Lexis EXam Invigilation System

Mike Wyer and Susan Eisenbach
Department of Computing
Imperial College
London, Great Britain, SW7 2BZ
email: mw@doc.ic.ac.uk

24th September 2001

Abstract

Over recent years, computers have been making their way into the classroom and lecture theatre. Overhead projectors, blackboards, and whiteboards have been joined or even displaced by smartboards and computer based multimedia presentations. Students with laptops are a common sight, and courses have their lecture notes on the web. Students are taking courses in programming, web-site design, computer graphics, and many other digital disciplines. Yet these courses are still being assessed with traditional pen and paper examinations.

The reasons for this are several including inertia, worries about security, and a lack of available tools to administer high-security exams with confidence.

At the request of the Academic Committee of the Imperial College Computing Department, we were asked to develop such a tool so that our students could sit an official examination using the Linux workstations in our teaching laboratories[?].

We embarked on a project to develop a system to provide necessary resources to candidates, prevent cheating, and securely retrieve and store the work done by the candidates during the exam. Here we describe the technologies, problems, and solutions encountered during the development of software to support exam invigilation.

The result of this work is the Lexis EXam Invigilation System, which was used to administer a first year programming exam on 21st March 2001 which comprised 110 students with access to 160 Linux workstations and lasted for three hours. At the end of which, the labs were restored to general access use.

According to the BBC, on the 2nd of April 2001, students sat the first paperless exam in the UK in a pilot scheme in Northern Ireland[?]. In fact, we beat them to it by several weeks.

1 Introduction

People learn to program by sitting in front of a machine and typing. However, formal programming examinations are usually hand written on paper. So the skill being tested is not the same as the one being learned.

At Imperial College, we have had years of experience in running low-priority, low-security programming tests on the standard lab systems. These tests consist of a few simple programming questions, with the students expected to code their answers within the allotted time, submitting via an automated email-based system. Given the small amount of credit available and suitable vigilance on the part of the test coordinator, it was felt that these tests did not warrant additional security measures on the workstations.

Students and staff preferred the computer based tests to traditional written papers— the students felt much more comfortable programming in an editor with the chance to run their code, and the staff were able to compile and run the submitted code directly,

which reduced the burden of checking syntax and correct solutions of the given problem by hand. In addition, the perennial problem of reading handwriting was removed.

Our regulations are such that one of the necessary conditions for passing the programming course is that a student must pass the final examination. With the popularity of the programming tests, we were asked to investigate the feasibility of running examinations securely on the lab systems. We were given the task of configuring our lab machines in such a way that students could safely take an official University of London examination on them.

At the time, our computing labs consisted of over 200 PCs, ranging from 233 MHz Pentiums to 800 MHz Athlons, all running RedHat Linux[?].

2 Requirements

Although most people are familiar with the security arrangements which accompany an official examination, they are not often encountered in a systems administration context.

These requirements are taken from the specification document discussed and agreed by the Academic Committee in the Computing Department of Imperial College.

2.1 Aims

Provide:

- Familiar lab-like environment during exams
- All resources necessary to complete exam
- Secure environment for completing exam
- Secure means of collecting exam work

Ensure:

- No access to unauthorised data
- No access to other users on network
- No distraction or interference from other users on network

2.2 Further details

Some exams may involve providing students with templates, stub code fragments, or other data. Likewise, the student will be required to create or modify files as part of the exam. The details of files to be provided and collected must be discussed in detail and in advance with the exam administrator. The students will not have access to shared network volumes, so any files needed for the exam will need to be provided by the examination system. Standard applications will be available.

Each completed exam submission must be securely stored and associated with the right candidate number. Exam submissions must not be accessible to anyone except the authorised agents of the University.

As with any other examination, students will only be allowed access to permitted resources. In addition to the usual physical precautions of a written exam (no books, paper, phones, radios, tattoos, etc), the student should not have access to unauthorised stored data or communication systems on the workstation. The additional methods are:

1. Data previously stored on hard disk in a writable area
2. Data on removable media (floppy or zip disk)
3. Data on network device (home directory, bitbucket)
4. Communication via network

It is also important to make sure that other users on the network do not interfere with the student during the exam, the on-line equivalent of the noisy mob in the corridor.

