the "nosuid" option. Additionally, these partitions can be mounted as "noexec" and "nodev" for extra security. **[DONE]**

16. Increase security of additional SSH access routes:

Beyond the primary SSH servers we create firewall holes to allow some users, who have specific academic requirements, to be able to login directly to their machines. We should also enhance the security on those machines and increase the amount of logging and auditing which is done. Alongside these security improvements, we should work to discourage the use of such SSH firewall holes in all but the most essential cases. There are often more secure ways of achieving the same results which are also better for the user once the access requirements are fully understood.

17. Improve the management of the access.conf file:

The current support in the LCFG auth component for managing the access.conf file is insufficient. For instance, it is not possible to block access to individual users whilst still allowing all others. We would also like to be able to have more fine-grained control over the permitted origins of logins from different sets of users.

18. Consider profiling user behaviour:

We could develop a system that automatically identifies the profiles of "normal" login behaviour for our users. It would then be able to flag-up any potentially abnormal activity. This could be rather difficult and require considerable work, particularly in producing a sufficiently capable system which doesn't overload the Computing Team with false-positives. It would also require extensive discussion and acceptance from the School.

19. Use SELinux:

We do not currently use the SELinux framework which would almost certainly have prevented this intrusion occurring. To run it in full "strict" mode might require a great deal of effort to make it work with LCFG. However, it should be possible to get a lot of the benefits whilst using the lesser "targetted" mode, which is the standard for Redhat systems. It is definitely worth some further investigation.

20. Shared database for failed logins:

We should investigate whether it is possible to create a database of failed logins which is shared across all DICE machines with external SSH access. This could be used to automatically block all SSH access for hosts which are attacking our systems.

21. SSH Honeypot:

We could use an SSH honeypot which has an SSH firewall hole but which cannot actually be accessed by any users due to restrictions in the SSH daemon config and the `access.conf` file. The location of this service would not be advertised to our users so it would only be found by attackers who are port-scanning our network for targets. This could then feed into a shared database of banned hosts.

22. User Education:

We need to work much harder to educate our users about the risk they pose to our systems. It is clear that the main route into our systems for attackers is our user accounts. The way in which our users behave can do a lot to protect our infrastructure from further intrusions. In particular we should:

1. Raise awareness of the locations from where it is (or is not) safe to login to our systems.
2. Strongly promote the running of anti-virus software
3. Document how to check home Linux machines for trojans
4. Document how to install and configure Kerberos for access from home

23. Computing Team Knowledge:

It is clear that we did not previously have a thorough understanding of the risks posed to our systems. We should do more to keep up-to-date with current attack methods and exploits being used, for example by monitoring the ISC and US-CERT websites and reading relevant computing security magazines such as "*Hack in the Box*" [19]. We also need to develop a greater knowledge of how to harden our systems and defend against attacks. Furthermore, it is inevitable that there will be further successful intrusions into our infrastructure. Consequently we must do more to develop operational strategies to deal with these intrusions when they occur.

24. Increase Internal Traffic Monitoring:

We could do more to monitor our internal traffic flow. This would allow us to spot compromised machines which are generating unusual network traffic. See DevProj #98 (Investigate sFlow), #102 (Intrusion Detection System), #103 (Scanning for compromised machines) and #133 (Perimeter filter logging) for details.

8 Conclusions

The results of the investigation and the discussion show that in most ways the SSH service was well configured and well managed. Several issues which require improvement have been identified but none of these were directly responsible for the root exploit. The most important aspect requiring substantial improvement is clearly that of monitoring so that we can quickly identify any intrusion. Some work has already been completed to enhance the logging and further work will be started soon.

In the past the focus of most attacks was systems based on Microsoft Windows. The perception was that Linux-based systems were generally secure enough to avoid intrusion. It may well be that this lack of interest from attackers gave Linux system administrators a false sense of security. It is clear that the threat of attack against Linux-based systems is now far greater than it was in the past. This has been seen in the attacks against the Linux kernel.org infrastructure [7], and many Linux distributors such as Debian and Redhat. For example, only quite recently the Fedora project has forced all their developers to change passwords and SSH keys [8] in response to "the large number of high profile sites with security breaches in recent months". In response to this evidence we have to assume that our systems are now continously under threat from persistent non-targetted attacks (e.g. port scanning for weakly protected services and attempting to find users with weak passwords).