3 Investigation

Development time was limited, so it was important to investigate currently available solutions. Several commercial products exist, for example WebCT[?] and Blackboard[?], but they are windows based and only offer support for traditional style exams. Indeed,

a paper by Braun and Crable[?] strongly suggests in-house development as an alternative to the existing tools.

Although no existing package provided all the facilities we needed, there was a good chance that some of the individual tasks could be covered by one of the many security tools, packages, and utilities available for Linux[?]. The project to build a system to help administer examinations was dubbed the Lexis EXam Invigilation System.

3.1 Network Access

A way of severely restricting the network was needed, and the most obvious and effective method would be to simply disconnect the network during any exam. Our network topology and hardware are such that this is a fairly straightforward option. The target machines would then be required to function correctly without any network. This raised several concerns about reliability, synchronisation, and monitoring.

What would happen if a machine had a fatal problem during the exam, say a hard disk head crash? How long would it take to recover any data, if it was possible at all? These issues encouraged us to look at other solutions to the problem. Leaving the network connected also introduces problems. There were still reliability issues, cheating might be easier and the whole exam could be open to external attack.

It was vital that the worst-case failure of any of the constituent systems would not invalidate the exam. In order for Lexis to be a success, the safety, security, and reliability of pen and paper had to be matched.

We investigated Linux kernel level firewalling as an alternative to complete network disconnection. Linux 2.2 was the stable kernel at the time, so the ipchains interface was evaluated[?]. The evaluation proved to be very positive.

Using ipchains would give us precise control over the network traffic to and from each workstation involved in the exam. While this is not a novel idea to anyone who has been using ipchains, the key factor is that using ipchains provides an easy way to achieve *temporary* network security while still allowing certain connections. The "certain connections" we had in mind were specifically OpenSSH[?] connections to a central server.

Most firewalling schemes are permanent; the rules are designed to be in place for perpetuity. With Lexis, the rules are in place for a few hours. Not only do the rules have to be automatically applied, they have to be removed as well. While the techniques involved are straightforward, the implementation must be absolutely reliable.

3.2 System Security

We needed a strategy to prevent cheating- access to unauthorised data, tools, or other users.

Many UNIX systems use the **chroot** system call to restrict processes to a limited "sandbox" environment. This works very well for daemons which have a specific function and whose resource requirements (libraries, device files) are known in advance. In order to provide a similar setup for an exam, we would be forced to replicate a large percentage of the existing filesystem so that candidates would have access to the X Window System, window managers, all the editors and compilers needed for the exam, and so on.

Not only would all this file copying take a long time, it would take up more disk space than was available, and it would be very hard to show that security had actually improved.

A similar strategy would be to dedicate a partition on the disk to Lexis, and dual-boot to specially configured OS and filesystem. As before disk space would be limited, and this approach has other drawbacks: we would have to provide compatible versions of the programming languages needed for the exam, along with having to provide a file transfer, security, and monitoring system. Although the security aspect would be simpler, we would still have to manage installation of up to three operating systems on the machine (Linux, Windows, and LexisOS, whatever that turned out to be).

We also considered creating a root filesystem image on the network which all the clients could mount, but this brought several more problems: using NFS (version 2) is not a good way to increase security, and where would the candidate's files be stored? If

we wanted to use the local disk, we would still be stuck with the problem of sanitising the filesystem and preventing use of data or programs stored on that disk.

It seemed that no matter which approach we took we would need to come up with a simple, practical, and general way to secure Linux in a systematic fashion. And if we could do that, then why not just run the exam from our newly-secured Linux environment which already had all the tools and configuration necessary to run lab software?

We started to analyse the types of activity that would be considered "cheating". It turns out that many of the activities which constitute illegal behaviour by students are privileged operations on the system. Operations such as mounting disks and creating trusted network sockets require either root access or set-uid root file permissions. By remounting the root filesystem without set-uid bits active, we eliminate the danger from setuid binaries. This cuts the risk from existing exploits of setuid code, and provides protection from trojans (eg a suid shell installed before the exam).

A useful side effect of this operation is that some system binaries that are installed setuid root (notably *man* and *ssh*) are also disabled. This would prevent the student logged into the machine from using the OpenSSH client to attack the only open network channel (the ssh link to the Lexis server).

3.3 Reliability

One of the key concerns from academics involved in the development of Lexis was that of reliability: what would happen if a PC crashed during the exam? While we could think of many analogous situations for a traditional paper-based exam which would be equally catastrophic, we wanted to show that a PC-based solution could improve upon the security and reliability of pen and paper.

To provide some protection from hardware failure, we decided that all client machines would dump the exam files to a central server on a regular basis. This would provide flexibility to cope with any situation that might arise— if the candidate accidentally deleted important files, we would be able to restore them (at the request of the examinations officer); if a candidate disagreed with the marking of the exam and claimed that Lexis was responsible, we would be able to provide a detailed history of that candidate's work during the exam; if a machine failed, we would be able to restore the last dump to a different machine and let the candidate continue with minimal disruption. The main aim was to be able to support any decision made by the examinations officer.

We discovered a neat solution to the problem of maintaining security through reboots, which also provided a way to control other aspects of Lexis: we made use of runlevel 4.

3.4 Runlevel 4

Runlevels are a standard feature of SysV-style init. Runlevels 0, 1, and 6 are reserved, and levels 2, 3, and 5 have (thanks to LSB[?]) fairly standard definitions across distributions. Runlevel 4 is available for use on many linux systems. By using a runlevel specifically for lexis, we can use **init**[?] to handle transitions into and out of exam state, as well as booting securely when an exam is in progress.

To start an exam, we create a new set of config files for the system, then change to runlevel 4. On changing runlevel, **init** stops services from the last runlevel and starts services for the new runlevel. We create a lexis service that only runs in runlevel 4 that carries out any changes that need to be done on starting an exam or booting during an exam, including signalling the server that the workstation is ready for use and turning on the firewall rules.

This approach should remove a lot of the system management burden from Lexis and place it onto standard system processes. Unfortunately, the **init** supplied with RedHat 6.2 proved extremely unreliable during initial testing, often changing runlevel without running stop or start scripts. This meant that a large amount of the functionality of init (stopping services, restarting them) would have to be replicated in Lexis to ensure reliability (we have discovered to our cost that it is much easier for us to reimplement rather than trying to get Red Hat Software Inc[?] to fix their product or accept patches from us).

3.5 Exam files

To make the dumping easier, we decided to restrict the candidates to a specific area of the disk. */exam* would be used to contain all the exam files and be the working area of the candidate. We only expect one candidate to use each machine, but any candidate could conceivably sit at any machine. We settled on the idea of a common home directory since this would mean we would only have to create the files needed for the exam once, and we would create special lexis accounts that would only be valid for the duration of the exam. All the lexis accounts would be in a 'lexis' group which would have access to */exam*.

Special lexis accounts would be necessary for several reasons:

- Our site uses Kerberos[**?**], which relies on network access for authentication, so candidates would not be able to log in during the exam
- According to University Examination regulations, candidates must only be identified by a candidate number. Using normal logins would compromise the candidates' confidentiality.

On our systems, this would necessitate disabling kerberos access, and providing new local lexis accounts with appropriate passwords. Since physical access to the machines would be controlled by the usual exam invigilators, and we would need some way of associating candidate number with submitted files, we decided to make the username and password the candidate number. This would provide a double check at login that the candidate was using the right candidate number, and that all files owned by the candidate would be tagged with their candidate number.

4 Design

We decided on a client/server architecture, where the workstations which the candidates will use are the clients, and a central machine which monitors the exam and stores submitted answers from the candidates is the server. The overall structure of a lexis session is summarised in Figure **??**, showing how each client is individually firewalled to the server, and the points at which various illegal activities are stopped.

Figure 1: Lexis Architecture

The lexis protocol is very simple. All communication is in ASCII over an openssh link. All commands consist of a single line (terminated with a single newline). When the client is invoked by the server, the server sends its version number. If the client version matches, the client returns 'ok'. If the client and server versions do not match, or the client is not being run as the root user, an error is returned instead. For all subsequent commands, the client will return 'ok' if the call succeeds (after any expected output) or an error message if it fails. All error messages include the hostname of the client.

File transfer is accomplished using base64 encoding to make binary data safe to send over the ASCII link and MD5 checksumming to ensure data integrity. This ensures the clients get the files they are supposed to from the server, and to make sure that the server receives valid dumps from the client.

4.1 Lexis client

The main goal with the client was to keep it safe and simple. The client files would have to be distributed to the clients ahead of time, and it would be extremely difficult (or even impossible) to make changes to the client during an exam. So the client software would have to provide the capability to cope with any situation that might occur during an exam.