The main weak point in our systems is clearly that they are used by real people. If we could eradicate all users from a system it would be very secure and almost entirely pointless. Instead we must focus on educating our users so that they have a much better awareness of the risk they present to the entire computing infrastructure of the School of Informatics. We also need to do more to help our users properly manage the security of their personal machines and make appropriate decisions about locations from which it is safe to login. A single weak password or a keylogger on an external machine can clearly have a large impact on our systems.

There are a number of issues which remain unclear and probably are not resolvable. In particular, although we know which version of the kernel was later targetted, we do not know what security hole was used to initially acquire *root* privileges. We also have not found any source code for the kernel modules or evidence as to how rebuilds were triggered when necessary. The response to removal of the modules from the kernel (for instance, when we upgraded the kernel) was always rapid but we don't know if this was fully automated or whether regular manual attention was been given to the root-kit. Knowledge of the level of manual attention required could make it clear if this was just a random automated drive-by attack or whether it was more targetted.

The issues identified during the investigation and subsequent review have been gathered together as a set of proposals. As part of the initial response phase many of these have already been completed. The rest are either ongoing or will form parts of various projects which are expected to start soon. Although we were slow to identify that the systems were compromised, since the point of discovery the response has been swift and assured. We have learnt a number of valuable lessons from this episode and there is no doubt that the security of the DICE infrastructure as a whole will be increased thanks to the work being done in response to this compromise.

Acknowledgements

I would like to acknowledge all the valuable contributions made to this report by my colleagues in the Computing Team with the School of Informatics. The response to this system compromise has been very much a team effort. I am also grateful for the very helpful feedback on this report provided by a number of external peer reviewers.

References

[1] *Scientific Linux* - http://www.scientificlinux.org/

[2] *Redhat Enterprise Linux* - http://www.redhat.com/rhel/

[3] *LCFG: Large-Scale Unix Configuration System* - http://www.lcfg.org/

[4] *LCFG: The Next Generation* - Paul Anderson and Alastair Scobie, http://www.lcfg.org/doc/ukuug2002.pdf

[5] *Redhat Bug Tracker* - http://bugzilla.redhat.com/

[6] *Linux Audit* - http://people.redhat.com/sgrubb/audit/

[7] *CERT: Linux servers under 'Phalanx' attack* - The Register, 27th August 2011, http://www.theregister.co.uk/2011/08/31/linux_kernel_security_breach/

[8] *Fedora: Mandatory password and ssh key change* http://lists.fedoraproject.org/pipermail/announce/2011-October/003005.html

[9] *Active attacks using stolen SSH keys* - Internet Storm Center Diary, 26th August 2008, http://isc.sans.edu/diary.html?storyid=4937

[10] *SSH Key-based Attacks* - US-CERT, 26th August 2008, http://www.us-cert.gov/current/archive/2008/08/27/archive.html#ssh_key_based_attacks

[11] *chkrootkit* - http://www.chkrootkit.org/

[12] *Rootkit Hunter* - http://www.rootkit.nl/projects/rootkit_hunter.html

[13] *AIDE* - http://aide.sourceforge.net/

[14] *The Hacker's Choice* - http://thc.org/

[15] *vlogger* - http://thc.org/releases.php?q=vlogger

[16] *Hardening RHEL5* - http://people.redhat.com/sgrubb/files/hardening-rhel5.pdf

[17] *Guide to the Secure Configuration of Red Hat Enterprise Linux 5* - National Security Agency, February 28th 2011, http://www.nsa.gov/ia/_files/os/redhat/rhel5-guide-i731.pdf

[18] *Native Host Intrusion Detection with RHEL6 and the Audit Subsystem* - http://people.redhat.com/sgrubb/audit/audit_ids_2011.pdf

[19] *Hack in the Box* http://magazine.hackinthebox.org/

Useful Software

- AIDE - http://aide.sourceforge.net/
- Linux Audit - http://people.redhat.com/sgrubb/audit/
- chkrootkit - http://www.chkrootkit.org/
- DenyHosts - http://denyhosts.sourceforge.net/
- fail2ban - http://www.fail2ban.org/wiki/index.php/Main_Page
- OSSEC - http://en.wikipedia.org/wiki/OSSEC
- rkhunter - http://www.rootkit.nl/projects/rootkit_hunter.html
- SSHGuard - http://www.sshguard.net
- Samhain - http://www.la-samhna.de/samhain/