We made the decision to use an OpenSSH2[**?**] to connect the server to the client. This would provide a simple STDIN/STDOUT communications channel between the server and client, as well as the means to get full remote shell access on the client from the server, to fix any problems remotely.

There would be just one program that communicated with the server (with others to accomplish specifc tasks as necessary), and it would receive commands from the server and respond to them. At no point should the client be sending unsolicited data to the server. This meant that there would be no

Figure 2: Main Lexis Components

need to compromise the server by trying to enable the server to trust the clients.

4.2 Lexis server

With the server, we wanted a straightforward system to manage connections with the clients, send and receive files, and respond to commands from the administrator. Since the clients would have limited functionality, most of the data processing would be done on the server, such as working out who had logged into which machine.

5 Software

Lexis uses a client-server approach, where the individual workstations are clients, and there are one or more central servers which communicate with the clients. The client software consists of three programs: **lexis_startup**, which is called by **init**[?] when switching to runlevel 4 (either at the start of an exam or on booting during an exam); **lexis_active** which is called by **sshd**[?] when a connection is made by the server; and **lexis_warning**, which is a simple X program that warns existing users that an exam is about to start. The lexis session is managed on the server by a single process, **lexis_server**. The dump files stored on the server can be queried using the **lexis_who** and **lexis_extract** scripts.

The interactions between the main scripts—**lexis_server**, **lexis_active** and **lexis_startup**—are shown in Figure **??**.

5.1 Components

5.1.1 lexis_active

Most of the client-side code is in **lexis_active**, such as the file transfer mechanism, authentication setup, **ipchains** configuration, and runlevel control. It is a straightforward perl[?] script which reads commands on stdin and produces output on stdout, and contains just over 400 lines of real code (stripping comments and whitespace). It is designed to be invoked as a root process at the remote end of an ssh connection, and will abort if the calling uid is not zero. The commands are summarised in Figure **??**.

5.1.2 lexis_startup

One-off operations at the start and end of the exam are performed by lexis_startup, which is a SysV-style initscript. It is called by init when changing to runlevel 4 or when booting in runlevel 4. In either case, lexis_startup remounts the root filesystem with SUID bits turned off, clears tmp directories, shuts down non-lexis services, redirects any remote syslogging to a local file (we don't want the system to lock up trying to contact a host its own firewall rules are blocking), opens a connection to the lexis server, and updates the X display manager (gdm or kdm).

On changing out of runlevel 4, the root filesystem is re-mounted with suid bits set, remote syslogging re-enabled, and the X setup restored.

The display manager update is very simple, but necessary: we install a new logo to make it obvious the machine is ready for taking an exam, and restart X since it's /tmp/ lock-file has been removed, and it will automatically log out any existing users. The logo we use (Figure **??**) is an adaptation of the classic Linux mascot, Tux, and shows him behind bars. The writing on his chest is "IC Outside", a logo we apply to the systems we build in Imperial College Computing Department.

There is scope for more paranoia in lexis_startup. The original idea was to recurse through the entire directory structure looking for world- or group-writable directories and clearing them. This strategy proved unworkable when we found that a number of standard tools (xemacs for example) use writable directories for storing site packages, or similarly updateable files. While individual cases (like xemacs) can be fixed on a site-wide basis, it would be incredibly dangerous to include code to remove or hide such directories automatically.

For the time being, we make the assumption that */tmp* and */var/tmp* are the only world-writable local directories. If lexis starts being used at a large

init	Clear */exam* and make the machine ready for use in an exam.
add server	Add the given IP address to the firewall rules and the list of hosts to contact when booting.
delete server	Remove the give IP address from firewall rules and the list of hosts to contact when booting.
port	Connect to the given port on the servers when booting.
rootpw	Set the root password for the current session.
user	Add the given username as a candidate.
users	Add the given list of whitespace separated usernames as candidates.
file	Transfer the given file to the client. If the filename is a relative path, transfer to */exam*, otherwise treat as an absolute path. Unpack gzipped tar files.
gen_passwd	Use current user list and root password to generate new */etc/passwd* and */etc/shadow* files. Install new PAM configuration files. Keep a backup of original configuration.
restore_passwd	Restore original */etc/passwd*, */etc/shadow*, and PAM files.
warn	Run **lexis_warning** for the given number of seconds.
kill	Kill all processes with uid > 100. Unmount any network filesystems.
start	Write out firewall configuration. Write out server and port settings. Change to runlevel 4. Exit.
dump	Return a gzipped tar file of */exam*.
ok	Return "ok".
quit	Restore original configuration. Clear */exam*. Change to runlevel 5. Exit.

Figure 3: lexis_active commands

Figure 4: Lexis Logo

Figure 5: DTD for lexis_server config file

number of sites, then more advanced techniques will become necessary.

The current lexis_startup is implemented in about 100 lines of perl.

5.1.3 lexis_warning

To warn any existing users that an exam is about to start we use lexis_warning, which is a simple Perl-Tk script that connects to the local X server. It turns the root window to a given colour (red by default) and pops up a small window containing a warning about the impending exam. The popup beeps in an irritating fashion every second until the current user acknowledges it. It is mainly intended for use when a lexis session is scheduled during a normal lab period— it's not necessary when the rooms have been cleared and checked for a full official examination.

5.1.4 lexis_server

The lexis server process is the heart of the Lexis system. It deals with data from a number of sources: there is a main config file, a network port for listening for new lexis clients, the connections to lexis clients, and also interactive input from the operator. The system is designed to enable one operator to manage many lexis clients at the same time from the same server process.

The server is configured using XML. The DTD is shown in Figure **??**, and Figure **??** shows an example config file.

The main config tag contains attributes describing where to store dump files and how often they should be taken, which port to listen on for booting lexis clients.

Figure 6: Config file for lexis_server

While the config file defines "start" and "stop" times, they are for information only, as Lexis does not yet start and end exams automatically. It is technically feasible to trigger these events, but development time was tight, and the staff in charge of the exam wanted to retain control over the start and end time of the exam in case of special circumstances.

Implementing auto start and finish would entail putting more critical code onto the client, which is something we wanted to avoid while the system develops. Also, the overhead of 200 machines all trying to dump to the server at precisely the same moment could cause problems on the server, and we didn't want to risk losing any candidate's work.

The rest of the config file contains a list of files to transfer to the clients, the list of candidate names, and a description of the hostnames of the client machines. The clients machines can be specified individually by name, or using a shortcut for ranges of machines. The example file would add the following machines as clients: lab25, dynamic01, dynamic02, ..., dynamic28

Multiple server processes can communicate with the same client machine; each connection will spawn its own **lexis_active** process. We have used this technique with a modified lexis_server to create a separate dumping process in case of any problems or long-running jobs on the main server process.

Required perl modules: Term::ReadKey, FileHandle, File::Copy, DirHandle, MIME::Base64, MD5, IPC::Open2, IO::Socket, IO::Select, Net::DNS, XML::Simple

5.1.5 lexis_who

In order to find out which candidates had logged into which machines, we developed **lexis_who**, which is a simple perl script that queries the dump files stored on the server. It uses the files created on login to determine the user of the machine, for example *.xsession-errors*.

5.1.6 lexis_extract

Once the exam was over, we needed a way to extract specific files from the dumps, so that the answers to the various questions could be sent to the right marker. We wrote **lexis_extract** to achieve this, and to provide a framework for any other processing Lexis users might want to perform on the dumps. There are perl and ruby[**?**] versions of **lexis_extract**, with different default tasks. The ruby version is much more powerful than the perl version, and at 120 lines is twice as long.

5.2 Installation and minimum requirements

The minimum requirements for the Lexis client code are OpenSSH 2, Perl (with MD5 and MIME::Base64 modules), ipchains, and SysV style init. The processing requirements on the client are minimal; Lexis is designed to keep out of the way of the candidate as much as possible, so the greatest load on the system is likely to be any compilers the candidate is using. The lexis client code is written in Perl, so it is possible for sites to customise the code to their specific requirements. Likewise, if other Operating Systems provide firewall rules in a similar way to **ipchains**, then Lexis can be ported to that OS (especially other UNIX variants). Lexis is not designed for Windows systems.

The lexis client install consists of **lexis_active** and **lexis_warning** in */usr/local/bin*, **lexis_startup** installed as a runlevel 4 startup script (and all other services removed from runlevel 4), a 'lexis' system group for ownership of */exam,* and finally all clients will need the SSH2 public key the server will be using to contact them.

The use of **lexis_warning** is optional, and can either be omitted, or replaced with a suitable equivalent for the site in question. If you choose to use **lexis_warning,** the perl Tk module will also be needed. The lexis client code can be easily made into an RPM or other package format. In which case, some additional security can be obtained by changing **lexis_server** to run 'rpm -V lexis-client && /usr/local/bin/lexis_active' on the remote client machine.

The requirements for the lexis server are somewhat stricter. The current **lexis_server** maintains a constant OpenSSH 2 connection for every client machine,

there is also the overhead of MD5 and base64 on all client dumps, along with any processing of the dump files that needs to be done during the exam. We used an Athlon 800 with 512MB of RAM to manage an exam with 160 client machines, but the machine was running very low on resources (we had to increase the file-max limit several times at the start of the exam to enable all the connections to succeed).

The main limit is one of time— the server was originally written as a single thread, so as the number of client machines increases the time to complete each stage of the exam process rises significantly. With 120 client machines, every second that a client takes to complete a task equates to 2 minutes for the lab as a whole. 30 seconds is not an unreasonable time for a client machine to transfer all the files it needs, generate MD5 crypted passwords for 100 users, shut down all non-essential system processes, change runlevel, and restart X. Unfortunately that means it would take an hour for the whole lab to startup. The current version of the server has some very simplistic multi-threading capabilities (call **fork()** for groups of 5 client requests), but it can still take a while for the whole set of client machines to complete intensive tasks (the initial startup is far and away the longest lexis process; dumps and file transfers complete in a matter of seconds for the whole lab).

6 Security

First and foremost, Lexis is a security product. Its sole function is to provide a safe environment for taking exams. Its success is measured by how successful it is in that area: ie. how secure is Lexis?

6.1 Client

If a candidate obtained root privileges, they would be able to circumvent or disable all the restrictions enforced by Lexis. For example, they would be able to drop firewall rules, connect to other hosts on the network, and access stored files via NFS.

Root privileges could be gained by a number of means: using the root password, rebooting the machine to single user mode, using a boot floppy, or installing a Trojan Horse on the client machine before the exam. Lexis takes a number of approaches to prevent successful exploitation of any of these techniques.

The root password is unique to each lexis exam, and is only stored on the local machine in an MD5 encrypted form. Any rebooting of the machine will generate a warning on the server when the ssh connection is dropped. The local LILO configuration is protected with a password to prevent booting in single user mode. The boot sequence can be re-ordered in the PC BIOS to prevent booting from floppy (although this cannot be easily automated).

Making use of a trojan horse would require root access prior to exam, although even if this were done, set-uid binaries would not be effective. The greatest risk from an approach such as this would be to hide unauthorised information on the machine. The candidate would have to do this to all machines that might be used for the test in order for it to work. A tool such as tripwire[**?**] might be useful for checking system integrity if this sort of exploit were a concern.

In general, a large effort is required to subvert Lexis; easy attacks are already blocked, risky attacks such as rebooting would be easily visible to exam invigilators or the lexis adminstrator during an exam, and other attacks require previous root access to the workstation, which could also be detected.

6.2 Server

The security of the server is of paramount concern; the root user on the lexis server can get root access to any lexis client. They would also have full access to the dumps. Lexis does not provide specific security for the server, as the setup will vary greatly depending on available tools, site policy, security awareness of academics involved in the exam, and also the general setup of the network (DNS servers, NFS servers if needed, and so on).

Lexis depends on DNS resolution for the forward and reverse lookup of client hostnames. This could be provided on the server, and so the server could then be firewalled exclusively to the lexis clients. The approach we took was to use ipchains to restrict the server to the local network (not just the lexis clients),

and close all ports except ssh, while restricting ssh access to the minimum subset of users who needed access to the server for the exam.

The possible attacks we have considered are: security compromise by client, DOS by client to prevent other candidates finishing exam, DOS from outside, security compromise from outside to tamper with stored dumps. None of these are easily solved by a simple toolkit approach- each lexis server will have different security requirements depending on the importance of the exam, the environment, other uses of the machine, means of transferring submitted exam answers to markers.

The server is a much more traditional security problem than the lexis client, as it needs to be secure before, during, and after exam. There is the usual compromise between ease and speed of use against security risks. The policy on each site must be the responsibility of the exam officer, but a good basis is minimal services, firewalled to lexis clients only during exam, encrypted dumps, restricted logins to exam personnel only. Lexis does not yet support encrypted dumps, but the feature would be simple to add, whether a symmetric key is set by the exam coordinator at the start of the session, or alternatively encrypting each dump for the users who are going to mark it (this depends on a reliable Public Key Infrastructure).

7 Lexis in use

Lexis was developed in order to satisfy a requirement from the Academic Committee that the First Year undergraduate programming exam would be taken on lab machines. That requirement gave us a strict deadline for completion of development and testing of Lexis, including a successful demonstration that the system would be stable and reliable in use.

While we were confident that the techniques used by Lexis were secure and met the needs of the examiners, we had no way of knowing how well the system would scale, how it would perform under load, and how it would cope with unexpected failures.

Early testing revealed a number of problems with the communication between server and client. Client crashes would cause a fatal error on the server when it tried to read from the filehandle connected to the client. Server crashes would leave zombie **lexis_active** processes running on the clients. These problems were successfully resolved by simplifying the client code and extending the server. We made the client block on input, so when the channel died, it would simply exit. The server was made much more resilient, trapping the PIPE signal, and removing clients from the active connection list at the first problem.

Unfortunately, these changes meant we had to sacrifice some functionality on the client; we had hoped to be able to asynchronously notify the server on significant events (login, logout, reboot, attempted network access, syslog messages), but there was no way to achieve this with the simpler client.

7.1 First test

The first proper test of Lexis was supposed to be a normal programming test, much like the many that had been taken before, only this time with Lexis providing security. Unfortunately, a known bug in the lab software occurred during a demonstration of Lexis to the test coordinator. Even though the problem was completely unrelated to Lexis, the coordinator didn't feel confident enough to run the actual test with Lexis. There was a great deal of disappointment all round, and there was still the problem of successfully demonstrating a full Lexis test before the main exam two weeks later.

The day before the main exam, a number of students were due to sit another programming test. This would be the final chance for Lexis to prove itself before the big exam, and also the largest number of clients tried so far.

At this stage, the server was still using a single thread of execution, processing each client sequentially. It was painfully slow, but it was also reliable, coping with all the failure cases the Teaching Assistants could think up— rebooting the client, unplugging a client completely and asking for the files to be restored elsewhere, deleting files and asking for the originals to be restored. Likewise, the system proved resilient against the security attacks they

attempted— all unauthorised network packets were blocked. They tried sending mail, and although the command succeeded, the messages were only queued on the client machine, and could not be sent on until the firewall rules were lifted.

The test coordinator emailed us to say:

> "Thanks very much. Lets hope it goes as smoothly tomorrow as it did today."

However, the speed issue was critical. With about 40 machines taking part in the programming test, it had taken over 30 minutes to get them all into an exam state. With 160 machines scheduled for use the following day, we could not afford a two hour wait for the system to start. Given that the system was basically reliable, and a complete rewrite was out of the question given the time restrictions, we needed to find a simple way to speed up lexis operations.

The solution we settled on at the time was to use a very simple **fork()**-based approach: each request going to more than one client machine would be broken down into batches of 5 (selectable at runtime) and a new process forked to execute each batch. While this would increase our resource requirements, it increased the responsiveness of the system by an order of magnitude without compromising the security or reliability of the already-tested code.

7.2 First Lexis Exam, 21st March 2001

The computer labs were cleared the night before the main exam, and we started Lexis before the students arrived. While we had considered having a separate server for each area of the labs (this exam used four of the five rooms we had available), in the end we were able to coordinate and run the entire exam from one server. There were 160 machines, and 110 candidates.

The exam got underway with very few problems. One student had difficulty accessing files immediately after logging in, but transferred to a spare machine straight away. The problem turned out to be a corrupted filesystem from a prior hardware fault that no-one had bothered reporting before.

A short time later we received a number of reports of exam files being corrupted. Specifically, a library file provided by lexis in */exam* and vital to the exam was being over-written with binary data. This caused a minor panic among the exam administators who had a number of distressed candidates unable to continue their work. It was very simple to send out fresh copies of the file in question to all the affected clients. That enabled the candidates to continue while we analyzed the cause.

Again, the problem was not actually caused by Lexis. An urgent investigation revealed that the library was being overwritten by graphics data, specifically a screen shot of the file manager. It turned out that one of the common keystroke combinations in the editor used by the candidates caused the file manager to dump a screen shot of the current window into the selected file. Once that was sorted out, the exam continued in a routine fashion.

We used **lexis_who** to print out a list of which candidates were using which machine, which was then checked off against the list prepared by the exam coordinator. This revealed several machines where earlier errors had caused the server to drop the connection to the client. We had assumed this would make the machine unusable for the client, but lexis clients proved to be more robust than we thought, and the candidates were still using the machines. We added them back into the client list and they responded and started dumping again.

In response to this problem, **lexis_server** has been amended to check for dropped clients that should be active.

Automatic dumps were happening every five minutes for over three hours. In total we took 6600 dumps, totalling over 60MB of data.

We received no complaints from the students, and those we spoke to after the exam were greatly in favour of Lexis exams over paper-based exams, especially when it came to programming.

8 Conclusions

We believe that Lexis is the first general tool for managing on-line paperless exams on the Linux platform. Lexis enables computing skills to be securely examined in an environment that provides the same tools

that the candidates are used to. Lexis can be used for any type of exam, from a multiple choice quiz to a full essay paper, although it is especially suited to situations where computers are a normal tool for the task in question.

Lexis is not designed to completely automate the process of University examinations- it won't start and stop exams by itself, won't grant extra time for latecomers, it can't mark the answers, and it certainly can't write the questions. What it can do is provide a secure framework for managing minimum privilege access to a local network of linux workstations, while automatically backing up files at regular intervals. These facilities can be put to a number of uses, not limited to exams or tests.

One application that has been discussed with us is that of kiosk systems: a series of Linux workstations available for public use in an insecure environment. Lexis could be used to restrict user activity on the kiosk machines, while also restricting network access to securely maintained proxy servers for access to email or the web. This approach would significantly cut down the potential for abuse of the systems. The advantages of using Lexis on this type of system are that it works on a standard install, and is avilable now, so it could be used as temporary protection while a dedicated system was developed.

Lexis was developed by system administration personnel to support a decision by the Academic Committee. The academics wanted a computer-based examination system for reasons of convenience, progress, and to satisfy student requests. The project progressed with the academics requesting features and suggesting failure scenarios, and the systems group suggesting pros and cons of various strategies and providing a system security perspective. Unusually for this type of collaboration, the academics were happy to accept the security restrictions, and the developers were able to provide all the requested features.

What does the future hold for Lexis? We have just completed another programming test with Lexis, and the coming academic year promises many more. We've just ported Lexis from our old RedHat setup to a new standard SuSE install. That took a day to support SuSE-specific tools and configuration—the same code now runs on both platforms. Other Universities in the UK have expressed an interest in Lexis, and we would like to see it in use at other sites.

Availability

Lexis is released under the GNU Public Licence, and can be downloaded from **http://www.doc.ic.ac.uk/~mw/lexis/**

References

[1] Computing Support Group web pages, **www.doc.ic.ac.uk/csg/**

[2] Linux, Linus Torvalds

[3] RedHat Software, **www.redhat.com**

[4] SuSE, **www.suse.com**

[5] init(8), standard Sys V root process

[6] OpenSSH, **www.openssh.com**

[7] Linux IP Firewalling Chains HOWTO **http://www.samba.org/ipchains/**

[8] Perl, Larry Wall et al

[9] Ruby **http://www.ruby-lang.org**

[10] Linux Standard Base **http://www.lsb.org**

[11] MIT Kerberos **http://www.mit.edu/kerberos/**

[12] Campen, San Diego State University **http://coe.sdsu.edu/eet/Articles/Paperless/start.htm**

[13] Braun,Crable "Administering Exams Electronically: Issues, Techniques, and Assessment", **www.isworld.org/ais.ac.98/proceedings/track26/braun.p**

[14] BBC News, **http://news.bbc.co.uk/hi/english/education/r**

[15] WebCT, **www.webct.com**

[16] blackboard, **www.blackboard.com**

[17] TripWire, **sourceforge.net/projects/tripwire**

The Authors

Mike Wyer is a recent graduate of Imperial College who currently works as a Systems Administrator in the Department of Computing. He has had a long-term interest in examinations and computers, having worked on exam registration in a final-year group project.

Susan Eisenbach is a Reader in the Department of Computing where she is responsible for the teaching programme. Her research interests include programming languages for distributed computing.